Succinct Data Structures for All

Daniel Saad Nogueira Nunes  Mauricio Ayala-Rincón

Grupo de Teoria da Computação
Universidade de Brasília

5th February 2015

VII Workshop de Matemática Aplicada — MAT/UnB

XII Seminário informal (,mas formal!) do Grupo de Teoria da Computação da UnB — GTC/UnB
Summary

1. Introduction
2. Bitmaps
3. Wavelet Trees
4. Compressed Indices
5. Current Work
6. References
Summary

1. Introduction

2. Bitmaps

3. Wavelet Trees

4. Compressed Indices

5. Current Work

6. References

There are more problems than people.

Ricardo Baeza-Yates
1 Introduction

- Memory Hierarchy
- Basic Concepts
Memory Hierarchy

Figure: Memory Hierarchy.
Figure: Gap between memory and cpu performance.
Memory Hierachy

Motivation
- Moore’s law: \# transistors grow exponentially.
- CPU speed and Memory capacity grows as well.
- Memory Access does not share the same result!
- We should use faster memories until we can!
Definition (Succinct Data Structures)

- A Succinct Data Structure ($SDS$) uses an amount of space that is “close” to the information-theoretic lower bound. [Jac89]
- But also allows efficient queries!
- The use of $SDS$s is encouraged in the actual scenario.
1 Introduction

- Memory Hierarchy
- Basic Concepts
Empirical Entropy

- Defined for every finite and individual string.
- Can be used to measure the performance of compression algorithms.
- Does not take in account the input distribution.
Basic Concepts

Definition (Zero\textsuperscript{th} Order Empirical Entropy)

- Zero\textsuperscript{th} order entropy can be defined as:
  \[
  H_0(S) = - \sum_{c \in \Sigma} \frac{n_c}{n} \log \frac{n_c}{n}
  \]  
  \hspace{1cm} (1)

  where \( n_c \) stands for the frequency of \( c \) in \( S \).
  - Does not consider any context to encode a symbol.
  - \( nH_0 \): lower bound to a zero-order compressor.
  - Huffman compression is bounded at \( nH_0 \) bits.
I take a whole life story and compress it into three minutes.

Harlan Howard
Bitmaps

- Bitmaps: the core of SDS.
- A sequence $S \in \{0, 1\}^*$. 
- Can represent information in compact space.
- Often three operations are supported:
  - Rank.
  - Select.
  - Access.
Rank Queries

Definition (Rank)

\[ \text{Rank}_1(B, i) = \text{number of 1's in } B[0, i] \]
\[ \text{Rank}_0(B, i) = \text{number of 0's in } B[0, i] \]

- Focus on \text{Rank}_1, since \text{Rank}_0(B, i) = n - \text{Rank}_1(B, i).
Definition (Select)

\[ Select_1(B, i) = \text{Position of the } i^{th} 1 \text{ in } B \]

\[ Select_0(B, i) = \text{Position of the } i^{th} 0 \text{ in } B \]

- Notice that \( \text{Rank}(Select(x)) = x \wedge Select(\text{Rank}(x)) = x \) iff \( B[x] = 1 \).
- Can be answered in \( O(1) \) time as well.
• Time is short!
• I need you to believe that *Rank* and *Select* queries can be answered in $O(1)$ time and $n + o(n)$ bits.
Summary

1. Introduction

2. Bitmaps

3. Wavelet Trees

4. Compressed Indices

5. Current Work

6. References

If you think in terms of a year, plant a seed; if in terms of ten years, plant trees; if in terms of 100 years, teach the people.

