Crochemore's String Matching Algorithm: Simplification and Extensions

Juha Kärkkäinen   Dominik Kempa   Simon J. Puglisi

University of Helsinki, Finland

Prague Stringology Conference 2013

Outline

1. Introduction
   - Problem
   - Existing solutions
   - Our contribution
2. Simplified Crochemore's algorithm
   - Original vs. simplified
   - Basic tools
   - Simplified algorithm description
3. Generalizations
   - Motivation
   - Longest prefix matching
   - Sparse longest prefix matching

Problem

Exact String Matching

Given strings $\mathrm{T}[1..n]$ and $\mathrm{P}[1..m]$ find all positions i such that $\mathrm{T}[i..i+m-1] = \mathrm{P}[1..m]$.

Problem

Exact String Matching

Given strings $\mathrm{T}[1..n]$ and $\mathrm{P}[1..m]$ find all positions i such that $\mathrm{T}[i..i+m-1] = \mathrm{P}[1..m]$.

- Algorithms available for many models and variants, e.g., compressed, higher-dimentions, "jumbled", multiple patterns

Problem

Exact String Matching

Given strings $\mathrm{T}[1..n]$ and $\mathrm{P}[1..m]$ find all positions i such that $\mathrm{T}[i..i+m-1] = \mathrm{P}[1..m]$.

- Algorithms available for many models and variants, e.g., compressed, higher-dimentions, "jumbled", multiple patterns
- `http://www.dmi.unict.it/~faro/smart/` lists over 80 algorithms **for the basic variant**. Many of them are time-optimal: $\mathcal{O}(n+m)$,

Problem

Exact String Matching

Given strings $T[1..n]$ and $P[1..m]$ find all positions i such that $T[i..i+m-1] = P[1..m]$.

- Algorithms available for many models and variants, e.g., compressed, higher-dimentions, "jumbled", multiple patterns
- `http://www.dmi.unict.it/~faro/smart/` lists over 80 algorithms **for the basic variant**. Many of them are time-optimal: $\mathcal{O}(n+m)$,
- but only few are also space-optimal, that is, use $\mathcal{O}(1)$ machine words in addition to the input strings

Time-space optimal string matching

Algorithm	Prepr.	Key idea
Rytter [2003]	yes	lex-maximal suffix
Gasieniec et al. [1995]	yes	zooming
Crochemore [1992]	no	lex-maximal suffix
Crochemore and Perrin [1991]	yes	critical factorization
Galil, Seiferas [1981]	yes	perfect factorization

Our contribution

1. Simplified version of Crochemore's algorithm

Our contribution

1. Simplified version of Crochemore's algorithm
    - much simpler shifting rule

Our contribution

1. Simplified version of Crochemore's algorithm
   - much simpler shifting rule
   - simpler analysis and extensions

Our contribution

1. Simplified version of Crochemore's algorithm
   - much simpler shifting rule
   - simpler analysis and extensions
   - still time-space-optimal

Our contribution

1. Simplified version of Crochemore's algorithm
   - much simpler shifting rule
   - simpler analysis and extensions
   - still time-space-optimal
2. Generalizations

Our contribution

1. Simplified version of Crochemore's algorithm
    - much simpler shifting rule
    - simpler analysis and extensions
    - still time-space-optimal
2. Generalizations
    - *pattern matching* → *longest prefix matching*

Our contribution

1. Simplified version of Crochemore's algorithm
   - much simpler shifting rule
   - simpler analysis and extensions
   - still time-space-optimal
2. Generalizations
   - *pattern matching* $\rightarrow$ *longest prefix matching*
   - *longest prefix matching* $\rightarrow$ *sparse longest prefix matching*

Outline

Original vs. simplified

Original

```
Algorithm Crochemore
z ← 0; c ← 1
(i, j, k, p) ← (0, 1, 1, 1)
while i ≤ n do
  while z + c ≤ n and c ≤ m and T[z + c] = P[m] do
    c ← c + 1
  if z + c = n + 1 or c = m + 1 then
    report(z)
  if z + c = n + 1 then
    c ← c − 1
  (i, j, k, p) ← next(P, T[z + c], i, j, k, p)
  if P[1..i] = suf(pref(P[i + 1..c − 1], p)) then
    if j − i > p then
      z ← z + p
      c ← c − p + 1
      j ← j − p
    else
      z ← z + p
      c ← c − p + 1
      (i, j, k, p) ← (0, 1, 1, 1)
  else
    z ← z + max(i, min(c − i, j)) + 1
    c ← 1
    (i, j, k, p) ← (0, 1, 1, 1)
```

