Text Indexing

Lecture 05: Burrows-Wheeler Transform

Florian Kurpicz
Recap: Text-Compression

**Definition: LZ77 Factorization [ZL77]**

Given a text $T$ of length $n$ over an alphabet $\Sigma$, the LZ77 factorization is
- a set of $z$ factors $f_1, f_2, \ldots, f_z \in \Sigma^+$, such that
- $T = f_1 f_2 \ldots f_z$ and for all $i \in [1, z]$ $f_i$ is
- single character not occurring in $f_1 \ldots f_{i-1}$ or
- longest substring occurring $\geq 2$ times in $f_1 \ldots f_i$

$T = \text{abababbabababa}$
- $f_1 = a$
- $f_2 = b$
- $f_3 = abab$
- $f_4 = bbb$
- $f_5 = aba$
- $f_6 = $

**Definition: LZ78 Factorization [ZL78]**

Given a text $T$ of length $n$ over an alphabet $\Sigma$, the LZ78 factorization is
- a set of $z$ factors $f_1, f_2, \ldots, f_z \in \Sigma^+$, such that
- $T = f_1 f_2 \ldots f_z$, $f_0 = \epsilon$ and for all $i \in [1, z]$
- if $f_1 \ldots f_{i-1} = T[1..j-1]$, then $f_i$ is the longest prefix of $T[j..n]$, such that
  $$\exists k \in [0, i), \alpha \in \Sigma \cup \{\}$ : $f_k = f_i \alpha$$

$T = \text{abababbabababa}$
- $f_1 = a$
- $f_2 = b$
- $f_3 = ab$
- $f_4 = abb$
- $f_5 = bb$
- $f_6 = aba$
- $f_7 = $
Burrows-Wheeler Transform \([BW94]\) (1/2)

**Definition: Burrows-Wheeler Transform**

Given a text \(T\) of length \(n\) and its suffix array \(SA\), for \(i \in [1, n]\) the Burrows-Wheeler transform is

\[
BWT[i] = \begin{cases} 
T[SA[i] - 1] & SA[i] > 0 \\
$ & SA[i] = 0
\end{cases}
\]

The definition is not very descriptive. An easy way to compute the BWT is to find the way to make it faster. What can we do with the BWT?
Definition: Burrows-Wheeler Transform

Given a text $T$ of length $n$ and its suffix array $SA$, for $i \in [1, n]$ the Burrows-Wheeler transform is

$$BWT[i] = \begin{cases} T[SA[i] - 1] & SA[i] > 0 \\ $ & SA[i] = 0 \end{cases}$$

---

<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>$LCP$</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>
</tr>
<tr>
<td>$BWT$</td>
<td>a</td>
<td>b</td>
<td>$c$</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>
**Definition: Burrows-Wheeler Transform**

Given a text $T$ of length $n$ and its suffix array $SA$, for $i \in [1, n]$ the Burrows-Wheeler transform is

$$BWT[i] = \begin{cases} T[SA[i] - 1] & SA[i] > 0 \\ $ & SA[i] = 0 \end{cases}$$

- character before the suffix in $SA$-order
- choose characters cyclic $\$ for first suffix

<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$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$BWT$</td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

Burrows-Wheeler Transform $[BW94]$ (1/2)
Definition: Burrows-Wheeler Transform

Given a text $T$ of length $n$ and its suffix array $SA$, for $i \in [1, n]$ the Burrows-Wheeler transform is

$$BWT[i] = \begin{cases} T[SA[i] - 1] & SA[i] > 0 \\
$ & SA[i] = 0 \end{cases}$$

- character before the suffix in $SA$-order
- choose characters cyclic $\$ for first suffix

- can compute $BWT$ in $O(n)$ time
- for binary alphabet $O(n/\sqrt{\log n})$ time and $O(n/\log n)$ words space is possible [KK19]
Burrows-Wheeler Transform [BW94] (1/2)

Definition: Burrows-Wheeler Transform
Given a text $T$ of length $n$ and its suffix array $SA$, for $i \in [1, n]$ the Burrows-Wheeler transform is

$$BWT[i] = \begin{cases} T[SA[i] - 1] & SA[i] > 0 \\ $ & SA[i] = 0 \end{cases}$$

- character before the suffix in $SA$-order
- choose characters cyclic $\$ for first suffix

- can compute $BWT$ in $O(n)$ time
- for binary alphabet $O(n / \sqrt{\lg n})$ time and $O(n / \lg n)$ words space is possible [KK19]

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

- definition is not very descriptive
- easy way to compute $BWT$
- what can we do with the $BWT$
**Burrows-Wheeler Transform (2/2)**

**Definition: Cyclic Rotation**

Given a text $T$ of length $n$, the $i$-th cyclic rotation is

$$T^{(i)} = T[i..n]T[1..i]$$

- $i$-th cyclic rotation is concatenation of $i$-th suffix and $(i-1)$-th prefix

---

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\tau(1)$</th>
<th>$\tau(2)$</th>
<th>$\tau(3)$</th>
<th>$\tau(4)$</th>
<th>$\tau(5)$</th>
<th>$\tau(6)$</th>
<th>$\tau(7)$</th>
<th>$\tau(8)$</th>
<th>$\tau(9)$</th>
<th>$\tau(10)$</th>
<th>$\tau(11)$</th>
<th>$\tau(12)$</th>
<th>$\tau(13)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ababcabcabba$</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>$b$</td>
</tr>
<tr>
<td></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>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></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>$a$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td></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>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></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>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
</tbody>
</table>
**Definition: Cyclic Rotation**

Given a text $T$ of length $n$, the $i$-th cyclic rotation is

$$T^{(i)} = T[i..n]T[1..i]$$

- $i$-th cyclic rotation is concatenation of $i$-th suffix and $(i - 1)$-th prefix

**Definition: Burrows-Wheeler Transform (alt.)**

Given a text $T$ and a matrix containing all its cyclic rotations in lexicographical order as columns, then the **Burrows-Wheeler transform** of the text is the last row of the matrix

$$T = ababcabcabba\$\,$$

$$\begin{array}{cccccccccccc}
T^{(1)} & T^{(2)} & T^{(3)} & T^{(4)} & T^{(5)} & T^{(6)} & T^{(7)} & T^{(8)} & T^{(9)} & T^{(10)} & T^{(11)} & T^{(12)} & T^{(13)} \\
\hline 
 a & b & a & b & c & a & b & c & a & b & b & a & \$ \\
b & a & b & c & a & b & c & a & b & b & a & \$ & a \\
a & b & c & a & b & c & a & b & b & a & \$ & a \\
b & c & a & b & c & a & b & b & a & \$ & a & b & a & b \\
c & a & b & c & a & b & b & a & \$ & a & b & a & b & a \\
a & b & c & a & b & b & a & \$ & a & b & a & b & a & b \\
b & c & a & b & b & a & \$ & a & b & a & b & a & b & a \\
c & a & b & b & a & \$ & a & b & a & b & a & b & a & c \\
a & b & b & a & \$ & a & b & a & b & a & b & c \\
b & b & a & \$ & a & b & a & b & c & a & b \\
a & b & a & \$ & a & b & a & b & c & b & a \\
b & a & \$ & a & b & a & b & c & b & a & b \\
a & \$ & a & b & a & b & c & b & a & b & b \\
\$ & a & b & a & b & c & b & a & b & b & a \\
\end{array}$$
**Burrows-Wheeler Transform (2/2)**

**Definition: Cyclic Rotation**

Given a text $T$ of length $n$, the $i$-th **cyclic rotation** is

$$T^{(i)} = T[i..n]T[1..i]$$

- $i$-th cyclic rotation is concatenation of $i$-th suffix and $(i - 1)$-th prefix

