

Aufln 18 & 19



# String Matching

- Input:

- Array com o texto:  $T[1..n]$
- Array com o padrão a encontrar:  $P[1..m]$

- Output:

- Shift Válido (valid shift): s

$$T[s+1] = P[1]$$

:

$$T[s+m] = P[m]$$

Exemplo:

Texto T: a b c a b a a b c a b a c

Padrão P: a b a a

↓

a b c a b a a b c a b a c

a b a a

valid shift: 3

## Algoritmo Naive

Naive String Matching ( $T, P$ )

let  $n = T.size$

let  $m = P.size$

for  $s=0$  to  $n-m$

| let  $found = \text{true}$

| for  $i=1$  to  $m$

| | if  $T[i+s] \neq P[i]$

| | |  $found = \text{false}$ ; break

| | if  $found$

| | print "O padrão ocorre q shift" + s

Complexidade:

$O(n \cdot m)$

Nº de iterações:  $((n-m)+1) \cdot m$

## Naïve String Matcher - Exemplo

Naïve String Matching ( $T, P$ )

let  $n = T.size$

let  $m = P.size$

for  $s=0$  to  $n-m$

: let found = true

: for  $i=1$  to  $m$

: : if  $T[i+s] \neq P[i]$

: : found = false; break

: if found

: print "O padrão ocorre q shift" + s

\*  $s=1$      $T: a c a a b c$

$P: a \overset{?}{\underset{3}{a}} b$

\*  $s=2$      $T: a c a a b c$

$P: \overset{?}{a} a b$

Estamos a comparar

c com a uma segunda vez  
desnecessariamente

\*  $s=3$      $T: a c a a b c$     ✓

$P: a a b$

Ideia: Usar pré-processamento

para evitar comparações desnecessárias.

## Algoritmo de Rabin-Karp - Simplificado

Text: a a a a a b  
n=6

Pattern: a a b  
m=3

a - 1  
b - 2  
c - 3  
d - 4

Associamos a cada símbolo do alfabeto  
de trabalho  $\Sigma$  um número único

\* Sliding Hash:

$$h(i+1) = h(i) - c(T[i]) + c(T[i+m])$$

hash da janela  $T[i+1, \dots, i+m]$

- Consegrimos associar o pedaço a um código?

h: Pattern  $\rightarrow$  Cade

- Vamos tentar:  $h(P[1..m]) = c(P[1]) + \dots + c(P[m])$

$$\begin{aligned} h(aab) &= c(a) + c(a) + c(b) \\ &= 1 + 1 + 2 \\ &= 4 \end{aligned}$$

① a a a a a b

③ a a a a a a b

② a a a a a ab

④ a a a a ab

Hatch

## Algoritmo de Rabin-Karp - Simplificado

Text:  $c \underset{n=6}{\underset{1 \ 2 \ 3 \ 4 \ 5}{|}} c \underset{6 \ 7 \ 8 \ 9 \ 10 \ 11}{|} a \underset{|}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$

Pattern:  $d \underset{n=3}{|} b \underset{|}{|} a$

$$h(P) = 4 + 2 + 1 = 7$$

$a - 1$   
 $b - 2$   
 $c - 3$   
 $d - 4$   
 $e - 5$

} Associamos a cada símbolo do alfabeto  
de trabalho  $\Sigma$  um número único

$$\textcircled{1} \quad c \underset{3+3+1=7}{|} c \underset{|}{|} a \underset{|}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$$

spurious hit

$$\textcircled{7} \quad c \underset{7-1+4=10}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$$

X

$$\textcircled{2} \quad c \underset{7-3+3=7}{|} \textcircled{O} \underset{|}{|} a \underset{|}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$$

SA

$$\textcircled{8} \quad c \underset{10-1+2=11}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$$

X

$$\textcircled{3} \quad c \underset{7-3+3=7}{|} c \underset{|}{|} \textcircled{O} \underset{|}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$$

SA

$$\textcircled{9} \quad c \underset{11-5+1=7}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$$

$$\textcircled{4} \quad c \underset{7-1+1=7}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$$

SA

✓ Hit

$$\textcircled{5} \quad c \underset{7-3+1=5}{|} c \underset{|}{|} c \underset{|}{|} a \underset{|}{|} a \underset{|}{|} e \underset{|}{|} d \underset{|}{|} b \underset{|}{|} a$$

