Text Indexing

Lecture 08: LZ and BWT Compressed Indexes

Florian Kurpicz
PINGO

https://pingo.scc.kit.edu/669011
Recap: FM-Index and $r$-Index

- based on backwards-search
- used to answer rank-queries on BWT

```
Function BackwardsSearch($P[1..n]$, $C$, rank):
1    $s = 1$, $e = n$
2    for $i = m, \ldots, 1$ do
3        $s = C[P[i]] + \text{rank}_{P[i]}(s - 1) + 1$
4        $e = C[P[i]] + \text{rank}_{P[i]}(e)$
5        if $s > e$ then
6            return $\emptyset$
7        return $[s, e]$
```
Recap: FM-Index and $r$-Index

- based on backwards-search
- used to answer rank-queries on BWT

### FM-Index

- build wavelet tree directly on BWT
- wavelet tree can be $H_0$ compressed
- blind to repetitions

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

```plaintext
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          return \emptyset
7  return [s, e]
```

Recap: FM-Index and $r$-Index
Recap: FM-Index and $r$-Index

- based on **backwards-search**
- used to answer **rank**-queries on **BWT**

### FM-Index
- build wavelet tree directly on **BWT**
- wavelet tree can be $H_0$ compressed
- blind to repetitions

### $r$-Index
- many arrays with $r$ entries
- build wavelet tree on one of these arrays
- size in numbers of **BWT** runs $r$

---

<table>
<thead>
<tr>
<th>Function</th>
<th>BackwardsSearch($P[1..n]$, $C$, $rank$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$s = 1$, $e = n$</td>
</tr>
<tr>
<td>2</td>
<td><strong>for</strong> $i = m, \ldots, 1$ <strong>do</strong></td>
</tr>
<tr>
<td>3</td>
<td>$s = C[P[i]] + rank_{P[i]}(s - 1) + 1$</td>
</tr>
<tr>
<td>4</td>
<td>$e = C[P[i]] + rank_{P[i]}(e)$</td>
</tr>
<tr>
<td>5</td>
<td><strong>if</strong> $s &gt; e$ <strong>then</strong></td>
</tr>
<tr>
<td>6</td>
<td><strong>return</strong> $\emptyset$</td>
</tr>
<tr>
<td>7</td>
<td><strong>return</strong> $[s, e]$</td>
</tr>
</tbody>
</table>
Different Types of Compression

### Statistical Coding

- based on frequencies of characters
- results in size $|T| \cdot H_k(T)$
  - $k$-th order empirical entropy
- good if frequencies are skewed
- blind to repetitions
  \[ |\underbrace{T \ldots T}_\ell| \cdot H_k(\underbrace{T \ldots T}_\ell) \approx \ell |T| \cdot H_k(T) \]
# Different Types of Compression

<table>
<thead>
<tr>
<th>Statistical Coding</th>
<th>LZ-Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>- based on frequencies of characters</td>
<td>- references to previous occurrences</td>
</tr>
<tr>
<td>- results in size $</td>
<td>T</td>
</tr>
<tr>
<td>$\updownarrow k$-th order empirical entropy</td>
<td>- good for repetitions</td>
</tr>
<tr>
<td>- good if frequencies are skewed</td>
<td>- index in this lecture</td>
</tr>
<tr>
<td>- blind to repetitions $\ell_\ell</td>
<td>T</td>
</tr>
</tbody>
</table>

\[
\ell |T| \cdot H_k(T) \approx \ell_\ell |T| \cdot H_k(T)
\]
## Different Types of Compression

<table>
<thead>
<tr>
<th>Statistical Coding</th>
<th>LZ-Compression</th>
<th>BWT-Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>based on frequencies of characters</td>
<td>references to previous occurrences</td>
<td>used in powerful index</td>
</tr>
<tr>
<td>results in size $</td>
<td>T</td>
<td>\cdot H_k(T)$</td>
</tr>
<tr>
<td>$k$-th order empirical entropy</td>
<td>good for repetitions</td>
<td></td>
</tr>
<tr>
<td>good if frequencies are skewed</td>
<td>index in this lecture</td>
<td></td>
</tr>
<tr>
<td>blind to repetitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>T \ldots T</td>
<td>\cdot H_k(T \ldots T) \approx \ell</td>
</tr>
</tbody>
</table>
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 $f_1 \ldots f_i$