**Definition: Burrows-Wheeler Transform (alt.)**

Given a text $T$ and a matrix containing all its cyclic rotations in lexicographical order as **columns**, then the **Burrows-Wheeler transform** of the text is the last **row** of the matrix
Definition: Cyclic Rotation
Given a text $T$ of length $n$, the $i$-th cyclic rotation is

$$T^{(i)} = T[i..n]T[1..i]$$

- $i$-th cyclic rotation is concatenation of $i$-th suffix and $(i - 1)$-th prefix

Definition: Burrows-Wheeler Transform (alt.)
Given a text $T$ and a matrix containing all its cyclic rotations in lexicographical order as columns, then the Burrows-Wheeler transform of the text is the last row of the matrix

$$T = \text{ababcabcabba}\$$

<table>
<thead>
<tr>
<th>$\tau(13)$</th>
<th>$\tau(12)$</th>
<th>$\tau(1)$</th>
<th>$\tau(9)$</th>
<th>$\tau(6)$</th>
<th>$\tau(3)$</th>
<th>$\tau(11)$</th>
<th>$\tau(2)$</th>
<th>$\tau(10)$</th>
<th>$\tau(7)$</th>
<th>$\tau(4)$</th>
<th>$\tau(8)$</th>
<th>$\tau(5)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>$$$</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>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>a</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>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>$$$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>
First and Last Row

- two important rows in the matrix
- other rows are not needed at all
- there is a special relation between the two rows
  
  later this lecture

**First Row** $F$
- contains all characters or the text in sorted order

**Last Row** $L$
- is the BWT itself

$$T = ababcabcabba\$$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$a$</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>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$a$</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>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
</tbody>
</table>
Properties of the BWT: Rank of Characters

Definition: Rank

Given a text $T$ over an alphabet $\Sigma$, the rank of a character at position $i \in [1, n]$ is

$$rank(i) = |\{j \in [1, i]: T[i] = T[j]\}|$$

- rank is number of same characters that occur before in the text
- mark ranks of characters w.r.t. text not BWT

<table>
<thead>
<tr>
<th>$T = ababcabcabba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau(13)\tau(12)\tau(1)\tau(9)\tau(6)\tau(3)\tau(11)\tau(2)\tau(10)\tau(7)\tau(4)\tau(8)\tau(5)$</td>
</tr>
<tr>
<td>$\tau(13)\tau(12)\tau(1)\tau(9)\tau(6)\tau(3)\tau(11)\tau(2)\tau(10)\tau(7)\tau(4)\tau(8)\tau(5)$</td>
</tr>
<tr>
<td>F $\tau(13)$ $\tau(12)$ $\tau(1)$ $\tau(9)$ $\tau(6)$ $\tau(3)$ $\tau(11)$ $\tau(2)$ $\tau(10)$ $\tau(7)$ $\tau(4)$ $\tau(8)$ $\tau(5)$</td>
</tr>
<tr>
<td>$\tau(13)$ $\tau(12)$ $\tau(1)$ $\tau(9)$ $\tau(6)$ $\tau(3)$ $\tau(11)$ $\tau(2)$ $\tau(10)$ $\tau(7)$ $\tau(4)$ $\tau(8)$ $\tau(5)$</td>
</tr>
<tr>
<td>$F$ $a$ $a$ $a$ $a$ $a$ $b$ $b$ $b$ $b$ $b$ $c$ $c$ $c$</td>
</tr>
<tr>
<td>a $b$ $b$ $b$ $b$ $a$ $a$ $b$ $c$ $c$ $a$ $a$ $a$</td>
</tr>
<tr>
<td>b $a$ $a$ $b$ $c$ $c$ $b$ $a$ $a$ $a$ $b$ $b$ $b$</td>
</tr>
<tr>
<td>a $b$ $b$ $a$ $a$ $a$ $c$ $b$ $b$ $b$ $c$ $c$ $c$</td>
</tr>
<tr>
<td>b $a$ $c$ $b$ $b$ $b$ $a$ $a$ $b$ $c$ $a$ $a$ $a$</td>
</tr>
<tr>
<td>c $b$ $a$ $a$ $b$ $c$ $a$ $b$ $b$ $a$ $a$ $a$ $a$</td>
</tr>
<tr>
<td>a $c$ $b$ $b$ $a$ $b$ $c$ $a$ $a$ $b$ $b$ $a$ $b$ $b$</td>
</tr>
<tr>
<td>b $a$ $c$ $a$ $b$ $c$ $a$ $b$ $b$ $a$ $a$ $b$ $a$ $a$</td>
</tr>
<tr>
<td>c $b$ $a$ $b$ $a$ $b$ $a$ $b$ $c$ $b$ $a$ $a$ $a$</td>
</tr>
<tr>
<td>a $c$ $b$ $c$ $b$ $a$ $b$ $b$ $a$ $a$ $a$ $b$ $a$</td>
</tr>
<tr>
<td>b $a$ $b$ $a$ $a$ $c$ $a$ $b$ $a$ $c$ $b$ $a$ $b$ $b$</td>
</tr>
<tr>
<td>b $b$ $a$ $b$ $a$ $a$ $c$ $c$ $b$ $a$ $a$ $a$ $b$ $b$</td>
</tr>
</tbody>
</table>
**Definition: Rank**

Given a text $T$ over an alphabet $\Sigma$, the rank of a character at position $i \in [1, n]$ is

$$\text{rank}(i) = |\{ j \in [1, i] : T[i] = T[j]\}|$$

- rank is number of same characters that occur before in the text
- mark ranks of characters w.r.t. text not BWT

Properties of the BWT: Rank of Characters

$T = \text{ababcabcabba}$

<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>rank</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

$F$

<table>
<thead>
<tr>
<th>$\tau(13)$</th>
<th>$\tau(12)$</th>
<th>$\tau(1)$</th>
<th>$\tau(9)$</th>
<th>$\tau(6)$</th>
<th>$\tau(3)$</th>
<th>$\tau(11)$</th>
<th>$\tau(2)$</th>
<th>$\tau(10)$</th>
<th>$\tau(7)$</th>
<th>$\tau(4)$</th>
<th>$\tau(8)$</th>
<th>$\tau(5)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>c</td>
<td>$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>$$</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>
</tr>
<tr>
<td>c</td>
<td>b</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>a</td>
<td>$$</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

$L$
Properties of the BWT: Rank of Characters

Definition: Rank

Given a text $T$ over an alphabet $\Sigma$, the rank of a character at position $i \in [1, n]$ is

$$\text{rank}(i) = |\{j \in [1, i] : T[i] = T[j]\}|$$

- rank is number of same characters that occur before in the text
- mark ranks of characters w.r.t. text not BWT

<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>rank</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

$T = \text{ababcabcabba}\$

\[\begin{array}{cccccccccccccccc}
\tau(1)&\tau(2)&\tau(3)&\tau(4)&\tau(5)&\tau(6)&\tau(7)&\tau(8)&\tau(9)&\tau(10)&\tau(11)&\tau(12)&\tau(13)
\hline
F & 1 & 5 & 1 & 4 & 3 & 2 & 5 & 1 & 4 & 3 & 2 & 1 & 2 \\
\end{array}\]

\[\begin{array}{cccccccccccccccc}
T & a & b & a & b & c & a & b & c & a & b & b & a & $\$
\end{array}\]

\[\begin{array}{cccccccccccccccc}
L & a & b & $ & c & c & b & b & a & a & a & a & b & b \\
\end{array}\]
Properties of the BWT: Rank of Characters

Definition: Rank

Given a text $T$ over an alphabet $\Sigma$, the rank of a character at position $i \in [1, n]$ is

$$rank(i) = |\{j \in [1, i]: T[i] = T[j]\}|$$

- rank is number of same characters that occur before in the text
- mark ranks of characters w.r.t. text not BWT

