Karlsruhe Institute of Technology

Text Indexing

Lecture 09: LZ Compressed Indeces

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License: www.creativecommons.org/licenses/by-sa/4.0 | commit 0cd47f0 compiled at 2023-12-18-13:33

PINGO

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

Recap: FM-Index and *r*-Index

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

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

$s = 1, e = n$

for $i = m, \ldots, 1$ **do**

  $s = C[P[i]] + rank_{P[i]}(s - 1) + 1$

  $e = C[P[i]] + rank_{P[i]}(e)$

  if $s > e$ **then**

    return $\emptyset$

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*)**:**

$s = 1, e = n$
for $i = m, \dots, 1$ **do**
  $s = C[P[i]] + rank_{P[i]}(s - 1) + 1$
  $e = C[P[i]] + rank_{P[i]}(e)$
  if $s > e$ **then**
    return $\emptyset$
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*

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

```
s = 1, e = n
for i = m, ..., 1 do
    s = C[P[i]] + rank_{P[i]}(s − 1) + 1
    e = C[P[i]] + rank_{P[i]}(e)
    if s > e then
        return ∅
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
  $|\underbrace{T \dots T}_{\ell}| \cdot H_k(\underbrace{T \dots T}_{\ell}) \approx \ell|T| \cdot H_k(T)$

Different Types of Compression

Statistical Coding

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

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

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

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 Σ, 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 ≥ 2 times in $f_1 \ldots f_i$

