Text Indexing

Lecture 09: LZ Compressed Indeces

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License @cc-by-sa: www.creativecommons.org/licenses/by-sa/4.0 | commit 0cd47f0 compiled at 2023-12-18-13:33
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. \( \text{for } i = m, \ldots, 1 \text{ do} \)
3. \( s = C[P[i]] + \text{rank}_{P[i]}(s - 1) + 1 \)
4. \( e = C[P[i]] + \text{rank}_{P[i]}(e) \)
5. \( \text{if } s > e \text{ then} \)
6. \( \quad \text{return } \emptyset \)
7. \( \text{return } [s, e] \)
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** $\text{BackwardsSearch} (P[1..n], C, rank)$:

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

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

\[
\text{Function } \text{BackwardsSearch}(P[1..n], C, \text{rank}) : \\
\begin{align*}
1 & \quad s = 1, \quad e = n \\
2 & \quad \text{for } i = m, \ldots, 1 \text{ do} \\
3 & \quad \quad s = C[P[i]] + \text{rank}_{P[i]}(s - 1) + 1 \\
4 & \quad \quad e = C[P[i]] + \text{rank}_{P[i]}(e) \\
5 & \quad \text{if } s > e \text{ then} \\
6 & \quad \quad \quad \text{return } () \\
7 & \quad \text{return } [s, e]
\end{align*}
\]
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

$$|\underbrace{T \ldots T}_\ell| \cdot \frac{H_k(T \ldots T)}{\ell} \approx \ell |T| \cdot H_k(T)$$
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
  $$|T \ldots T| \cdot H_k(T \ldots T) \approx \ell |T| \cdot H_k(T)$$

**LZ-Compression**
- references to previous occurrences
- each LZ factor can be encoded in $O(1)$ space
- good for repetitions
- index in this lecture

---

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
  $$|T \ldots T| \cdot H_k(T \ldots T) \approx \ell |T| \cdot H_k(T)$$

LZ-Compression

- references to previous occurrences
- each LZ factor can be encoded in $O(1)$ space
- good for repetitions
- index in this lecture
### 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></td>
<td></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></td>
<td></td>
</tr>
<tr>
<td>- good if frequencies are skewed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- blind to repetitions $</td>
<td>T\ldots T</td>
<td>\cdot H_k(T\ldots T) \approx \ell</td>
</tr>
<tr>
<td>- references to previous occurrences</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- each LZ factor can be encoded in $O(1)$ space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- good for repetitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- index in this lecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- used in powerful index</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- theoretical insight in this lecture</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 $f_1 \ldots f_i$

$T = \text{abababbbababa}$

- $f_1 = a$
- $f_2 = b$
- $f_3 = \text{abab}$
- $f_4 = \text{bbb}$
- $f_5 = \text{aba}$
- $f_6 = \$$
**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
- \( T = f_1f_2 \ldots f_z \) 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 \( f_1 \ldots f_i \)

**Example**

\( T = abababbbaba$ \)

- \( f_1 = a \)
- \( f_2 = b \)
- \( f_3 = abab \)
- \( f_4 = bbb \)
- \( f_5 = aba \)
- \( f_6 = $ \)

**Now**

- LZ-compressed replacement for wavelet trees
- **rank** and **access** queries \( \checkmark \) **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
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 $n/s$
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 $B_iB_{i+1}$ 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
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

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

if \( |B^u| \) is small enough, store text explicitly
\[ |B^u| \in \Theta(\log_\sigma n) \]
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 $B_i B_{i+1}$

leaves on last level store text explicitly

| $B^u| = n/(s^\ell_u - 1)$
- if $|B_u|$ is small enough, store text explicitly
- $|B^u| \in \Theta(lg_\sigma n)$

PINGO how many blocks are there per level?
Lemma: Number of Blocks per Level

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

Proof (Sketch)

Let $\ell > 0$ be a level in the block tree and $C = B_i - 1 + 1$. 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 the end of an 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 \(3^\tau z\).