Simplified

```
Algorithm Crochemore
i ← 1; ℓ ← 0
(s, p) ← (0, 0)
while i ≤ n − m do
  while T[i + ℓ] = P[1 + ℓ] do
    update(ℓ, s, p)
  if ℓ = m then
    report(i)
  if 3p ≤ j and P[1..s] = P[p..p + s) then
    i ← i + p
    j ← j − p
  else
    i ← i + ⌊j/3⌋ + 1
    j ← 0
```

Fundamental concept

Definition

An integer p is a period of string X if $p \leq |X|$ and X is a prefix of $X[1..p]^\infty$. The shortest period of X is denoted $per(X)$.

Fundamental concept

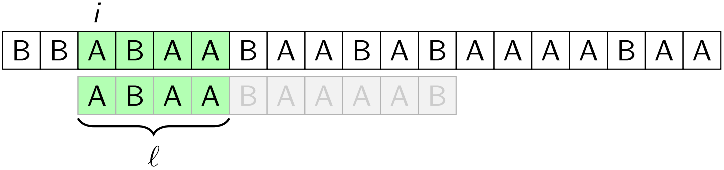
Definition

An integer p is a period of string X if $p \leq |X|$ and X is a prefix of $X[1..p]^\infty$. The shortest period of X is denoted $per(X)$.

Example

Periods of $X = ABAABAAB$ are: 3, 6 and 8 thus $per(X) = 3$.

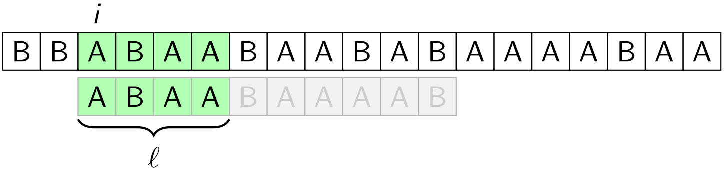
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i+\ell] = \mathrm{P}[1+\ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

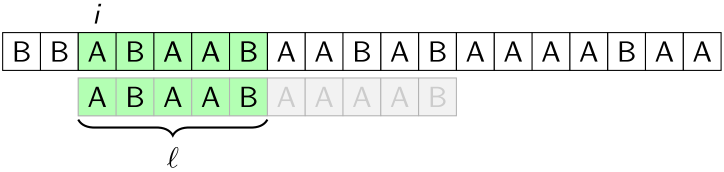
Basic principle



Algorithm `Morris-Pratt`

```
i ← 1; ℓ ← 0
while i ≤ n − m do
    while T[i + ℓ] = P[1 + ℓ] do
        ++ℓ
    if ℓ = m then report(i)
    Δ ← per[ℓ]              ▶ per[ℓ] = per(P[1..ℓ])
    ℓ′ ← ℓ − per[ℓ]
    i ← i + Δ; ℓ ← ℓ′
```

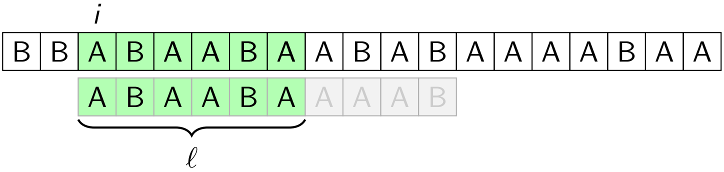
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $T[i+\ell] = P[1+\ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(P[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

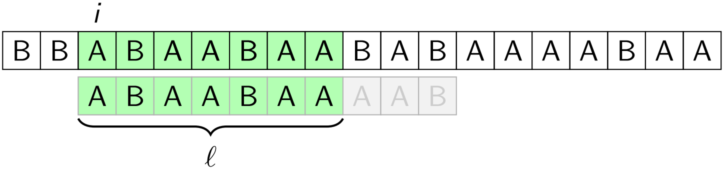
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $T[i + \ell] = P[1 + \ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(P[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

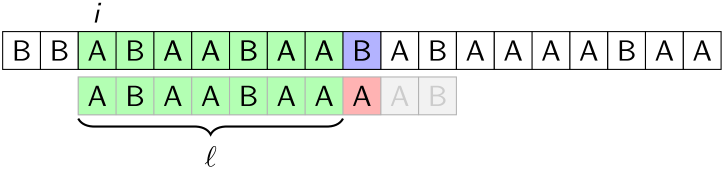
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

