

## Outline

## Cuprins

1	Recapitulare	1
2	Algoritmul Knuth-Morris-Pratt	2
3	Expresii regulate	8

## 1 Recapitulare

### String Searching (Matching) Problem

*Input* Două șiruri:  $s = s[0] \dots s[n-1]$ , numit subiect sau text, și  $p = p[0] \dots p[m-1]$ , numit pattern.

*Output* Prima apariție a patternului  $p$  în textul  $s$ , dacă există;  $-1$ , altfel.

### Algoritmul naiv (brute force)

- $O(n \cdot m)$  în cazul cel mai nefavorabil,  $O(\min(n, m))$  în cazul cel mai favorabil
- numărul mediu de comparații  $\leq 2(n+1-m)$

Întrebare: putem obține  $O(n)$  în cazul cel mai nefavorabil?

### Algoritmul Rabin-Karp

- utilizează tehnica tabelor de dispersie (hash)
- trebuie să fie ușor de calculat și de comparat valorile hash
- complexitatea în cazul cel mai nefavorabil  $O(n \cdot m)$ , dar foarte puțin probabil să apară în practică
- complexitatea medie  $O(m+n)$
- extensibil la cazul bidimensional (imagini)

### Algoritmul Boyer-Moore

- regula caracterului rău: pentru cazul cel mai nefavorabil are complexitatea  $O(m \cdot n)$
- Regula sufixului bun:
  - complexitate  $O(n+m)$  dacă patternul  $p$  nu apare în subiect; altfel rămâne  $O(n \cdot n)$
  - totuși, cu o simplă modificare (regula Galil, 1979) se poate obține  $O(n+m)$  în toate cazurile
  - algoritmul original al lui Boyer-Moore (1977) utilizează o variantă simplificată a regulei sufixului bun
  - Richard Colen (1991) a stabilit o limită de  $3n$

## 2 Algoritmul Knuth-Morris-Pratt

Algoritmul naiv<sup>1</sup>

a	b	c	b	a	b	a	b	a	a	b	c	b	a	b
				=	=	=	=	=	≠					
				a	b	a	b	a	c	a				

Intuiția<sup>2</sup>

a	b	c	b	a	b	a	b	a	a	b	c	b	a	b
				=	=	=	=	=	≠					
				a	b	a	b	a	c	a				
					a	b	a	b	a	c	a			
						a	b	a	b	a	c	a		

Intuiția<sup>3</sup>

?	?	?	?	a	b	a	b	a	?	?	?	?	?	?
				=	=	=	=	=	≠					
				a	b	a	b	a	?	?				
					a	b	a	b	?	?	?			
						a	b	a	?	?	?	?		

Pentru pattern-ul *ababaca*, dacă la o poziție  $i$  se potrivesc exact 5 caractere, nu există nicio șansă ca pattern-ul să se potrivească la poziția  $i + 1$ .

Ideea

?	?	?	?	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	?	?	?	?	?	?	?
				=	=	=	=	=	=	=	≠						
				$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	?						
								=	=	=							
								$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	?		

Ideea

?	?	?	?	$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	?	?	?	?	?	?	?
				=	=	=	=	=	=	=	≠						
				$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	?						
								=	=	=							
								$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	?		

<sup>1</sup>Exemplu din [CLRS]

<sup>2</sup>Exemplu din [CLRS]

<sup>3</sup>Exemplu din [CLRS]

## Ideea

?	?	?	?	$x_1$	...	$x_k$	...	$x_1$	...	$x_k$	?	?	?	?
				=	=	=	=	=	=	=	≠			
				$x_1$	...	$x_k$	...	$x_1$	...	$x_k$	?			

Ne interesează cea mai mare valoare a lui  $k$  astfel încât  $x_1 \dots x_k$  să fie atât prefix cât și sufix al părții din pattern care s-a potrivit.

