Exploring Regular Structures in Strings

17. July 2018

PhD Defense of Dominik Köppl

Jury Members

- Prof. Dr. Thomas Schwentick
- Prof. Dr. Johannes Fischer
- Prof. Dr. Shunsuke Inenaga
- Prof. Dr. Sven Rahmann
managing massive text data

various types:

- documents
- versioned source code
- biological data

range: gigabyte – terabyte
managing massive text data

various types:
- documents
- versioned source code
- biological data

range: gigabyte – terabyte

problems
- text indexing
- text compression
- text analysis
I. Text data structures

- Sparse suffix array

II. Factorizations

- Compute $a, b$

III. Regular structures

- Find gapped repeats, squares in influence construction of 3
I. text data structures

- sparse suffix array

compute

II. factorizations

- Lempel-Ziv

III. regular structures

- find gapped repeats squares in influence of construction of
I. text data structures

II. factorizations

III. regular structures

sparse suffix array

compute

find

Lempel-Ziv

gapped repeats

squares
I. text data structures

II. factorizations

III. regular structures

sparse suffix array

compute

find

influence construction of

Lempel-Ziv

u g u
gapped repeats

u u
squares
I. text data structures

sparse suffix array

II. factorizations

compute

III. regular structures

influence construction of

find

Lempel-Ziv

gapped repeats

squares
setting

\[ T = \square \text{ text} \]
setting

\[ T = \begin{array}{cccc}
1 & \cdots & n \\
\end{array} \text{ text} \]
setting

\[ T = \begin{array}{ccccccc}
1 & \ldots & n \\
\in \Sigma
\end{array} \text{ text} \]

\[ \sigma := |\Sigma| = n^{O(1)} \]
setting

\[ T = \begin{array}{cccccc}
1 & \cdots & n \\
\in & \Sigma \\
\end{array} \text{ text} \]

\[ \sigma := |\Sigma| = n^{O(1)} \]

- word-RAM model
- \( T \) loaded into RAM (not measured)
I. sparse suffix sorting
suffix sorting

\[ T = \]

\[ T = \]
suffix sorting

- sort all suffixes lexicographically

\[ T = \]
suffix sorting

- sort all suffixes lexicographically

\[ T = \text{sparse suffix array} \]

\[ \text{sparse LCP array} \]
suffix sorting

- sort all suffixes lexicographically
- lengths of the longest common prefix (LCP) between adjacent suffixes.
- solved in $O(n)$ time and words of space

\[ T = \] 

LCP array
suffix array
suffix sorting

- sort all suffixes lexicographically
- lengths of the longest common prefix (LCP) between adjacent suffixes.
- solved in $O(n)$ time and words of space
- sometimes need only suffixes starting at $p_1, \ldots, p_m$

$LCP = \{LCP[p_1], \ldots, LCP[p_m]\}$

sort

$LCP$ array

suffix array
suffix sorting

- sort all suffixes lexicographically
- lengths of the longest common prefix (LCP) between adjacent suffixes.
- solved in $O(n)$ time and words of space
- sometimes need only suffixes starting at $p_1, \ldots, p_m$
sparse suffix sorting

- set of $m$ text positions $p_1, \ldots, p_m$
- sort suffixes starting at these positions

given text in RAM and $m = o(n)$, we want
- $o(n)$ time
- $O(m) = o(n)$ space
sparse suffix sorting

\[ T = \]

\[ O\left(\sqrt{\log \sigma} + \log \log n \right) \times \log m \times \log n \] time

\[ O(m) \] space

if text is overwriteable

\[ o(n) \] time if \( c = o(n) \) and \( m = o(n) \)
sparse suffix sorting

$p_1, \ldots, p_m$: online, arbitrary order

$T = p_1, \ldots, p_m$

$\text{LCE query compared positions}$

$O\left( c \left( \sqrt{\log \sigma} + \log \log n \right) + n \right)$ time

$O\left( m \right)$ space

if text is overwriteable

$O\left( n \right)$ time if $c = o\left( n \right)$ and $m = o\left( n \right)$
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query

\[ T = p_1 p_2 \]

LCE query

\[ O\left( c\left( \sqrt{\log \sigma} + \log \log n \right) + m \log m \log n \log \ast n \right) \] time

\[ O\left( m \right) \] space

if text is overwriteable

\[ o\left( n \right) \] time if $c = o\left( n \right)$ and $m = o\left( n \right)$
sparse suffix sorting

- \( p_1, \ldots, p_m \): online, arbitrary order
- compare two suffixes with LCE query

\[
T = p_1 p_2
\]