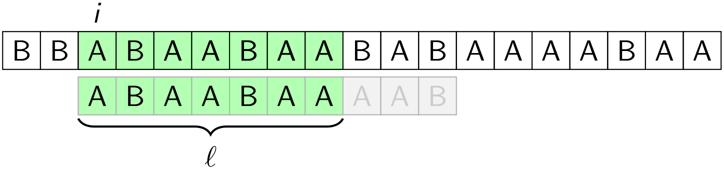
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
    while $T[i+\ell] = P[1+\ell]$ **do**
        $++\ell$
    if $\ell = m$ **then** $report(i)$
    $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(P[1..\ell])$
    $\ell' \leftarrow \ell - per[\ell]$
    $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

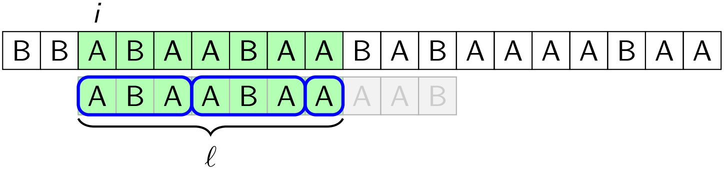
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i+\ell] = \mathrm{P}[1+\ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** *report*(i)
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

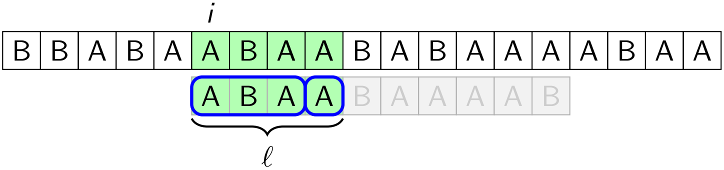
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
    while $\mathrm{T}[i+\ell] = \mathrm{P}[1+\ell]$ **do**
        $++\ell$
    if $\ell = m$ **then** $report(i)$
    $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
    $\ell' \leftarrow \ell - per[\ell]$
    $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

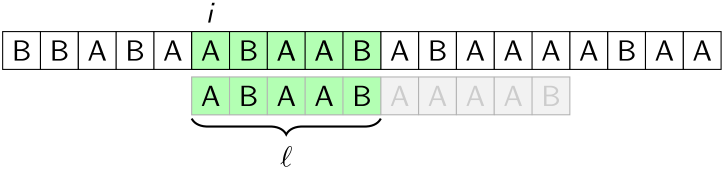
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i+\ell] = \mathrm{P}[1+\ell]$ **do**
    $\text{++}\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

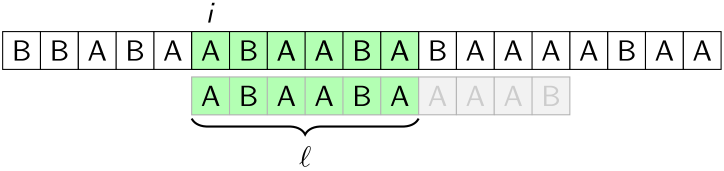
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i+\ell] = \mathrm{P}[1+\ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

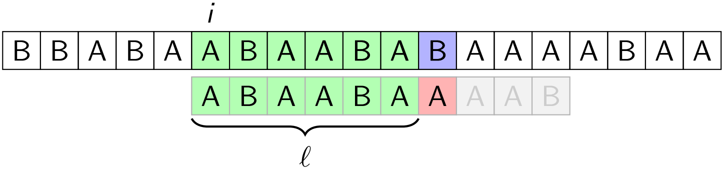
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

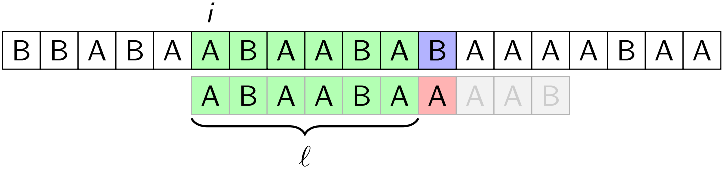
Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \le n - m$ **do**
    while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
        $++\ell$
    if $\ell = m$ **then** *report*(i)
    $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
    $\ell' \leftarrow \ell - per[\ell]$
    $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Basic principle



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $T[i + \ell] = P[1 + \ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(P[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

- Time: $\mathcal{O}(n + m)$ (each step increases the value of $i + \ell \leq 2n$)

Basic tools cont'd

Definition

Let $MaxSuf(X)$ be the lex-maximal suffix of X.

