Real-time string matching in sublinear space

Leszek Gąsieniec, Roman Kolpakov
Department of Computer Science,
University of Liverpool, Liverpool, UK

String matching problem

$T = t_1 \ldots t_n$ — text

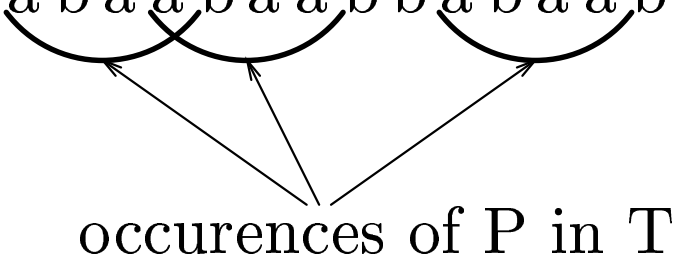
$P = p_1 \ldots p_m$ — pattern $(m \leq n)$

The problem is to find all occurrences of P in T

Ex:

P = a b a a

T = a b a b a a b a a b b a b a a b



Next function

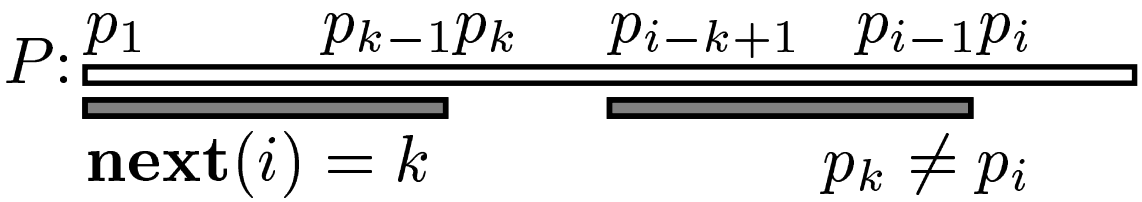
$P = p_1 \dots p_m$

next : $\{1, \dots, m\} \to \{0, 1, \dots, m-1\}$

next(i) is the largest positive k s.t.

$p_1 \dots p_{k-1} = p_{i-k+1} \dots p_{i-1}$ and $p_i \neq p_k$

if such k exists (**next**$(i) = 0$ otherwise).



Ex: $P = abaaababababc$

next$(1) = 0$; **next**$(2) = 1$; **next**$(3) = 0$;

next$(4) = 2$; **next**$(5) = 2$; **next**$(6) = 1$;

next$(7) = 0$; **next**$(8) = 4$; **next**$(9) = 0$;

next$(10) = 4$; **next**$(11) = 0$;

next$(12) = 4$; **next**$(13) = 3$

next can be computed in $O(m)$ time

Knuth-Morris-Pratt algorithm

KMP algorithm:

compute **next** for $p_1 \dots p_m\$$

$i := 1$; {pointer to pattern symbol p_i}

for $j := 1$ **to** n {pointer to text symbol t_j}

    while $i > 0$ **and** $p_i \neq t_j$

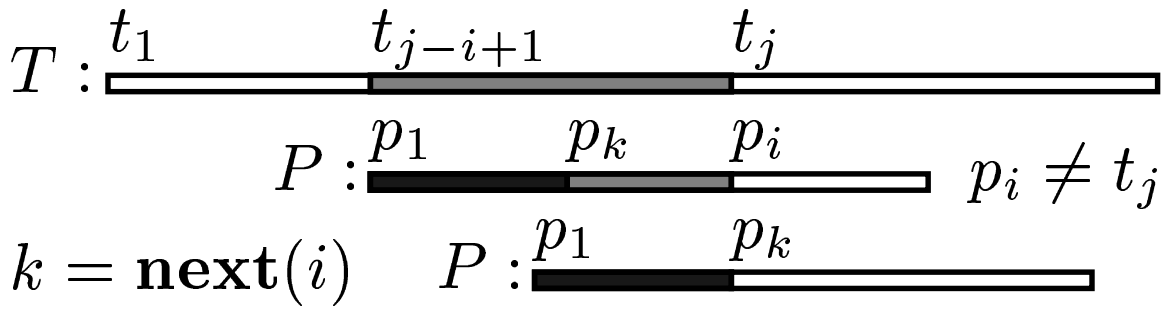
        $i :=$ **next**(i);

    if $i = m$

        report an occurence of P ending in t_j;

