Reducing Squares in Suffix Arrays

Peter Leupold
University of Leipzig

Prague Stringology Conference 2014
What I Really Want

The duplication history for a string $aabcbabcbbc$. The direction of reductions is top to bottom:
Why I Want This

- For theoretical reasons (number of normal forms):

\[ \text{duproots}(n) := \max\{|R| : R \text{ set of all normal forms of a string of length } n\} \]
Why I Want This

- For theoretical reasons (number of normal forms):

  \[ \text{duproots}(n) := \max\{|R| : R \text{ set of all normal forms of a string of length } n\} \]

- Could be useful for compression (Ilie et al.)
Why I Want This

• For theoretical reasons (number of normal forms):

\[ \text{duproots}(n) := \max\{|R| : R \text{ set of all normal forms of a string of length } n\} \]

• Could be useful for compression (Ilie et al.)

• Use duplication for phylogenetic trees

• ...
Possible duplication histories interesting for biological investigations.

In the article


the way in which 17 populations of fungi had evolved was induced from looking at the duplication histories of their genomes.
Phylogenetic Trees
Phylogenetic Trees
Phylogenetic Trees
Phylogenetic Trees

aba

abba

? ?

aba

abbba

ababa

aba

ababa
The Big Problem

Theorem (PSC 2009)

For every positive integer \( \ell \) there are words of length \( \ell \) over a four-letter alphabet whose number \( N \) of normal forms under eliminating squares is bounded by:

\[
\frac{1}{30} 110^{\frac{\ell}{42}} \leq N \leq 2^\ell.
\]

No efficient algorithm for all cases.
Different square reductions in periodic factor:

\[
\begin{array}{ccc}
\text{abcabcba} & \text{abcabcba} & \text{abcabcba} \\
\downarrow & \downarrow & \leftarrow \\
\text{abcba} & \\
\end{array}
\]

**Lemma**

Let \( w \) be a string with period \( k \). Then any deletion of a factor of length \( k \) will lead to the same result.
Naive Computation of all Strings Reachable from $w$ by Reduction of Squares

**Input:** string: $w$;

**Data:** stringlist: $S$ (contains $w$);

1. while $(S$ nonempty) do
2.     $x := \text{POP}(S)$;
3.     Construct the suffix array of $x$;
4.     if (there are runs in $x$) then
5.         foreach run $r$ do
6.             Reduce one square in $r$;
7.             Add new string to $S$;
8.         end
9.     end
10. else output $x$;
11. end
Naive Computation of all Strings Reachable from $w$ by Reduction of Squares

**Input:** string: $w$;

**Data:** stringlist: $S$ (contains $w$);

1. **while** ($S$ nonempty) do
2.     $x := \text{POP}(S)$;
3.     Construct the suffix array of $x$;
4.     **if** (there are runs in $x$) **then**
5.         **foreach** run $r$ **do**
6.             Reduce one square in $r$;
7.             Add new string to $S$;
8.         **end**
9.     **end**
10.    **else** output $x$;
11. **end**
Modification of the suffix array by deletion of $bcb$ in $abcbbcba$

<table>
<thead>
<tr>
<th>SA</th>
<th>LCP</th>
<th></th>
<th>SA</th>
<th>LCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>a</td>
<td>7−3=4</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>abcbbcba</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>ba</td>
<td>6−3=3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>bbcba</td>
<td>⇒</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>bcba</td>
<td>5−3=2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>bcbbcba</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>cba</td>
<td>4−3=1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>cbbcba</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Modification of the suffix array by deletion of *bcb* in *abcbbcba*

<table>
<thead>
<tr>
<th>SA</th>
<th>LCP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>abcbbcba</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>ba</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>bbcba</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>bcba</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>bcbbcba</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>cba</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>cbbcbba</td>
</tr>
</tbody>
</table>

\[ 7 - 3 = 4 \quad 1 \quad \text{a} \]
\[ 6 - 3 = 3 \quad 1 \quad \text{ba} \]
\[ 5 - 3 = 2 \quad 0 \quad \text{bcba} \]
\[ 4 - 3 = 1 \quad \text{cba} \]

We also need: ISA

---

P. Leupold  Reducing Squares in Suffix Arrays (12)
The First Small Problem

New suffixes

P. Leupold  Reducing Squares in Suffix Arrays (13)
The First Small Problem

New suffixes

Not so new?
The First Small Problem

New suffixes

Not so new?

Deletions treated by:

Suffixes right of the deletion:

Order and LCP remain the same.

Deleting the left half of the square.
LCP is not greater than $\ell + n$:

Order unchanged, because $\ell + n$ first letters remain the same.
LCP greater than $\ell + n$:

Some letter within the prefix of length LCP might change
Computing the new suffix array

Lemma (Condition for Change in Position)
Let the LCP of two strings $z$ and $uvw$ be $k$ and let $z < uvw$. Then $z$ and $uvvw$ have the same LCP and $z < uvvw$ unless $\text{LCP}(z, uvw) \geq |uv|$; in the latter case also $\text{LCP}(z, uvvw) \geq |uv|$.

Lemma (No further changes to the left)
Let $\text{LCP}[\text{ISA}[j]] = k$ in the suffix array of a string $w$ of length $n + 1$. Then for $i < j$ we always have $\text{LCP}[\text{ISA}[i]] \leq k + j - i$. 
Computing the New Suffix Array

**Input:** string: w; arrays: SA, LCP; length and pos of square: n, k;

```
1 for j = n + k to |w| − 1 do
2     SAnew[j] := SA[j] - n;
3 end
4 i := k − 1;
5 while (LCP[i] > n + k − i AND i ≥ 0) do
6     compute SAnew of w[i . . . k − 1]w[k + n . . . |w| − 1];
7     compute new LCP[i];
8     /* with methods of Salson et al. */
9     i := i − 1;
10 end
11 for j = 0 to i + m do
12     SAnew[j] := SA[j];
13 end
```
Other Small Problems

- Efficient decision whether string has exponentially many ancestors
- Different examples from \texttt{abcabcabc}c with several normal forms
- Strategy for traversing the duplication history graph
- Store only changes instead of new suffix array
- Is a different method for run detection better?
- Are there strings over three letters with exponentially many normal forms?
Why I keep thinking about the LCP...