Given $T = \text{abababbbbaba}$

- $f_1 = a$
- $f_2 = b$
- $f_3 = \text{abab}$
- $f_4 = \text{bbb}$
- $f_5 = \text{aba}$
- $f_6 = \$
LZ-Compressed Index

**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 $f_1 \ldots f_i$

$T = \text{abababbbabab}$

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

**Now**

- LZ-compressed replacement for wavelet trees
- *rank* and *access* queries select also supported
- LZ-compression better than $H_k$-compression
Definition: Block Tree (1/4)

Given a text $T$ of length $n$ over an alphabet of size $\sigma$
- $\tau, s \in \mathbb{N}$ greater 1
- assume that $n = s \cdot \tau^h$ for some $h \in \mathbb{N}$

append $s$ until $n$ has this form

A **block tree** is a
- perfectly balanced tree with height $h$
- that may have leaves at higher levels such that
  - the root has $s$ children,
  - each other inner node has $\tau$ children
Definition: Block Tree (2/4)

In a block tree, leaves at
- the last level store characters or substrings of $T$
- at higher levels store special leftward pointer

Each node $u$
- represents a block $B^u$
- which is a substring of $T$ identified by a position

The root represents $T$ and its children consecutive blocks of $T$ of size $n/s$
Block Trees (3/4)

Definition: Block Tree (3/4)

Let $\ell_u$ be the level (depth) of node $u$
- the level of the root is 0

Let $B_1, B_2, \ldots$ be the blocks represented at level $\ell_u$ from left to right
- for any $i$, $B_i$ and $B_{i+1}$ are consecutive in $T$
- if $B_i B_{i+1}$ are the leftmost occurrence in $T$, the nodes representing the blocks are marked
**Block Trees (4/4)**

**Definition: Block Tree (4/4)**

If node $u$ is marked, then
- it is an internal node
- with $\tau$ children

otherwise, if node $u$ is not marked, then
- $u$ is a leaf storing
- pointers to nodes $v_i, v_{i+1}$ at the same level
  - that represent blocks $B_i$ and $B_{i+1}$
  - covering the leftmost occurrence of $B^u$
- offset to the occurrence of $B^u$ in $B_iB_{i+1}$

leaves on last level store text explicitly

\[ |B^u| = n/(s\tau^{\ell_u-1}) \]

if $|B^u|$ is small enough, store text explicitly

\[ |B^u| \in \Theta(\lg \sigma n) \]
Lemma: Number of Blocks per Level

The number of blocks in any level \( \ell > 0 \) in the block tree is at most \( 3\tau z \).
Lemma: Number of Blocks per Level
The number of blocks in any level \( \ell > 0 \) in the block tree is at most \( 3^\tau \tau \zeta \).

Proof (Sketch)
Let \( \ell > 0 \) be a level in the block tree and
- \( C = B_{i-1}B_iB_{i+1} \) a concatenation of three consecutive blocks at level \( \ell - 1 \)
- not containing the end of an LZ factor
- thus a leftwards occurrence in \( T \).
Lemma: Number of Blocks per Level

The number of blocks in any level \( \ell \) > 0 in the block tree is at most \( 3\tau z \).

Proof (Sketch)

Let \( \ell > 0 \) be a level in the block tree and

- \( C = B_{i-1}B_iB_{i+1} \) a concatenation of three consecutive blocks at level \( \ell - 1 \)
- not containing the end of an LZ factor
- thus a leftwards occurrence in \( T \)

\( B_{i-1} \) and \( B_{i+1} \) can only be marked if \( B_i \) is marked

- \( B_i \) is marked if it contains end of LZ factor
- there are only \( z \) LZ factors
Lemma: Number of Blocks per Level
The number of blocks in any level $\ell > 0$ in the block tree is at most $3\tau z$

Proof (Sketch)
Let $\ell > 0$ be a level in the block tree and
- $C = B_{i-1}B_iB_{i+1}$ a concatenation of three consecutive blocks at level $\ell - 1$
- not containing the end of an LZ factor
- thus a leftwards occurrence in $T$

$B_{i-1}$ and $B_{i+1}$ can only be marked if $B_i$ is marked
- $B_i$ is marked if it contains end of LZ factor
- there are only $z$ LZ factors