X

\* Sliding Hash:

$$h(i+1) = h(i) - C(T[i]) + C(T[i+m])$$

↳ hash da janela  $T[i+1, \dots, i+m]$

## Algoritmo de Rabin-Karp - Simplificado

Text:  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$   
 $n=6$

Pattern:  $d \ b \ a$

$$n=3$$

$$h(P) = 4 + 2 + 1 = 7$$

$a - 1$   
 $b - 2$   
 $c - 3$   
 $d - 4$   
 $e - 5$

Associamos a cada símbolo do alfabeto  
 de trabalho  $\Sigma$  um número único

$$3+3+1 = 7 \text{ (Spurious Match)}$$

①  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

$$7-3+3 = 7 \text{ (Spurious Match)}$$

②  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

$$7-3+3 = 7 \text{ (Spurious Match)}$$

③  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

$$7-1+1 = 7 \text{ (Spurious Match)}$$

④  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

$$7-3+1 = 5 \text{ (Not a match)}$$

⑤  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

⑥  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

$$5-3+5 = 7 \text{ (Spurious Match)}$$

⑦  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$   $\uparrow$   
 $7-1+4 = 10$   
 (Not a match)

⑧  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$   $\uparrow$   
 $10-1+2 = 11$   
 (Not a match)

⑨  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$   $\uparrow$   
 $11-5+1 = 7$   
 (Real match)

## Algoritmo de Rabin-Karp - Simplificado<sup>2</sup>

Text:  $c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a$   
 $n=6$

Pattern:  $d \ b \ a$   
 $m=3$

a - 1  
 b - 2  
 c - 3  
 d - 4  
 e - 5

} Associamos a cada símbolo do alfabeto  
 do trabalho  $\Sigma$  um número único

\* Como colisões de hashes?

$$\begin{aligned}
 & -dba \rightarrow \\
 & C(d) \times \beta^2 + C(b) \times \beta^1 + C(a) \times \beta^0 \\
 & = 4 \times 5^2 + 2 \times 5 + 1 \\
 & = 4 \times 25 + 10 + 1 \\
 & = 111
 \end{aligned}$$

$$\begin{aligned}
 \beta : & \underbrace{\Sigma} \\
 & \hookrightarrow \text{tamanho do alfabeto}
 \end{aligned}$$

\* Função de Hash

$$\begin{aligned}
 h(P[1..m]) &= C(P[1]) \cdot \beta^{m-1} \times C(P[2]) \cdot \beta^{m-2} \times \dots \times C(P[m]) \cdot \beta^0 \\
 &= \sum_{i=1}^m C(P[i]) \cdot \beta^{m-i}
 \end{aligned}$$

## Algoritmo de Rabin-Karp - Simplificado<sup>2</sup>

### \* Função de hash

$$h(P[1..m]) = C(P[1]) \cdot \beta^{m-1} + C(P[2]) \cdot \beta^{m-2} + \dots + C(P[m]) \cdot \beta^0$$
$$= \sum_{i=1}^m C(P[i]) \cdot \beta^{m-i}$$

### \* Hash Deslizante

$$h(i+1) = (h(i) - \beta^{m-1} \cdot C(T[i])) \cdot \beta + C(T[i+m])$$

↳ Hash da janela  $T[i+1, \dots, i+m]$

Observação: No livro/slides da cadeira a fórmula é diferente porque se assume que que  $h(i)$  é o hash da janela  $\bar{q}$  comes na posição i+1. A adaptação é trivial.

## Algoritmo de Rabin-Karp - Simplificado 2

Text:  $\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{matrix}$   
 $n=6$