        $i :=$ **next**$(m+1)$;

    else $i := i + 1$;



Real-time string matching

Def: *On-line* string matching — input text symbols are assessed consecutively and any occurence of pattern is reported during assessing the last symbol of the occurence

Def: *Real-time* string matching — on-line string matching such that the time for assessment of an input text symbol is bounded by a constant

$f_0 = b$, $f_1 = a$, $f_n = f_{n-1} f_{n-2}$ for $n \geq 2$

$|f_n| = F_n$ — n-th Fibonacci number

f_n and $f_{n-2} f_{n-1}$ coincide in the first $F_n - 2$ letters and don't coincide in the $(F_n - 1)$-th letter

Ex: $f_5 = \underline{abaaba}ba$

$f_3 f_4 = \underline{abaaba}ab$

if $P = f_n$ then $\mathbf{next}(F_n - 1) = F_{n-1} - 1$

$\Downarrow$

the **while**-loop in KMP algorithm can be executed at least n times for the same text symbol t_j

$\Downarrow$

KMP algorithm is not a real-time algorithm

Real-time KMP modification

compute **next** for $p_1 \ldots p_m\$$

$i := 1$; $j' = 1$;

for $j := 1$ **to** n

    KMP_STEP();

    if $j' \leq j$ **then** KMP_STEP();

KMP_STEP():

if $p_i = t_{j'}$

    if $i = m$ **then**

        report an occurence of P ending in $t_{j'}$;

        $i :=$ **next**$(m+1)$;

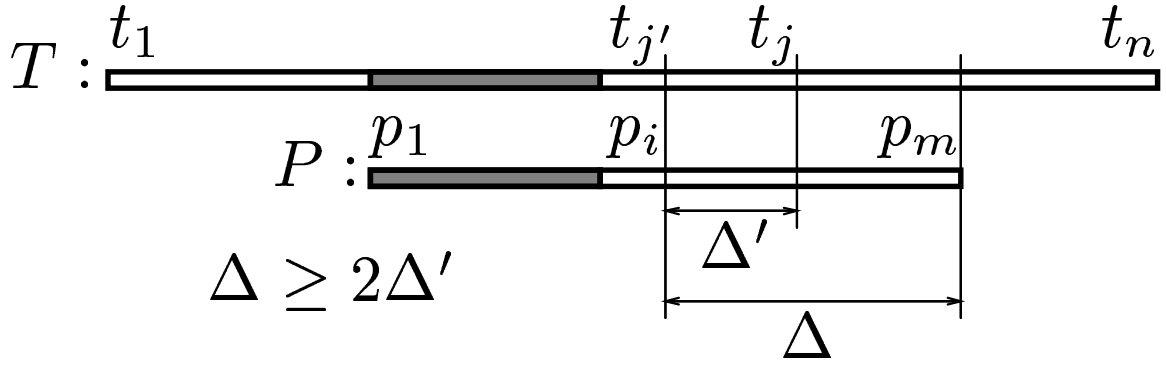
    else $i := i + 1$;

    $j' := j' + 1$;

else $\{p_i \neq t_{j'}\}$

    if next$(i) > 0$ **then** $i :=$ **next**(i);

    else $j' := j' + 1$; $i := 1$;



a) the algorithm finds any occurrence $T[u..v]$ of P during the processing the end symbol t_v;
b) $\Delta' \leq m/2$.