Basic tools cont'd

Definition

Let $\mathrm{M}\mathit{axSuf}(X)$ be the lex-maximal suffix of X.

Example

A
ANA
ANANA
BANANA
NA
NANA

$X = \mathrm{BANANA}$

Basic tools cont'd

Definition

Let $MaxSuf(X)$ be the lex-maximal suffix of X.

Example

A
ANA
ANANA
BANANA
NA
NANA

X = BANANA

Basic tools cont'd

Definition

Let $MaxSuf(X)$ be the lex-maximal suffix of X.

Example

A
ANA
ANANA
BANANA
NA
NANA

$$X = \text{BANANA}$$
$$MaxSuf(X) = \text{NANA}$$

Basic tools cont'd

Definition

Let $MaxSuf(X)$ be the lex-maximal suffix of X.

Definition

A pair of integers (s, p) is called *MS-decomposition* of X if $MaxSuf(X) = X[s..n]$ and $per(X[s..n]) = p$.

Basic tools cont'd

Definition

Let $MaxSuf(X)$ be the lex-maximal suffix of X.

Definition

A pair of integers (s, p) is called *MS-decomposition* of X if $MaxSuf(X) = X[s..n]$ and $per(X[s..n]) = p$.

Example

MS-decomposition of $X = BANANA$ is $(3, 2)$.

Simplified algorithm overview

- Only $\mathcal{O}(1)$ space is available $\Rightarrow$ no $per[1..n]$ precomputed

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Simplified algorithm overview

- Only $\mathcal{O}(1)$ space is available $\Rightarrow$ no $per[1..n]$ precomputed

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Simplified algorithm overview

- We keep MS-decomposition (s, p) of matching prefix $\mathrm{P}[1..\ell]$

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Simplified algorithm overview

- We keep MS-decomposition (s, p) of matching prefix $\mathrm{P}[1..\ell]$

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$; $(s, p) \leftarrow (0, 0)$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Simplified algorithm overview

- We keep MS-decomposition (s, p) of matching prefix $\mathsf{P}[1..\ell]$

Algorithm `Crochemore`

```
i ← 1; ℓ ← 0; (s, p) ← (0, 0)
while i ≤ n − m do
    while T[i + ℓ] = P[1 + ℓ] do
        ++ℓ; update(s, p)            ▶ O(1) (amortized)
    if ℓ = m then report(i)
    Δ ← per[ℓ]        ▶ per[ℓ] = per(P[1..ℓ])
    ℓ′ ← ℓ − per[ℓ]
    i ← i + Δ; ℓ ← ℓ′
```

Simplified algorithm overview

- We keep MS-decomposition (s, p) of matching prefix $\mathrm{P}[1..\ell]$
- Recall that $p = per(MaxSuf(\mathrm{P}[1..\ell]))$

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$; $(s, p) \leftarrow (0, 0)$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$; update(s, p) ▶ $\mathcal{O}(1)$ (amortized)
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Simplified algorithm overview

- We keep MS-decomposition (s, p) of matching prefix $\mathrm{P}[1..\ell]$
- Recall that $p = per(MaxSuf(\mathrm{P}[1..\ell]))$
  - Lemma: If $per(\mathrm{P}[1..\ell]) \leq \ell/3$ then $per(\mathrm{P}[1..\ell]) = p$

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$; $(s, p) \leftarrow (0, 0)$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    ++ℓ; update(s, p) ▶ $\mathcal{O}(1)$ (amortized)
  if $\ell = m$ **then** $report(i)$
  $\Delta \leftarrow per[\ell]$ ▶ $per[\ell] = per(\mathrm{P}[1..\ell])$
  $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Simplified algorithm overview

- We keep MS-decomposition (s, p) of matching prefix $\mathrm{P}[1..\ell]$
- Recall that $p = per(MaxSuf(\mathrm{P}[1..\ell]))$
  - Lemma: If $per(\mathrm{P}[1..\ell]) \leq \ell/3$ then $per(\mathrm{P}[1..\ell]) = p$

