Practically-efficient String Matching

Maxime Crochemore

King's College London

Maxime.Crochemore@kcl.ac.uk
http://www.dcs.kcl.ac.uk/staff/mac/

Examples

⋆ Naive search with backward scan (1)

```
a a c a a a a a b a b a a b a b . .
a b a a a a
  a b a a a a
    a b a a a a
      a b a a a a
        a b a a a a
          . . .
```

⋆ Naive search with backward scan (2)

```
a b a b a a a b a b b a a b a b . .
a b a a a b a b
  a b a a a b a b
    a b a a a b a b
      a b a a a b a b
        . . .
```

Right-to-left scan — shift

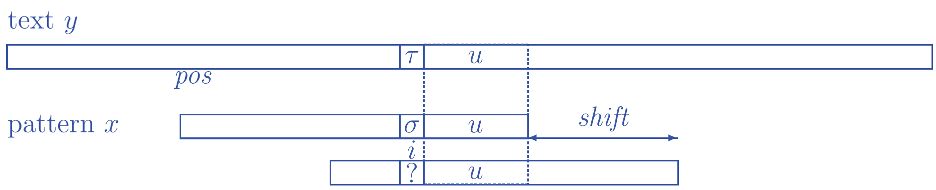
`text`	···	c	g	c	t	c	**g**	**c**	g	c	t	**a**	t	c	g	···
pattern		c	g	c	t	a	**g**	**c**								
						c	**g**	**c**	t	a	g	c				
								c	g	c	t	**a**	g	c		

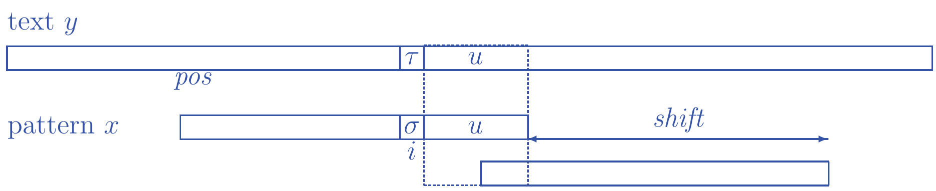
- ⋆ Match shift: good-suffix rule (function d)
- ⋆ Occurrence shift heuristics: bad-character rule
- ⋆ Extra rules if memorization

---

while window on text **do**
  $u \longleftarrow$ longest common suffix of window and pattern
  if $u =$ pattern **then** report a match
  shift window $d(u)$ places to the right

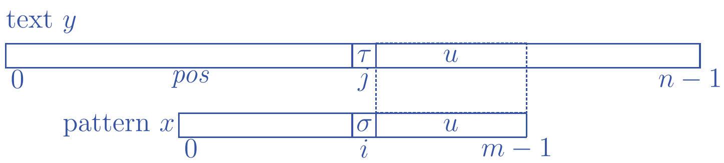
Match shift





- ⋆ **Precomputation**
  of rightmost occurrences of u's: $O(m)$
- ⋆ Second shift length = a period of x
- ⋆ Table D implements the good-suffix rule:
  $shift = d(u) = D[i]$

BM algorithm



★ No memorization of previous matches

BM(string x, y; integer m, n)

  $pos \longleftarrow 0$

  while $pos \leq n - m$ **do**

    $i \longleftarrow m - 1$

    while $i \geq 0$ **and** $x[i] = y[pos + i]$ **do** $i \longleftarrow i - 1$

    if $i = -1$ **then**

      output('x occurs in y at position ', pos)

      $pos \longleftarrow pos + period(x)$

    else

      $pos \longleftarrow pos + D[i]$

Suffix displacement

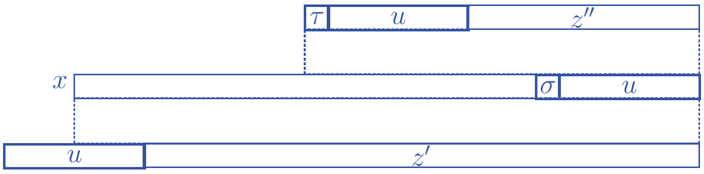
★ Displacement function d:

$d(u) = \min\{|z| > 0 \mid (x \text{ suffix of } uz)$ or

$(\tau uz \text{ suffix of } x \text{ and } \tau u \text{ not suffix of } x, \text{ for } \tau \in \Sigma)\}$

★ Displacement table D:

$D[i] = d(x[i+1\,..\,m-1]), \text{ for } i = 0, .., m-1$

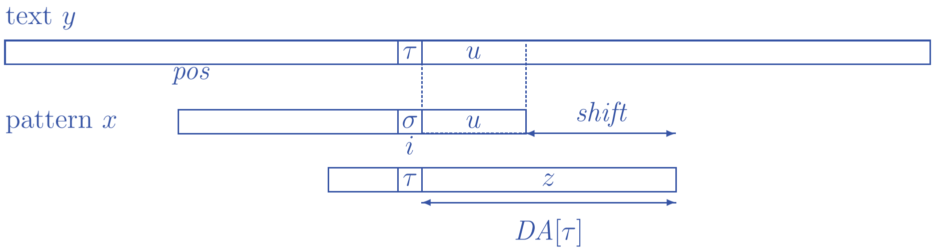


★ Note 1: u is a (strict) border of uz''

★ Note 2: $|z'|$ is a period of uz' (thus, $|z'|$ is a period of x)

Lemma 1 *Table D can be computed in linear time.* [see Page 22]

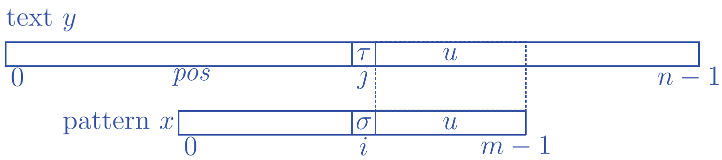
Occurrence shift



- Table DA implements the bad-character rule:
  $DA[\tau] = \min\{|z| > 0 \mid \tau z \text{ suffix of } x\} \cup \{|x|\}$
- $shift = DA[\tau] - |u| = DA[\tau] - m + i + 1$

COMPUTE_DA(string x; integer m)
 for all σ **in** Σ **do**
  $DA[\sigma] = m$
 for $i \longleftarrow 0$ **to** $m-2$ **do**
  $DA[x[i]] = m - i - 1$
 return DA

BM with occurrence shift



- Use of DA in addition to D

BM(string x, y; integer m, n);
 $pos \longleftarrow 0$
 while $pos \leq n - m$ **do**
  $i \longleftarrow m - 1$
  while $i \geq 0$ **and** $x[i] = y[pos + i]$ **do**
   $i \longleftarrow i - 1$
  if $i = -1$ **then**
   output('x occurs in y at position ', pos)
   $pos \longleftarrow pos + period(x)$
  else
   $pos \longleftarrow pos + \max\{D[i], DA[y[pos + i]] - m + i + 1\}$

Complexity of BM

★ **Preprocessing phase**

match shift	$O(m)$
occurrence shift	$O(m + \text{card}\,\Sigma)$

★ **Search phase** (finding all occurrences)

running time	$O(n \times m)$
minimum number of comparisons	n/m
maximum number of comparisons	$n \times m$

★ **Extra space**

for shift functions	$O(m + \text{card}\,\Sigma)$
can be reduced to	$O(m)$

Symbol comparisons in variants of BM

★ For finding the first occurrence

[Knuth, Morris, Pratt, 1977]	$\leq 7 \times n$
[Guibas, Odlysko, 1980]	$\leq 4 \times n$
[Cole, 1990]	$\leq 3 \times n$

Theorem 1 *If period$(x) > m/2$, BM searching algorithm performs at most $3n - n/m$ symbol comparisons. The bound is tight.*

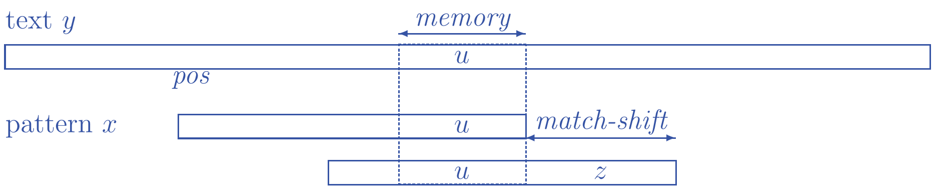
Proof difficult

Symbol comparisons in variants of BM (followed)

- ⋆ For finding all occurrences

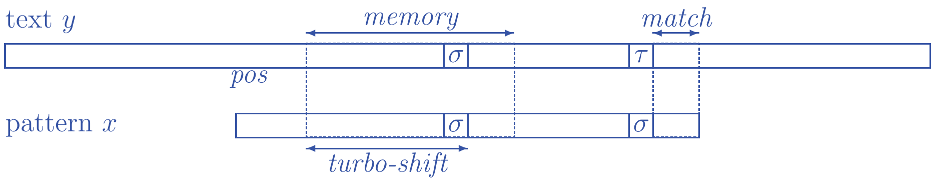
[Galil, 1979]	$O(n)$
$\longrightarrow$ prefix memorization	$O(1)$ extra space
[Crochemore *et alii*, 1991]	$\leq 2 \times n$
$\longrightarrow$ last-suffix memorization	$O(1)$ extra space
$\longrightarrow$ **Turbo-BM**	
[Apostolico, Giancarlo, 1986]	$\leq 1.5 \times n$
$\longrightarrow$ all-suffix memorization	$O(m)$ extra space