$T = ababcabcabba$

<table>
<thead>
<tr>
<th>T</th>
<th>a b a b c a b c a b b a $</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank</td>
<td>1 1 2 2 1 3 3 2 4 4 5 5 1</td>
</tr>
</tbody>
</table>
Properties of the BWT: Rank of Characters

**Definition: Rank**

Given a text $T$ over an alphabet $\Sigma$, the *rank* of a character at position $i \in [1, n]$ is

$$\text{rank}(i) = |\{j \in [1, i] : T[i] = T[j]\}|$$

- rank is number of same characters that occur before in the text
- mark ranks of characters w.r.t. text not BWT

$T = ababcabcabba$

<table>
<thead>
<tr>
<th>Rank</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>3</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>4</th>
<th>5</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
</table>

$T$ = a b a b c a b c a b b a $

| Rank | 1 | 1 | 2 | 2 | 1 | 3 | 3 | 2 | 4 | 4 | 5 | 5 | 1 |
Properties of the BWT: Rank of Characters

**Definition: Rank**

Given a text $T$ over an alphabet $\Sigma$, the rank of a character at position $i \in [1, n]$ is

$$\text{rank}(i) = |\{j \in [1, i] : T[i] = T[j]\}|$$

- rank is number of same characters that occur before in the text
- mark ranks of characters w.r.t. text not BWT
- order of ranks is the same in first and last row

$T = ababcabcabba$

<table>
<thead>
<tr>
<th>Rank</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
</tr>
</tbody>
</table>

$\tau(1) \tau(2) \tau(3) \tau(4) \tau(5) \tau(6) \tau(7) \tau(8) \tau(9) \tau(10) \tau(11) \tau(12) \tau(13)$
LF-Mapping (1/2)

- want to map characters from last to first row
- why do we want this?
  - helps with pattern matching
  - transform BWT back to $T$

**Definition: LF-mapping**

Given a text $T$ of length $n$ and its suffix array $SA$, then the LF-mapping is a permutation of $[1, n]$, such that

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

- similar to definition of BWT
- requires $SA$ or explicitly saving LF-mapping

\[
T = ababcabcabba$
\]

| F | $|a|a|a|a|a|b|b|b|b|b|c|c|
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | $|b|b|b|b|a|a|b|c|c|a|a|
| b | a | a | b | c | c | $ | b | a | a | b | b|
| a | b | b | a | a | a | a | c | $ | b | b | b|
| b | a | c | $ | b | b | b | a | a | b | c | a |
| c | b | a | a | b | c | a | b | b | a | a | $ |
| a | c | b | b | a | a | b | c | a | $ | b | b |
| b | a | c | $ | b | c | a | b | a | b | b | b |
| c | b | a | b | a | b | a | b | c | b | a | a |
| a | c | b | c | b | a | b | b | a | a | $ | b | a |
| b | a | b | a | a | $ | c | a | b | b | a | c | b |
| b | b | a | b | b | a | a | $ | c | c | b | a | a |
| a | b | $ | c | c | b | b | a | a | a | a | b | b |

| L | a | b | $ | c | c | b | b | a | a | a | a | b | b |
LF-Mapping (1/2)

- want to map characters from last to first row
- why do we want this?
  - helps with pattern matching
  - transform BWT back to $T$

**Definition: LF-mapping**

Given a text $T$ of length $n$ and its suffix array $SA$, then the LF-mapping is a permutation of $[1, n]$, such that

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

- similar to definition of BWT
- requires $SA$ or explicitly saving LF-mapping

**Example**

$$T = ababcabcabba$$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\tau(1)$</th>
<th>$\tau(2)$</th>
<th>$\tau(3)$</th>
<th>$\tau(4)$</th>
<th>$\tau(5)$</th>
<th>$\tau(6)$</th>
<th>$\tau(7)$</th>
<th>$\tau(8)$</th>
<th>$\tau(9)$</th>
<th>$\tau(10)$</th>
<th>$\tau(11)$</th>
<th>$\tau(12)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<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>$c$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
</tbody>
</table>

LF-Mapping (2/2)
LF-Mapping (1/2)

- want to map characters from last to first row
- why do we want this?
  - helps with pattern matching
  - transform BWT back to T

Definition: LF-mapping

Given a text $T$ of length $n$ and its suffix array $SA$, then the LF-mapping is a permutation of $[1, n]$, such that

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

- similar to definition of BWT
- requires SA or explicitly saving LF-mapping

$T = ababcabcabba$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>$$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>$$$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

F: Forward

L: Last
**LF-Mapping (1/2)**

- want to map characters from last to first row
- why do we want this?
  - helps with pattern matching
  - transform BWT back to $T$

**Definition: LF-mapping**

Given a text $T$ of length $n$ and its suffix array $SA$, then the LF-mapping is a permutation of $[1, n]$, such that

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

- similar to definition of BWT
- requires $SA$ or explicitly saving LF-mapping

---

**Diagram**

$T = ababcabcabba$

$$\begin{array}{cccccccccccc}
\tau(0) & \tau(1) & \tau(2) & \tau(3) & \tau(4) & \tau(5) & \tau(6) & \tau(7) & \tau(8) & \tau(9) & \tau(10) & \tau(11) & \tau(12) & \tau(13) \\
F & $ & a & a & a & a & a & b & b & b & b & b & b & c & c \\
& a & b & b & b & b & a & a & b & c & c & a & a & b & b \\
& b & a & a & b & c & c & $ & b & a & a & a & b & b & b \\
& a & b & b & a & a & a & a & c & $ & b & b & b & c & \\
& b & a & c & $ & b & b & b & a & a & b & c & a & a & a \\
& c & b & a & a & a & c & a & b & b & a & a & $ & b & \\
& a & b & b & b & a & a & b & c & a & $ & b & b & \\
& b & a & c & $ & b & c & a & b & a & b & b & a & a \\
& c & b & a & b & a & b & a & b & c & b & a & a & $ & \\
& a & c & b & c & b & a & b & b & a & a & $ & b & a & \\
& b & a & a & a & $ & c & a & b & b & a & c & b & \\
& b & a & b & b & a & a & $ & c & c & b & a & a & \\
& b & b & a & b & a & a & $ & c & c & b & a & a & a \\
& b & a & b & b & a & a & a & b & b & b & b & a & c \\
& a & b & $ & c & c & b & b & a & a & a & a & b & b \\
\end{array}$$
**LF-Mapping (1/2)**

- want to map characters from last to first row
- why do we want this?
  - helps with pattern matching
  - transform BWT back to $T$

**Definition: LF-mapping**

Given a text $T$ of length $n$ and its suffix array $SA$, then the LF-mapping is a permutation of $[1, n]$, such that

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

- similar to definition of BWT
- requires $SA$ or explicitly saving LF-mapping

**Example:**

$T = ababcabcabba$

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

**LF-Mapping (2/2)**
want to map characters from last to first row
why do we want this?
  - helps with pattern matching
  - transform BWT back to $T$

**Definition: LF-mapping**

Given a text $T$ of length $n$ and its suffix array $SA$, then the LF-mapping is a permutation of $[1, n]$, such that

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

similar to definition of BWT

requires $SA$ or explicitly saving LF-mapping
### LF-Mapping (2/2)

**Definition: C-Array and Rank-Function**

Given a text $T$ of length $n$ over an alphabet $\Sigma$, $\alpha \in \Sigma$, and $i \in [1, n]$ then

$$C[\alpha] = |i \in [1, n] : T[i] < \alpha|$$

and

$$\text{rank}_\alpha(i) = |\{j \in [1, i] : T[j] = \alpha\}|$$

- $C$ contains total number of smaller characters
- $\text{rank}_\alpha$ contains number of $\alpha$'s in prefix $T[1..i]$
- $\text{rank}_\alpha$ can be computed in $O(1)$ time [FM00]
LF-Mapping (2/2)

