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 $i \leq j \in [1..n]$

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

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

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

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

| $T$ | a | b | a | b | c | a | b | c | a | b | b | a | $|
| $SA$ | 13 | 12 | 1 | 9 | 6 | 3 | 11 | 2 | 10 | 7 | 4 | 8 | 5 |
| $LCP$ | 0 | 0 | 1 | 2 | 2 | 5 | 0 | 2 | 1 | 1 | 4 | 0 | 3 |
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

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

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

Suffix Tree
1973

Suffix Array
1993

LCP Array
1993

Memory Requirements

...babab... aabaac...

...83095...

...a a b b a a b b...
(Compressed) Text Indices #Ad

Memory Requirements

- Suffix Tree (1973)
- Suffix Array (1993)
- LCP Array (1993)
- BWT (1994)
- Wavelet Tree (2000)
- FM-Index (2000)
- r-Index (2018)
- Block Tree (2021)

String Sorting

LCE Queries

(Patricia) Tries

Bit Vectors and Rank/Select Queries

EM Hashing

Succinct Data Structures

Compression

abca
ba
cc
83095...

babab...
aabac...

01110001
01001
101
10
11
0

a: 0
b: 4
c: 5

abccaca0
0110001
01001
1001
0
10
11
0

abccaaca
01110001
01001
101
10
11
0

a: 0
b: 4
c: 5

JEA '21, ALENEX '18,'20, SPIRE '19

Submitted (arXiv:2205.04745)

ALENEX '19, BigData '18

Florian Kurpicz | Advanced Data Structures | 08 Compressed Suffix Array

Institute of Theoretical Informatics, Algorithm Engineering
(Compressed) Text Indices #Ad
(Compressed) Text Indices #Ad

Memory Requirements

- **Suffix Tree**
  - Year: 1973
  - Diagram:

- **Suffix Array**
  - Year: 1993
  - Diagram:

- **LCP Array**
  - Year: 1993
  - Diagram:

- **BWT**
  - Year: 1994
  - Diagram:

- **Wavelet Tree**
  - Year: 2000
  - Diagram:

- **FM-Index**
  - Year: 2000
  - Diagram:

- **r-Index**
  - Year: 2018
  - Diagram:

- **Block Tree**
  - Year: 2021
  - Diagram:

- **Compression**
  - Diagram:

- **String Sorting**

- **LCE Queries**

- **(Patricia) Tries**

- **Bit Vectors and Rank/Select Queries**

- **EM Hashing**

- **Succinct Data Structures**

- **Compression**
The definition of the \( \Psi \) function is given as follows:

**Definition: \( \Psi \) Function**

Given a suffix array \( SA \) of length \( n \),

\[
\Psi(i) = SA^{-1}[SA[i] + 1]
\]
**Definition: Ψ Function**

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

$$\Psi(i) = SA^{-1}[SA[i] + 1]$$
**Ψ Function**

Definition: Ψ Function

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

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

<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>

|   | $|$ | a | $|$ | a | a | a | b | b | b | b | c | c | a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|   | a | b | a | b | b | b | b | b | a | b | a | b | b |
|   | a | b | a | c | c | c | a | b | b | c | b | b | c |
|   | a | b | c | c | b | a | b | c | a | a | c | b | a |
|   | a | b | a | b | a | b | b | a | $|$ | a | b | b | $|$ |
|   | a | b | a | b | a | b | b | a | $|$ | a | b | b | $|$ |
Definition: \( \Psi \) Function

Given a suffix array \( SA \) of length \( n \),

\[
\Psi(i) = SA^{-1}[SA[i] + 1]
\]
**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
Definition: $\Psi$ 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 \(\text{currently } O(n \log n) \text{ bits}\)
Recovering $\Psi$ from $SA$