$T =$ `abababbbbaba$`

- $f_1 =$ `a`
- $f_2 =$ `b`
- $f_3 =$ `abab`
- $f_4 =$ `bbb`
- $f_5 =$ `aba`
- $f_6 =$ `$`

LZ-Compressed Index

Karlsruhe Institute of Technology

Definition: LZ77 Factorization [ZL77]

Given a text T of length n over an alphabet Σ, 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 ≥ 2 times in $f_1 \ldots f_i$

$T =$ abababbbbaba$

- $f_1 =$ 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 ℹ *select* also supported
- LZ-compression better than H_k-compression

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

Definition: Block Tree (1/4)

Given a text T of length n over an alphabet of size σ

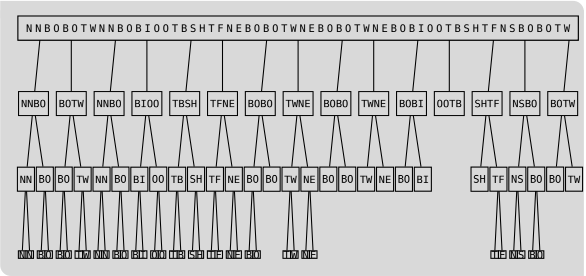
- $\tau, s \in \mathbb{N}$ greater 1
- assume that $n = s \cdot \tau^h$ for some $h \in \mathbb{N}$
  - ℹ append \$s until n has this form

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 τ children



Block Trees (2/4)

Definition: Block Tree (2/4)

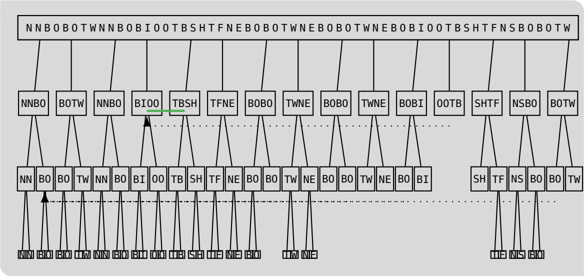
In a block tree, leaves at

- 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



Block Trees (3/4)

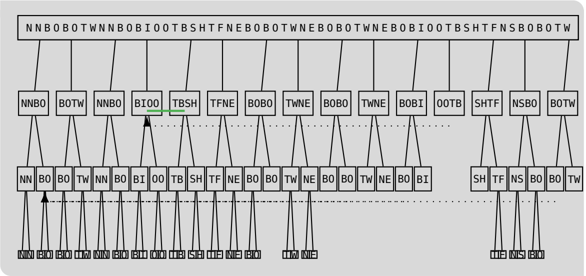
Definition: Block Tree (3/4)

Let ℓ_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 ℓ_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)

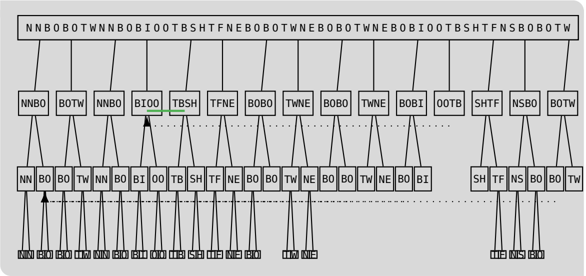
If node u is marked, then

- it is an internal node
- with τ 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)

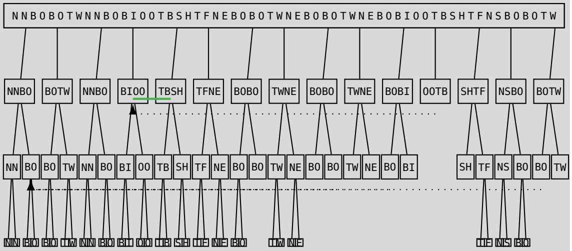
If node u is marked, then

- it is an internal node
- with τ 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