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 \( \ell > 0 \) 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

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 \( \lfloor \lg n \rfloor \) bits
  - requiring \( O(\lg \sigma) \) 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 $> 0$ 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 $O(\lg \sigma)$ bits per block
- $h = \lg_\tau \frac{n \lg \sigma}{s \lg n}$ 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 $T$
- $B_{i-1}$ 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 if 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 $O(\lg \sigma)$ bits per block
- $h = \lg_{\tau \frac{n\lg \sigma}{\lg n}}$ and $O(s)$ pointers to top level
- rounding up length adds $\leq O(\tau)$ 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 $h = \lg_{\tau} \frac{n \log \sigma}{s \log n}$. The block tree requires

$$O((s + z \tau \log_{\tau} \frac{n \log \sigma}{s \log n}) \log n)$$

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

$\text{time } O(\lg \tau \frac{n \lg \sigma}{s} \frac{n}{\lg n})$
Access Queries in Block Trees

- 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

Time $O(\lg \tau \frac{n \lg \sigma}{s \lg n})$
Rank Queries in Block Trees

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

**Rank Query**

Given position $i$ and character $\alpha$ return $\text{rank}_{\alpha}(T, i)$

- 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 for each character
- else, follow pointer and continue

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

- example on the board
Rank Queries in Block Trees

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

Rank Query

Given position $i$ and character $\alpha$ return $\text{rank}_{\alpha}(T, i)$

- 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 for each character
- Else, follow pointer and continue

Time $O(\frac{n\lg \sigma}{s\lg n})$
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
- $O(n(1 + \log \frac{s}{z}))$ 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

$O(n(1 + \log \frac{s}{z}))$ expected time and $O(n)$ space

Only expected construction time!

Queries very fast in practice

Construction very slow in practice

Space-efficient construction of block trees
**Construction of Block Trees**

<table>
<thead>
<tr>
<th><strong>O(n)</strong> Working Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>- build Aho-Corasick automaton for containing all pairs of consecutive unmarked blocks</td>
</tr>
<tr>
<td>- identify unmarked blocks on next level</td>
</tr>
<tr>
<td>- $O(n(1 + \lg(\frac{z}{\tau})))$ time and $O(n)$ space</td>
</tr>
</tbody>
</table>

**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
- $O(n(1 + \lg \frac{z}{\bar{s}}))$ 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 Monte Carlo algorithm
- $O(n(1 + \lg \frac{z}{\bar{s}}))$ 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
- $O(n(1 + \lg_{\frac{\tau}{s}}))$ 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
- $O(n(1 + \lg_{\frac{\tau}{s}}))$ expected time and $O(n)$ space
- only expected construction time!

- queries very fast in practice
- construction very slow in practice
- space-efficient construction of block trees
## State-of-Block-Tree-Construction

<table>
<thead>
<tr>
<th>Method</th>
<th>Reference</th>
<th>Working Space</th>
<th>Time</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aho-Corasic</td>
<td>[Bel+21]</td>
<td>$O(n)$</td>
<td>$O(n(1 + \log_{\tau}(z\tau/s)))$</td>
<td>no</td>
</tr>
<tr>
<td>Fingerprints</td>
<td>[Bel+21]</td>
<td>$O(s + z\tau \log_{\tau} \left(\frac{n \log \sigma}{s \log n}\right))$</td>
<td>$O(n(1 + \log_{\tau}(z\tau/s)))$</td>
<td>yes (slow)</td>
</tr>
<tr>
<td>LPF Array</td>
<td>[KopplKM2023LPFBlockTrees]</td>
<td>$O(n)$</td>
<td>$O(n(1 + \log_{\tau}(z\tau/s)))$</td>
<td>yes (fast)</td>
</tr>
</tbody>
</table>
Lempel-Ziv Parse

A A A A B B A A A B B A B B A A A
Lempel-Ziv Parse
Lempel-Ziv Parse
Lempel-Ziv Parse
Lempel-Ziv Parse
Lempel-Ziv Parse
Lempel-Ziv Parse
Lempel-Ziv Parse

A A A A B B A A A A B B A A A A
Lempel-Ziv Parse
Lempel-Ziv Parse

LPF

PrevOcc
Our Algorithm (Marking of Nodes)

LPF

|0| 3| 2| 1| 0| 1| 6| 5| 4| 3| 2| 5| 4| 3| 2| 1|

PrevOcc

|-1| 1| 2| 3| -1| 5| 2| 3| 4| 5| 6| 4| 5| 6| 7| 8|

A A A A B B A A A B B A A A A A A A A A A
Our Algorithm (Marking of Nodes)

\[\text{LPF: } 0 \ 3 \ 2 \ 1 \ 0 \ 1 \ 6 \ 5 \ 4 \ 3 \ 2 \ 5 \ 4 \ 3 \ 2 \ 1\]

\[\text{PrevOcc: } -1 \ 1 \ 2 \ 3 \ -1 \ 5 \ 2 \ 3 \ 4 \ 5 \ 6 \ 4 \ 5 \ 6 \ 7 \ 8\]

\[\text{A A A A B B A A A B B A A A A A B B A B A B A A A}\]
Our Algorithm (Marking of Nodes)
Our Algorithm (Marking of Nodes)