Definition: $C$-Array and $Rank$-Function

Given a text $T$ of length $n$ over an alphabet $\Sigma$, $\alpha \in \Sigma$, and $i \in [1, n]$ then

$$C[\alpha] = |i \in [1, n]: T[i] < \alpha|$$

and

$$rank_\alpha(i) = |\{j \in [1, i]: T[j] = \alpha\}|$$

- $C$ contains total number of smaller characters
- $rank_\alpha$ contains number of $\alpha$’s in prefix $T[1..i]$
- $rank_\alpha$ can be computed in $O(1)$ time [FM00]

$T$ = a b a b c a b c a b b a $

$rank$ = 1 1 2 2 1 3 3 2 4 4 5 5 1

- rank now on BWT
- $C$ is exclusive prefix sum over histogram

$\text{BWT}$
LF-Mapping (2/2)

**Definition: C-Array and Rank-Function**

Given a text $T$ of length $n$ over an alphabet $\Sigma$, $\alpha \in \Sigma$, and $i \in [1, n]$ then

$$C[\alpha] = \left| \{ i \in [1, n] : T[i] < \alpha \} \right|$$

and

$$\text{rank}_\alpha(i) = \left| \{ j \in [1, i] : T[j] = \alpha \} \right|$$

- $C$ contains total number of smaller characters
- $\text{rank}_\alpha$ contains number of $\alpha$’s in prefix $T[1..i]$
- $\text{rank}_\alpha$ can be computed in $O(1)$ time [FM00]

**Definition: LF-Mapping (alt.)**

Given a $BWT$, its $C$-array, and its $\text{rank}$-Function, then

$$LF(i) = C[BWT[i]] + \text{rank}_{BWT[i]}(i)$$
Reversing the BWT (1/2)

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text
Reversing the BWT (1/2)

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text
Reversing the BWT (1/2)

- characters (w.r.t. text) preserve order in L and F
- \(LF\)-mapping returns previous character in text

\[
\text{L} = \text{ababcabcabba} \$
\]
Reversing the BWT (1/2)

- characters (w.r.t. text) preserve order in \( L \) and \( F \)
- \( LF \)-mapping returns previous character in text

\[
T = \text{ababcabcabba}$
\]

<table>
<thead>
<tr>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\backslash$</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>L</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

\( LF \)
Reversing the BWT (1/2)

- characters (w.r.t. text) preserve order in L and F
- LF-mapping returns previous character in text

\[ T = \text{ababcabcabba} $\]

\[ L \]
\[ \text{a} \quad \text{b} \quad \$ \quad \text{c} \quad \text{c} \quad \text{b} \quad \text{b} \quad \text{a} \quad \text{a} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \quad \text{b} \quad \text{c} \quad \text{c} \quad \text{b} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \quad \text{a} \quad \text{a} \quad $\]

\[ LF \]
\[ 2 \quad 7 \quad 1 \quad 12 \quad 13 \quad 8 \quad 9 \quad 3 \quad 4 \quad 5 \quad 6 \quad 10 \quad 11 \]
Reversing the BWT (2/2)

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text

<table>
<thead>
<tr>
<th>L</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>$</td>
<td>12</td>
</tr>
<tr>
<td>c</td>
<td>13</td>
</tr>
<tr>
<td>c</td>
<td>8</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>a</td>
<td>11</td>
</tr>
<tr>
<td>a</td>
<td>12</td>
</tr>
<tr>
<td>a</td>
<td>13</td>
</tr>
</tbody>
</table>

$L[1] = \$, $k = 1$
$L[2] = a$, $k = \text{LF}(1) = 2$
$L[3] = b$, $k = \text{LF}(2) = 7$
$L[4] = b$, $k = \text{LF}(7) = 9$
$L[5] = a$, $k = \text{LF}(9) = 4$
$L[6] = c$, $k = \text{LF}(4) = 12$

...
Reversing the BWT (2/2)

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text

- $T[n] = \$ \text{ and } T^{(n)}$ in first row TODO
- apply $LF$-mapping on result to obtain any character

$$T[n - i] = L[LF(LF(\ldots(LF(1))\ldots))]$$

$i-1$ times
Reversing the BWT (2/2)

- characters (w.r.t. text) preserve order in \(L\) and \(F\)
- \(LF\)-mapping returns previous character in text
- \(T[n] = $\) and \(T^{(n)}\) in first row TODO
- apply \(LF\)-mapping on result to obtain any character

\[
T[n - i] = L[LF(LF(\ldots(LF(1))\ldots))]_{i-1 \text{ times}}
\]

- \(T[13] = $\) and \(k = 1\)

<table>
<thead>
<tr>
<th>L</th>
<th>a</th>
<th>b</th>
<th>$</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
Reversing the BWT (2/2)

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text

- $T[n] = $ and $T^{(n)}$ in first row TODO
- apply $LF$-mapping on result to obtain any character

\[ T[n - i] = L[LF(LF(\ldots(LF(1))\ldots))] \]

\[ i-1 \text{ times} \]

<table>
<thead>
<tr>
<th>L</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>0</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF</td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

- $T[13] = $ and $k = 1$
- $T[12] = L[1] = a$ and $k = LF(1) = 2$
Reversing the BWT (2/2)

- characters (w.r.t. text) preserve order in L and F
- LF-mapping returns previous character in text

- $T[n] = \$ \text{ and } T^{(n)} \text{ in first row TODO}$
- apply LF-mapping on result to obtain any character

$$T[n - i] = L[LF(LF(...(LF(1))...))]$$

<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>0</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>LF</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

- $T[13] = \$ \text{ and } k = 1$
- $T[12] = L[1] = a \text{ and } k = LF(1) = 2$
Reversing the BWT (2/2)

- Characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text

- $T[n] = $ and $T^{(n)}$ in first row TODO
- Apply $LF$-mapping on result to obtain any character

\[ T[n - i] = L[LF(LF(\ldots(LF(1))\ldots))] \]

\[ i \text{ times} \]

<table>
<thead>
<tr>
<th>L</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>0</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF</td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

- $T[13] = $ and $k = 1$
- $T[12] = L[1] = a$ and $k = LF(1) = 2$
Reversing the BWT (2/2)

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text
- $T[n] = $ and $T^{(n)}$ in first row TODO
- apply $LF$-mapping on result to obtain any character

$$T[n - i] = L[LF(LF(\ldots(LF(1))\ldots))]$$

$i - 1$ times

<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>0</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$LF$</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

- $T[13] = $ and $k = 1$
- $T[12] = L[1] = a$ and $k = LF(1) = 2$
- $T[9] = L[9] = a$ and $k = LF(9) = 4$
Reversing the BWT (2/2)

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text

$T[n] = \$ \ $ and $T^{(n)}$ in first row TODO

apply $LF$-mapping on result to obtain any character

$$T[n - i] = L[LF(LF(\ldots(LF(1))\ldots))]$$

$i-1$ times

<table>
<thead>
<tr>
<th>L</th>
<th>a</th>
<th>b</th>
<th>$</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

- $T[13] = \$ \ $ and $k = 1$
- $T[12] = L[1] = a$ and $k = LF(1) = 2$
- $T[9] = L[9] = a$ and $k = LF(9) = 4$
Reversing the BWT (2/2)

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text

- $T[n] = \$ \text{ and } T^{(n)}$ in first row TODO
- apply $LF$-mapping on result to obtain any character

$$T[n - i] = L[LF(LF(\ldots(LF(1))\ldots))]$$

$i - 1$ times

$$
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c}
\hline
L & \text{a} & \text{b} & \$ & \text{c} & \text{c} & \text{b} & \text{b} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} \\
\hline
\text{LF} & 2 & 7 & 1 & 12 & 13 & 8 & 9 & 3 & 4 & 5 & 6 & 10 & 11 \\
\hline
\end{array}
$$

