Text Indexing: Lecture 2
Simon Gog – gog@ira.uka.de
Navigating in the text via $\Psi$ and LF

2.5 The Burrows-Wheeler Transform

We have already defined the Burrows-Wheeler transformed string $T_{BWT}$ in Section 2.3. In this section, we will describe the relation of $T_{BWT}$ with the $LF$ and $\Psi$ function.

Before doing this, we first have a look at the interesting history of the Burrows-Wheeler transform. David Wheeler had the idea of the character reordering already in 1978. It then took 16 years and a collaboration with Michael Burrows until the seminal technical report at Digital Equipment Corporation [BW94] was published. The reader is referred to [ABM08] for the historical background of this interesting story. Today, the Burrows-Wheeler Transform plays a key role in text data compression. The most prominent example for a compressor based on this transformation is the bzip2 application.

However, one can not only compress the text but also index it. To show how this is possible, we have to present the relation between the $LF$ and $\Psi$ function and $T_{BWT}$.

![Diagram](a) and (b) show the $LF$ function and the $\Psi$ function and how it can be expressed by $T_{BWT}$ and $F$ for the running example $T = u m u l m u n d u m u l m u m$. 

1 http://www.bzip.org
Self Index
Does not only provide search functionality but also efficient reconstruction of any substring of the original text.

LF mapping
For every suffix $j = SA[i]$, $LF(i)$ is the position of $j - 1$ (the previous suffix in the text) in $SA$. It holds:

$$LF[i] = C[BWT[i]] + rank(i, BWT[i], BWT)$$

I.e. we can decode text backwards. Starting from the last suffix $\$$ at SA-position 0, we can decode the whole text.
### Turning the FM-Index into a Self Index

#### Inverse Suffix Array

<table>
<thead>
<tr>
<th>i</th>
<th>SA</th>
<th>ISA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>14 $</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>6 dumulmum$</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>11 lrum$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3 lrumderulmum$</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>8 m$</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>15 murnulmum$</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>9 murnmulmumulmum$</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>1 mum$</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>13 murnmulmumulmum$</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>5 mumulmurnulmum$</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10 umulmum$</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2 umulmurnulmum$</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>7 um$</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>12 umulmum$</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>4 umulmurnumulmumulmum$</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>0 undumulmum$</td>
</tr>
</tbody>
</table>

- **Inverse permutation of SA:**
  \[ ISA[SA[i]] = i \]

- **Given suffix x. Where does x occur in SA?**

**Express LF:**
\[ LF[i] = ISA[SA[i] - 1 \mod n] \]

**Express \( \Psi \):**
\[ \Psi[i] = ISA[SA[i] + 1 \mod n] \]
Implement $\Psi$ via WT over BWT

$\Psi$ calculation
$\Psi[i] = select(rank(i, F[i], F), F[i], BWT)$

Operation select
Given a sequence $X$, a symbol $c$, and an integer $i$. Operation $select(i, c, X)$ returns the position of the $i$-th occurrence of $c$ in $X$.

Exercise
- Assume that there is a data structure which solves select queries on bitvectors in constant time using $o(n)$ space. Show how select can be implemented in $\log \sigma$ time and $o(n \log \sigma)$ bits for a sequence of length $n$ over an alphabet of size $\sigma$.
- What is the maximal size of the set \{ $i \mid \Psi[i] > \Psi[i + 1]$ \}?
Sampling (for locate)

Fix a sampling rate $s$. Add a bitvector $B$ of length $n$ with $B[i] = 1$ if $SA[i] \equiv 0 \mod s$. Store the samples in array $SA'$ of size $n/s$. I.e. for all $i$ with $B[i] = 1$, $SA'[\text{rank}(i, 1, B)] = SA[i]$.

Pseudo-code for accessing $SA[i]$

See blackboard.
Compressing the Index

Definitions

\( H_0(X) \) – zeroth order empirical entropy

Given a sequence \( X \) of length \( n \) over alphabet \( \Sigma \). Let \( n_c \) be the number of occurrences of \( c \in \Sigma \) in \( X \).

\[
H_0(X) = \sum_{c \in \Sigma, n_c > 0} \frac{n_c}{n} \log \frac{n}{n_c}
\]

Provides a lower bound to the number of bits needed to compress \( X \) using a compressor which just considers character frequencies.
Elias-Fano Coding Elias [1974], Fano [1971]

Given a non-decreasing sequence $X$ of length $m$ over alphabet $[0..n]$. $X$ can be represented using $2m + m \log \frac{n}{m} + o(m)$ bits while each element can still be accessed in constant time.

This representation can also be used to represent a bitvector (e.g. $n$ is bitvector length, $m$ the number of set bits, and $X$ the position of the set bits)
Compressing the Index

How does Elias-Fano coding work?