  - $|B^u \in \Theta(\lg_\sigma n)|$

Block Trees (4/4)

Definition: Block Tree (4/4)

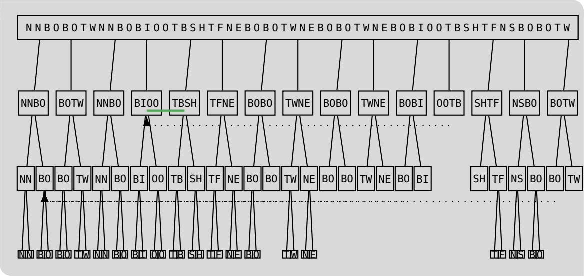
If node u is marked, then

- it is an internal node
- with τ 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
  - $|B^u \in \Theta(\lg_\sigma n)|$
- **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 > 0 in the block tree is at most $3\tau z$

Block Trees are LZ Compressed (1/2)

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

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

Block Trees are LZ Compressed (1/2)

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

- $C = B_{i-1}B_iB_{i+1}$ 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

Block Trees are LZ Compressed (1/2)

Karlsruhe Institute of Technology

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

- $C = B_{i-1}B_iB_{i+1}$ 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 τ children

Block Trees are LZ Compressed (1/2)

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

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 τ children

Block Trees are LZ Compressed (1/2)

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

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 τ children

Block Trees are LZ Compressed (1/2)

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
  ℹ 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 τ children

Block Trees are LZ Compressed (1/2)

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
  ℹ 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

