Computing All Distinct Squares in Linear Time for Integer Alphabets

Hideo Bannai ¹   Shunsuke Inenaga ¹   Dominik Köppl ²

¹Department of Informatics, Kyushu University, Japan
²Department of Computer Science, TU Dortmund, Germany

CPM’17

*with python a one-liner: [i**2 for i in range(1,n)]*
squares

<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>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>
squares

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

fact: leftmost squares $\equiv$ distinct squares
squares

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

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

fact
leftmost squares ≡ distinct squares
works on distinct squares

#distinct squares
\[ \leq 2n \quad \text{Fraenkel and Simpson’98} \]
\[ \leq 2n - \Theta(\lg n) \quad \text{Ilie’07} \]
\[ \leq \lfloor 11n/6 \rfloor \quad \text{Deza et al.’15} \]
\[ \leq n \quad \text{(yet) unknown} \]

algorithms computing all distinct squares
\[ \mathcal{O}(\sigma n) \text{ time} \quad \text{Gusfield and Stoye’04} \]
\[ \mathcal{O}(n) \text{ time} \quad \text{Crochemore’14} \]
\[ \mathcal{O}(n) \text{ time} \quad \text{this paper} \]

where
- \( \sigma \): alphabet size
- \( n \): text length
setting

given
- text $T$
- $n := |T|$ text length
- alphabet of size $\sigma = n^{O(1)}$

problem
find all distinct squares

goal: $O(n)$ time
naive solution

- create suffix array and LCP array
- iterate over each text position $i$
- iterate over all possible periods $p$
- compare $T[i..]$ with $T[i + p..]$ $\forall 1 \leq i, p \leq n$
- if found a square $\Rightarrow$ check whether already reported

$$\Rightarrow \mathcal{O}\left(\sum_{i} n \cdot \sum_{p} n \cdot t_{\lambda}\right)$$

- $t_{\lambda}$: time for look-up ($t_{\lambda} = n \log \sigma$ for a simple trie)
better solutions

idea to get faster
  - check only at certain text position
  - check only periods up to a threshold

sufficient: all borders of Lempel-Ziv factors

idea from

[Gusfield and Stoye’04]
computing all distinct squares in $\mathcal{O}(\sigma n)$ time
Lempel-Ziv parsing
Lempel-Ziv parsing

```
: a b a b a a a b a b a
```

F1 F2 F3 F4 F5
Lempel-Ziv parsing

\[ \text{a b a b a a a a b a a b a b a} \]
Lempel-Ziv parsing

ab\quad ab\quad ba\quad a\quad a\quad a\quad b\quad a\quad b\quad a\quad a\quad b\quad a\quad b\quad a\quad a
Lempel-Ziv parsing
Lempel-Ziv parsing

```
  a b a b a
```

```
  a a b a a
```

```
  b a b a b a
```
Lempel-Ziv parsing

$F_1 F_2 F_3 F_4 F_5$

a b a b a a a b a b a a a a b a b a b a b a
computing squares

\[ a \ b \ a \ b \ a \ a \ a \ a \ b \ a \ b \ a \ b \ a \ a \ a \ b \ a \ b \ a \ b \ a \]

reported squares:

\[ \bullet \ a b a b a \]
\[ \bullet \ a a \]
\[ \bullet \ a b a b a \]

problems:

\[ \bullet \ reporting \ duplicates \]
\[ \bullet \ baba \ not \ found \]
computing squares

\[
\begin{array}{cccccc}
F_1 & F_2 & F_3 & F_4 & F_5 \\
ab & ba & ba & aa & ab \\
ba & ab & ba & ba & ba \\
\end{array}
\]

reported squares:

- ♦abab
- ♦aa
- ♦abab

problems:

- ♦reporting duplicates
- ♦baba not found
computing squares

\[
\begin{array}{cccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} \\
\text{b}(F_2) & p & q
\end{array}
\]

reported squares:

\begin{itemize}
  \item ♦abab
  \item ♦aa
  \item ♦abab
\end{itemize}

problems:

\begin{itemize}
  \item ♦reporting duplicates
  \item ♦baba not found
\end{itemize}
computing squares

\[ a | b | a | b | a | a | a | b | a | b | a | b | a \]

reported squares:

\[ ♦abab \]
\[ ♦aa \]
\[ ♦abab \]

problems:

\[ ♦reporting duplicates \]
\[ ♦baba not found \]
computing squares

reported squares:

- abab
computing squares

reported squares:
- abab
- abab
- abab

problems:
- reporting duplicates
- baba not found
computing squares

reported squares:

- abab

problems:

- reporting duplicates
- baba not found
computing squares

reported squares:
- abab
- aa
computing squares

```
abab
```

reported squares:
- abab
- aa

```
abab
```

reported squares:
- abab
- aa

problems:
- reporting duplicates
- baba not found
computing squares

reported squares:
- abab
- aa
computing squares

reported squares:
- abab
- aa
- abab

problems:
- reporting duplicates
- baba not found
computing squares

| a | b | a | b | a | a | a | b | a | b | a | b | a |

reported squares:
- abab
- aa
- abab

problems:
- reporting duplicates
- baba not found
suffix link walk: construct suffix tree
suffix link walk: decorate squares
suffix link walk: rotate
time bounds

- suffix link walk traversals $O(n)$ nodes
- number of suffix links to a particular node $\leq \sigma$
- $\Rightarrow$ a long traversal can happen $O(\sigma)$ times

hence $O(n\sigma)$ time [Gusfield and Stoye’04]
$O(n)$ time goal

problem

- ∄ dictionary with $O(1)$ access/update time
- store lists, be careful about uniqueness!

solution

use longest previous factor table (LPF)!
longest previous factor table

a b a b a a a a b a b a

a b a b a a a a b a b a
longest previous factor table
longest previous factor table

```
a b a b a a a a a b a b a
```
longest previous factor table

\[
\begin{array}{cccccccccccc}
0 & 0 & 3 & & & & 2 & & & & & \\
\end{array}
\]

abababa a a b a b a b a
longest previous factor table

```
 0 0 3 2 1 2 5 4 3 2 1
a b a b a a a b a b a
```
longest previous factor table

0 0 3 2 1 2 5 4 3 2 1

a b a b a a a b a b a
longest previous factor table

```
0 0 3 2 1 2 5 4 3 2 1
```

```
  a b a b b a a a b a b a
```
longest previous factor table
longest previous factor table
longest previous factor table
longest previous factor table

0 0 3 2 1 2 5 4 3 2 1

a b a b a a a b a b a

0 0 3 2 1 2 5 4 3 2

a b a b a a a b a b a

0 0 3 2 1 2 5 4 3 2

a b a b a a a b a b a

0 0 3 2 1 2 5 4 3 2

a b a b a a a b a b a

0 0 3 2 1 2 5 4 3 2

a b a b a a a b a b a

0 0 3 2 1 2 5 4 3 2

a b a b a a a b a b a

0 0 3 2 1 2 5 4 3 2

a b a b a a a b a b a

0 0 3 2 1 2 5 4 3 2

a b a b a a a b a b a
longest previous factor table

0 0 3 2 1 2 5 4 3 2 1

a b a b a a a b a b a
computing squares

0 0 3 2 1 2 5 4 3 2 1
a b a b a a a a b a b a

reported squares:

♦abab
♦baba
♦aa

new techniques:
♦right rotate found squares
♦skip if LPF[i] ≥ 2p
computing squares

<table>
<thead>
<tr>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>$F_4$</th>
<th>$F_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

a b a b a a a b a b a

reported squares:

- $\text{abab}$
- $\text{baba}$
- $\text{aa}$

new techniques:
- Right rotate found squares
- Skip if LPF[$i$] $\geq 2p$
computing squares

reported squares:

♦abab
♦baba
♦aa

new techniques:
♦right rotate found squares
♦skip if LPF\[i\] ≥ 2p
computing squares

reported squares:

♦abab
♦baba
♦aa

new techniques:
♦right rotate found squares
♦skip if \(\text{LPF}[i] \geq 2p\)
computing squares

reported squares:

- abab
computing squares

reported squares:
- abab
- baba

new techniques:
- right rotate found squares
- skip if LPF\[i\] ≥ 2p
computing squares

reported squares:
- abab
- baba

new techniques:
- right rotate found squares
computing squares

reported squares:
- abab
- baba

new techniques:
- right rotate found squares
computing squares

reported squares:
- abab
- baba
- aa

new techniques:
- right rotate found squares
- skip if $\text{LPF}[i] \geq 2p$
computing squares

reported squares:
- abab
- baba
- aa

new techniques:
- right rotate found squares
- skip if LPF\[i\] ≥ 2p
computing squares

reported squares:

- abab
- baba
- aa

new techniques:

- right rotate found squares

\[
\begin{array}{|c|c|c|c|c|}
\hline
0 & 0 & 3 & 2 & 1 \\
\hline
0 & 1 & 2 & 5 & 4 \\
\hline
a & b & a & b & a \\
\hline
\end{array}
\]
computing squares

reported squares:
- abab
- baba
- aa

new techniques:
- right rotate found squares
- skip if LPF[i] ≥ 2p
RMQs on LPF

naive right rotation of square $S$ takes $\mathcal{O}(|S|)$ time
$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

right rotations by RMQ on LPF
RMQs on LPF

naive right rotation of square $S$ takes $\mathcal{O}(|S|)$ time

$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

right rotations by RMQ on LPF

```latex
\begin{array}{l}
\text{LPF} \\
\begin{array}{cccccccc}
a & b & a & a & a & a & b & a & a & a & a & a \\
p & & & & & & & & & & & 2p \\
\end{array}
\end{array}
```
RMQs on LPF

naive right rotation of square $S$ takes $\mathcal{O}(|S|)$ time
$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

right rotations by RMQ on LPF
RMQs on LPF

naive right rotation of square $S$ takes $\mathcal{O}(|S|)$ time

$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

right rotations by RMQ on LPF
RMQs on LPF

naive right rotation of square $S$ takes $O(|S|)$ time
$\Rightarrow O(n^2)$ total time!

our approach

right rotations by RMQ on LPF

$\#RMQs \leq 3 \times$ number of newly found squares
time analysis

known:
- number of distinct squares \( \text{occ} = \mathcal{O}(n) \)

algorithm:
- build data structures (suffix array, etc.) \( \mathcal{O}(n) \)
- query on \( \mathcal{O}(n) \) positions:
  - for a square with LCP \( \mathcal{O}(1) \)
  - find right rotated squares with RMQ \( \mathcal{O}(\text{occ}) \) total

\( \mathcal{O}(n + \text{occ}) = \mathcal{O}(n) \) total time
### Experiments

| collection  | $\sigma$ | $z$ | $\max_x |F_x|$ | $\text{occ}$ | time |
|------------|---------|-----|----------------|-------------|------|
| dblp.xml   | 97      | 7,035,342 | 1060 | 7412 | 70   |
| proteins   | 26      | 20,875,097 | 45,703 | 3,108,339 | 245  |
| dna        | 17      | 13,970,040 | 97,966 | 132,594 | 310  |
| english    | 226     | 13,971,134 | 987,766 | 13,408 | 2639 |
| einstein   | 125     | 49,575    | 906,995 | 18,192,737 | 3953 |

- 200 MiB collections from Pizza&Chili corpus
- $\sigma$: alphabet size
- $z$: $\#$ Lempel-Ziv factors
- time: time in seconds

**time bottleneck: RMQs**
finding all distinct squares in linear time

**techniques**
- modification of [Gusfield and Stoye’04]
- using LPF array

**open problems**
- cope without RMQ
- create MAST in $\mathcal{O}(n)$ time

**further results**
- computing online in $\mathcal{O}\left(n \lg^2 \lg n / \lg \lg \lg n\right)$ time
- in linear time:
  - decorating suffix tree with information of all squares
  - building topology of the minimal augmented suffix tree (MAST)

Thank you for listening. Any questions are welcome!
finding all distinct squares in linear time

techniques

- modification of [Gusfield and Stoye’04]
- using LPF array

open problems

- cope without RMQ
- create MAST in $O(n)$ time

further results

- computing online in $O\left(n \lg^2 \lg n / \lg \lg \lg n\right)$ time
- in linear time:
  - decorating suffix tree with information of all squares
  - building topology of the minimal augmented suffix tree (MAST)

Thank you for listening. Any questions are welcome!