Turbo-BM method



- ⋆ **Features**
  - **Stores** the last match in case of match shift (*memory*)
  - **Jumps** on *memory*
  - Uses **turbo-shifts**
- ⋆ **Preprocessing**
  same as BM algorithm
- ⋆ **Search**
  $O(1)$ extra space to store *memory*: (length, right position)
- ⋆ Note: *match-shift* is a period of uz since u is a border of it

Turbo-shift



- $\textit{turbo-shift} = |\textit{memory}| - |\textit{match}|$
- **Use in Turbo-BM**:

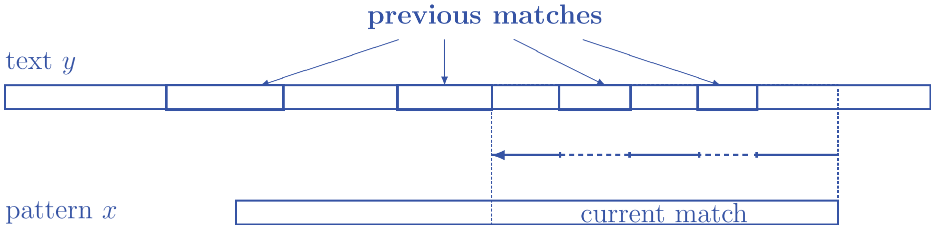
  $\textit{shift} \leftarrow \max\{\textit{match-shift}, \textit{occ-shift}, \textit{turbo-shift}\}$;
  if $(\textit{shift} = \textit{match-shift})$ **then**
      set memory;
  else
      { $\textit{shift} \leftarrow \max\{\textit{shift}, \textit{match}+1\}$; no memory; }

Tight number of comparisons for Turbo-BM

Theorem 2 *The Turbo-BM searching algorithm runs in time $O(n)$. It makes no more than $2n$ symbol comparisons.*

Proof difficult

AG method



⋆ **Features**

– **Stores** all previous matches (suffixes of pattern)

– Uses the table Suf

$Suf[i]$ = longest suffix of x ending at i in x

⋆ **Extra preprocessing**

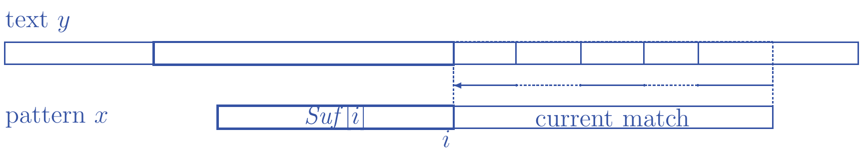
computing Suf on the pattern: $O(m)$ time and space

⋆ **Search**

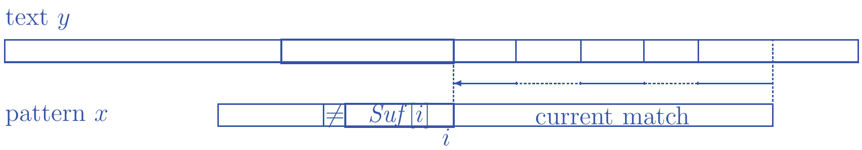
$O(m)$ extra space to store the matches

Rules for shifts in AG

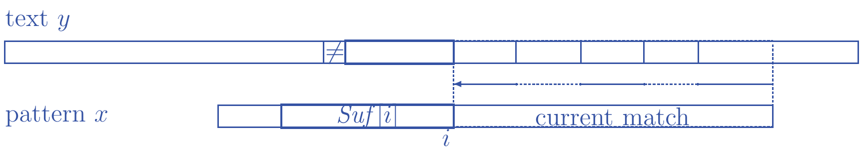
- **match**



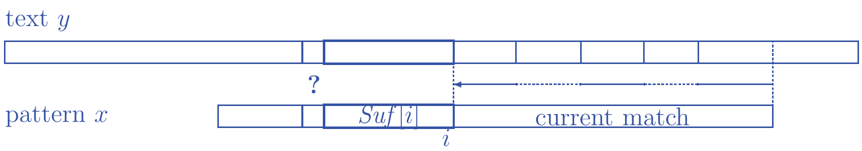
- **mismatch**



- **mismatch**



- **jump**



Tight number of comparisons for AG

Lemma 2 *The AG algorithm makes at most $\frac{n}{2}$ comparisons on text characters previously compared.*

Theorem 3 *The AG searching algorithm runs in time $O(n)$. It makes no more than $1.5n$ symbol comparisons.*