Proof (Sketch)

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

- $C = B_{i-1} B_i B_{i+1}$ 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 τ children

Block Trees are LZ Compressed (1/2)

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
  - $O(\lg_\sigma n)$ 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

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 τ children

Block Trees are LZ Compressed (1/2)

Karlsruhe Institute of Technology

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
  ℹ 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

- $C = B_{i-1} B_i B_{i+1}$ 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 τ children

Block Trees are LZ Compressed (2/2)

Lemma: Space Requirements of Block Trees

Given a text T of length n over an alphabet of size σ 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

$$O((s + z\tau \lg_\tau \frac{n \lg \sigma}{s \lg n}) \lg n) \text{ bits of space,}$$

where z is the number of LZ77 factors of T

Block Trees are LZ Compressed (2/2)

Lemma: Space Requirements of Block Trees

Given a text T of length n over an alphabet of size σ 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

$$O((s + z\tau \lg_\tau \frac{n \lg \sigma}{s \lg n}) \lg n) \text{ bits of space,}$$

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

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

- example on the board

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

- example on the board

- **PINGO** can we answer rank queries the same way?

Rank Queries in Block Trees

- for each block add histogram $Hist_{B_u}$ 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 α return $rank_\alpha(T, i)$

- follow nodes that represent block containing $T[i]$
- remember $Hist_{B_u}[\alpha]$
- of not marked follow pointer and consider offset
- at leaf, if last level, compute local rank binary 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 $Hist_{B_u}$ 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 α return $rank_\alpha(T, i)$

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

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

- 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
- $O(n(1 + \lg_\tau \frac{z}{s}))$ time and $O(n)$ space

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

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
- $O(n(1 + \lg_\tau \frac{z}{s}))$ 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 ℹ Monte Carlo algorithm
- $O(n(1 + \lg_\tau \frac{z}{s}))$ 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

Method	Reference	Working Space	Time	I
Aho-Corasic	[Bel+21]	$O(n)$	$O(n(1+\log_\tau(z\tau/s)))$	n
Fingerprints	[Bel+21]	$O(s + z\tau\log_\tau(\frac{n\log\sigma}{s\log n}))$	$O(n(1+\log_\tau(z\tau/s)))$ expected	y
LPF Array	[**KopplKM2023LPFBlockTrees**]	$O(n)$	$O(n(1+\log_\tau(z\tau/s)))$	y

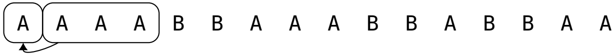
Lempel-Ziv Parse

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

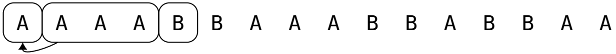
Lempel-Ziv Parse

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

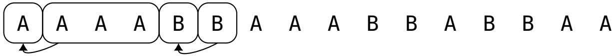
Lempel-Ziv Parse



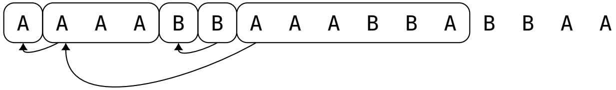