Each marked block results in $\tau$ children
Lemma: Number of Blocks per Level

The number of blocks in any level \( \ell > 0 \) in the block tree is at most \( 3\tau z \)

- \( O(\tau z) \) blocks per level

Proof (Sketch)

Let \( \ell > 0 \) be a level in the block tree and

- \( C = B_{i-1}B_iB_{i+1} \) a concatenation of three consecutive blocks at level \( \ell - 1 \)
- not containing the end of an LZ factor
- thus a leftwards occurrence in \( T \)

\( B_{i-1} \) and \( B_{i+1} \) can only be marked if \( B_i \) is marked
- \( B_i \) is marked if it contains end of LZ factor
- there are only \( z \) LZ factors

Each marked block results in \( \tau \) children
Lemma: Number of Blocks per Level

The number of blocks in any level \( \ell > 0 \) in the block tree is at most \( 3\tau z \).

- \( O(\tau z) \) blocks per level
- unmarked block requires \( O(\lg n) \) bits of space

Proof (Sketch)

Let \( \ell > 0 \) be a level in the block tree and
- \( C = B_{i-1}B_iB_{i+1} \) a concatenation of three consecutive blocks at level \( \ell - 1 \)
- not containing the end of an LZ factor
- thus a leftwards occurrence in \( T \)

\( B_{i-1} \) and \( B_{i+1} \) can only be marked if \( B_i \) is marked
- \( B_i \) is marked if it contains end of LZ factor
- there are only \( z \) LZ factors

Each marked block results in \( \tau \) children.
Lemma: Number of Blocks per Level

The number of blocks in any level $> 0$ in the block tree is at most $3\tau z$

- $O(\tau z)$ blocks per level
- unmarked block requires $O(\lg n)$ bits of space
- marked block requires $O(\tau \lg n)$ bits of space

charged to child

Proof (Sketch)

Let $\ell > 0$ be a level in the block tree and

- $C = B_{i-1}B_iB_{i+1}$ a concatenation of three consecutive blocks at level $\ell - 1$
- not containing the end of an LZ factor
- thus a leftwards occurrence in $T$

$B_{i-1}$ and $B_{i+1}$ can only be marked if $B_i$ is marked
- $B_i$ is marked if it contains end of LZ factor
- there are only $z$ LZ factors

Each marked block results in $\tau$ children
Lemma: Number of Blocks per Level

The number of blocks in any level \( \ell > 0 \) in the block tree is at most \( 3 \tau z \)

- \( O(\tau z) \) blocks per level
- unmarked block requires \( O(\lg n) \) bits of space
- marked block requires \( O(\tau \lg n) \) bits of space
  - charged to child
- last level has \( O(\tau z) \) blocks with plain text
  - \( O(\lg \sigma n) \) symbols of \( \lceil \lg n \rceil \) bits
  - requiring \( O(\lg n) \) bits per block

Proof (Sketch)

Let \( \ell > 0 \) be a level in the block tree and

- \( C = B_{i-1}B_iB_{i+1} \) a concatenation of three consecutive blocks at level \( \ell - 1 \)
- not containing the end of an LZ factor
- thus a leftwards occurrence in \( T \)
- \( B_{i-1} \) and \( B_{i+1} \) can only be marked if \( B_i \) is marked
- \( B_i \) is marked if it contains end of LZ factor
- there are only \( z \) LZ factors

Each marked block results in \( \tau \) children
Lemma: Number of Blocks per Level

The number of blocks in any level \( \ell > 0 \) in the block tree is at most \( 3\tau z \)

- \( O(\tau z) \) blocks per level
- unmarked block requires \( O(\lg n) \) bits of space
- marked block requires \( O(\tau \lg n) \) bits of space
  - charged to child
- last level has \( O(\tau z) \) blocks with plain text
  - \( O(\lg, n) \) symbols of \( \lfloor \lg n \rfloor \) bits
  - requiring \( O(\lg n) \) bits per block
- \( h = \lg_{\tau} n \frac{\lg \sigma}{\lg n} \) and \( O(s) \) pointers to top level

Proof (Sketch)

Let \( \ell > 0 \) be a level in the block tree and
- \( C = B_{i-1}B_iB_{i+1} \) a concatenation of three consecutive blocks at level \( \ell - 1 \)
- not containing the end of an LZ factor
- thus a leftwards occurrence in \( T \)

\( B_i \) and \( B_{i+1} \) can only be marked if \( B_i \) is marked
- \( B_i \) is marked if it contains end of LZ factor
- there are only \( z \) LZ factors

Each marked block results in \( \tau \) children
Lemma: Number of Blocks per Level

The number of blocks in any level \( \ell > 0 \) in the block tree is at most \( 3\tau z \).