- $T[13] = \$ \text{ and } k = 1$
- $T[12] = L[1] = a \text{ and } k = LF(1) = 2$
- $T[10] = L[7] = b \text{ and } k = LF(7) = 9$
- $T[9] = L[9] = a \text{ and } k = LF(9) = 4$
- $\ldots$

Properties of the BWT: Runs

- BWT is reversible
- can be used for lossless compression

Definition: Run (simplified)

Given a text $T$ of length $n$, we call its substring $T[i..j]$ a run, if

- $T[k] = T[\ell]$ for all $k, \ell \in [i, j]$ and
- $T[i - 1] \neq T[i]$ and $T[j + 1] \neq T[j]$

(To be more precise, this is a definition for a run of a periodic substring with smallest period 1, but this is not important for this lecture.)
Properties of the BWT: Runs

- **BWT is reversible**
- can be used for lossless compression

**Definition: Run (simplified)**

Given a text $T$ of length $n$, we call its substring $T[i..j]$ a **run**, if

- $T[k] = T[\ell]$ for all $k, \ell \in [i, j]$ and
- $T[i - 1] \neq T[i]$ and $T[j + 1] \neq T[j]$

(To be more precise, this is a definition for a run of a periodic substring with smallest period 1, but this is not important for this lecture.)

<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>0</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

- **BWT** contains lots of runs
- same context is often grouped together
Compressing the BWT: Run-Length Compression

**Definition: Run-Length Encoding**

Given a text $T$, represent each run $T[i..i+\ell)$ as tuple

$(T[i], \ell)$

**Example:**

$T = \text{ababcabcabba}$

<table>
<thead>
<tr>
<th>$T$</th>
<th>BWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>$$</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>b</td>
</tr>
<tr>
<td>7</td>
<td>b</td>
</tr>
<tr>
<td>8</td>
<td>a</td>
</tr>
<tr>
<td>9</td>
<td>a</td>
</tr>
<tr>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>11</td>
<td>b</td>
</tr>
<tr>
<td>12</td>
<td>b</td>
</tr>
<tr>
<td>13</td>
<td>b</td>
</tr>
</tbody>
</table>

- $(a, 1)$
- $(b, 1)$
- $(\$, 1)$
- $(c, 2)$
- $(b, 2)$
- $(a, 4)$
- $(b, 2)$
Compressing the BWT: Move-to-Front

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the **MTF encoding** $MTF(T)$ of the text is computed as follows:

- start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$
- scan text from left to right, for character $T[i]$
  - append position of $T[i]$ in $X$ to $MTF(T)$ and
  - move $T[i]$ to front of $X$

- MTF encoding can easily be reverted
- consists of many small numbers
- runs are preserved
- use Huffman on encoding **no theoretical improvement but good in practice**
Compressing the BWT: Move-to-Front

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the MTF encoding $MTF(T)$ of the text is computed as follows:

- Start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$
- Scan text from left to right, for character $T[i]$
  - Append position of $T[i]$ in $X$ to $MTF(T)$ and
  - Move $T[i]$ to front of $X$

- MTF encoding can easily be reverted
- Consists of many small numbers
- Runs are preserved
- Use Huffman on encoding

No theoretical improvement but good in practice

**Example**

$T = ababcabcabba$

<table>
<thead>
<tr>
<th>$T[i]$</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>0</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

$MTF = 2$ and $X = a, b, c$

$MTF = 2 3$ and $X = b, a, c$

$MTF = 2 3 3$ and $X = c, b, a$

$MTF = 2 3 3 4$ and $X = c, a, b$

$MTF = 2 3 3 4 1$ and $X = a, b, c$

$MTF = 2 3 3 4 1 1$ and $X = c, a, b$
Compressing the BWT: Move-to-Front

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the MTF encoding $MTF(T)$ of the text is computed as follows:

- Start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$.
- Scan text from left to right, for character $T[i]$:
  - Append position of $T[i]$ in $X$ to $MTF(T)$ and
  - Move $T[i]$ to front of $X$.

MTF encoding can easily be reverted.
- Consists of many small numbers.
- Runs are preserved.
- Use Huffman on encoding.

No theoretical improvement but good in practice.

$$T = ababcabcabba$$

<table>
<thead>
<tr>
<th>BWT</th>
<th>1 2 3 4 5 6 7 8 9 0 11 12 13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a b $ c c b b a a a b b</td>
</tr>
</tbody>
</table>

$X = $, a, b, c
**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the MTF encoding $MTF(T)$ of the text is computed as follows:

- Start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$.
- Scan text from left to right, for character $T[i]$:
  - Append position of $T[i]$ in $X$ to $MTF(T)$ and
  - Move $T[i]$ to front of $X$.

MTF encoding can easily be reverted.

- Consists of many small numbers.
- Runs are preserved.
- Use Huffman on encoding.

**Compressing the BWT: Move-to-Front**

| $T = \text{ababcabcabba}$ | $\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 11 & 12 & 13 \hline \end{array}$ |
|---------------------------|--------------------------|
| BWT | $a$ | $b$ | $\$ | $c$ | $c$ | $b$ | $b$ | $a$ | $a$ | $a$ | $b$ | $b$ |
| $X = \$, a, b, c |
| $MTF = 2$ and $X = a, \$, b, c |

MTF encoding:

- $MTF = 2$ and $X = a, \$, b, c
## Compressing the BWT: Move-to-Front

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the **MTF** encoding $MTF(T)$ of the text is computed as follows:

- Start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$.
- Scan text from left to right, for character $T[i]$: append position of $T[i]$ in $X$ to $MTF(T)$ and move $T[i]$ to front of $X$.

MTF encoding can easily be reverted.
- Consists of many small numbers.
- Runs are preserved.
- Use Huffman on encoding, no theoretical improvement but good in practice.

---

**Example:**

Let $T = ababcabcabba$. The BWT and MTF encoding are as follows:

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\text{BWT}$</th>
<th>$\text{MTF}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>2</td>
</tr>
<tr>
<td>$$</td>
<td>$$</td>
<td>3</td>
</tr>
<tr>
<td>$c$</td>
<td>$c$</td>
<td>4</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>5</td>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>6</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>7</td>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>8</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>9</td>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>10</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>11</td>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>12</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>13</td>
</tr>
</tbody>
</table>

MTF encoding of $T$: $MTF = 2 3 3 4 1 1 3 1 4 1 1 1 2 1$. The list $X$ is updated as follows:

- $X = \$, $a$, $b$, $c$.
- $MTF = 2$ and $X = a$, $\$, $b$, $c$.
- $MTF = 2 3$ and $X = b$, $a$, $\$, $c$. 

---

2021-11-22 Florian Kurpicz | Text Indexing | 05 Burrows-Wheeler Transform
**Compressing the BWT: Move-to-Front**

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the **MTF encoding** $MTF(T)$ of the text is computed as follows:

- Start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$.
- Scan the text from left to right, for character $T[i]$:
  - Append the position of $T[i]$ in $X$ to $MTF(T)$.
  - Move $T[i]$ to the front of $X$.

- MTF encoding can easily be reverted.
  - Consists of many small numbers.
  - Runs are preserved.
  - Use Huffman on encoding.

**Example**:

Let $T = ababcabcabba$.