Lempel-Ziv Parse



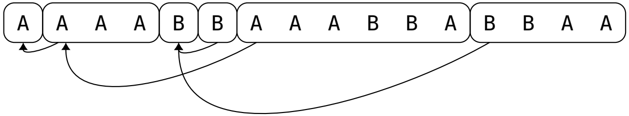
Lempel-Ziv Parse



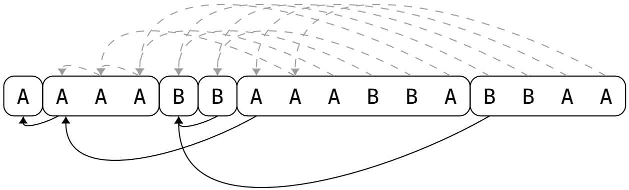
Lempel-Ziv Parse



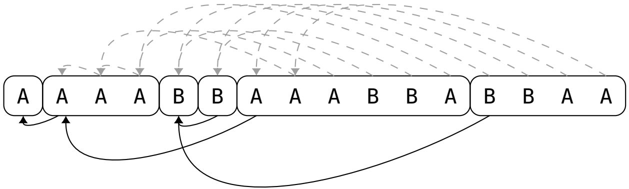
Lempel-Ziv Parse



Lempel-Ziv Parse

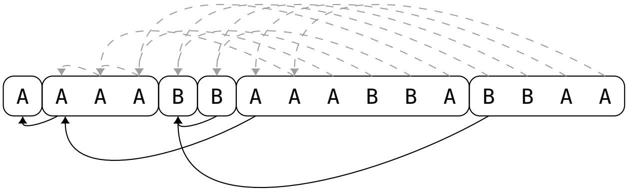


Lempel-Ziv Parse



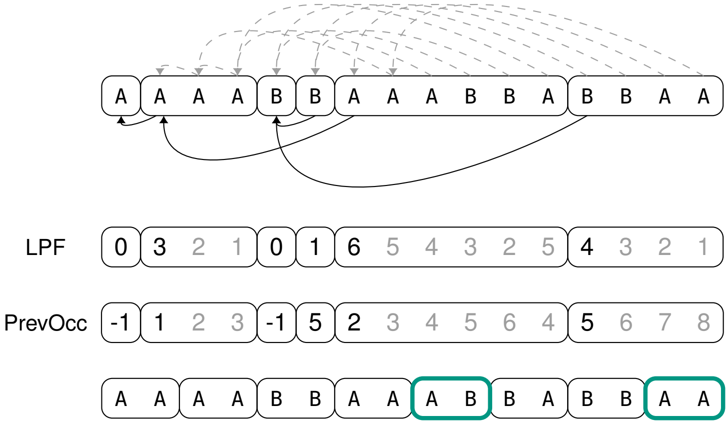
LPF	0	3	2	1	0	1	6	5	4	3	2	5	4	3	2	1

Lempel-Ziv Parse

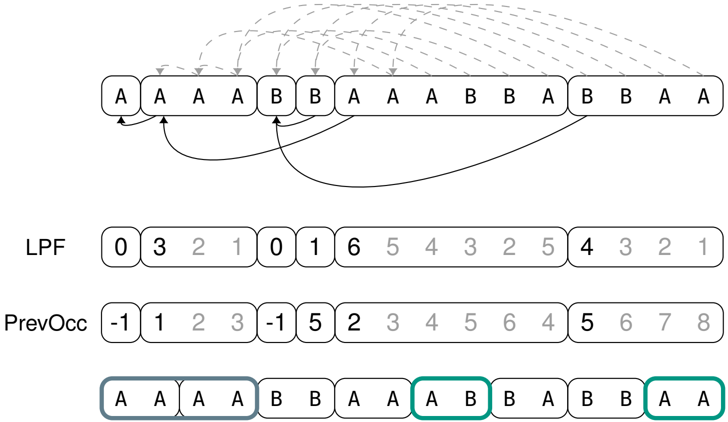


LPF	0	3	2	1	0	1	6	5	4	3	2	5	4	3	2	1
PrevOcc	-1	1	2	3	-1	5	2	3	4	5	6	4	5	6	7	8

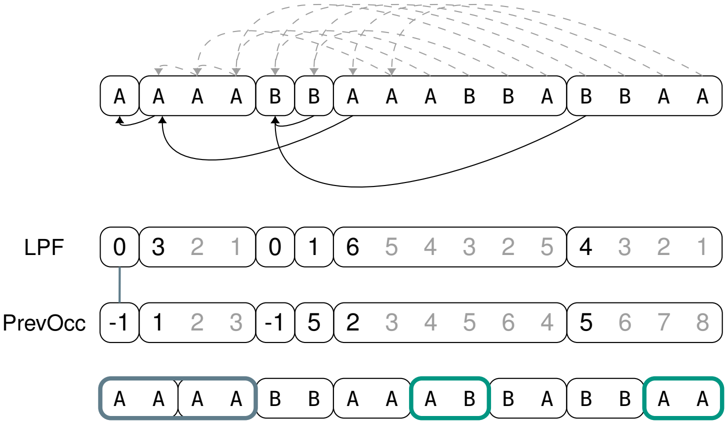
Our Algorithm (Marking of Nodes)



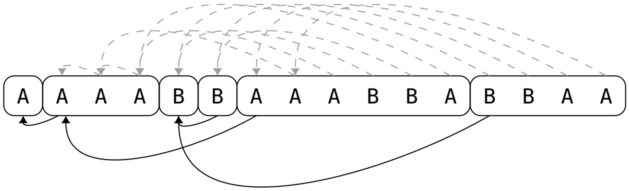
Our Algorithm (Marking of Nodes)



Our Algorithm (Marking of Nodes)

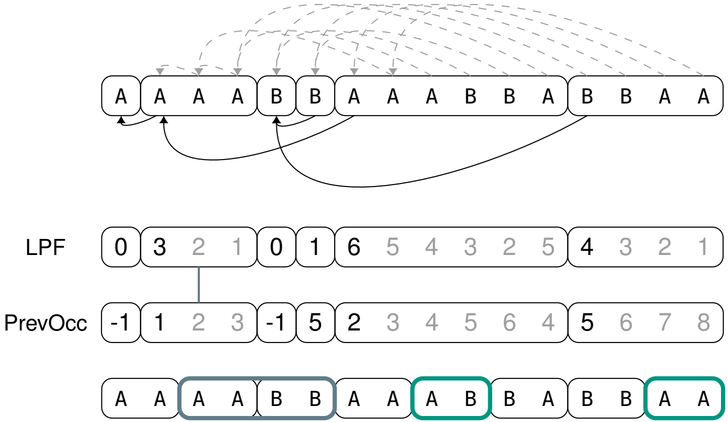


Our Algorithm (Marking of Nodes)

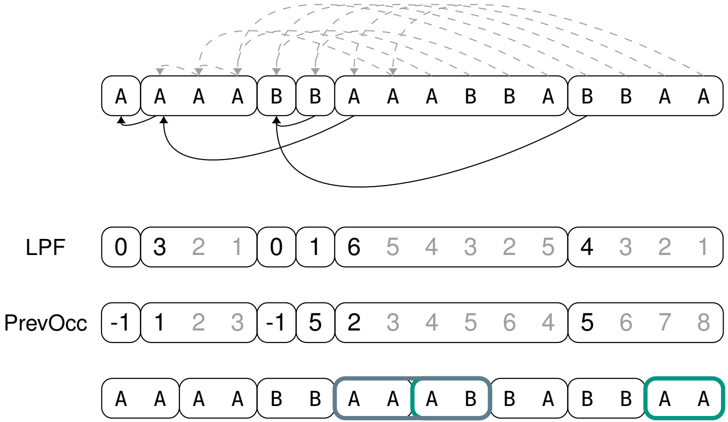


LPF	0	3	2	1	0	1	6	5	4	3	2	5	4	3	2	1
PrevOcc	-1	1	2	3	-1	5	2	3	4	5	6	4	5	6	7	8
	A	A	A	A	B	B	A	A	A	B	B	A	B	B	A	A

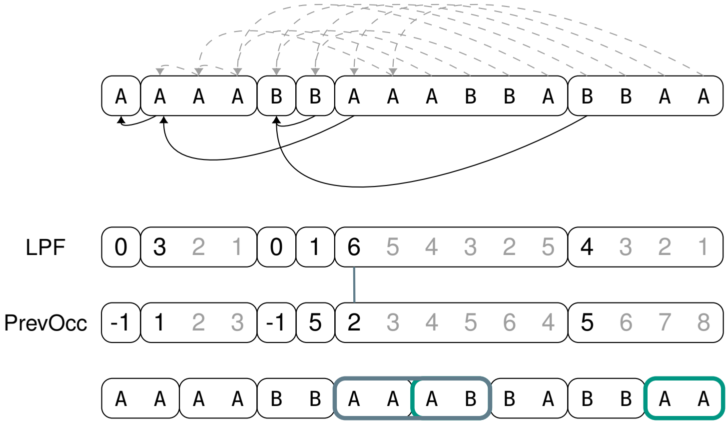
Our Algorithm (Marking of Nodes)



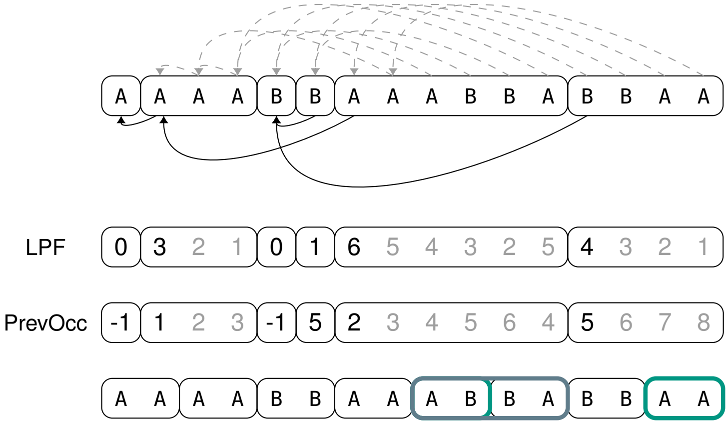
Our Algorithm (Marking of Nodes)



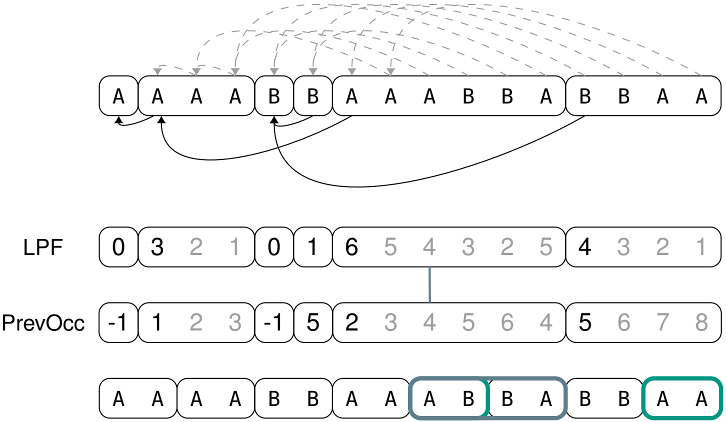
Our Algorithm (Marking of Nodes)



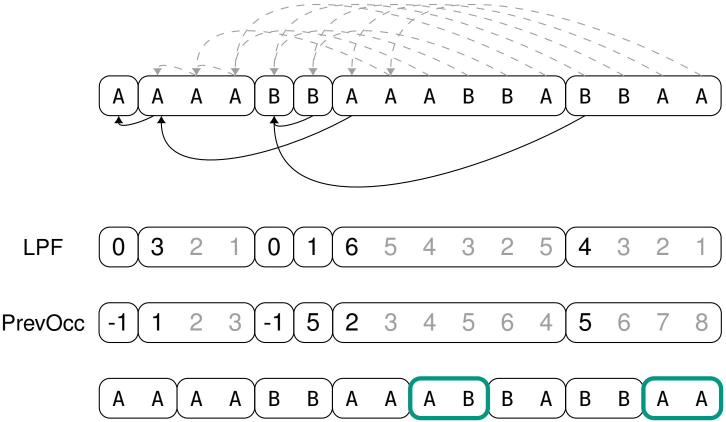
Our Algorithm (Marking of Nodes)



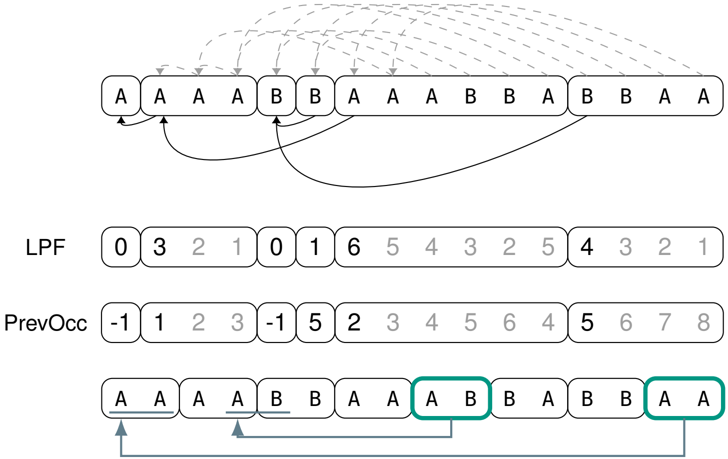
Our Algorithm (Marking of Nodes)



Our Algorithm (Marking of Nodes)



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

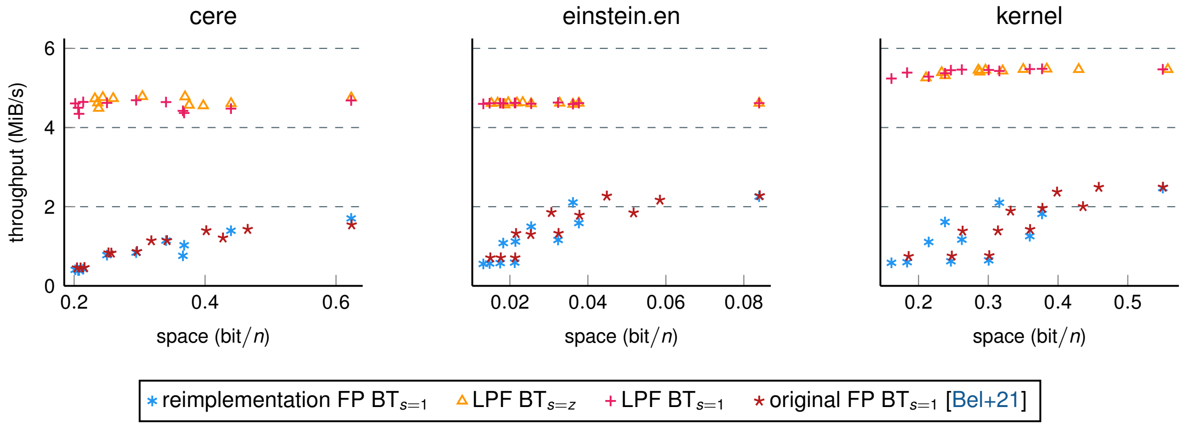
- highly tuned implementation
- tree consists only of bit and compact vectors
- tuning parameter
  - degree root $s = \{1, z\}$ (only we have $s = z$)