Confucius

Nunes, D.S.N & Ayala-Rincón, M.
Succinct Data Structures for All
GTC/UnB 17/56
Bitmaps Limitations

- Support only binary $\Sigma$.
- What if one wanted to answer general $\text{Rank}/\text{Select}$ queries over another alphabet?
- Example: $\text{Rank}_c(\text{ACTAGACCTAGACGAC}, 7) = 3$.
- Solution: Wavelet Trees ($\mathcal{WT}$) [GGV03].
- $\mathcal{WT}$s reduce general $\text{Rank}/\text{Select}$ queries on binary queries.
Wavelet Trees

S = aaabraacaaadaabraa

{a, b, c, d, r}

Figure: Wavelet tree for $S = aaabraacaaadaabraa$
Wavelet Trees

Bitmaps Limitations

- $\text{Rank}_b(S, 10) = ?$
- $\text{Select}_a(S, 11) = ?$
- $\text{Access}(S, 14) = ?$
Wavelet Trees

S = aaabraacaaadaabraa
{a,b,c,d,r}

012345678901234567
aaabraacaaadaabraa
000100010001000100

{a,b}

01234567890123
aaabaaaaaaabaa
0001000000100

{c,d,r}

0123
cdr
1011

{d,r}

012
rdr
101

Figure: Answering $\text{Rank}_b(S, 10)$
Wavelet Trees

\[ S = \text{aaabraacaaadaabreaa} \]
\[ \{a, b, c, d, r\} \]

\[ 012345678901234567 \]
\[ \text{aaabraacaaadaabreaa} \]
\[ 0001001000100100 \]

\[ \text{Rank}_0(B, 10) = 9 \]

**Figure**: Answering \( \text{Rank}_b(S, 10) \)
Wavelet Trees

$S = \text{aaababraacaadaabraa}$

$\{a,b,c,d,r\}$

$$
\text{012345678901234567}
\text{aaabraacaaaadaabraa}
\text{00010010001000100}
$$

$Rank_0(B, 10) = 9$

Figure: Answering $Rank_b(S, 10)$

Nunes, D.S.N & Ayala-Rincón, M. Succinct Data Structures for All GTC/UnB
Wavelet Trees

S = aaabracaaaaadaabraa

{a,b,c,d,r}

012345678901234567
aaabracaaaaadaabraa
00010010001000100

{a,b}

01234567890123
aabaaaaaaaabaa
00010000000100

{c,d,r}

0123
rcdr
1011

{d,r}

012
rdr
101

aaa

bb

C

dd

rr

Figure: Answering Selecta(S, 11)
\( S = \texttt{aaabraacaaadaabraa} \)

\( \{a,b,c,d,r\} \)

\[
\begin{array}{c}
012345678901234567 \\
\text{aaabraacaaadaabraa} \\
00010000001000100
\end{array}
\]

\{a,b\} \quad \{c,d,r\}

\[
\begin{array}{c}
01234567890123 \\
\text{aaabaaaaaaabaa} \\
00010000001000100
\end{array}
\]

\( \text{Select}_a(B, 11) = 12 \)

\[\begin{array}{c}
\{d,r\}
\end{array}\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[
\begin{array}{c}
012 \\
rdr \\
101
\end{array}
\]

\[\text{Figure : Answering } \text{Select}_a(S, 11)\]
Wavelet Trees

\[ S = \text{aaabraacaaadaabraa} \]
\[
\{a, b, c, d, r\}
\]

\[ \begin{aligned}
012345678901234567 \\
\text{aaabraacaaadaabраa} \\
000100100100100100
\end{aligned} \]

\[ \text{Select}_0(B, 12) = 16 \]

\[ \begin{aligned}
\{a, b\} & \quad \text{Select}_0(B, 11) = 12 \\
01234567890123 & \quad \text{rcdr} \\
\text{aaabaaaaaaabaa} & \quad 1011 \\
00010000000100 & \quad \{d, r\}
\end{aligned} \]

\[ \begin{aligned}
\{c, d, r\} & \quad \text{Select}_0(B, 12) = 16 \\
0123 & \quad \text{rdr} \\
\text{aaabraacaaadaabраa} & \quad 1011
\end{aligned} \]

\[ \begin{aligned}
\{a, b\} & \quad \text{Select}_0(B, 11) = 12 \\
01234567890123 & \quad \text{rcdr} \\
\text{aaabaaaaaaabaa} & \quad 1011 \\
00010000000100 & \quad \{d, r\}
\end{aligned} \]

\[ \begin{aligned}
\{c, d, r\} & \quad \text{Select}_0(B, 12) = 16 \\
0123 & \quad \text{rdr} \\
\text{aaabraacaaadaabраa} & \quad 1011
\end{aligned} \]