LCE query compared positions \( c \)
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query

\[ T = \]

\[ \text{compared positions } c \]
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query

$LCE$ query compared positions $c$

$T = p_2 \quad p_1 \quad p_3$

Space: $O(m)$

Time: $O\left(\mathcal{O}\left(\sqrt{\log \sigma} + \log \log n\right) + m \log m \log n \log^* n\right)$ if text is overwriteable

If $c = \mathcal{O}(n)$ and $m = \mathcal{O}(n)$
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query

\[
T = p_2 p_1 p_3
\]

compared positions $c$

LCE query
sparse suffix sorting

- \(p_1, \ldots, p_m\): online, arbitrary order
- compare two suffixes with LCE query

\[ T = p_1 \cdots p_2 \cdots p_3 \cdots \]

\[ \text{compared positions } c \]

\[ \text{LCE query} \]
sparse suffix sorting

- \( p_1, \ldots, p_m \): online, arbitrary order
- compare two suffixes with LCE query
- result:
  - \( O(c(\sqrt{\lg \sigma} + \lg \lg n) + m \lg m \lg n \lg^* n) \) time
  - \( O(m) \) space
  - if text is overwriteable

\[ T = \begin{array}{c}
\text{compared positions } c
\end{array} \]
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query
- result:
  - $O(c(\sqrt{\lg \sigma} + \lg \lg n) + m \lg m \lg n \lg^* n)$ time
  - $O(m)$ space
  - if text is overwriteable
  - $o(n)$ time if $c = o(n)$ and $m = o(n)$
**sparse suffix sorting**

<table>
<thead>
<tr>
<th>time</th>
<th>words</th>
<th>authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>$n + O(1)$</td>
<td>Goto’17, Li+’16</td>
</tr>
<tr>
<td>$O(n^2/m)$</td>
<td>$O(m)$</td>
<td>Kärkkäinen+’06</td>
</tr>
<tr>
<td>$O(n)$ whp.</td>
<td>$O(m)$</td>
<td>Prezza’18</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(m)$</td>
<td>open problem</td>
</tr>
</tbody>
</table>

our result

- $O\left( c\left( \sqrt{\log \sigma} + \log \log n \right) + m \log m \log n \log^* n \right)$ time
- $O(m)$ space

Fischer, I, Köppl

Deterministic sparse suffix sorting on rewritable texts.

In Proc. LATIN, **2016**
time bound

\[ O\left( c\left( \sqrt{\lg \sigma} + \lg \lg n \right) + m \lg m \lg n \lg^* n \right) \]

LCE data structure
- built on \( c \)
- is mergeable
Candidate: ESP-tree

Cormode, Muthu. ’07
Candidate: ESP-tree

- grammar tree

Cormode, Muthu. ’07
Candidate: ESP-tree

- grammar tree
- same substrings have mostly same parsing
Candidate: ESP-tree

- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings
Candidate: ESP-tree

- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings

Cormode, Muthu. '07
Candidate: ESP-tree

- grammar tree
- same substrings have mostly same parsing
- **LCE query**: compare nodes instead of strings

Cormode, Muthu. ’07
Candidate: ESP-tree

- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings
Candidate: ESP-tree

- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings

special case: repetitions!
greedy parsing

\[ a^{3k+0} = \begin{array}{cccc}
B & \cdots & B & B \\
aaa & \cdots & aaa & aaa
\end{array} \]
greedy parsing

\[ a^{3k+0} = B \cdots B B B \]
\[ a^{3k+1} = B \cdots B B A A \]
greedy parsing

\[ a^{3k+0} = \begin{array}{cccc} B & \cdots & B & B \\ aaa & \cdots & aaa & aaa \\ \end{array} \]

\[ a^{3k+1} = \begin{array}{cccc} B & \cdots & B & A \\ aaa & \cdots & aaa & aa \\ \end{array} \]

\[ a^{3k+2} = \begin{array}{cccc} B & \cdots & B & A \\ aaa & \cdots & aaa & aa \\ \end{array} \]
### Greedy Parsing