Galil 1981:
The algorithm is a real-time algorithm which solves the string matching problem in $O(m+n)$ time and $O(m)$ space

Space efficient string matching

Can string matching problem be solved in $o(m)$ space?

Galil, Seiferas 1983:
Crochemore, Perrin 1991:
Gąsieniec, Plandowski, Rytter 1995:
String matching problem can be solved in $O(m+n)$ time and $O(1)$ space

Gąsieniec, Potapov 2003:
On-line algorithm which solves the string matching problem in $O(m+n)$ time and $O(1)$ space

All the algorithms are not real-time

Our problem

Can real-time string matching problem be solved in $o(m)$ space?

Partial next function

for any real x

$$\mathbf{next}_x(i) = \begin{cases} \mathbf{next}(i) & \text{if} \quad \mathbf{next}(i) > x, \\ 0 & \text{otherwise.} \end{cases}$$

Ex: $P = abaaababababc$

$\mathbf{next}_1(1) = 0; \quad \mathbf{next}_1(2) = 0;$

$\mathbf{next}_1(3) = 0; \quad \mathbf{next}_1(4) = 2;$

$\mathbf{next}_1(5) = 2; \quad \mathbf{next}_1(6) = 0;$

$\mathbf{next}_1(7) = 0; \quad \mathbf{next}_1(8) = 4;$

$\mathbf{next}_1(9) = 0; \quad \mathbf{next}_1(10) = 4;$

$\mathbf{next}_1(11) = 0; \quad \mathbf{next}_1(12) = 4;$

$\mathbf{next}_1(13) = 3$

$\mathbf{list}[\mathbf{next}_1]$:

$(4,2)(5,2)(8,4)(10,4)(12,4)(13,3) =$

$\underbrace{(4,2)(5,2)}_{\alpha_1}\underbrace{(8,4)(10,4)(12,4)}_{\alpha_2}\underbrace{(13,3)}_{\alpha_3}$

$\mathbf{list}^*[\mathbf{next}_1] = \alpha_1\alpha_2\alpha_3$ — list of ranges

$\mathbf{value}(\alpha_2) = 4, \quad \mathbf{begin}(\alpha_2) = 8,$

$\mathbf{end}(\alpha_2) = 12, \quad \mathbf{step}(\alpha_2) = 2$

$\Downarrow$

each range requires $O(1)$ space.

Lemma:

$\mathbf{list}^*[\mathbf{next}_x]$ contains $O(m/x)$ ranges and requires $O(m/x)$ space.

Let $\tau \geq 2$

$m_i = \lfloor m^{i/\tau} \rfloor$, $P_i = P[1..m_i]$, $i = 1, \ldots, \tau$

h_0 — **next** function for $P_1\$$

h_i — $\mathbf{next}_{m^{i/\tau}}$ function for $P_{i+1}\$$,
$i = 1, \ldots, \tau - 1$

Ex: $P = abaaababababab$, $\tau = 2$

$m = |P| = 12$, $m_1 = 3$, $P_1 = aba\$$

$h_0(1) = 0$, $h_0(2) = 1$

$h_0(3) = 0$, $h_0(4) = 2$

$\mathbf{list}^*[h_1] = \alpha_2$

h_0, $\mathbf{list}^*[h_1]$,..., $\mathbf{list}^*[h_{\tau-1}]$ require at total of $O(m^{1/\tau} \cdot \tau)$ space.

KMP_STEP computes for each j largest k, s.t., $p_1 \dots p_k = t_{j-k+1} \dots t_j$

$\mathbf{maxpref}_0(j)$ — largest $k \leq m^{1/\tau}$, s.t., $p_1 \dots p_k = t_{j-k+1} \dots t_j$ ($\mathbf{maxpref}_0(j) = 0$ if there is no such k)
$\mathbf{maxpref}_i(j)$, $i = 1, \dots, \tau - 1$, — largest k, s.t., $p_1 \dots p_k = t_{j-k+1} \dots t_j$ and $m^{i/\tau} < k \leq m^{(i+1)/\tau}$ ($\mathbf{maxpref}_i(j) = 0$ if there is no such k)