**Figure:** Answering $\text{Select}_a(S, 11)$
Wavelet Trees

S = aaabraacaaadaabraa
\{a, b, c, d, r\}

Figure: Answering \textit{Access}(S, 14)
Wavelet Trees

$S = \text{aaabraacaaadaabrra}$

$\{a,b,c,d,r\}$

$012345678901234567$

$\text{aaabrracaaadaabrra}$

$00001000100001000100$

$Rank_0(B,14) = 12$

Figure: Answering $Access(S, 14)$
Wavelet Trees

$$S = \text{aaabraqaacaadaabbraa}$$

$$\{a,b,c,d,r\}$$

$$\begin{array}{c}
012345678901234567 \\
aaabraqaacaadaabbraa \\
000100100100100100
\end{array}$$

$$\text{Rank}_0(B, 14) = 12$$

Figure: Answering Access($$S, 14$$)
Wavelet Trees

Properties

- $\sigma$ leaves and $\sigma - 1$ internal nodes.
- Height: $\lceil \log \sigma \rceil$.
- Rank, Select and Access in $O(\log \sigma)$ time.
- Space: $O(n \log \sigma + \sigma \log n)$.
- Space (with no pointers): $O(n \log \sigma)$.
- Can achieve $O(n H_0)$ bits if shaped as a Huffman tree.
- It is a **self-index**: entirely replaces the original sequence.
Wavelet Trees

\[ S = \text{aaabracaaadaabraa} \quad \{a,b,c,d,r\} \]

\[
\begin{array}{c}
012345678901234567 \\
\text{aaabracaaadaabraa} \\
000110010001001100 \\
\{b,c,d,r\} \\
\text{aaaaaaa}
\end{array}
\]

\[
\begin{array}{c}
012345 \\
brcdrbr \\
110011 \\
\{c,d\} \\
01 \\
cd \\
01
\end{array}
\]

\[
\begin{array}{c}
\text{dd} \\
bb
\end{array}
\]

\[
\begin{array}{c}
\text{rr}
\end{array}
\]

\[
\begin{array}{c}
c
\end{array}
\]

**Figure:** Huffman-shaped WT.

Nunes, D.S.N & Ayala-Rincón, M.  
Succinct Data Structures for All  
GTC/UnB 25/56
Summary

1. Introduction

2. Bitmaps

3. Wavelet Trees

4. Compressed Indices

5. Current Work

6. References

Suffix Arrays... The permutation in Stringology

Roberto Grossi
The Suffix Tree ($ST$) is a well-known index in the literature which represents all the suffixes in $O(n \log n)$ bits or $O(n)$ words.

- Can be constructed in $O(n)$ time.
- Demands too much space in practice.
S = aaabraacaaadaabraa$
0123456789012345678

Figure: Suffix Tree for aaabraacaaadaabraa$. 
Suffix Array

- Suffix Array (SA): compact alternative to STs.
- Integer array containing the position of suffixes in lexicographical order induced by $\Sigma$.
- Can be built in $O(n)$ time.
- Can handle bigger texts.
- Inverse SA: Integer array containing the lexicographical order of the $i^{th}$ suffix.
### Suffix Array