<table>
<thead>
<tr>
<th>$\Psi$</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>$\Psi$</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>

| $T$  | a | b | a | b | a | b | c | a | b | c | a | b | a | $\$ |
|------|---|---|---|---|---|---|---|---|---|----|----|----|----|
| $SA$| 13| 12| 1 | 9 | 6 | 3 | 11| 2 | 10| 7 | 4 | 8 | 5  |

Replacing $SA$ with $\Psi$

Which number does *in this example* not occur?
Replacing $SA$ with $\Psi$

Which number does in this example not occur?
Answer: 3
Replacing $SA$ with $\Psi$

- which number does in this example not occur? Answer: 3
- how to obtain $SA[j]$ using $\Psi$

| $T$  | a | b | a | b | c | a | b | c | a | b | b | a | $  
| SA  | 13| 12| 1 | 9 | 6 | 3 | 11| 2 | 10| 7 | 4 | 8 | 5  
| $\Psi$ | - | 1 | 8 | 9 | 10| 11| 2 | 6 | 7 | 12 | 13| 4 | 5  

| $\Psi$ | a | a | a | b | a | b | b | a | b | c | c | a | a  
| $\Psi$ | b | b | b | c | c | $|b$ | b | b | c | a | b | b  
| $\Psi$ | c | c | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | a | b | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | b | b | $|b$ | b | a | $|b$ | b | a | c | a | b | a  

| $\Psi$ | c | c | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | a | b | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | b | b | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | c | c | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | a | b | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | b | b | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | c | c | $|b$ | b | a | $|b$ | b | a | c | a | b | a  
| $\Psi$ | a | b | $|b$ | b | a | $|b$ | b | a | c | a | b | a
Replacing $SA$ 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
- $n - \#\text{steps}$ is SA value
- requires $O(n)$ time
Replacing $SA$ 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
- $n - \#\text{steps}$ 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
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

<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$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>$</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>$</td>
<td>$</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>$</td>
<td>b</td>
<td>$</td>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>$</td>
<td>b</td>
<td>a</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

<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$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>-</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>-</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td>-</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td>-</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</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

<table>
<thead>
<tr>
<th>$\Psi$</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>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</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>
<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>

what is better? PINGO

<table>
<thead>
<tr>
<th>$\Psi$</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>$\Psi$</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>

```plaintext
1 2 3 4 5 6 7 8 9 10 11 12 13
```

```plaintext
Ta b a b c a b c a b b a $\$
```

```
SA 13 12 1 9 6 3 11 2 10 7 4 8 5
```

```
Ψ - 1 8 9 10 11 2 6 7 12 13 4 5
```

<table>
<thead>
<tr>
<th>$\Psi$</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>$\Psi$</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>

```
Ψ - 1 8 9 10 11 2 6 7 12 13 4 5
```

<table>
<thead>
<tr>
<th>$\Psi$</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>$\Psi$</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>

```
Ψ - 1 8 9 10 11 2 6 7 12 13 4 5
```

```
Ψ - 1 8 9 10 11 2 6 7 12 13 4 5
```
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

<table>
<thead>
<tr>
<th>$\Psi$</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>$^A$</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>
<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>

$^A$ 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
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

<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>$a$</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>
</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>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>b</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$c$</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td>$\Psi$-</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>
</tbody>
</table>
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: $O(\log n)$
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: $O(\log n)$
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: \( O(\log n) \)
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: $O(\log n)$
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></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>$$</td>
</tr>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>$$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>$$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>$$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>$$</td>
</tr>
</tbody>
</table>
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></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>

<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>$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>$a$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>$a$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>$a$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>$a$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>$a$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>$a$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

8/19 2024-06-10 Florian Kurpicz | Advanced Data Structures | 08 Compressed Suffix Array Institute of Theoretical Informatics, Algorithm Engineering
Speeding Up Lookups in $\Psi$ (2/2)

1. every log $n$-th entry in $\Psi$
2. every log $n$-th step in $\Psi$
3. 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></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>$$</td>
</tr>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$$</td>
<td>a</td>
<td>$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>$$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>a</td>
</tr>
</tbody>
</table>
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 $T[SA[i+1]..n] \leq T[SA[i+1]+1..n]$

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

Note that not all increasing intervals belong to the same character.

