Efficient algorithms for two extensions of LPF table: the power of suffix arrays

M. Crochemore  C. S. Iliopoulos
M. Kubica       W. Rytter
T. Waleń

SOFSEM 2010
Preliminaries

Input: a string $y[0..n-1]$.

Auxiliary algorithms:
- the suffix array (SUF),
- the longest common prefix array (LCP),
- range minimum/maximum query (RMQ) for SUF and LCP.

Can be done in $O(n)$ time.
We consider two variants of the classical problem:

The Longest Previous Factor Problem (LPF)

\[ \text{LPF}[i] = \text{the largest such } k, \text{ that } y[i \ldots i + k] \text{ appears before (possibly overlapping)}. \]
Introduction

We consider two variants of the classical problem:

**The Longest Previous Factor Problem (LPF)**

\[ \text{LPF}[i] = \text{the largest such } k, \text{ that } y[i \ldots i + k] \text{ appears before (possibly overlapping)}. \]

- Well studied.
- Can be computed in \( O(n) \) time.
The Longest Previous Reversed Factor Problem (LPrF)

$LPrF[i] =$ the largest such $k$, that $\text{rev}(y[i .. i + k])$ appears before (without overlapping).
The Longest Previous Reversed Factor Problem (LPrF)

$LPrF[i] =$ the largest such $k$, that $\text{rev}(y[i \ldots i + k])$ appears before (without overlapping).

- Generalises a factorization of strings used to extract certain types of palindromes [Kolpakov, Kucherov, 2008].
- Applications in compression of genetic sequences (in combination with LPF) [Grumbach, Tahi, 1993].
The Longest Previous Non-Overlapping Factor Problem (LPnF)

$LPnF[i] =$ the largest such $k$, that $y[i \ldots i + k]$ appears before (without overlapping).

Emerged from a version of Ziv-Lempel factorization. Decomposition of a string into already processed factors. Application in algorithms computing repetitions in strings [Crochemore, 1986], [Main, 1989], [Kolpakov, Kucherov, 1999].
Introduction

The Longest Previous Non-Overlapping Factor Problem (LPnF)

\[ LPnF[i] = \text{the largest such } k, \text{ that } y[i \ldots i + k] \text{ appears before (without overlapping)}. \]

- Emerged from a version of Ziv-Lempel factorization.
- Decomposition of a string into already processed factors.
- Application in algorithms computing repetitions in strings [Crochemore, 1986], [Main, 1989], [Kolpakov, Kucherov, 1999].
Efficient algorithms for two extensions of LPF table
The Alternating Search Technique

Assumptions

We assume, that the following operations are given, and take $O(1)$ time:

- $Val(k)$ — non-increasing (for $i \leq k \leq j$),
- $Candidate(k)$ — a predicate,
- $FirstMin(i, j)$ — first position $k \in [i \ldots j]$ with the minimum value of $Val(k)$,
- $NextCand(i, j)$ — any candidate $k \in [i \ldots j)$.
Goal

For a given range \([i \ldots j]\), find a *candidate* \(k\) maximizing \(Val(k)\).
The Alternating Search Technique

**Goal**

For a given range $[i..j]$, find a candidate $k$ maximizing $Val(k)$.

**Alternating-Search($i, j$)**

Running time: $O(Val(k_{opt}) - Val(j) + 1)$
Computation of the LPrF table

- Calculate SUF and LCP for \( x = y \# \text{rev}(y) \).
- \( \text{LPrF}[i] = \max \{ \text{RMQ}(\text{LCP}[i \ldots j]) : j > 2n - i \} \)
Computation of the LPrF table

- Calculate SUF and LCP for \( x = y \# \text{rev}(y) \).
- \( \text{LPrF}[i] = \max\{\text{RMQ}(\text{LCP}[i..j]) : j > 2n - i\} \)

Example
Computation of the LPrF table

- \( \text{LPrF}[i + 1] \geq \text{LPrF}[i] - 1 \)

An instance of the alternating search (using: SUF and LCP for \( x \), and RMQ).

\( \mathcal{O}(n) \) running time.
Computation of the LPrF table

- \( \text{LPrF}[i + 1] \geq \text{LPrF}[i] - 1 \)

- An instance of the alternating search (using: SUF and LCP for \( x \), and RMQ).
- \( O(n) \) running time.
Computation of the LPnF table

\[ LPnF[i + 1] \geq LPnF[i] - 1 \]

Boundary case (squares) — using runs (Kolpakov, Kucherov, 1999).

General case — the alternating search (using: SUF and LCP for \( y \), and RMQ).

\( O(n) \) running time.

M. Crochemore, C.S. Iliopoulos, M. Kubica, W. Rytter, T. Waleń
Efficient algorithms for two extensions of LPF table
Computation of the LPnF table

- \( \text{LPnF}[i + 1] \geq \text{LPnF}[i] - 1 \)

\[
\begin{align*}
\text{LPnF}[i] & \quad \text{LPnF}[i + 1] \\
b \ a \ a \ b \ a & \quad b \ a \ a \ b \ a
\end{align*}
\]

- Boundary case (squares) — using runs [Kolpakov, Kucherov, 1999].
- General case — the alternating search (using: SUF and LCP for \( y \), and RMQ).
- \( O(n) \) running time.
Summary

Our results

- The LPrF and LPnF tables can be computed in $O(n)$ time.
- The optimal parsing of a text, using factors and/or reverse factors can be computed in $O(n)$ time.
Thank you for your attention!