Algorithm `Crochemore`

1: $i \leftarrow 1$; $\ell \leftarrow 0$; $(s, p) \leftarrow (0, 0)$
2: **while** $i \leq n - m$ **do**
3: **while** $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
4: ++ℓ; update(s, p) ▶ $\mathcal{O}(1)$ (amortized)
5: **if** $\ell = m$ **then** $report(i)$
6: **if** $per(\mathrm{P}[1..\ell]) \leq \ell/3$ **then** ▶ $\mathcal{O}(1)$ (amortized)
7: $\Delta \leftarrow p$; $\ell' \leftarrow \ell - p$ ▶ $per(\mathrm{P}[1..\ell]) = p$
8: $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Simplified algorithm overview

- We keep MS-decomposition (s, p) of matching prefix $\mathrm{P}[1..\ell]$
- Recall that $p = per(MaxSuf(\mathrm{P}[1..\ell]))$
  - Lemma: If $per(\mathrm{P}[1..\ell]) \leq \ell/3$ then $per(\mathrm{P}[1..\ell]) = p$

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$; $(s, p) \leftarrow (0, 0)$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$; update(s, p) ▶ $\mathcal{O}(1)$ (amortized)
  if $\ell = m$ **then** $report(i)$
  if $per(\mathrm{P}[1..\ell]) \leq \ell/3$ **then** ▶ $\mathcal{O}(1)$ (amortized)
    $\Delta \leftarrow p$; $\ell' \leftarrow \ell - p$ ▶ $per(\mathrm{P}[1..\ell]) = p$
  else
    $\Delta \leftarrow \lfloor \ell/3 \rfloor + 1$ ▶ safe shift
    $(\ell', s, p) \leftarrow (0, 0, 0)$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Simplified algorithm overview

- We keep MS-decomposition (s, p) of matching prefix $\mathrm{P}[1..\ell]$
- Recall that $p = per(MaxSuf(\mathrm{P}[1..\ell]))$
  - Lemma: If $per(\mathrm{P}[1..\ell]) \leq \ell/3$ then $per(\mathrm{P}[1..\ell]) = p$

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$; $(s, p) \leftarrow (0, 0)$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $++\ell$; update(s, p) ▶ $\mathcal{O}(1)$ (amortized)
  if $\ell = m$ **then** $report(i)$
  if $per(\mathrm{P}[1..\ell]) \leq \ell/3$ **then** ▶ $\mathcal{O}(1)$ (amortized)
    $\Delta \leftarrow p$; $\ell' \leftarrow \ell - p$ ▶ $per(\mathrm{P}[1..\ell]) = p$
  else
    $\Delta \leftarrow \lfloor \ell/3 \rfloor + 1$ ▶ safe shift
    $(\ell', s, p) \leftarrow (0, 0, 0)$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

- Time: $\mathcal{O}(n + m)$ (each step increases the value of $3i + \ell \leq 4n$)

Outline

Motivation

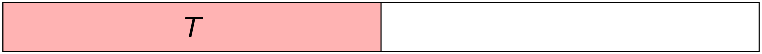
- Scan-based lightweight LZ77 factorization:

Motivation

- Scan-based lightweight LZ77 factorization:
  - text is stored on disk

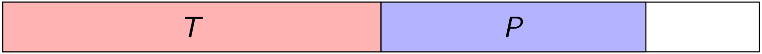
Motivation

- Scan-based lightweight LZ77 factorization:
  - text is stored on disk
  - a prefix T of text was already processed



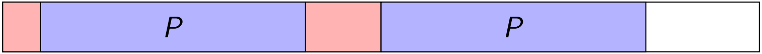
Motivation

- Scan-based lightweight LZ77 factorization:
  - text is stored on disk
  - a prefix T of text was already processed
  - goal: the length of the longest P following T that occurs twice in TP



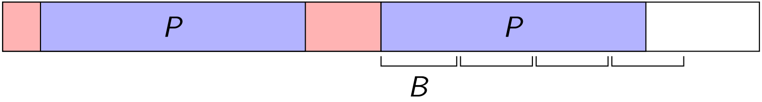
Motivation

- Scan-based lightweight LZ77 factorization:
  - text is stored on disk
  - a prefix T of text was already processed
  - goal: the length of the longest P following T that occurs twice in TP



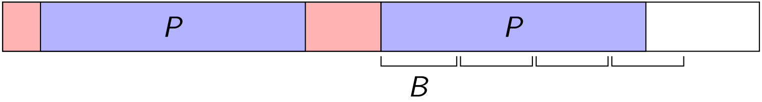
Motivation

- Scan-based lightweight LZ77 factorization:
  - text is stored on disk
  - a prefix T of text was already processed
  - goal: the length of the longest P following T that occurs twice in TP
- P can have length $\gg B$ (where B = memory budget)