<table>
<thead>
<tr>
<th>$a^{3k+0}$</th>
<th>[ \begin{array}{cccc} B &amp; \cdots &amp; B &amp; B \end{array} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ \begin{array}{cccc} aaa &amp; \cdots &amp; aaa &amp; aaa \end{array} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$a^{3k+1}$</th>
<th>[ \begin{array}{cccc} B &amp; \cdots &amp; B &amp; B \end{array} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ \begin{array}{cccc} aaa &amp; \cdots &amp; aaa &amp; aa \end{array} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$a^{3k+2}$</th>
<th>[ \begin{array}{cccc} B &amp; \cdots &amp; B &amp; B \end{array} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ \begin{array}{cccc} aaa &amp; \cdots &amp; aaa &amp; aa \end{array} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$a^{3k+3}$</th>
<th>[ \begin{array}{cccc} B &amp; \cdots &amp; B &amp; B \end{array} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ \begin{array}{cccc} aaa &amp; \cdots &amp; aaa &amp; aa \end{array} ]</td>
</tr>
</tbody>
</table>
$T = \begin{array}{cccccccc}
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{aa} & \text{ba} & \text{ba} \\
\end{array}$
\( T = \)

\[
\begin{array}{cccccc}
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{aa} \\
\text{ba} & \text{ba} & \\
\end{array}
\]

\[
\downarrow \text{prepend a}
\]

\( \text{a} T = \)

\[
\begin{array}{cccccccccc}
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{ba} & \text{ba} \\
\end{array}
\]
\[ T = \begin{array}{cccccccc}
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{aa} & \text{ba} & \text{ba} \\
\end{array} \]

\[ aT = \begin{array}{cccccccc}
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{ba} & \text{ba} \\
\end{array} \]

.prepend \ a
\( T = \) 

\[
\begin{array}{cccccccc}
\text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{A} & \text{A} & \text{N} & \text{N} \\
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{aa} & \text{ba} & \text{ba} \\
\end{array}
\]

\[
\text{ESP-tree}(T)
\]

\[
\begin{array}{cccccccc}
\text{F} & \text{M} & \text{D} & \text{D} & \text{C} & \text{G} \\
\end{array}
\]

\[
\text{ESP-tree}(aT)
\]

\[
\begin{array}{cccccccc}
\text{E} & \text{D} & \text{U} & \text{N} & \text{N} \\
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{ba} & \text{ba} \\
\end{array}
\]

\( aT = \) 

\[
\begin{array}{cccccccc}
\text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{A} & \text{N} & \text{N} \\
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{ba} & \text{ba} \\
\end{array}
\]

\[
\text{prepend a}
\]

12
Prepending a character changes $O(\lg n \lg^* n)$ nodes.

Cormode, Muthu’07
Prepending a character changes $\mathcal{O}(\lg n \lg^* n)$ nodes.}

\[ T = \text{aaa} \quad \text{aaa} \quad \text{aaa} \quad \text{aaa} \quad \text{aa} \quad \text{aa} \quad \text{ba} \quad \text{ba} \]

Fischer, I, Köppl

Deterministic sparse suffix sorting in the restore model.

Submitted to TALG, 2018
HSP trees

\[ T = \text{aaa} \quad \text{aaa} \quad \text{aaa} \quad \text{aaa} \quad \text{aa} \quad \text{aa} \quad \text{ba} \quad \text{ba} \]

\[ \text{ESP-tree}(T) \]

- F
- D
- B
- A
- N
- G
- M
- C

\[ T = 3 \]

\[ T = 14 \]
HSP trees

ESP-tree\( (T) \):

\[
\begin{array}{cccccc}
F & & & & & M \\
D & D & & & & G \\
B & B & B & B & A & A \\
& & & & & \\
\end{array}
\]

\[
T = \begin{array}{cccccc}
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{aa} \\
& & & & & \\
\text{ba} & \text{ba} & & & & \\
\end{array}
\]

HSP-tree\( (T) \):

\[
\begin{array}{cccccc}
& & & & & \\
a_9 & & & & a_7 & \\
a_3 & a_3 & a_3 & a_3 & a_2 & a_2 \\
& & & & N & N \\
\end{array}
\]

\[
T = \begin{array}{cccccc}
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} & \text{aa} & \text{aa} \\
& & & & & \\
\text{ba} & \text{ba} & & & & \\
\end{array}
\]
HSP trees

\[
aT = \begin{array}{c}
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aa} \\
\text{ba} \\
\text{ba}
\end{array}
\]

HSP-tree(aT)

\[
\begin{array}{c}
a_9 \\
a_3 \\
a_3 \\
a_3 \\
a_3 \\
a_3 \\
a_2 \\
N \\
N
\end{array}
\]

\[
T = \begin{array}{c}
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aa} \\
\text{aa} \\
\text{ba} \\
\text{ba}
\end{array}
\]