Example on the board
does $\Psi$ have some structure?

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

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

/chalkboard-◎eacher
Compressing Ordered Sequences

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

\[\Delta\text{-Encoding}\]
- store difference between entries
- scanning whole sequence up to value when decoding

\[\text{Elias-Fano (Lecture 05)}\]
- upper and lower halves
- upper half represented in bit vector \( p_i + i \)
- lower half plain bit compressed

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 10 & 20 & 21 & 22 & 32 \\
0 & 1 & 2 & 4 & 7 & 10 & 20 & 21 & 22 & 30 & & & \\
\end{array}
\]

- 0: 000000
- 1: 000001
- 2: 000010
- 4: 000100
- 7: 000111
- 10: 001010
- 20: 010100
- 21: 010101
- 22: 010110
- 30: 100000

Upper: 11101101000111000100
Lower: 00 01 10 00 11 10 00 01 10 00
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>Number</th>
<th>Tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>2</td>
<td>000010</td>
</tr>
<tr>
<td>3</td>
<td>000100</td>
</tr>
<tr>
<td>4</td>
<td>000111</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>001010</td>
</tr>
<tr>
<td>20</td>
<td>010100</td>
</tr>
<tr>
<td>21</td>
<td>010101</td>
</tr>
<tr>
<td>22</td>
<td>010110</td>
</tr>
<tr>
<td>30</td>
<td>100000</td>
</tr>
</tbody>
</table>

Upper: \( 11101101000111000100 \)

Lower: \( 00011000111000011000 \)
Recap: Elias-Fano Coding Space

Lemma: Elias-Fano Coding

Given an array containing \( n \) distinct integers from a universe \( U = [0, n) \), 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)

\[ u = 512, \quad n = 8, \quad q = 64 \]

\( (0, 3, 17, 89, 128, 132, 500, 511) \)

\( (0, 0), (0, 3), (0, 7), (1, 25), (2, 0), (2, 4), (7, 52), (7, 63) \)

store quotient \((x/q)\) using Elias-Fano

store remainder \((x \mod q)\) 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
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 $(x/q, x\%q)$

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
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 \( (x/q, x\%q) \)

- \( u = 512, n = 8, q = 64 \)
- \( (0, 3, 17, 89, 128, 132, 500, 511) \)
- \( \{0, 0\}, \{0, 3\}, \{0, 7\}, \{1, 25\}, \{2, 0\}, \{2, 4\}, \{7, 52\}, \{7, 63\} \)
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
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
Elias-Fano coding for each increasing interval
- 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 $(x/q, x \% q)$

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

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

more precise: two additional bits per character

$u = 512, n = 8, q = 64$

$(0, 3, 17, 89, 128, 132, 500, 511)$

$\{0, 0\}, \{0, 3\}, \{0, 7\}, \{1, 25\}, \{2, 0\}, \{2, 4\}, \{7, 52\}, \{7, 63\}$
**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
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>
<thead>
<tr>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$ a b a b c a b c a b b a $</td>
</tr>
<tr>
<td><strong>SA</strong> 13 12 1 9 6 3 11 2 10 7 4 8 5</td>
</tr>
<tr>
<td>$\Psi$ - 1 8 9 10 11 2 6 7 12 13 4 5</td>
</tr>
<tr>
<td><strong>NEW</strong> 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
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
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?
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

<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>

Right half (SA) still big
How to recurs?
SA half consists only of odd values
- for value $x$ store $(x - 1)/2$
- reversible since all values are odd
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

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

<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

<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>$Ψ$</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></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>
```
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>$Ψ$</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

<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>( \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>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>
</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>
</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

<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)

- recurses \( \log \log n \) 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.

<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>
**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

<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>
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

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