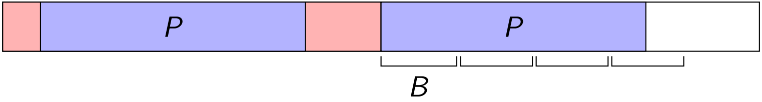
Motivation

- Scan-based lightweight LZ77 factorization:
  - text is stored on disk
  - a prefix T of text was already processed
  - goal: the length of the longest P following T that occurs twice in TP
- P can have length $\gg B$ (where B = memory budget)
- to keep the algorithm lightweight, we cannot index text



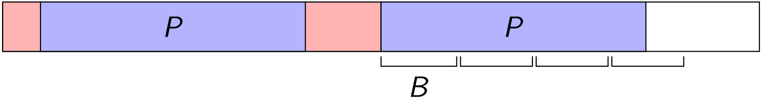
Motivation

- Scan-based lightweight LZ77 factorization:
  - text is stored on disk
  - a prefix T of text was already processed
  - goal: the length of the longest P following T that occurs twice in TP
- P can have length $\gg B$ (where B = memory budget)
- to keep the algorithm lightweight, we cannot index text
- solution: search each block using string matching



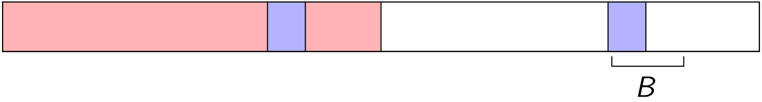
Longest prefix matching

- if the block under consideration contains the phrase endpoint we need to compute its length



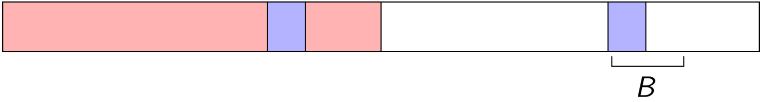
Longest prefix matching

- if the block under consideration contains the phrase endpoint we need to compute its length



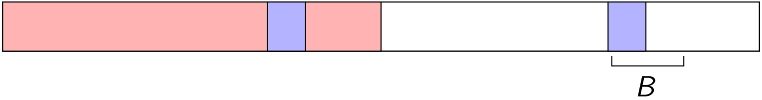
Longest prefix matching

- if the block under consideration contains the phrase endpoint we need to compute its length
  - *longest prefix matching* problem



Longest prefix matching

- if the block under consideration contains the phrase endpoint we need to compute its length
  - *longest prefix matching* problem

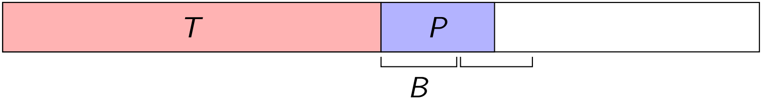


Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$; $\ell_{max} \leftarrow 0$
while $i \leq n - m$ **do**
  while $T[i + \ell] = P[1 + \ell]$ **do**
    $++\ell$
  if $\ell > \ell_{max}$ **then**
    $\ell_{max} \leftarrow \ell$
  $\Delta \leftarrow per[\ell]$; $\ell' \leftarrow \ell - per[\ell]$
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

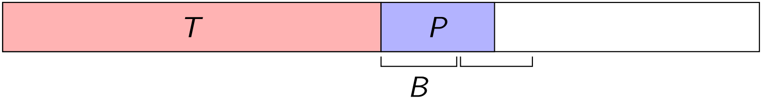
Sparse longest prefix matching

- searching for occurrences of prefixes of 2nd block should be restricted to positions where occurrences of the 1st block end



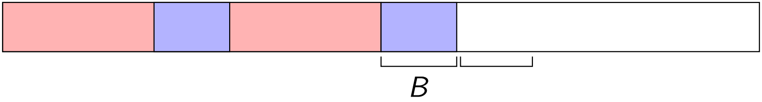
Sparse longest prefix matching

- searching for occurrences of prefixes of 2nd block should be restricted to positions where occurrences of the 1st block end
  - **sparse** *longest prefix matching* problem