HSP-tree(T)

\[
\begin{array}{c}
a_9 \\
a_7 \\
a_3 \\
a_3 \\
a_3 \\
a_2 \\
a_2 \\
N \\
N
\end{array}
\]

 prepend \(a\)

\[
T = \begin{array}{c}
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aa} \\
\text{aa} \\
\text{ba} \\
\text{ba}
\end{array}
\]
HSP trees

HSP-tree(\(aT\))

\[aT = \begin{array}{cccc}
a_9 & a_8 & N_2 \\
a_3 & a_3 & a_3 & a_3 \\
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} \\
\end{array}\]

prepend a

HSP-tree(\(T\))

\[T = \begin{array}{cccc}
a_9 & a_7 & N_2 \\
a_3 & a_3 & a_3 & a_2 \\
\text{aaa} & \text{aaa} & \text{aaa} & \text{aaa} \\
\end{array}\]
HSP trees

\[ aT = \]

\[ T = \]

\[ HSP\text{-}tree(T) \]

\[ HSP\text{-}tree(aT) \]

\[ aT = \]

\[ aT = \]

\[ \text{prepend } a \]

\[ \text{prepend } a \]
HSP trees

\[ a_T = \begin{array}{c}
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{aaa} \\
\text{ba} \\
\text{ba}
\end{array} \]

node changes

ESP-tree \( \mathcal{O}(\lg^2 n \lg^* n) \) nodes \rightarrow \)

HSP-tree \( \mathcal{O}(\lg n \lg^* n) \) nodes

\[ T = a_3 a_8 a_3 a_3 a_3 a_3 a_2 N N \]
time bound

\[ \mathcal{O}(c(\sqrt{\log \sigma} + \log \log n)) + \mathcal{O}(m \log m \log n \log^* n) \]

construction on \( \mathcal{O}(c) \) characters

search tree

HSP-trees

\[ \mathcal{O}(\log n \log^* n) \] time per query
time bound

\[ \mathcal{O}(c(\sqrt{\lg \sigma} + \lg \lg n)) + \mathcal{O}(m \lg m (\lg n \lg^* n)) \]

construction on \( \mathcal{O}(c) \) characters

LCE queries

HSP-trees

- \( \mathcal{O}(\lg n \lg^* n) \) time per query
- is mergeable in \( \mathcal{O}(\lg n \lg^* n) \) time

\[ a_5 + a_4 = a_9 \]

\[ a_3 \ a_2 \ a_3 \ a_2 \ a_3 \ a_3 \ a_3 \]
time bound

\[ O\left( c\left( \sqrt{\lg \sigma} + \lg \lg n \right) \right) + \left\{ \begin{array}{l}
\text{LCE queries} \\
\text{search tree}
\end{array} \right\}
\]

Construction on \( O(c) \) characters

**HSP-trees**
- \( O(\lg n \lg^* n) \) time per query
- is mergeable in \( O(\lg n \lg^* n) \) time
- is storable in text space

![Diagram of HSP-trees](attachment:image.png)
time bound

\[ \mathcal{O}(c(\sqrt{\lg \sigma} + \lg \lg n)) \] 
construction on \( \mathcal{O}(c) \) characters

\[ + \begin{cases} 
\mathcal{O}(m \lg m) \lg n \lg^* n) 
\end{cases} \]
LCE queries

HSP-trees

- \( \mathcal{O}(\lg n \lg^* n) \) time per query
- is mergeable in \( \mathcal{O}(\lg n \lg^* n) \) time
- is storable in text space

\[ \begin{array}{c}
a_5 \\
a_3 \quad a_2 \\
\text{aaa} \quad \text{aa}
\end{array} + 
\begin{array}{c}
a_4 \\
a_2 \quad a_2 \\
\text{aa} \quad \text{aa}
\end{array} = 
\begin{array}{c}
a_9 \\
a_3 \quad a_3 \quad a_3 \\
\text{aaa} \quad \text{aaa} \quad \text{aaa}
\end{array} \]
II. Lempel-Ziv factorization

1. LZ77
2. LZ78
LZ77

\[ T = a a b a b a a a $ \]

Coding:
LZ77

\[ T = \textcolor{gray}{a} \textcolor{black}{a} \textcolor{gray}{b} \textcolor{black}{a} \textcolor{gray}{b} \textcolor{black}{a} \textcolor{gray}{a} \textcolor{black}{a} $ \]