Proof rather difficult

Worst-case example

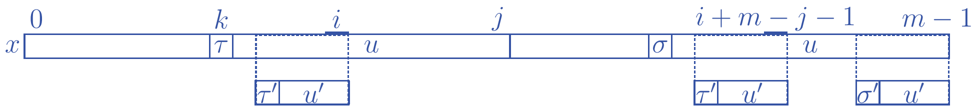
```
a a b a a a b a a b a a a b a a b a a a b ...
a a b a a a b
        a a b a a a b
          a a b a a a b
            a a b a a a b
              a a b a a a b
                      a a b a a a b
                        a a b a a a b
                          a a b a a a b
                            a a b a a a b
```

pattern $= a^{m-1}ba^mb$, text $= (a^{m-1}ba^mb)^e$

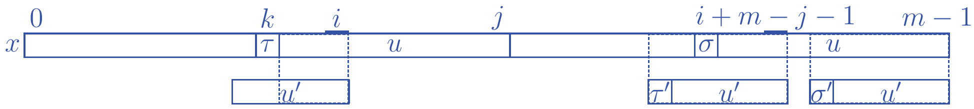
number of comparisons $= 2m+1+(3m+1)e = \frac{3m+1}{2m+1}n-m \approx 1.5n$

Computing the table of suffixes

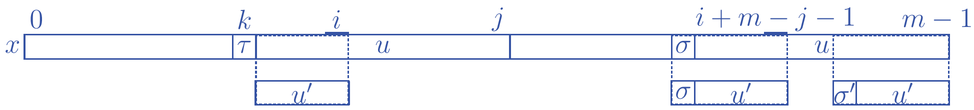
- ⋆ i given: $Suf[j]$ known for $i < j \leq m-1$
  $k = \min\{j - Suf[j] \mid i < j < m\}$ (leftmost considered position)
  $\sigma \neq \tau$; $\sigma' \neq \tau'$; $Suf[i+m-j-1] = |u'|$



- ⋆ **If** $Suf[i+m-j-1] < i-k$: $Suf[i] = Suf[i+m-j-1]$



- ⋆ **If** $Suf[i+m-j-1] > i-k$: $Suf[i] = i-k$



- ⋆ **If** $Suf[i+m-j-1] = i-k$: find $Suf[i]$ by scanning **from position** k
  Yields a new k and a new j $(= i)$

Extra preprocessing for AG

- ⋆ Linear-time computation of table Suf

COMPUTE_SUF(string x; integer m)
    $Suf[m-1] \longleftarrow m$; $k \longleftarrow m-1$
    for $i \longleftarrow m-2$ **downto** 0 **do**
        if $i > k$ **and** $Suf[i+m-j-1] \neq i-k$ **then**
            $Suf[i] \longleftarrow \min\{Suf[i+m-j-1], i-k\}$
        else
            $j \longleftarrow i$; $k \longleftarrow \min\{i, k\}$;
            while $k \geq 0$ **and** $x[k] = x[k+m-j-1]$ **do**
                $k \longleftarrow k-1$;
            $Suf[i] \longleftarrow j-k$
    return Suf

From *Suf* to D

i	0	1	2	3	4	5	6	7	8	9	10
$x[i]$	a	b	a	a	a	b	a	b	a	b	a
$Suf[i]$	1	0	3	1	1	0	3	0	5	0	11

⋆ **Initializing D using the periods of x only**

$Suf[2] = 3 \Longrightarrow$ period 8 ; $Suf[0] = 1 \Longrightarrow$ period 10 ; period 11

8	8	8	8	8	8	8	8	10	10	11

⋆ **Accounting for occurrences inside x of its suffixes**

$Suf[3] = 1 \Longrightarrow D[9] \leq 7$; $Suf[8] = 5 \Longrightarrow D[5] \leq 2$

	8	8	8	8	8	$\cancel{8}$	8	$\cancel{8}$	10	$\cancel{10}$	$\cancel{11}$
						2		$\cancel{8}$		$\cancel{10}$	$\cancel{9}$
								4		$\cancel{7}$	$\cancel{5}$
										6	$\cancel{3}$
											1
$D[i]$	8	8	8	8	8	2	8	4	10	6	1

Computing the displacement table D with the table *Suf*

⋆ Linear-time computation of table D

COMPUTE_D(string x; integer m; table D)
 $j \longleftarrow 0$;
 for $i \longleftarrow m-2$ **downto** -1 **do**
  if $i = -1$ **or** $Suf[i] = i+1$ **then**
   while $j < m-i-1$ **do**
    $D[j] \longleftarrow m-i-1$;
    $j \longleftarrow j+1$;
 for $i \longleftarrow 0$ **to** $m-2$ **do**
  $D[m - Suf[i] - 1] \longleftarrow m-i-1$;
 return D