## Notății

- reamintim: frontieră (bordură) a unui șir  $t$  - un factor (subșir) care este și prefix și sufix al lui  $t$
- notăm:  $\text{maxFr}(i)$  - frontiera maximă a lui  $p[0..i-1]$  care e factor propriu ( $\neq p[0..i-1]$ )  
 $f[i] = |\text{maxFr}(i)|$  (lungimea frontierei (bordurii) maxime a lui  $p[0..i-1]$ )
- exemplu:  $p = \begin{matrix} a & b & a & b & a & c & a \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{matrix}$

$i$	$\text{maxFr}(i)$	$f[i]$
1	$\varepsilon$	0
2	$\varepsilon$	0
3	$a$	1
4	$ab$	2
5	$aba$	3
6	$\varepsilon$	0

- să vedem pe un exemplu cum poate fi utilizat eficient  $f[i]$

### Exemplu 1/6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$a$	$b$	$c$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$c$	$a$
=	=	≠												
$a$	$b$	$a$	$b$	$a$	$c$	$a$								
0	1	2	3	4	5	6								

- eșec la pozițiile  $i = k = 2$  (poziția  $i$  în subiect, poziția  $k$  în pattern)
- $f[k] = f[2] = 0$
- se face un salt egal cu  $k - f[k] = 2 - 0 = 2$
- următoarele poziții ce se vor compara:  $i = 2, k = 0$  ( $k$  devine  $f[k]$ )

### Exemplu 2/6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$a$	$b$	$c$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$c$	$a$
		≠												
		$a$	$b$	$a$	$b$	$a$	$c$	$a$						
		0	1	2	3	4	5	6						

- eșec la pozițiile  $i = 2, k = 0$
- $f[0] = ?$
- se face un salt egal cu  $k - f[k] = 0 - f[0] = 1$ , deci luăm  $f[0] = -1$
- următoarele poziții ce se vor compara:  $i = 3, k = 0$  (se incrementează cu 1 atât  $i$  cât și  $k$ )

#### Exemplu 3/6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$a$	$b$	$c$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$c$	$a$
			$\neq$											
			$a$	$b$	$a$	$b$	$a$	$c$	$a$					
			0	1	2	3	4	5	6					

- eșec la pozițiile  $i = 3, k = 0$
- $f[0] = -1$
- se face un salt egal cu  $k - f[k] = 0 - f[0] = 1$
- următoarele poziții ce se vor compara:  $i = 4, k = 0$  (se incrementează cu 1 atât  $i$  cât și  $k$ )

#### Exemplu 4/6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$a$	$b$	$c$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$c$	$a$
			$=$	$=$	$=$	$=$	$=$	$\neq$						
			$a$	$b$	$a$	$b$	$a$	$c$	$a$					
			0	1	2	3	4	5	6					

- eșec la pozițiile  $i = 9, k = 5$
- $f[5] = 3$
- se face un salt egal cu  $k - f[k] = 5 - f[5] = 2$
- următoarele poziții ce se vor compara:  $i = 9, k = 3$  ( $k$  devine  $f[k]$ )

#### Exemplu 5/6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$a$	$b$	$c$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$b$	$a$	$c$	$a$
			$=$	$=$	$=$	$=$	$=$	$\neq$						
			$a$	$b$	$a$	$b$	$a$	$c$	$a$					
			0	1	2	3	4	5	6					

- eșec la pozițiile  $i = 11, k = 5$
- $f[5] = 3$
- se face un salt egal cu  $k - f[k] = 5 - f[5] = 2$
- următoarele poziții ce se vor compara:  $i = 11, k = 3$  ( $k$  devine  $f[k]$ )

### Exemplu 6/6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	b	c	b	a	b	a	b	a	b	a	b	a	c	a
								=	=	=	=	=	=	=
								a	b	a	b	a	c	a
								0	1	2	3	4	5	6

- s-a găsit prima apariție

### Algoritmul KMP în Alk

```

KMP(s, n, t, m, f) {
    i = 0;
    k = 0;
    while (i < n) {
        while (k != -1) && (p[k] != s[i])
            k = f[k];
        if (k == m-1)
            return i-m+1; /* gasit p in s */
        else {
            i = i+1;
            k = k+1;
        }
    }
    return -1; /* p nu apare in s */
}

```

### Timpul de execuție

Reamintim funcția eșec pentru exemplul precedent:

0	1	2	3	4	5	6
a	b	a	b	a	c	a
-1	0	0	1	2	3	0

#### 1. Observații:

- pentru orice  $k$ ,  $-1 \leq f[k] < k$ .
- valoarea lui  $k$  va crește de cel mult  $n$  ori (ca și  $i$ )
- la fiecare iterație while interioară  $k$  descrește, dar va fi  $\geq -1$
- per total,  $k$  nu va putea descrește de mai multe ori de câte ori crește
- deci while interior va face cel mult  $n$  iterații în total

#### 2. Concluzie: timpul de execuție pentru KMP este $O(n)$

### Funcția eșec $f$ : introducere

- deoarece  $f$  este utilizată atunci când o comparație eșuează,  $f$  se numește și funcție eșec (failure function)
- notată și cu  $\pi$  (de exemplu în [CLR])

- reamintim că  $f[i] = |maxFr p[0..i-1]|$  (lungimea frontierei maxime a lui  $p[0..i-1]$ )

$a$	$b$	$a$	$b$	$a$	$c$	$a$
-1	0	0	1	2	3	0

O implementare naivă poate avea complexitatea  $O(m^3)$  (exercițiu pentru acasă).

Dacă presupunem că  $f[0..i-1]$  a fost deja calculat, cum calculăm eficient  $f[i]$ ?

### Funcția eșec $f$ : domeniul problemei

- notatie:  $u \leq_{fr} v$  ddacă  $u \leq_{pref} v$  și  $u \leq_{suff} v$
- definiția formală a lui  $maxFr(v)$   
 $maxFr(v) <_{fr} v$   
 $(\forall w) w <_{fr} v$  implică  $w \leq_{fr} maxFr(v)$
- notatie:  $maxFr^0(v) = v$ ,  $maxFr^{i+1}(v) = maxFr(maxFr^i(v))$
- avem:  
 $maxFr^{i+1}(v) <_{fr} maxFr^i(v) <_{fr} \dots <_{fr} maxFr^1(v) maxFr^0(v) = v$

**Theorem 1.**  $u \leq_{fr} v$  ddacă există  $i \geq 0$  a.î.  $u = maxFr^i(v)$ .

**Corollary 2.**  $u <_{fr} v$  ddacă există  $i > 0$  a.î.  $u = maxFr^i(v)$ .

### Funcția eșec $f$ : calcul

- reamintim că  $f[i] = |maxFr(p[0..i-1])|$
- rezultă că  $f[i] = f^k[i-1] + 1$ , unde  $k$  este cel mai mic întreg cu proprietatea  $p[f^k[i-1] + 1] = p[i]$
- adică ne uităm la prefixele lui  $p$  care sunt sufixe ale lui  $p[0..i-2]$  și-l luăm pe cel mai mare cu proprietatea că următorul caracter coincide cu  $p[i-1]$ , unde
- prefixele lui  $p$  care sunt sufixe ale lui  $p[0..i-2]$ :  $maxFr(p[0..i-2])$ ,  $maxFr^2(p[0..i-2])$ ,  $maxFr^3(p[0..i-2])$ , și
- $|maxFr^k(p[0..i-2])| = f^k[i-1]$

Observație:

$$\begin{aligned}
 |maxFr^2(p[0..i-1])| &= |maxFr(maxFr(p[0..i-1]))| \\
 &= maxFr(p[0..f[i]-1]) \\
 &= f[f[i]] \\
 &= f^2[i] \\
 |maxFr^3(p[0..i-1])| &= |maxFr(maxFr^2(p[0..i-1]))| \\
 &= maxFr(p[0..f^2[i]-1]) \\
 &= f[f^2[i]] \\
 &= f^3[i] \\
 &\dots
 \end{aligned}$$

### Calculul funcției eșec: reprezentarea în Alk

```

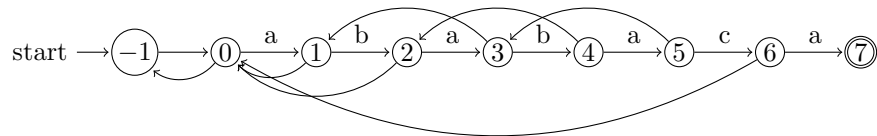
f[0] = -1;
k = -1;
for (i = 1; i <= m; ++i) {
    while(k >= 0 && p[k+1] != p[i])
        k = f[k];
    k = k + 1;
    f[i] = k;
}

```

Timp de execuție:  $\Theta(m)$ .