  - degree other nodes $\tau = \{2, 4, 8, 16\}$
  - number characters in leaves $b = \{2, 4, 8, 16\}$

- 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)

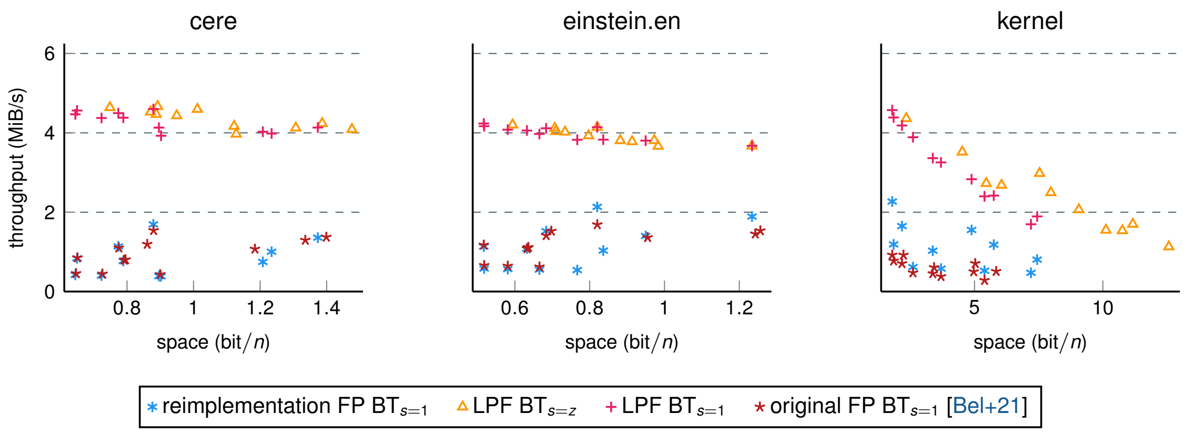
- repetitive instances from P&C corpus
- non-repetitive instances from P&C corpus

Highly Repetitive Inputs (Access Only)

Karlsruhe Institute of Technology



Highly Repetitive Inputs (with Rank and Select Support)

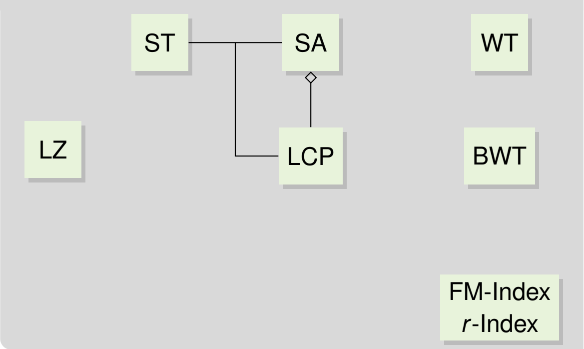


Conclusion and Outlook

This Lecture

- block trees

Linear Time Construction



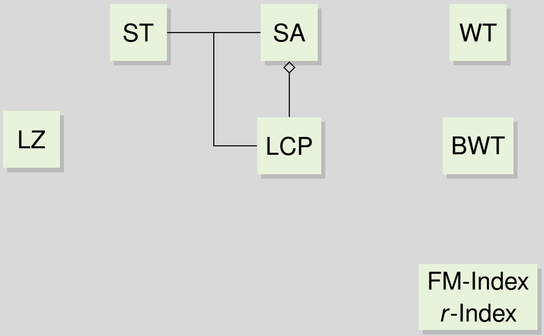
Conclusion and Outlook

This Lecture

- block trees

- efficient block tree construction
- linear time block tree construction

Linear Time Construction



Conclusion and Outlook

This Lecture

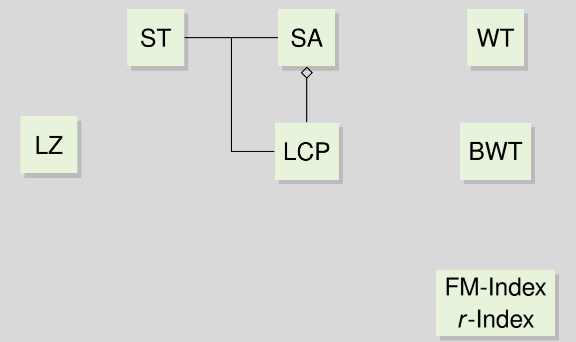
- block trees

- 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



Bibliography I

[Bel+21] Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. "Block Trees". In: *J. Comput. Syst. Sci.* 117 (2021), pages 1–22. DOI: `10.1016/j.jcss.2020.11.002`.

[BW94] Michael Burrows and David J. Wheeler. *A Block-Sorting Lossless Data Compression Algorithm*. Technical report. 1994.

[KK20] Dominik Kempa and Tomasz Kociumaka. "Resolution of the Burrows-Wheeler Transform Conjecture". In: *FOCS*. IEEE, 2020, pages 1002–1013. DOI: `10.1109/FOCS46700.2020.00097`.

[ZL77] Jacob Ziv and Abraham Lempel. "A Universal Algorithm for Sequential Data Compression". In: *IEEE Trans. Inf. Theory* 23.3 (1977), pages 337–343. DOI: `10.1109/TIT.1977.1055714`.