**Table**: Suffix Array for `aaabracaaadaabraa$`.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$SA[i]$</th>
<th>$SA^{-1}[i]$</th>
<th>$T_{SA[i]}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>18</td>
<td>3</td>
<td>$$</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>6</td>
<td>$a$</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>10</td>
<td>$aa$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>14</td>
<td><code>aaabracaaadaabraa$</code></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>18</td>
<td><code>aaadaabraa$</code></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>7</td>
<td><code>abraa$</code></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>11</td>
<td><code>aabracaaadaabraa$</code></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>15</td>
<td><code>aacaadaabraa$</code></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>4</td>
<td><code>aadaabraa$</code></td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>8</td>
<td><code>abraa$</code></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>12</td>
<td><code>abraacaaadaabraa$</code></td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>16</td>
<td><code>aacaadaabraa$</code></td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>5</td>
<td><code>adaabraa$</code></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>9</td>
<td><code>abraa$</code></td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>13</td>
<td><code>braacaaadaabraa$</code></td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>17</td>
<td><code>caadaabraa$</code></td>
</tr>
<tr>
<td>16</td>
<td>11</td>
<td>2</td>
<td><code>dabraa$</code></td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>1</td>
<td><code>raa$</code></td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>0</td>
<td><code>raacaaadaabraa$</code></td>
</tr>
</tbody>
</table>
With SAs: space consumption is still an issue.

In order to manipulate huge texts one need more space-efficient data-structures!

Compressed Indices for All!

Figure: Space Consumption of Indices for HG.
4 Compressed Indices

- Basic Concepts
- CSA
- $FM$-Index
The Compressed Suffix Array (CSA) was originally developed by Grossi and Vitter [GV00].

Main idea: sample some suffix array entries and recover other by computation.

CSA core: $\Psi$ function.

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

$\Psi(i)$ is piecewise crescent for suffixes starting with the same symbol.

Allows compression by differential encoding.
## Suffix Array

**Table**: $\Psi$ function for `aaabraacaaadaabraa$`.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$SA[i]$</th>
<th>$SA^{-1}[i]$</th>
<th>$\Psi(i)$</th>
<th>$T_{SA[i]}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>18</td>
<td>3</td>
<td>3</td>
<td>$$$</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>6</td>
<td>0</td>
<td>a$</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>10</td>
<td>1</td>
<td>aa$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>14</td>
<td>6</td>
<td><code>aaabraacaaadaabraa$</code></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>18</td>
<td>8</td>
<td><code>aaadaabraa$</code></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>7</td>
<td>9</td>
<td><code>aabraa$</code></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td><code>abraacaaadaabraa$</code></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>15</td>
<td>11</td>
<td><code>aacaaadaabraa$</code></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>4</td>
<td>12</td>
<td><code>aadabraa$</code></td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>8</td>
<td>13</td>
<td><code>abaa$</code></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>12</td>
<td>14</td>
<td><code>abraacaaadaabraa$</code></td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>16</td>
<td>15</td>
<td><code>aacaadaabraa$</code></td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>5</td>
<td>16</td>
<td><code>adaabraa$</code></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>9</td>
<td>17</td>
<td><code>braa$</code></td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>13</td>
<td>18</td>
<td><code>braacaaadaabraa$</code></td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>17</td>
<td>4</td>
<td><code>caadaabraa$</code></td>
</tr>
<tr>
<td>16</td>
<td>11</td>
<td>2</td>
<td>5</td>
<td><code>daabraa$</code></td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>1</td>
<td>2</td>
<td><code>raa$</code></td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td><code>raacaaadaabraa$</code></td>
</tr>
</tbody>
</table>
The $\Psi$ function

- How $\Psi$ is efficiently coded?
- Basic idea: take $\Psi(i) - \Psi(i - 1)$ and apply Rice code.
- For increasing sequence $(0, 2, 5, 7, 9)$ one would have the bitmap:

$$B = \begin{array}{cccccc}
1 & 001 & 0001 & 001 & 001 \\
0 & 2 & 5 & 7 & 9
\end{array}$$

- $\Psi(i) = Select_1(B, i)$. 
How recover $SA$ entries?