Analiza e similară cu cea de la KMP.

### Funcția eșec reprezentată ca un automat



Un automat este format din:

- alfabet de intrare  $(a, b, c)$
- stări  $(-1, 0, 1, \dots, 7)$
- starea inițială  $(-1)$
- stare finală/acceptare  $(7)$
- tranziții spontane:  $(-1 \rightarrow 0, 0 \rightarrow -1, 1 \rightarrow 0, \dots)$
- tranziții etichetate:  $(0 \xrightarrow{a} 1, 1 \xrightarrow{b} 2, 2 \xrightarrow{a} 3, \dots)$

### 3 Expresii regulate

Motivație: pattern-uri în Emacs (sau alt editor similar)

Din documentație:

Pattern	Matches
.	Any single character except newline ("n").
\.	One period
[0-9]+	One or more digits
[^ 0-9]+	One or more non-digit characters
[A-Za-z]+	one or more letters
[-A-Za-z0-9]+	one or more letter, digit, hyphen
[_A-Za-z0-9]+	one or more letter, digit, underscore
[-_A-Za-z0-9]+	one or more letter, digit, hyphen, underscore
[[:ascii:]]+	one or more ASCII chars. (codepoint 0 to 127, inclusive)
[[:nonascii:]]+	one or more none-ASCII characters (For example, Unicode characters)
[\n\t ]+	one or more {newline character, tab, space}.

[lex]

Demo cu Emacs

#### Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație "*pattern*" *generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

**Definiție 1.** Mulțimea expresiilor regulate peste alfabetul  $\Sigma$  este definită recursiv astfel:

- $\epsilon$ , empty sunt expresii regulate
- orice caracter din  $\Sigma$  este o expresie regulată;
- dacă  $e_1, e_2$  sunt expresii regulate, atunci  $e_1e_2$  și  $e_1 + e_2$  sunt expresii regulate;
- dacă  $e$  este expresie regulată, atunci  $(e)$  și  $e^*$  sunt expresii regulate.

Arborele sintactic abstract: pe tabla.

#### Legătura cu pachetul <regex> din C++, Emacs

<regex>	expresia regulata
[abc]	a + b + c
\d sau [[:digit:]]	0 + 1 + ... + 9
[[:digit:]]*	(0 + 1 + ... + 9)*
[[:digit:]]+	(0 + 1 + ... + 9)(0 + 1 + ... + 9)*

#### Limbaajul definit de o expresie regulată

**Definiție 2.** Mulțimea de șiruri (limbaajul)  $L(e)$  definit de o expresie regulată  $e$  este definit recursiv astfel:

- $L(\epsilon) = \{\epsilon\}$  ( $\epsilon$  este șirul vid (de lungime zero)),  $L(empty) = \emptyset$
- dacă  $e$  este un caracter atunci  $L(e) = \{e\}$ ;
- dacă  $e = e_1e_2$  atunci  $L(e) = L(e_1)L(e_2) = \{w_1w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$ ;
- dacă  $e = e_1 + e_2$  atunci  $L(e) = L(e_1) \cup L(e_2)$ ;
- dacă  $e = e_1^*$  atunci  $L(e) = \cup_k L(e_1^k)$ , unde  $L(e_1^0) = \{\epsilon\}$ ,  $L(e_1^{k+1}) = L(e_1^k)L(e_1)$ ;



- dacă  $e = (e_1)$  atunci  $L(e) = L(e_1)$ .

**Exemplu:** Fie alfabetul  $A = \{a, b, c\}$ . Avem  $L(a(b + a)c) = \{abc, aac\}$  și  $L((ab)^*) = \{\varepsilon, ab, abab, ababab, \dots\} = \{(ab)^k \mid k \geq 0\}$ . sfex

### Căutare cu expresii regulate

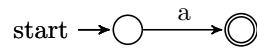
*Input* Un text  $s$ , un pattern  $p$  exprimat ca o expresie regulată. [2ex] *Output:* Prima apariție a unui șir din limbajul definit de expresia regulată [3ex]

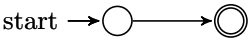
Algoritmul de căutare utilizează un automat asociat patternului, similar ca la KMP.

