Text Indexing

Lecture 09: LZ Compressed Indices

Florian Kurpicz
PINGO

https://pingo.scc.kit.edu/309703
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 $r$

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

1. $s = 1, e = n$
2. for $i = m, \ldots, 1$ do
3.   
4.   
5.   
6.   
7. return $[s, e]$
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
  $$|T \ldots T| \cdot H_k(T) \approx \ell |T| \cdot H_k(T)$$

**LZ-Compression**
- References to previous occurrences
- Each LZ factor can be encoded in $O(1)$ space
- Good for repetitions
- Index in this lecture

**BWT-Compression**
- Used in powerful index
- Theoretical insight in this lecture
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 $f_1 \ldots f_i$

Example:

$T = \text{abababbbaba}$

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

Now

- LZ-compressed replacement for wavelet trees
- rank and access queries $\textbf{select}$ 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

Block Trees [Bel+21] (1/4)
Block Trees (2/4)

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 \( n/s \)
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.
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

$|B^u| = n/(s \tau^\ell_u - 1)$

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

PINGO how many blocks are there per level?
Block Trees are LZ Compressed (1/2)

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 $O(\lg \sigma)$ bits per block
- $h = \lg_\tau \frac{n \lg_\sigma}{s \lg n}$ 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

$$h = \log_{\tau} \frac{n \log \sigma}{s \log n}.$$  

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 $T$.

- $s = z$ results in a tree of height $O(\log_{\tau} \frac{n \log \sigma}{z \log n})$
- space requirements $O(z \tau \log_{\tau} \frac{n \log \sigma}{z \log n} \log n)$ 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

$\text{time } O(\lg \tau \frac{n \lg \sigma}{s \lg n})$
for each block add histogram $Hist_{Bu}$ for prefix of $T$ up to block (not containing)

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

---

**Example**

Example on the board

---

PINGO what can be problematic with block tree construction?
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

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

- 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>$O(s + z \tau \log_{\tau}(\frac{n \log \sigma}{s \log n}))$</td>
<td>$O(n(1 + \log_{\tau}(z \tau/s)))$</td>
<td>expected yes</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</td>
</tr>
</tbody>
</table>
Our Algorithm (Marking of Nodes)
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**

- **reimplementation FP BT** $s=1$
- **LPF BT** $s=z$
- **LPF BT** $s=1$
- **original FP BT** $s=1$ [Bel+21]
Highly Repetitive Inputs (with Rank and Select Support)

![Graphs showing throughput vs. space for different inputs: cere, einstein.en, kernel.]

- cere
- einstein.en
- kernel

Legend:
- * reimplementation FP $B_{T=1}$
- △ LPF $B_{T=z}$
- + LPF $B_{T=1}$
- ★ original FP $B_{T=1}$ [Bel+21]
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

<table>
<thead>
<tr>
<th>ST</th>
<th>SA</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LZ</td>
<td>LCP</td>
<td>BWT</td>
</tr>
</tbody>
</table>

FM-Index
r-Index
Bibliography I