- \( O(\tau z) \) blocks per level
- unmarked block requires \( O(\lg n) \) bits of space
- marked block requires \( O(\tau \lg n) \) bits of space
- last level has \( O(\tau z) \) blocks with plain text
  - \( O(\lg_{\sigma} n) \) symbols of \( \lceil \lg n \rceil \) bits
  - requiring \( O(\lg n) \) bits per block
- \( h = \lg_{\tau} \frac{n \lg_{\sigma} \lg n}{\sigma \lg n} \) and \( O(s) \) pointers to top level
- rounding up length adds \( \leq O(h\tau) \) blocks per level

Proof (Sketch)

Let \( \ell > 0 \) be a level in the block tree and

- \( C = B_{i-1}B_iB_{i+1} \) a concatenation of three consecutive blocks at level \( \ell - 1 \)
- not containing the end of an LZ factor
- thus a leftwards occurrence in \( T \)

\( B_{i-1} \) and \( B_{i+1} \) can only be marked if \( B_i \) is marked

- \( B_i \) is marked if it contains end of LZ factor
- there are only \( z \) LZ factors

Each marked block results in \( \tau \) children
Lemma: Space Requirements of Block Trees

Given a text $T$ of length $n$ over an alphabet of size $\sigma$ and integers $s, \tau > 1$, a block tree of $T$ has height $h = \lg_\tau \frac{n \lg \sigma}{s \lg n}$. The block tree requires

$$O((s + z\tau \lg_\tau \frac{n \lg \sigma}{s \lg n}) \lg n)$$

bits of space,

where $z$ is the number of LZ77 factors of $T$
Lemma: Space Requirements of Block Trees

Given a text $T$ of length $n$ over an alphabet of size $\sigma$ and integers $s, \tau > 1$, a block tree of $T$ has height

$$h = \lg_\tau \frac{n \lg \sigma}{s \lg n}.$$  

The block tree requires

$$O((s + z \tau \lg_\tau \frac{n \lg \sigma}{s \lg n}) \lg n)$$

bits of space,

where $z$ is the number of LZ77 factors of $T$.

- $s = z$ results in a tree of height $O(\lg_\tau \frac{n \lg \sigma}{z \lg n})$.
- space requirements $O(z \tau \lg_\tau \frac{n \lg \sigma}{z \lg n})$ bits
- however $z$ not known.
queries are easy to realize
if not supported directly, additional information can be stored for blocks

Access Query

Given position \( i \) return \( T[i] \)
- follow nodes that represent block containing \( T[i] \)
- of not marked follow pointer and consider offset
- at leaf, if last level, return character
- else, follow pointer and continue

\[ \text{time } O(\lg_{\tau} n \frac{\lg \sigma}{s \lg n}) \]
Rank Queries in Block Trees

- for each block add histogram \( Hist_{Bu} \) for prefix of \( T \) up to block (not containing)
- \( O(\sigma (s + z^\tau ) \frac{n\lg n}{s\lg n} \lg n) \) bits of space

**Rank Query**

Given position \( i \) and character \( \alpha \) return \( rank_\alpha (T, i) \)

- follow nodes that represent block containing \( T[i] \)
- remember \( Hist_{Bu}[\alpha] \)
- of not marked follow pointer and consider offset
- at leaf, if last level, compute local rank \( \odot \) binary rank for each character
- else, follow pointer and continue

- time \( O(\lg \frac{n\lg \sigma}{s\lg n}) \)

- example on the board
Construction of Block Trees

\( O(n) \) Working Space

- build Aho-Corasick automaton for containing all pairs of consecutive unmarked blocks
- identify unmarked blocks on next level
- \( O(n(1 + \lg \frac{z}{s})) \) time and \( O(n) \) space

Pruning

Size of block tree can be reduced further by identifying some blocks that are not necessary. Those blocks can easily be identified by examining the \( O(s + z) \) working space.