Coding: a
LZ77

$T = \text{aabaabaa}$

Coding: $a(1,1)$
LZ77

\[ T = \text{aabababa} \$

Coding: \( a(1,1)b \)
LZ77

\[ T = aababaaba \$

Coding: \( a(1,1)b(2,3) \)
LZ77

\[ T = \textcolor{gray}{a} \textcolor{orange}{a} \textcolor{gray}{b} \textcolor{blue}{a} \textcolor{purple}{b} \textcolor{blue}{a} \textcolor{gray}{a} \$ \]

Coding: \( a(1,1)b(2,3)(1,1) \)
LZ77

$T = \text{aabababa}a$  

Coding: $a(1,1)b(2,3)(1,1)\$

(1,1) (2,3) (1,1)
LZ77

![Graph showing time and bits of space with various functions including \(n^2\), \(n \log^3 n\), \(n \log n \log \log \sigma\), \(n(\log \sigma + \log \log n)\), \(n \log \sigma \), \(n \log \log \sigma\), and others.](image-url)
LZ77

Fischer, I, Köppl, Sadakane
Lempel-Ziv factorization powered by space efficient suffix trees.
Algorithmica, 2018
suffix tree

A suffix tree is a compressed trie (a tree data structure) in which each path from root to leaf corresponds to a suffix of the input string. The root node contains the special marker symbol '1'. Each internal node in the tree represents a substring of the input string, and each edge is labeled with a symbol from the input alphabet. The tree is constructed such that each leaf node corresponds to a suffix of the input string, and the path from the root to a leaf node represents the suffix in the input string. This structure allows for efficient string operations, such as finding the longest common prefix among suffixes, or locating a substring within the input string.
suffix tree

pre-order number

suffix number

walk from leaf to root
mark visited nodes
select leaves in text order
visited nodes witnesses reference
suffix tree

- walk from leaf to root
suffix tree
- walk from leaf to root
- mark visited nodes
suffix tree
- walk from leaf to root
- mark visited nodes
suffix tree
- walk from leaf to root
- mark visited nodes
suffix tree

- walk from leaf to root
- mark visited nodes
suffix tree
- walk from leaf to root
- mark visited nodes
- select leaves in text order
suffix tree

- walk from leaf to root
- mark visited nodes
- select leaves in text order
suffix tree
- walk from leaf to root
- mark visited nodes
- select leaves in text order
suffix tree
- walk from leaf to root
- mark visited nodes
- select leaves in text order
- visited nodes witnesses reference

factor at 2 has reference 1
suffix trees

construction

- $O(n)$ time
- $O(n \lg n)$ bits working space

space-efficient $O(n)$ time constructions:

- $(1 + \varepsilon)n \lg n + O(n)$ bits with $0 < \varepsilon \leq 1$
- $O(n \lg \sigma)$ bits due to Munro+’17

Farach-Colton+’00

succinct

compressed
succinct suffix tree

<table>
<thead>
<tr>
<th>ISA</th>
<th>$n \lg n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>$\varepsilon n \lg n$</td>
</tr>
<tr>
<td>BP</td>
<td>$4n + o(n)$</td>
</tr>
<tr>
<td>LCP</td>
<td>$2n + o(n)$</td>
</tr>
<tr>
<td>RMQ</td>
<td>$2n + o(n)$</td>
</tr>
<tr>
<td>...</td>
<td>$\mathcal{O}(n)$</td>
</tr>
</tbody>
</table>

total space: $(1 + \varepsilon)n \lg n + \mathcal{O}(n)$ bits.
succinct suffix tree

<table>
<thead>
<tr>
<th>ISA</th>
<th>LZ77 factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>n \lg n</td>
</tr>
<tr>
<td>BP</td>
<td>\varepsilon n \lg n</td>
</tr>
<tr>
<td>LCP</td>
<td>4n + o(n)</td>
</tr>
<tr>
<td>RMQ</td>
<td>2n + o(n)</td>
</tr>
<tr>
<td>...</td>
<td>\mathcal{O}(n)</td>
</tr>
</tbody>
</table>

\[
\text{total space: } (1 + \varepsilon)n \lg n + \mathcal{O}(n) \text{ bits.}
\]

goal: overwrite ISA with LZ77 factors.
succinct suffix tree

<table>
<thead>
<tr>
<th>ISA</th>
<th>LZ77 factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td></td>
</tr>
<tr>
<td>BP</td>
<td></td>
</tr>
<tr>
<td>LCP</td>
<td></td>
</tr>
<tr>
<td>RMQ</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