LPF:

\[
\begin{array}{cccccccc}
0 & 3 & 2 & 1 & 0 & 1 & 6 & 5 & 4 & 3 & 2 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

PrevOcc:

\[
\begin{array}{cccccccc}
-1 & 1 & 2 & 3 & -1 & 5 & 2 & 3 & 4 & 5 & 6 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]
Our Algorithm (Marking of Nodes)

LPF: 0 3 2 1 0 1 6 5 4 3 2 5 4 3 2 1
PrevOcc: -1 1 2 3 -1 5 2 3 4 5 6 4 5 6 7 8

A A A A B B B B A A A A B B A A A A

A A A A B B A A A B B A A A A A A
Our Algorithm (Marking of Nodes)

![Diagram of marking nodes](image)
Our Algorithm (Marking of Nodes)

LPF

PrevOcc

A A A A B B A A A B B A A
Our Algorithm (Marking of Nodes)

LPF: 0 3 2 1 0 1 6 5 4 3 2 5 4 3 2 1

PrevOcc: -1 1 2 3 -1 5 2 3 4 5 6 4 5 6 7 8

A A A A B B A A A B B A A
Our Algorithm (Marking of Nodes)

LPF

PrevOcc

A A A A B B A A A B B A B A A A
Our Algorithm (Marking of Nodes)

LPF: 0 3 2 1 0 1 6 5 4 3 2 5 4 3 2 1

PrevOcc: -1 1 2 3 -1 5 2 3 4 5 6 4 5 6 7 8

A A A A B B A A A B B A A A
Our Algorithm (Marking of Nodes)
Experimental Evaluation

- highly tuned implementation
- tree consists only of bit and compact vectors
- tuning parameter
  - degree root $s = \{1, z\}$ (only we have $s = z$)
  - degree other nodes $\tau = \{2, 4, 8, 16\}$
  - number characters in leaves $b = \{2, 4, 8, 16\}$
Experimental Evaluation

- highly tuned implementation
- tree consists only of bit and compact vectors
- tuning parameter
  - degree root $s = \{1, z\}$ (only we have $s = z$)
  - degree other nodes $\tau = \{2, 4, 8, 16\}$
  - number characters in leaves $b = \{2, 4, 8, 16\}$
- original FP BT [Bel+21]
- our reimplementation of the original FP BT
- our LPF BT construction with $s = 1$ and $s = z$
- dynamic programming variants
- parallelization
- no comparison with wavelet trees (faster)
- repetitive instances from P&C corpus
- non-repetitive instances from P&C corpus
Highly Repetitive Inputs (Access Only)

- **cere**
- **einstein.en**
- **kernel**

<table>
<thead>
<tr>
<th>throughput (MiB/s)</th>
<th>space (bit/n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0</td>
<td>0.6</td>
</tr>
<tr>
<td>4.0</td>
<td>0.4</td>
</tr>
<tr>
<td>2.0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

- Reimplementation FP $BT_{s=1}$
- LPF $BT_{z}$
- LPF $BT_{s=1}$
- Original FP $BT_{s=1}$ [Bel+21]
Highly Repetitive Inputs (with Rank and Select Support)

Throughput (MiB/s) vs. space (bit/n)

- **cere**
- **einstein.en**
- **kernel**

Markers:
- * reimplementations FP BT$_{s=1}$
- ▲ LPF BT$_{s=z}$
- + LPF BT$_{s=1}$
- * original FP BT$_{s=1}$ [Bel+21]
Conclusion and Outlook

This Lecture
- block trees

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- \( r \)-Index
Conclusion and Outlook

This Lecture

- block trees
- efficient block tree construction
- linear time block tree construction

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- r-Index
Conclusion and Outlook

This Lecture
- block trees
- efficient block tree construction
- linear time block tree construction

Next Lecture
- move data structure and relation of BWT runs and LZ factors

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- r-Index
Bibliography I