### Automatul asociat unei expresii regulate

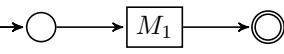
– cazul de bază

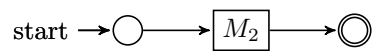
$e$  este o litera (un simbol)  $a \in \Sigma$



$e$  este  $\varepsilon$  start  $\rightarrow$  

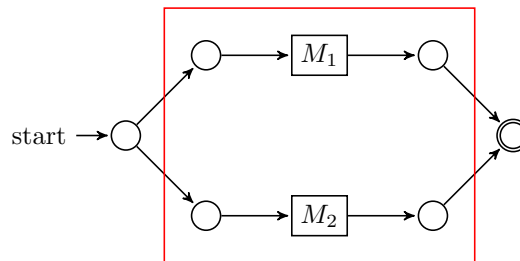
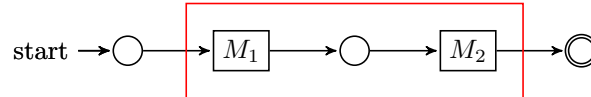
$e$  este empty start  $\rightarrow$  

pentru cazul inductiv presupunem: start  $\rightarrow$  



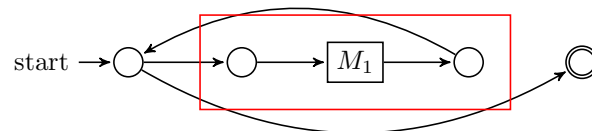
### Automatul asociat unei expresii regulate

$e = e_1 e_2$ :



$e = e_1 + e_2$ :

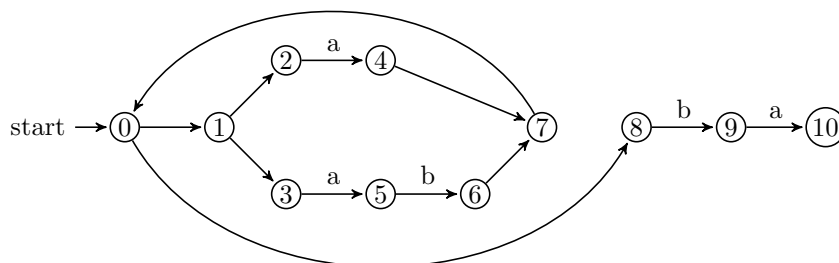
### Automatul asociat unei expresii regulate



$e = e_1^*$ :

### Exemplu

$$e = (a + ab)^*ba$$



*Detaliile procesului de construcție pe tablă*

### Utilizarea automatului în căutare

*Pe tablă*

### Construcții mai performante

- automatului Brzozowski (1964)
- construcția unui automat determinist
- utilizând funcțiile **first** și **follow** (Berry, Setti, 1986)
- paralelizare (Myer, A Four Russians Algorithm for Regular Expression Pattern Matching)
- o altă construcție pentru automatul nedeterminist este Glushkov-McNaughton-Yamada (1960-1961), care poate fi și paralelizată (Navarro & Raffinot, 2004)

Mai multe detalii despre expresii regulate și automatele lor la cursul LFAC din anul II.

### Complexitatea căutării cu expresii regulate

Presupunem că lungimea expresiei regulate este  $m$  (numărul de caractere fără operatori) și  $m_\Sigma = |\Sigma \cup \{., +, *\}|$ .

**Theorem 3** (Thomson, 1968). *Problema căutării cu expresii regulate poate fi rezolvată în timpul  $O(mn)$  cu automate nedeterministe și spațiu  $O(m)$ .*

**Theorem 4** (Kleene, 1956). *Problema căutării cu expresii regulate poate fi rezolvată în timpul  $O(n + 2^{m_\Sigma})$  cu automate deterministe și spațiu  $O(2^{m_\Sigma})$ .*

**Theorem 5** (Myers, 1992). *Problema căutării cu expresii regulate poate fi rezolvată în timpul  $O(mn/\log n)$  cu automate deterministe și spațiu  $O(mn/\log n)$ .*