- $n \lg n$
- $\varepsilon n \lg n$
- $4n + o(n)$
- $2n + o(n)$
- $O(n)$

$O(n)$

total space: $(1 + \varepsilon)n \lg n + O(n)$ bits.
goal: overwrite ISA with LZ77 factors.
problem: RMQ, LCP, and SA depend on ISA.
smallest leaf
$O(1)$ time

smallest leaf

$T[i..]$

$T[i + 1..]$
The diagram illustrates a computational process with the following key elements:

- **Smallest Leaf**: Node 1
- **O(1) Time**: Operation indicated by the arrow from node 2 to node 1
- **O(n) Time**: Operation indicated by the arrow from node 10 to node 13

The process begins at node 1, which is the smallest leaf. From there, the operation takes O(1) time to transition to node 2. Subsequently, node 2 triggers an O(n) time operation, leading to node 10. This process continues through the tree, with each path indicating the time complexity of the operations involved.
$\Psi O(1)$ time

$O(n)$ time
only readable characters

label?

\( T \overset{a}{\rightarrow} T[1..] \)

\( T \overset{a}{\rightarrow} T[2..] \)

\( O(n) \) time

\( \psi \)

\( O(1) \) time

smallest leaf
LZ78

$T = \text{aababa$a$}$

1 2 3 4 5 6 7 8

Coding:

0 LZ trie
LZ78

$T = \text{a a b a b a a a }$ $\$

1 2 3 4 5 6 7 8

Coding: (0, a)

LZ trie

0

1

a
LZ78

\[ T = a a a b a b a a a $ \]

Coding: \((0, a)(1, b)\)
LZ78

\[ T = a \ a \ a \ b \ a \ b \ a \ a \ a \$ \]

1 2 3 4 5 6 7 8

Coding: (0, a) (1, b) (2, a)

LZ trie

0

1

2

3

aba
LZ78

$T = a a b a b a a a \$$

Coding: (0, a) (1, b) (2, a) (1, \$)

LZ trie

Diagram of the LZ trie for the string $T = a a b a b a a a \$$. The coding steps are illustrated with arrows connecting the nodes in the trie.
two approaches

1. suffix tree based
2. LZ trie based
1. suffix tree based

<table>
<thead>
<tr>
<th>time</th>
<th>bits</th>
<th>authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(n))</td>
<td>(O(n \lg n))</td>
<td>Nakashima+’15</td>
</tr>
<tr>
<td>(O(n/\varepsilon))</td>
<td>((1 + \varepsilon)n \lg n + O(n))</td>
<td>Fischer, I, Köppl, Sadakane</td>
</tr>
<tr>
<td>(O(n))</td>
<td>(O(n \lg \sigma))</td>
<td></td>
</tr>
</tbody>
</table>

Fischer, I, Köppl, Sadakane

Lempel-Ziv factorization powered by space efficient suffix trees.

Algorithmica, 2018
- mark root
- move to highest unmarked ancestor

\[
\begin{array}{c}
\text{mark root} \\
\text{move to highest unmarked ancestor}
\end{array}
\]
- mark root
- move to highest unmarked ancestor

select leaves in text order

last marked node is parent of new leaf
- mark root
- move to highest unmarked ancestor

select leaves in text order
last marked node is parent of new leaf
- mark root
- move to highest unmarked ancestor
- select leaves in text order
- mark root
- move to highest unmarked ancestor
- select leaves in text order
- mark root
- move to highest unmarked ancestor
- select leaves in text order
- last marked node is parent of new leaf

![Diagram of a tree with nodes marked and an explanation of the process of marking and selecting leaves.]

- The process starts by marking the root of the tree.
- The next step involves moving to the highest unmarked ancestor of the node being considered.
- Then, select the leaves in text order, starting from the node marked just before.
- The last marked node before a new leaf is added becomes the parent of that new leaf.
## 2. LZ trie based

<table>
<thead>
<tr>
<th>Time</th>
<th>Bits</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}(n \lg \sigma)$</td>
<td>$\mathcal{O}(z \lg z)$</td>
<td>Lempel, Ziv’78</td>
</tr>
<tr>
<td>$\mathcal{O}(n + z \frac{\lg^2 \lg \sigma}{\lg \lg \lg \sigma})$</td>
<td>$\mathcal{O}(z \lg z)$</td>
<td>Fischer, Gawrychowski’15</td>
</tr>
<tr>
<td>$\mathcal{O}(n)$ whp.</td>
<td>$\mathcal{O}(z \lg(\sigma z))$</td>
<td></td>
</tr>
</tbody>
</table>