- Basic idea: store explicitly only $k = \frac{n}{\log^\epsilon n}$ entries of $SA$.
  - $o(n)$ bits of space if $\epsilon > 1$

- Mark sampled entries with a 1 in a bitmap.

- Apply $\Psi$ at maximum $l \leq k$ times until finding a sampled entry.

$$SA[i] = SA[\Psi^l(i)] - l \mod n$$
**Suffix Array**

<table>
<thead>
<tr>
<th>$i$</th>
<th>$SA[i]$</th>
<th>$SA^{-1}[i]$</th>
<th>$\Psi(i)$</th>
<th>$T_{SA[i]}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>18</td>
<td>3</td>
<td>3</td>
<td>$$$</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>6</td>
<td>0</td>
<td>$a$$</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>10</td>
<td>1</td>
<td>$aal$$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>14</td>
<td>6</td>
<td>$aaabraaacaadaabraa$</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>18</td>
<td>8</td>
<td>$aaadaabraa$</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>7</td>
<td>9</td>
<td>$aabraa$</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>$abraacaaadaabraa$</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>15</td>
<td>11</td>
<td>$aacaadaabraa$</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>4</td>
<td>12</td>
<td>$adaabraa$</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>8</td>
<td>13</td>
<td>$abraa$</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>12</td>
<td>14</td>
<td>$abraacaaadaabraa$</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>16</td>
<td>15</td>
<td>$aacaadaabraa$</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>5</td>
<td>16</td>
<td>$adaabraa$</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>9</td>
<td>17</td>
<td>$braa$</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>13</td>
<td>18</td>
<td>$braacaaadaabraa$</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>17</td>
<td>4</td>
<td>$aaadaabraa$</td>
</tr>
<tr>
<td>16</td>
<td>11</td>
<td>2</td>
<td>5</td>
<td>$daabraa$</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>1</td>
<td>2</td>
<td>$raa$</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td>$raacaaadaabraa$</td>
</tr>
</tbody>
</table>

- $\Psi(2) = 1 \rightarrow \Psi(1) = 0 \rightarrow \Psi(0) = 3 \rightarrow \Psi(3) = 6 \rightarrow SA[6] = 1$
Summary

4 Compressed Indices

- Basic Concepts
- CSA
- FM-Index
The \textit{FM} family of indices is based on the Burrows-Wheeler Transform (\textit{BWT}) \cite{FM05}.

Same idea as \textit{CSA}: sample some entries and recover others by computation.

The \textit{BWT} is computed by sorting the cyclical suffixes and taking the last column.
aaabraacaaadaabraham$
abraacaaadaabraham$a
abraacaaadaabraham$aa
braacaaadaabraham$aaa
raacaaadaabraham$aaab
acaaadaabraham$aaabab
caadaabraham$aaabrac
aadaabraham$aaabracac
daabraah$aaabraacaa
abraa$aaabraacaaa
abraa$aaabraacaaad
abraa$aaabraacaaaad
braa$aaabraacaaada
raa$aaabraacaaadaab
aa$aaabraacaaadaabr
a$aaabraacaaadaabra
$aaabraacaaadaabraham$

\[\text{BWT for } S = \text{aaabraacaaadaabraham}$.}
The \textit{BWT} is a permutation based on the original text.

- Has a close relation to \textit{SAs}.
- \( BWT[i] = T[SA[i] - 1 \mod n] \).
- Same symbols tend to be grouped together.
- Eases the compression.
By using the BWT we can walk through the suffix array.

\[ LF(i) = SA^{-1}[SA[i] - 1 \mod n] \]