COMP_MPREF$_i$, $i = 0, 1, \dots, \tau - 1$, — modification of KMP_STEP for computing of $\mathbf{maxpref}_i$, using $\mathbf{list}^*[h_i]$ (h_0 if $i = 0$)

Space efficient algorithm

compute h_0, **list**$^*[h_1]$,..., **list**$^*[h_{\tau-1}]$

$j_0 := j_1 := \ldots j_{\tau-1} := 1$;

{j_i – current position for which **maxpref**$_i$ is computed}

$l_0 := 1$; $l_1, \ldots, l_{\tau-1}$ are undefined;

{l_i – position of pattern symbol compared with t_{j_i}}

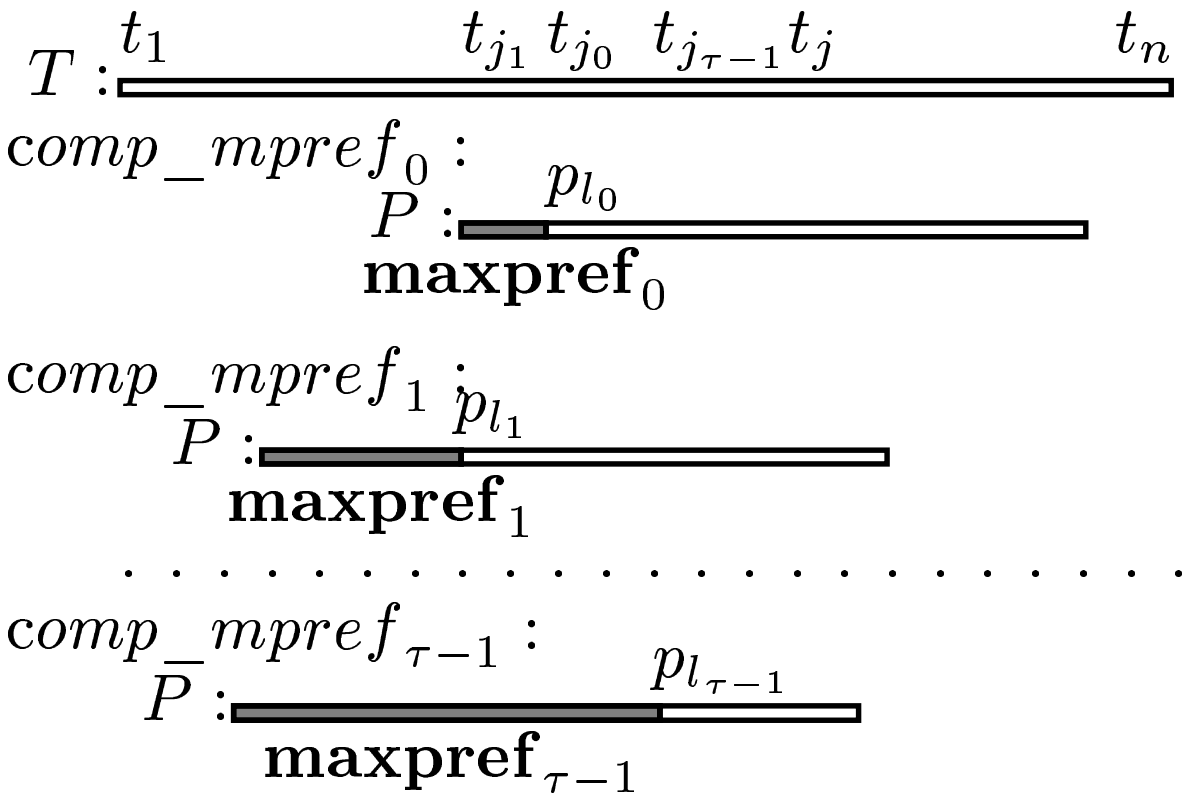
for $j := 1$ **to** n

    read the next symbol t_j;

    for $i := \tau - 1$ **down to** 0

        COMP_MPREF$_i$();

        if $j_i \leq j$ **then** COMP_MPREF$_i$();