- $z$: \# factors

---

Fischer, Köppl


In Proc. SPIRE, **2017**
LZ trie implementations

baseline:
- binary first-child-next-sibling
- ternary

new tries:
- hash: hash table representation
- compact hash: quotienting of hash
- rolling: store Karp-Rabin fingerprints in hash table

Bentley, Sedgewick’97
hash

LZ trie

0
  a
1
    $ b
      4
        a
          3

hash table

<table>
<thead>
<tr>
<th>key</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, a)</td>
<td>1</td>
</tr>
</tbody>
</table>
hash

LZ trie

hash key: (1, b)  
value: 2

hash table

<table>
<thead>
<tr>
<th>key</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,a)</td>
<td>1</td>
</tr>
</tbody>
</table>
hash

LZ trie

hash table

<table>
<thead>
<tr>
<th>key</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,b)</td>
<td>2</td>
</tr>
<tr>
<td>(0,a)</td>
<td>1</td>
</tr>
</tbody>
</table>

hash function

hash key: (1,b)
value: 2
III. regular structures

squares

```
  u  u
```

gapped repeats

```
  u  g  u
```
distinct squares

1 2 3 4 5 6 7 8 9 10 11

a b a b a a a a b a b a
distinct squares

squares
- abab at 1
- baba at 2
- aa at 5
- aa at 6
- abab at 7
- baba at 8
distinct squares

squares
- abab at 1
- baba at 2
- aa at 5
- aa at 6
- abab at 7
- baba at 8
distinct squares

leftmost squares

- abab at 1
- baba at 2
- aa at 5
- aa at 6
- abab at 7
- baba at 8
finding distinct squares

<table>
<thead>
<tr>
<th>algorithms</th>
<th>bits</th>
<th>authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{O}(n) )</td>
<td>( \mathcal{O}(n \lg n) )</td>
<td>Crochemore+’14</td>
</tr>
<tr>
<td>( \mathcal{O}(n \lg^\varepsilon n) )</td>
<td>( \mathcal{O}(n \lg \sigma) )</td>
<td></td>
</tr>
<tr>
<td>( \mathcal{O}\left(\frac{n}{\varepsilon}\right) )</td>
<td>( (2 + \varepsilon)n \lg n + \mathcal{O}(n) )</td>
<td></td>
</tr>
<tr>
<td>online: ( \mathcal{O}\left(\frac{n \lg^2 \lg n}{\lg \lg \lg n}\right) )</td>
<td>( \mathcal{O}(n \lg n) )</td>
<td></td>
</tr>
</tbody>
</table>

\( 0 < \varepsilon \leq 1 \)

Bannai, Inenaga, Köppl
Computing all distinct squares in linear time for integer alphabets.
In Proc. CPM, 2017
gapped repeats

1 2 3 4 5 6 7 8 9 10 11 12 13 14

momomosumomomo
gapped repeats

\[ i_L \quad u \quad i_R \]

\[ m \quad o \quad m \quad o \quad s \quad u \quad m \quad o \quad m \quad o \]

gapped repeat \((i_L, i_R, u)\)
gapped repeats

- gapped repeat \((i_L, i_R, u)\)
- *maximal* if it cannot be extended to the left nor to the right.
gapped repeats

- gapped repeat \((i_L, i_R, u)\)
- maximal if it cannot be extended
  - to the left nor
gapped repeats

- gapped repeat \((i_L, i_R, u)\)
- \textit{maximal} if it cannot be extended
  - to the left nor
  - to the right.
gapped repeats

- gapped repeat \((i_L, i_R, u)\)
- \textit{maximal} if it cannot be extended
  - to the left nor
  - to the right.
"α-gapped repeats"

maximal gapped repeat \((i_L, i_R, u)\)
\(\alpha\)-gapped repeats

- maximal gapped repeat \((i_L, i_R, u)\)
- \(g := \text{gap}\)
α-gapped repeats

- maximal gapped repeat \((i_L, i_R, u)\)
- \(g := \text{gap}\)
gapped palindromes

\[ (i_L, i_R, u) \text{ gapped palindrome} \]
gapped palindromes

\((i_L, i_R, u)\) gapped palindrome

- is \textit{maximal} if it cannot be extended
  - inwards nor
gapped palindromes

(i_L, i_R, u) gapped palindrome

is *maximal* if it cannot be extended

- inwards nor
gapped palindromes

\[(i_L, i_R, u)\] gapped palindrome

- is \textit{maximal} if it cannot be extended
  - inwards nor
  - outwards.
gapped palindromes

\[(i_L, i_R, u)\] gapped palindrome

is *maximal* if it cannot be extended
- inwards nor
- outwards.
�apped palindromes

\[(i_L, i_R, u)\] gapped palindrome

is maximal if it cannot be extended

- inwards nor
- outwards.