<table>
<thead>
<tr>
<th>MTF</th>
<th>BWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$b$</td>
</tr>
<tr>
<td>2 3</td>
<td>$a$</td>
</tr>
<tr>
<td>2 3 3</td>
<td>$b$</td>
</tr>
</tbody>
</table>

- $X = $, $a$, $b$, $c$
- $MTF = 2$ and $X = a$, $\$, $b$, $c$
- $MTF = 2 3$ and $X = b$, $a$, $\$, $c$
- $MTF = 2 3 3$ and $X = $, $b$, $a$, $c$
Compressing the BWT: Move-to-Front

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the **MTF encoding** $MTF(T)$ of the text is computed as follows:

- **MTF encoding** $MTF(T)$ of the text is computed as follows:
  - start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$
  - scan text from left to right, for character $T[i]$
    - append position of $T[i]$ in $X$ to $MTF(T)$ and
    - move $T[i]$ to front of $X$

- MTF encoding can easily be reverted
- consists of many small numbers
- runs are preserved
- use Huffman on encoding

- no theoretical improvement but good in practice

---

$T = \text{ababcabcabba}$

<table>
<thead>
<tr>
<th>MTF</th>
<th>BWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$ab$c</td>
</tr>
</tbody>
</table>

- $X = $, a, b, c
- $MTF = 2$ and $X = a, $, b, c
- $MTF = 2 \ 3$ and $X = b, a, $, c
- $MTF = 2 \ 3 \ 3$ and $X = $, b, a, c
- $MTF = 2 \ 3 \ 3 \ 4$ and $X = c, $, b, a
Compressing the BWT: Move-to-Front

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the MTF encoding $MTF(T)$ of the text is computed as follows:

- start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$
- scan text from left to right, for character $T[i]$
  - append position of $T[i]$ in $X$ to $MTF(T)$ and
  - move $T[i]$ to front of $X$

- MTF encoding can easily be reverted
- consists of many small numbers
- runs are preserved
- use Huffman on encoding

**Example**

$T = \text{ababcabcabba}$

<table>
<thead>
<tr>
<th>$\text{BWT}$</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>$0$</th>
<th>$11$</th>
<th>$12$</th>
<th>$13$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td></td>
</tr>
</tbody>
</table>

- $X = $, $a$, $b$, $c$
- $MTF = 2$ and $X = a$, $\$, $b$, $c$
- $MTF = 2\,3$ and $X = b$, $a$, $\$, $c$
- $MTF = 2\,3\,3$ and $X = $, $b$, $a$, $c$
- $MTF = 2\,3\,3\,4$ and $X = c$, $\$, $b$, $a$
- $MTF = 2\,3\,3\,4\,1$ and $X = c$, $\$, $b$, $a$
Compressing the BWT: Move-to-Front

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the MTF encoding $MTF(T)$ of the text is computed as follows:

- Start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$.
- Scan text from left to right, for character $T[i]$:
  - Append position of $T[i]$ in $X$ to $MTF(T)$ and
  - Move $T[i]$ to front of $X$.

- MTF encoding can easily be reverted.
- Consists of many small numbers.
- Runs are preserved.
- Use Huffman on encoding.

**Example**

Let $T = ababcabcabba$. The MTF encoding of $T$ is computed as follows:

1. Start with $X = [1, a, b, c]$.
2. Scan text from left to right, for each character:
   - Append position in $X$ to $MTF(T)$ and move to front of $X$.
3. The final MTF encoding is $MTF = 2 3 3 4 1 1$.

**Table**

<table>
<thead>
<tr>
<th>BWT</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>0</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

**MTF**

- $MTF = 2$ and $X = a, $, b, c$
- $MTF = 2 3$ and $X = b, a, $, c$
- $MTF = 2 3 3$ and $X = $, b, a, c$
- $MTF = 2 3 3 4$ and $X = c, $, b, a$
- $MTF = 2 3 3 4 1$ and $X = c, $, b, a$
- $MTF = 2 3 3 4 1 1$ and $X = c, $, b, a
Definition: Move-To-Front Encoding

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the MTF encoding $MTF(T)$ of the text is computed as follows:

- Start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$.
- Scan text from left to right, for character $T[i]$:
  - Append position of $T[i]$ in $X$ to $MTF(T)$ and
  - Move $T[i]$ to front of $X$.

MTF encoding can easily be reverted:
- CONSISTS OF MANY SMALL NUMBERS
- RUNS ARE PRESERVED
- USE HUFFMAN ON ENCODING

$T = \text{ababcabcabba}$

<table>
<thead>
<tr>
<th>BWT</th>
<th>$\text{ababcabcabba}$</th>
<th>$MTF$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a$ $b$ $$ $c$ $c$ $b$ $b$ $a$ $a$ $a$ $b$ $b$</td>
<td></td>
</tr>
<tr>
<td>$X$ =</td>
<td>$$, a, b, c</td>
<td></td>
</tr>
<tr>
<td>$MTF = 2$ and $X = a, $, b, c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$MTF = 2 3$ and $X = b, a, $, c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$MTF = 2 3 3$ and $X = $, b, a, c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$MTF = 2 3 3 4$ and $X = c, $, b, a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$MTF = 2 3 3 4 1$ and $X = c, $, b, a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$MTF = 2 3 3 4 1 1$ and $X = c, $, b, a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Compressing the BWT: Move-to-Front

**Definition: Move-To-Front Encoding**

Given a text $T$ over an alphabet $\Sigma = [1, \sigma]$, the **MTF encoding** $MTF(T)$ of the text is computed as follows:

- start with a list $X = \Sigma[1], \Sigma[2], \ldots, \Sigma[\sigma]$
- scan text from left to right, for character $T[i]$
  - append position of $T[i]$ in $X$ to $MTF(T)$ and
  - move $T[i]$ to front of $X$

- MTF encoding can easily be reverted
- consists of many small numbers
- runs are preserved
- use Huffman on encoding

**Example**

Let $T = ababcabcabba$. The BWT and MTF encoding are as follows:

- BWT: $a b \$, $c c b b a a a b b$
- MTF: $2$ and $X = a, \$, $b, c$
- MTF: $2 3$ and $X = b, a, \$, $c$
- MTF: $2 3 3$ and $X = \$, $b, a, c$
- MTF: $2 3 3 4$ and $X = c, \$, $b, a$
- MTF: $2 3 3 4 1$ and $X = c, \$, $b, a$
- MTF: $2 3 3 4 1 1$ and $X = c, \$, $b, a$

...
Pattern Matching using the BWT

Recap

Given a text $T$ of length $n$ over an alphabet $\Sigma$, $\alpha \in \Sigma$, and $i \in [1, n]$ then

$$C[\alpha] = |i \in [1, n]: T[i] < \alpha|$$

and

$$rank_\alpha(i) = |\{j \in [1, i]: T[j] = \alpha\}|$$

- interval for $\alpha$ is $[C[\alpha - 1], C[\alpha + 1]]$
- find sub-interval using $rank_\alpha$
- example on the board

- find interval of occurrences in $SA$ using $BWT$
- $SA$ is divided into intervals based on first character of suffix as seen during SAIS
- text from $BWT$ is backwards
- search pattern backwards
Backwards Search in the BWT

Function `BackwardsSearch(P[1..n], C, rank)`:

1. \( s = 1, e = n \)
2. for \( i = m, \ldots, 1 \) do
3. \( s = C[P[i]] + rank_{P[i]}(s - 1) + 1 \)
4. \( e = C[P[i]] + rank_{P[i]}(e) \)
5. if \( s > e \) then
6. \( \text{return } \emptyset \)
7. \( \text{return } [s, e] \)

- no access to text or SA required
- no binary search
- existential queries are easy
- counting queries are easy
- reporting queries require additional information
- example on the board
reporting queries require SA
storing whole SA requires too much space
better: sample every s-th SA position in SA′
Sampling the Suffix Array

- Reporting queries require SA
- Storing whole SA requires too much space
- Better: sample every $s$-th SA position in $SA'$