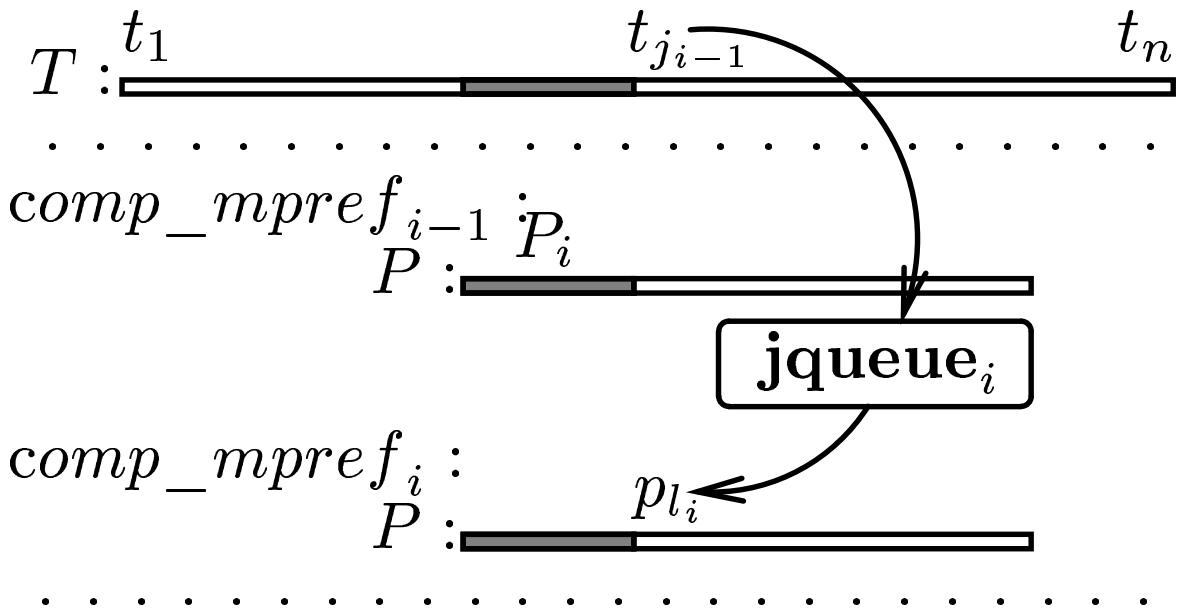
$$j - j_i \leq (|\mathbf{list}^*[h_i]| + 1)/2$$