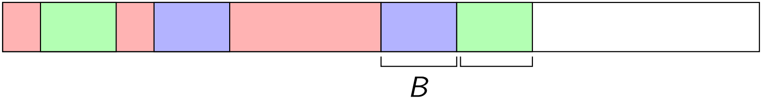
Sparse longest prefix matching

- searching for occurrences of prefixes of 2^{nd} block should be restricted to positions where occurrences of the 1^{st} block end
  - **sparse** *longest prefix matching* problem



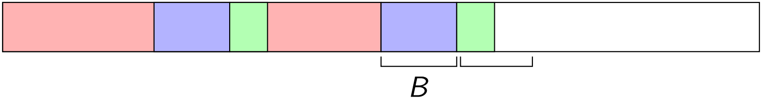
Sparse longest prefix matching

- searching for occurrences of prefixes of 2^{nd} block should be restricted to positions where occurrences of the 1^{st} block end
  - **sparse** *longest prefix matching* problem



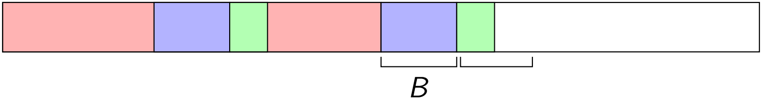
Sparse longest prefix matching

- searching for occurrences of prefixes of 2nd block should be restricted to positions where occurrences of the 1st block end
  - **sparse** *longest prefix matching* problem



Sparse longest prefix matching

- searching for occurrences of prefixes of 2^{nd} block should be restricted to positions where occurrences of the 1^{st} block end
  - **sparse** *longest prefix matching* problem



Algorithm `Morris-Pratt`

$i \leftarrow 1$; $\ell \leftarrow 0$; $\ell_{max} \leftarrow 0$
while $i \leq n - m$ **do**
    if $i \in \mathcal{A}$ **then** ▶ $\mathcal{A}$ = considered positions (ascending)
        while $T[i + \ell] = P[1 + \ell]$ **do**
            $++\ell$
    if $\ell > \ell_{max}$ **then**
        $\ell_{max} \leftarrow \ell$
    $\Delta \leftarrow per[\ell]$; $\ell' \leftarrow \ell - per[\ell]$
    $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Streaming issue

- in basic form of Crochemore's algorithm text is not streamed

Streaming issue

- in basic form of Crochemore's algorithm text is not streamed

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$; $(s, p) \leftarrow (0, 0)$
while $i \leq n - m$ **do**
    while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
        $++\ell$; update(s, p)
    if $\ell = m$ **then** *report*(i)
    if $per(\mathrm{P}[1..\ell]) \leq \ell/3$
        $\Delta \leftarrow p$; $\ell' \leftarrow \ell - p$
    else
        $\Delta \leftarrow \lfloor \ell/3 \rfloor + 1$ ▶ This can cause the value
        $(\ell', s, p) \leftarrow (0, 0, 0)$ ▶ of $i + \ell$ to decrease
    $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

Streaming issue

- in basic form of Crochemore's algorithm text is not streamed

Algorithm `Crochemore`

$i \leftarrow 1$; $\ell \leftarrow 0$; $(s, p) \leftarrow (0, 0)$
while $i \leq n - m$ **do**
  while $\mathrm{T}[i + \ell] = \mathrm{P}[1 + \ell]$ **do**
    $\text{++}\ell$; update(s, p)
  if $\ell = m$ **then** $\mathit{report}(i)$
  if $\mathit{per}(\mathrm{P}[1..\ell]) \leq \ell/3$
    $\Delta \leftarrow p$; $\ell' \leftarrow \ell - p$
  else
    $\Delta \leftarrow \lfloor \ell/3 \rfloor + 1$ ▶ This can cause the value
    $(\ell', s, p) \leftarrow (0, 0, 0)$ ▶ of $i + \ell$ to decrease
  $i \leftarrow i + \Delta$; $\ell \leftarrow \ell'$

- Observation: every streamed text symbol is compared to pattern, i.e., pattern stores recent text history

Conclusion

- minimalistic version of Crochemore's algorithm retaining all its key features

Conclusion

- minimalistic version of Crochemore's algorithm retaining all its key features
- after several modifications very useful in lightweight computation of LZ77 parsing

Conclusion

- minimalistic version of Crochemore's algorithm retaining all its key features
- after several modifications very useful in lightweight computation of LZ77 parsing
- prefix matching problems could be of independent interest

Conclusion

Thank you!