- **LF** moves to the previous suffix.
- **LF** and \( \Psi \) are very similar: \( LF(\Psi(i)) = \Psi(LF(i)) \).
- \( LF(i) = C[BWT[i]] + \text{Rank}_{BWT[i]}(BWT, i) \).
  - \( C[i] \) contains the \# of symbols in \( S \) which are lexicographically smaller than \( i \).
  - Why?
  - Previous suffixes starting with the same symbol will retain relative order.
  - They are contiguous in the first row!
$aaabraacaadaabrah$  $a$aaabraacaadaabrah
$aaabraacaadaabrah$aa  $aa$aaabraacaadaabrah
braacaadaabrah$aaa  aaabraacaadaabrah$
raacaadaabrah$aaab  aaadaabrah$aaabraac
aacaadaabrah$aaabr  aabraaaababraacaad
caadaabrah$aaabra  aacaaadaabrah$aaab
aaadaabrah$aaabraac  aadaabrah$aaabraacaad
aadaabrah$aaabraaca  abraa$aaabraacaad
adaabrah$aaabraacaac  abraacaadaabrah$aa
daabra$aaabraacaac  acaaadaabrah$aaabra
aabraa$aaabraacaaad  adaabra$aaabraacaac
abraa$aaabraacaadaaa  braaa$aaabraacaadaaa
braa$aaabraacaadaaa  braacaadaabrah$aa
raa$aaabraacaadaaab  caadaabrah$aaabra
a$aaabraacaadaab  daabra$aaabraacaad
a$aaabraacaadaab  raa$aaabraacaadaab
$aaabraacaadaab  raacaadaabrah$aaab

Figure: $LF(13) = C[a] + \text{Rank}_a(BWT, 13) - 1$
$aaabracaaadaabraa$
$aaabracaaadaabraa$a$
$abraacaaadaabraa$aa$
$braacaaadaabraa$aaa$
$raacaaadaabraa$aaab$
$acaaadaabraa$aaabr$
$caaadaabraa$aaabra$
$aaadaabraa$aaabraac$
$adaabraa$aaabraaca$
$daabraa$aaabraacaa$
$aaabraa$aaabraacaaad$
$abraa$aaabraacaaada$
$braa$aaabraacaaadaa$
$raa$aaabraacaaadaab$
$aa$aaabraacaaadaabr$
$a$aaabraacaaadaabra$
$aaabracaaadaabraa$

\[
\text{Figure : } LF(13) = 1 + 9 - 1 = 9.
\]
What changes from one \(\mathcal{FM}\)-Index to another: how to represent the \(\text{BWT}\).

- We need to know: \(C[i] + \text{Rank}_{\text{BWT}^i}(\text{BWT}, i)\);
- Simple and fast implementation:

\[
\text{BWT} \rightarrow \text{WT}(\text{BWT})
\]

- To achieve \(O(nH_0)\) bits, use Huffman-shaped \(\text{WT}\)s.
The best preparation for good work tomorrow is to do good work today.

Elbert Hubbard
Summary

Current Work

- Results
- To-do List
Results

- Compressed Suffix Tree with Low Peak Memory Usage [NA14].
  - Based on CSA and additional compressed information.
- Support of complex queries:
  - Longest Common Ancestor.
  - Suffix Link.
Figure: Proposed $CST$ using 13 bits per symbol.
Space and Memory Peak

Memory peak and space
Size(MB) vs Space(MB)

Figure: Proposed CST using 13 bits per symbol.
**Operations**

**Figure**: Proposed CST using 13 bits per symbol.

---

Nunes, D.S.N & Ayala-Rincón, M. Succinct Data Structures for All GTC/UnB 50/56
5 Current Work

- Results
- To-do List
To-do List

- Search for theoretical improvements in SDSs, which can lead to practical usage.
- Design fast and more efficient SDSs with low peak memory usage.
- Compare to others implementations. [ACN13, GBMP13]
Summary

1. Introduction
2. Bitmaps
3. Wavelet Trees
4. Compressed Indices
5. Current Work
6. References
References

Practical compressed suffix trees.

[FM05] Paolo Ferragina and Giovanni Manzini.
Indexing compressed text.

From theory to practice: Plug and play with succinct data structures.
References


References

Space-efficient static trees and graphs.

A compressed suffix tree based implementation with low peak memory usage.