$$\Downarrow$$

$$\max_i (j - j_i) = O(m^{1/\tau})$$

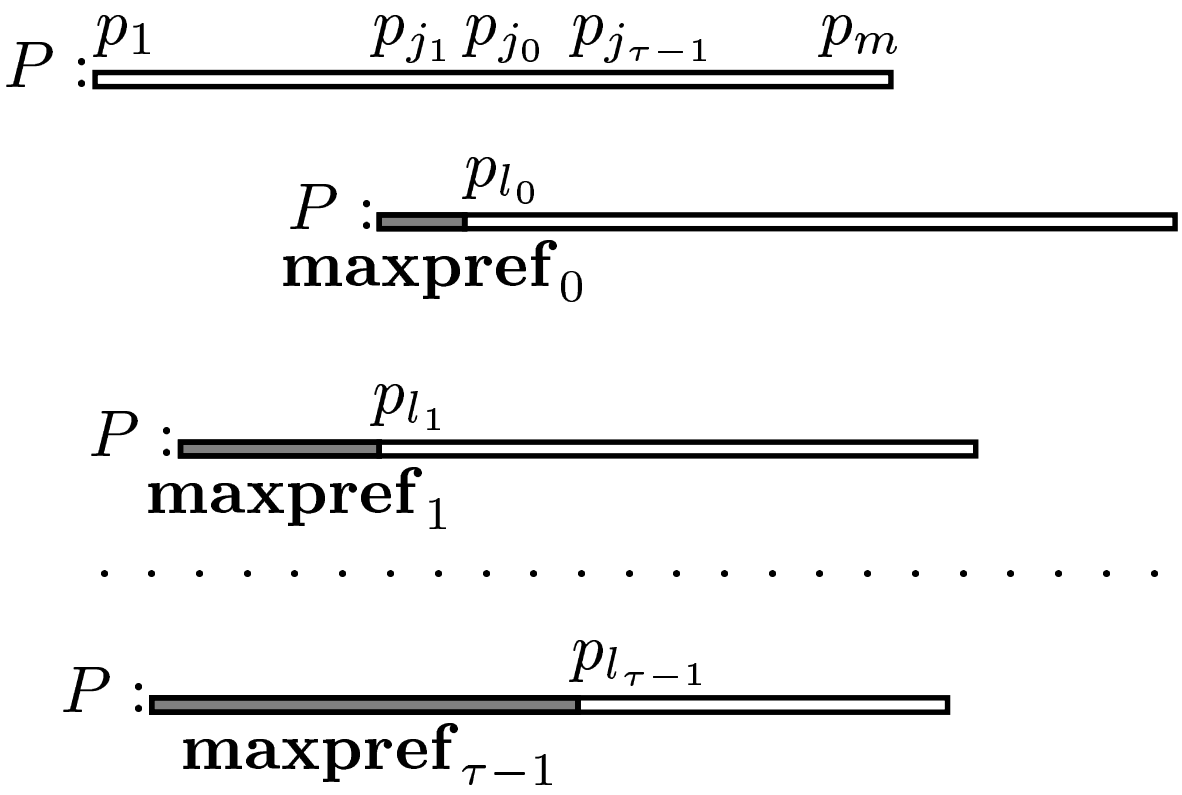
COMP_MPREF$_i$ begins computation of **maxpref**$_i$ only after COMP_MPREF$_{i-1}$ finds an occurence of P_i in T.

Positions of occurences of P_i are communicated to COMP_MPREF$_i$ by means of FIFO queue **jqueue**$_i$:



$$j - j_i = O(m^{1/\tau}) \Rightarrow |\mathbf{jqueue}_i| = O(m^{1/\tau})$$

Preprocessing



Preprocessing requires $O(m)$ time and
$O(m^{1/\tau} \cdot \tau)$ space

COMP_MPREF$_{\tau-1}$ reports occurrences of $P_\tau = P$ in T

The algorithm solves the string matching problem in $O(m+n)$ time and $O(m^{1/\tau} \cdot \tau)$ space, and time required for the test of each text symbol is bounded by $O(\tau)$.

Theorem: For any constant $\varepsilon > 0$ the string matching problem can be solved in real time and $O(m^\varepsilon)$ additional space.

Conclusions

Another version of algorithm:

$j - j_i \leq Km^{1/\tau}$, $K < m^{1/\tau}$

$m'_i = \left\lceil m^{i/\tau} + \frac{Km^{2/\tau}}{m^{1/\tau} - K} \right\rceil$, $P'_i = P[1..m'_i]$

$h'_i = \mathbf{next}_{m^{i/\tau}}$ for $P'_{i+1}\$$

COMP_MPREF$_i$ computes

$\mathbf{maxpref}'_i(j)$ which is largest k, s.t.,

$p_1 \ldots p_k = t_{j-k+1} \ldots t_j$ and

$m^{i/\tau} < k \leq m'_{i+1}$ ($k \leq m'_1$ for $i = 0$)

In this version we can avoid the use of

$\mathbf{jqueue}_i$.

Is this version is more space efficient?

Can string matching problem be solved in real time and $O(\log m)$ additional space?