How to find sampled position?
- Mark sampled positions in bit vector of size $n$
- If match occurs check if position is sampled
- Otherwise find sample using $LF$
- If $SA[i] = j$ then $SA[LF(i)] = j - 1$
Sampling the Suffix Array

- reporting queries require SA
- storing whole SA requires too much space
- better: sample every \( s \)-th SA position in \( SA' \)

how to find sampled position?
- mark sampled positions in bit vector of size \( n \)
- if match occurs check if position is sampled
- otherwise find sample using \( LF \)
- if \( SA[i] = j \) then \( SA[LF(i)] = j - 1 \)

- \( rank_1(i) \) in bit vector is number of sample
- \( SA'[rank_1(i)] \) is sampled value
- \( SA'[rank_1(i)] - \# \) steps till sample found is correct SA value
Sampling the Suffix Array

- Reporting queries require SA
- Storing whole SA requires too much space
- Better: sample every \( s \)-th SA position in \( SA' \)

- How to find sampled position?
  - Mark sampled positions in bit vector of size \( n \)
  - If match occurs check if position is sampled
  - Otherwise find sample using LF
  - If \( SA[i] = j \) then \( SA[LF(i)] = j - 1 \)

- \( rank_1(i) \) in bit vector is number of sample
- \( SA'[rank_1(i)] \) is sampled value
- \( SA'[rank_1(i)] - \text{#steps till sample found} \) is correct SA value

- Finding a sample requires \( O(s \cdot t_{rank}) \) time
std::vector<char/int/...>

- easy access
- very big: 1, 4, ... bytes per bit
Efficient Bit Vectors in Practice (1/3)

- `std::vector<char/int/...>`
  - easy access
  - very big: 1, 4, ... bytes per bit

- `std::vector<bool>`
  - bit vector in C++ (1 bit per byte)
  - easy access
  - layout depending on implementation
Efficient Bit Vectors in Practice (1/3)

- `std::vector<char/int/...>`
  - easy access
  - very big: 1, 4, ... bytes per bit

- `std::vector<bool>`
  - bit vector in C++ (1 bit per byte)
  - easy access
  - layout depending on implementation

- `std::vector<uint64_t>`
  - requires 8 bytes per bit(?)
  - store 64 bits in 8 bytes
  - how to access bits
# Efficient Bit Vectors in Practice (1/3)

<table>
<thead>
<tr>
<th>std::vector&lt;char/int/...&gt;</th>
<th>std::vector&lt;uint64_t&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>- easy access</td>
<td>- requires 8 bytes per bit(?)</td>
</tr>
<tr>
<td>- very big: 1, 4, ... bytes per bit</td>
<td>- store 64 bits in 8 bytes</td>
</tr>
<tr>
<td></td>
<td>- how to access bits</td>
</tr>
</tbody>
</table>

**std::vector<bool>**
- bit vector in C++ (1 bit per byte)
- easy access
- layout depending on implementation

- $i/64$ is position in 64-bit word
- $i\%64$ is position in word
# Efficient Bit Vectors in Practice (1/3)

<table>
<thead>
<tr>
<th><code>std::vector&lt;char/int/...&gt;</code></th>
<th><code>std::vector&lt;uint64_t&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>easy access</td>
<td>requires 8 bytes per bit(?)</td>
</tr>
<tr>
<td>very big: 1, 4, ... bytes per bit</td>
<td>store 64 bits in 8 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>std::vector&lt;bool&gt;</code></th>
<th><code>std::vector&lt;uint64_t&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>bit vector in C++ (1 bit per byte)</td>
<td>(i/64) is position in 64-bit word</td>
</tr>
<tr>
<td>easy access</td>
<td>(i%64) is position in word</td>
</tr>
<tr>
<td>layout depending on implementation</td>
<td></td>
</tr>
</tbody>
</table>

<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>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td></td>
</tr>
</tbody>
</table>

<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>63</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>62</td>
<td>63</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \ldots \]
// There is a bit vector
std::vector<uint64_t> bit_vector;

// access i-th bit
uint64_t block = bit_vector[i/64];
bool bit = (block >> (63 - (i % 64))) & 1ULL;
Efficient Bit Vectors in Practice (2/3)

```cpp
// There is a bit vector
std::vector<uint64_t> bit_vector;

// access i-th bit
uint64_t block = bit_vector[i/64];
bool bit = (block >> (63 - (i % 64))) & 1ULL;
```

```
0 1 2 3 4 5 ... 62 63
1 1 1 0 1 0 ... 1 0
```
// There is a bit vector
std::vector<uint64_t> bit_vector;

// access i-th bit
uint64_t block = bit_vector[i/64];
bool bit = (block >> (63 - (i % 64))) & 1ULL;
// There is a bit vector
std::vector<uint64_t> bit_vector;

// access i-th bit
uint64_t block = bit_vector[i/64];
bool bit = (block >> (63 - (i % 64))) & 1ULL;

---

Efficient Bit Vectors in Practice (2/3)
Efficient Bit Vectors in Practice (3/3)

- Fill bit vector from left to right:

```
0  1  2  3  4  5  ...  62  63
1  1  1  0  1  0  ...  1  0
```

- Fill bit vector right to left:

```
63  62  ...  5  4  3  2  1  0
0  1  ...  0  1  0  1  1  1
```

Assembler code:
- `mov ecx, edi`
- `not ecx`
- `shr rsi, cl`
- `mov eax, esi`
- `and eax, 1`

`(block >> (63-(i%64))) & 1ULL;`

`(block >> (i%64)) & 1ULL;`
### Efficient Bit Vectors in Practice (3/3)

#### Fill Bit Vector from Left to Right

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>62</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[(\text{block} \gg (63-(i\%64))) \& 1\text{ULL};\]

#### Fill Bit Vector Right to Left

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>...</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[(\text{block} \gg (i\%64)) \& 1\text{ULL};\]

---

**assembler code:**

```
mov ecx, edi
not ecx
shr rsi, cl
mov eax, esi
and eax, 1
```
Efficient Bit Vectors in Practice (3/3)

\[(\text{block} \gg (63-(i\%64))) \& 1\text{ULL};\]

- fill bit vector from left to right

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
1 & 1 & 1 & 0 & 1 & 0 \\
62 & 63 & & & & \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
& & & & & \\
1 & 0 & & & & \\
\end{array}
\]

- assembler code:
  - mov ecx, edi
  - not ecx
  - shr rsi, cl
  - mov eax, esi
  - and eax, 1

\[(\text{block} \gg (i\%64)) \& 1\text{ULL};\]

- fill bit vector right to left

\[
\begin{array}{cccccc}
63 & 62 & \ldots & 5 & 4 & 3 \\
0 & 1 & \ldots & 0 & 1 & 0 \\
1 & 1 & 1 & & & \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 0 & \ldots & 1 & 1 & 0 \\
0 & 0 & \ldots & 1 & 0 & 1 \\
0 & 1 & \ldots & 0 & 1 & 0 \\
\end{array}
\]
### Efficient Bit Vectors in Practice (3/3)

#### (block >> (63-(i%64))) & 1ULL;

- **fill bit vector from left to right**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>62</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

| 0 | 0 | 0 | 0 | 0 | 0 | ... | 1 | 0 |

- **assembler code:**
  - `mov ecx, edi`
  - `not ecx`
  - `shr rsi, cl`
  - `mov eax, esi`
  - `and eax, 1`

#### (block >> (i%64)) & 1ULL;

- **fill bit vector right to left**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>...</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| 0 | 0 | ... | 1 | 1 | 0 | 0 | 1 | 0 |