Replacing Aho-Corasick Automaton

Replace Aho-Corasick automaton with Karp-Rabin fingerprints to validate if matching fingerprints due to matching strings. This approach results in an expected time \( O(n(1 + \lg \frac{z}{s})) \) and space \( O(n) \) only for the expected construction time. In practice, queries are very fast, but construction is very slow.
Construction of Block Trees

\( O(n) \) Working Space

- build Aho-Corasick automaton for containing all pairs of consecutive unmarked blocks
- identify unmarked blocks on next level
- \( O(n(1 + \lg \frac{z}{s})) \) time and \( O(n) \) space

Pruning

- size of block tree can be reduced further
- some blocks not necessary
- those blocks can easily be identified
## Construction of Block Trees

### $O(n)$ Working Space
- build Aho-Corasick automaton for containing all pairs of consecutive unmarked blocks
- identify unmarked blocks on next level
- $O(n(1 + \lg_s \frac{z}{\tau}))$ time and $O(n)$ space

### $O(s + z\tau)$ Working Space
- replace Aho-Corasick automaton with Karp-Rabin fingerprints
- validate if matching fingerprints due to matching strings \(\mathbb{1}\) Monte Carlo algorithm
- $O(n(1 + \lg_s \frac{z}{\tau}))$ expected time and $O(n)$ space
- only expected construction time!

### Pruning
- size of block tree can be reduced further
- some blocks not necessary
- those blocks can easily be identified
Construction of Block Trees

**$O(n)$ Working Space**
- build Aho-Corasick automaton for containing all pairs of consecutive unmarked blocks
- identify unmarked blocks on next level
- $O(n(1 + \lg \frac{z}{s}))$ time and $O(n)$ space

**Pruning**
- size of block tree can be reduced further
- some blocks not necessary
- those blocks can easily be identified

**$O(s + z\tau)$ Working Space**
- replace Aho-Corasick automaton with Karp-Rabin fingerprints
- validate if matching fingerprints due to matching strings ⏯ Monte Carlo algorithm
- $O(n(1 + \lg \frac{z}{s}))$ expected time and $O(n)$ space
- only expected construction time!

- queries very fast in practice
- construction very slow in practice
- good topic for thesis 😊
Let $T$ be a text, then
- $r(T)$ is number of BWT runs of $T$
- $z(T)$ is number of LZ77 factors of $T$

Definition: Burrows-Wheeler Transform [BW94]

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>$$</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>
</tr>
</tbody>
</table>
Lemma: Number of BWT Runs

Let $T$ be a text of length $n$, then

$$r(T) \in O(z(T) \lg^2 n)$$

- $LCP[i]$ is irreducible if $i = 1$ or $BWT[i] \neq BWT[i - 1]$
- number of irreducible LCP-values is $r(T)$

Lemma: Sum of Irreducible LCP-Values

The number of irreducible LCP-Values in $[\ell, 2\ell]$ is in $O(z\ell \lg n)$

- $r(T)$ is number of irreducible LCP-values
- apply lemma for $[2^i, 2^{i+1})$ for $i \in [0, \lfloor \lg n \rfloor]$
- number of $LCP[i] = 0$ entries is $\sigma \leq z$
Relation Between BWT Runs and LZ Factors (3/3)

Lemma: Number of Occurrences of Substrings

For any $\ell > 1$, the number of distinct substrings of $T$ of length $\ell$ is $\leq z\ell$

Proof (Sketch)

- Consider any substring of length $\ell > 1$
- If substrings is contained in LZ factor, there is previous occurrence
- Distinct substrings overlap LZ factors
- There are at most $\ell$ substring per end of LZ factor

- Use number of distinct substrings
- To show that the number of irreducible LCP-values
- Is limited as stated in lemma
Conclusion and Outlook

This Lecture
- block trees
- \( r \in O(z \log^2 n) \)

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- \( r \)-Index
This Lecture
- block trees
- \( r \in O(z \lg^2 n) \)

Open Questions
- efficient block tree construction
- linear time block tree construction

Linear Time Construction
Conclusion and Outlook

This Lecture
- block trees
- $r \in O(z \lg^2 n)$

Open Questions
- efficient block tree construction
- linear time block tree construction

Next Lecture
- suffix array construction in different models of computation

Linear Time Construction

Graph showing relationships between data structures like ST, SA, WT, LZ, LCP, BWT, FM-Index, and r-Index.