- Divide each element into two parts: high-part and low-part.
- ⌈log m⌉ high-bits and ⌊log n⌋ − ⌈log m⌉ low bits
- Sequence of high-parts of X is also non-decreasing.
- Gap encode the high-parts and use unary encoding to represent gaps. Call result H.
  - i.e. for a gap of size $g_i$ we use $g_i + 1$ bits ($g_i$ zeros, 1 one).
  - Sum of gaps (= #zeros) is at most $2^{⌈log m⌉} ≤ 2^{log m} = m$
  - i.e. $H$ has size at most $2m$ (#zeros + #ones)
- Low-parts are represented explicitly.
Compressing the Index

How does Elias-Fano coding work?

\[ X = \begin{array}{cccccccc}
4 & 13 & 15 & 24 & 26 & 27 & 29 \\
00100 & 01101 & 01111 & 11000 & 11010 & 11011 & 11101 \\
1 & 0 & 3 & 1 & 3 & 3 & 6 & 0 & 6 & 2 & 6 & 3 & 7 & 1 \\
\end{array} \]

\[
\delta = \begin{array}{cccccccc}
1 & 2 & 0 & 3 & 0 & 0 & 1 \\
1-0 & 3-1 & 3-3 & 6-3 & 6-6 & 6-6 & 7-6 \\
\end{array} \]

\[
H = 01001100011101 \\
L = 0 \ 1 \ 3 \ 0 \ 2 \ 3 \ 1 \\
\]
Compressing the Index
How does Elias-Fano coding work?

Constant time access

- Add a select structure to $H$ (Okanohara and Sadakane [2007]).

```plaintext
00  access(i)
01      p ← select(i + 1, 1, H)
02      x ← p − i
03  return $x \cdot 2^{\lceil \log n \rceil − \lfloor \log m \rfloor} + L[i]$
```
Compressing the Index

Apply Elias-Fano coding to a $\Psi$-based CSA

- $\Psi$ consists of at most $\sigma$ non-decreasing sequences in the range $[0, n-1]$.

$$|\text{CSA}_\Psi| = \sum_{c \in \Sigma} \left( n_c (2 + \log \frac{n}{n_c}) + o(n_c) \right)$$

$$= \sum_{c \in \Sigma} 2n_c + n \sum_{c \in \Sigma} \frac{n_c}{n} \log \frac{n}{n_c} + o(n)$$

$$= 2n + H_0(\mathcal{T}) + o(n)$$

- $+\mathcal{O}(\sigma \log n)$ bits to handle character boundaries
Compressing the Index
Search in a \( \Psi \)-based CSA

- Compare pattern from left to right (forward search) to suffix \( SA[i] \)
- Use binary search on the interval \([0, n - 1]\).

```plaintext
compare(P, i)
  k ← 0
  while k < |P| do
    if \( C[P[k] + 1] - 1 < i \) then
      return -1 \( //P \) smaller than suffix
    else if \( C[P[k]] > i \) then
      return +1 \( //P \) larger than suffix
    k ← k + 1
  i ← \( \Psi[i] \)
return 0 \( //P \) equal to the first \( m \) character of the suffix
```
E.g. Elias-$\delta$ code. Let $\text{bin}(x)$ be the binary representation of $x$. Write $\text{bin}(\text{bin}(x)) - 1$ in unary, append the $\text{bin}(\text{bin}(x)) - 1$ least significant bits of $\text{bin}(x)$, and append the $\text{bin}(x) - 1$ least significant bits of $\text{bin}(x)$.

| $x_{(10)}$ | $x_{(\text{unary})}$ | $x_{(2)}$ | $x_{(\delta\text{-code})}$ | $|x_{\delta\text{-code}}|$ |
|-----------|-------------------|-----------|----------------|----------------|
| 1         | 01                | 1         | 1              | 1              |
| 2         | 001               | 10        | 0100           | 4              |
| 3         | 0001              | 11        | 0101           | 4              |
| 4         | 00001             | 100       | 01100          | 5              |
| 5         | 000001            | 101       | 01101          | 5              |
| 13        | 0000000000000001  | 1101      | 00100101       | 8              |

Elias-$\delta$ code for $x$ is $2 \log \log x + \log x + O(1)$. 

Length of
Compressing the Index
Space analysis of a $\Psi$-based CSA using Elias-$\delta$ code.

For each character $c$ gap-encode its increasing $\Psi$ sequence. E.g.

$$g_{c,i} = \Psi[C[c] + i] - \Psi[C[c] + i - 1]$$ for $i > 0$ and $g_{c,0} = \Psi[C[c]]$ for $i = 0$.

$$\sum_{c \in \Sigma} \sum_{i=0}^{n_c-1} \left( \log g_{c,i} + 2 \log \log g_{c,i} + O(1) \right)$$

$$\leq O(n) + \sum_{c \in \Sigma} \sum_{i=0}^{n_c-1} \left( \log \frac{n}{n_c} + 2 \log \log \frac{n}{n_c} \right)$$

$$= O(n) + n \sum_{c \in \Sigma} \frac{n_c}{n} \left( \log \frac{n}{n_c} + 2 \log \log \frac{n}{n_c} \right)$$

$$= nH_0(T) + O(n \log \log n)$$