- **assembler code:**
  - `mov ecx, edi`
  - `shr rsi, cl`
  - `mov eax, esi`
  - `and eax, 1`
Rank Queries in Bit Vectors (1/2)

\[
\begin{align*}
\text{rank}_\alpha(i) & \quad \text{# of } \alpha\text{s before } i \\
\text{select}_\alpha(j) & \quad \text{position of } j\text{-th } \alpha
\end{align*}
\]

<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>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
rank\(\alpha(i)\) # of \(\alpha\)s before \(i\)

select\(\alpha(j)\) position of \(j\)-th \(\alpha\)

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
\]
Rank Queries in Bit Vectors (1/2)

$\text{rank}_\alpha(i)$ # of $\alpha$s before $i$

$\text{select}_\alpha(j)$ position of $j$-th $\alpha$
Rank Queries in Bit Vectors (1/2)

\[ \text{rank}_\alpha(i) \# \text{ of } \alpha \text{ s before } i \]

\[ \text{select}_\alpha(j) \text{ position of } j\text{-th } \alpha \]

\[ \text{rank}_0(5) \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Rank Queries in Bit Vectors (1/2)

\[ \text{rank}\alpha(i) \] # of \(\alpha\)s before \(i\)

\[ \text{select}\alpha(j) \] position of \(j\)-th \(\alpha\)

Example:

- \(\text{rank}_0(5)\) = 2
- \(\text{select}_1(5)\) = 5
Rank Queries in Bit Vectors (1/2)

\[ \text{rank}_\alpha(i) \] \# of \( \alpha \)s before \( i \)

\[ \text{select}_\alpha(j) \] position of \( j \)-th \( \alpha \)

\[ \text{rank}_0(5) \]
\[ \text{select}_1(5) \]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rank Queries in Bit Vectors (1/2)

$$\text{rank}_\alpha(i)$$ # of \(\alpha\)s before \(i\)

$$\text{select}_\alpha(j)$$ position of \(j\)-th \(\alpha\)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

$$\text{rank}_0(5)$$
rank_\(\alpha\)(i) \# of \(\alpha\)s before \(i\)
select_\(\alpha\)(j) position of \(j\)-th \(\alpha\)

rank_0(5)

2

0 1 2 3 4 5 6 7 8 9
0 1 1 0 1 1 0 1 0 0

super-block

# of 0s w.r.t. BV
Rank Queries in Bit Vectors (1/2)

\[ \text{rank}_\alpha(i) \] \# of \( \alpha \)s before \( i \)

\[ \text{select}_\alpha(j) \] position of \( j \)-th \( \alpha \)

\[ \text{rank}_0(5) \]

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
\]
Rank Queries in Bit Vectors (1/2)

\( \text{rank}_\alpha(i) \) # of \( \alpha \)s before \( i \)

\( \text{select}_\alpha(j) \) position of \( j \)-th \( \alpha \)

- \( \text{rank}_0(5) \)
- # of 0s w.r.t. super-block
- # of 0s w.r.t. BV
for a bit vector of size $n$

blocks of size $s = \left\lfloor \frac{\lg n}{2} \right\rfloor$

super blocks of size $s' = s^2 = \Theta(\lg^2 n)$
for a bit vector of size $n$
blocks of size $s = \left\lfloor \frac{\lg n}{2} \right\rfloor$
super blocks of size $s' = s^2 = \Theta(\lg^2 n)$

for all $\left\lfloor \frac{n}{s'} \right\rfloor$ super blocks, store number of 0s from beginning of bit vector to end of super-block

$n/s' \cdot \lg n = O\left(\frac{n}{\lg n}\right) = o(n)$ bits of space
for a bit vector of size $n$
blocks of size $s = \lfloor \frac{\lg n}{2} \rfloor$
super blocks of size $s' = s^2 = \Theta(\lg^2 n)$

for all $\lfloor \frac{n}{s'} \rfloor$ blocks, store number of 0s from beginning of super block to end of block
$n/s' \cdot \lg s' = O(\frac{n \lg \lg n}{\lg n}) = o(n)$ bits of space

for all $\lfloor \frac{n}{s} \rfloor$ super blocks, store number of 0s from beginning of bit vector to end of super-block
$n/s' \cdot \lg n = O(\frac{n}{\lg n}) = o(n)$ bits of space
Rank Queries in Bit Vectors (2/2)

- for a bit vector of size $n$
- blocks of size $s = \lfloor \frac{\lg n}{2} \rfloor$
- super blocks of size $s' = s^2 = \Theta(\lg^2 n)$

- for all $\lfloor \frac{n}{s'} \rfloor$ super blocks, store number of 0s from beginning of super block to end of block
  - $n/s' \cdot \lg s' = O(\frac{n \lg \lg n}{\lg n}) = o(n)$ bits of space

- for all length-$s$ bit vectors, for every position $i$
  - store number of 0s up to $i$
  - $2^{\frac{\lg n}{2}} \cdot s \cdot \lg s = O(\sqrt{n} \lg n \lg \lg n) = o(n)$ bits of space

- for all $\lfloor \frac{n}{s} \rfloor$ blocks, store number of 0s from beginning of super block to end of block
  - $n/s \cdot \lg s = O(\frac{n \lg \lg n}{\lg n}) = o(n)$ bits of space
Rank Queries in Bit Vectors (2/2)

- for a bit vector of size \( n \)
  - blocks of size \( s = \lceil \frac{\lg n}{2} \rceil \)
  - super blocks of size \( s' = s^2 = \Theta(\lg^2 n) \)

- for all \( \lfloor \frac{n}{s'} \rfloor \) super blocks, store number of 0s from beginning of super block to end of block
  - \( n/s' \cdot \lg s' = O(\frac{n \lg \lg n}{\lg n}) = o(n) \) bits of space

- for all length-\( s \) bit vectors, for every position \( i \)
  - store number of 0s up to \( i \)
  - \( 2^{\frac{\lg n}{2}} \cdot s \cdot \lg s = O(\sqrt{n} \lg n \lg \lg n) = o(n) \) bits of space

- query in \( O(1) \) time

- \( rank_0(i) = i - rank_1(i) \)
The FM-Index (First Look) \[\text{[FM00]}\]

**Building Blocks of FM-Index**
- wavelet tree on BWT providing \textit{rank}-function
  \(\text{wavelet trees are topic of next lecture!}\)
- \textit{C}-array
- sampled suffix array with sample rate \(s\)
- bit vector marking sampled suffix array positions

**Space Requirements**
- wavelet tree: \(n[\lg \sigma](1 + o(1))\) bits
- \(\textit{C}-\text{array}: \sigma[\lg n]\) bits \(\Theta n(1 + o(1))\) bits if \(\sigma \geq \frac{n}{\lg n}\)
- sampled suffix array: \(\frac{n}{s}[\lg n]\) bits
- bit vector: \(n(1 + o(1))\) bits

**Lemma: FM-Index Space Requirements**
Given a text \(T\) of length \(n\) over an alphabet of size \(\sigma\), the FM-index requires \(O(n \lg \sigma)\) bits of space

**space and time bounds can be achieved with**
\(s = \lg_{\sigma} n\)
Conclusion and Outlook

This Lecture
- Burrows-Wheeler transform
- introduction to FM-index

Linear Time Construction

- ST
- SA
- LZ
- LCP
- BWT

Next Lecture
- wavelet trees
- more on FM-index
Conclusion and Outlook

This Lecture
- Burrows-Wheeler transform
- introduction to FM-index
- efficient bit vectors
- rank queries on bit vectors

Linear Time Construction

ST → SA
LZ → LCP → BWT
Conclusion and Outlook

This Lecture
- Burrows-Wheeler transform
- introduction to FM-index
- efficient bit vectors
- rank queries on bit vectors

Next Lecture
- wavelet trees
- more on FM-index

Linear Time Construction

ST  SA
LCP
LZ
BWT
Bibliography I