\[(i_L, i_R, u)\] is \(\alpha\)-gapped if \(g + u \leq \alpha u\)
Definition

\[
\begin{align*}
\text{occ}_R & := \# \text{ occurrences of max. } \alpha\text{-gapped}
\end{align*}
\]

\[
\begin{align*}
\text{occ}_P & \quad \text{repeats} \\
\text{occ}_\alpha & \quad \text{palindromes}
\end{align*}
\]

Problem

\[
\text{occ}_R, \text{occ}_P \leq ?
\]
occ$_R$- repeats

\[ O(\alpha^2 n) \quad \text{Kolpakov+’14} \]
\[ O(\alpha n) \quad \text{Crochemore+’15} \]
\[ \leq 18\alpha n \quad ^1 \]
\[ \leq 13\alpha n \quad ^2 \]

^1 Gawrychowski, I, Inenaga, Köppl, Manea
Tighter bounds and optimal algorithms for all maximal $\alpha$-gapped repeats and palindromes.
TOCS, 2018

^2 I, Köppl
Improved upper bounds on all maximal $\alpha$-gapped repeats and palindromes.
Accepted at TCS, 2018
occ- palindromes

\[ \leq 28\alpha n + 7n \quad 1 \]
\[ \leq 16\alpha n - 3n \quad 2 \]

1. Gawrychowski, I, Inenaga, Köppl, Manea
   Tighter bounds and optimal algorithms for all maximal \(\alpha\)-gapped repeats and palindromes.
   TOCS, 2018

2. I, Köppl
   Improved upper bounds on all maximal \(\alpha\)-gapped repeats and palindromes.
   Accepted at TCS, 2018
finding all maximal $\alpha$-gapped repeats

previous results: Tanimura+’15, Crochemore+’15

$O(\alpha n)$ time for constant $\sigma$. 
finding all maximal $\alpha$-gapped repeats

previous results: Tanimura+’15, Crochemore+’15

$O(\alpha n)$ time for constant $\sigma$.

*: same time with $\sigma = n^{O(1)}$

Gawrychowski, I, Inenaga, Köppl, Manea
Tighter bounds and optimal algorithms for all maximal $\alpha$-gapped repeats and palindromes.
TOCS, 2018
the big picture

- suffix tree
- LZ77
- distinct squares
- Runs
- max. $\alpha$-gapped palindromes
- maximal $\alpha$-gapped repeats
- LZ78
- sparse suffix sorting
sparse suffix sorting of \( m \) suffixes
- \( o(n) \) deterministic time if \( c = o(n) \) and \( m = o(n) \)
- \( O(m) \) space

LZ 77/78 in \( O(n) \) time with
- \( (1 + \varepsilon)n \lg n + O(n) \) bits
- \( O(n \lg \sigma) \) bits

LZ78 in \( O(n) \) time whp. and \( O(z \lg(z\sigma)) \) bits + practical!

finding all distinct squares
- \( O(n) \) time and \((2 + \varepsilon)n \lg n \) bits
- online near-linear time

all maximal \( \alpha \)-gapped repeats / palindromes
- find in \( O(\alpha n) \) time
- \# = \( O(\alpha n) \)

thank you for your attention!
summary

sparse suffix sorting of \( m \) suffixes
- \( o(n) \) deterministic time if \( c = o(n) \) and \( m = o(n) \)
- \( \mathcal{O}(m) \) space

LZ 77/78 in \( \mathcal{O}(n) \) time with
- \( (1 + \varepsilon)n \lg n + \mathcal{O}(n) \) bits
- \( \mathcal{O}(n \lg \sigma) \) bits

LZ78 in \( \mathcal{O}(n) \) time whp. and \( \mathcal{O}(z \lg(z\sigma)) \) bits + practical!

finding all distinct squares
- \( \mathcal{O}(n) \) time and \( (2 + \varepsilon)n \lg n \) bits
- online near-linear time

all maximal \( \alpha \)-gapped repeats / palindromes
- find in \( \mathcal{O}(\alpha n) \) time
- \# = \( \mathcal{O}(\alpha n) \)

string \( T \in \Sigma^* \), \( n := |T| \), \( \sigma := |\Sigma| = n^{O(1)} \), \( 0 < \varepsilon \leq 1 \), \( z : \# \) factors