Pattern:  $\begin{matrix} d & b & a \end{matrix}$   
 $n=3$   
 $h(P) = 421$

- ①  $\begin{matrix} 331 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix} \quad \text{X}$   
 $\uparrow$   
 $(331 - 300) \times 10 + 3 = 310 + 3 = 313$
- ②  $\begin{matrix} 313 \\ c & \textcircled{0} & a & c & c & a & a & e & d & b & a \end{math} \quad \text{X}$   
 $\uparrow$   
 $(313 - 300) \times 10 + 3 = 130 + 3 = 133$
- ③  $\begin{matrix} 133 \\ c & c & \textcircled{a} & c & c & a & a & e & d & b & a \end{math} \quad \text{X}$   
 $\uparrow$   
 $(133 - 300) \times 10 + 1 = 330 + 1 = 331$
- ④  $\begin{matrix} 331 \\ c & c & a & \textcircled{0} & c & a & a & e & d & b & a \end{math} \quad \text{X}$   
 $\uparrow$   
 $(331 - 300) \times 10 + 1 = 310 + 1 = 311$
- ⑤  $\begin{matrix} 311 \\ c & c & a & c & c & a & a & e & d & b & a \end{math} \quad \text{X}$   
 $\uparrow$   
 $(311 - 300) \times 10 + 5 = 110 + 5 = 115$
- ⑥  $\begin{matrix} 115 \\ c & c & a & c & c & a & a & e & d & b & a \end{math} \quad \text{X}$   
 $\uparrow$

$a - 1$   
 $b - 2$   
 $c - 3$   
 $d - 4$   
 $e - 5$   
 $\dots$   
 $j - 10$

Associamos a cada símbolo do alfabeto

de trabalho  $\Sigma$  um número único

$$(115 - 100) \times 10 + 4 = 150 + 4 = 154$$

⑦  $\begin{matrix} 154 \\ c & c & a & c & c & a & a & e & d & b & a \end{math} \quad \text{X}$

$$(154 - 100) \times 10 + 2 = 540 + 2 = 542$$

⑧  $\begin{matrix} 542 \\ c & c & a & c & c & a & a & e & d & b & a \end{math} \quad \text{X}$

$$(542 - 500) \times 10 + 1 = 420 + 1 = 421$$



## Algoritmo de Rabin-Karp - Simplificado 2

Text:  $\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{matrix}$   
 $n=6$

Pattern:  $\begin{matrix} d & b & a \end{matrix}$   
 $n=3$   
 $h(P) = 421$

①  $\begin{matrix} 331 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$a - 1$   
 $b - 2$   
 $c - 3$   
 $d - 4$   
 $e - 5$   
 $\dots$   
 $j - 10$

Associamos a cada símbolo do alfabeto  
de trabalho  $\Sigma$  um número único

②  $\begin{matrix} (331 - 300) \times 10 + 3 = 31 \times 10 + 3 = 313 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

③  $\begin{matrix} (313 - 300) \times 10 + 3 = 13 \times 10 + 3 = 133 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

④  $\begin{matrix} (133 - 300) \times 10 + 1 = 31 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

⑤  $\begin{matrix} (31 - 300) \times 10 + 1 = 31 \times 10 + 1 = 311 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

⑥  $\begin{matrix} (311 - 300) \times 10 + 5 = 11 \times 10 + 5 = 115 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$(115 - 100) \times 10 + 4 = 15 \times 10 + 4 = 154$$

⑦  $\begin{matrix} c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$(154 - 100) \times 10 + 2 = 54 \times 10 + 2 = 542$$

⑧  $\begin{matrix} c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$(542 - 500) \times 10 + 1 = 421$$

⑨  $\begin{matrix} c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

## Algoritmo de Rabin-Karp - Simplificado 2

Text:  $\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{matrix}$   
 $n=6$

Pattern:  $\begin{matrix} d & b & a \end{matrix}$   
 $n=3$   
 $h(P) = 421$

①  $\begin{matrix} 331 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$\uparrow \quad (331 - 300) \times 10 + 3 = 31 \times 10 + 3 = 313$$

②  $\begin{matrix} 313 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$\uparrow \quad (313 - 300) \times 10 + 3 = 13 \times 10 + 3 = 133$$

③  $\begin{matrix} 133 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$\uparrow \quad (133 - 300) \times 10 + 1 = 331$$

④  $\begin{matrix} 331 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$\uparrow \quad (331 - 300) \times 10 + 1 = 31 \times 10 + 1 = 311$$

⑤  $\begin{matrix} 311 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$\uparrow \quad (311 - 300) \times 10 + 5 = 11 \times 10 + 5 = 115$$

⑥  $\begin{matrix} 115 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

a - 1  
b - 2  
c - 3  
d - 4  
e - 5  
...  
j - 10

} Associamos a cada símbolo do alfabeto  
de trabalho  $\Sigma$  um número único

Problema: Overflow of os códigos de hash.

$$(115 - 100) \times 10 + 4 = 15 \times 10 + 4 = 154$$

⑦  $\begin{matrix} 154 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$(154 - 100) \times 10 + 2 = 54 \times 10 + 2 = 542$$

⑧  $\begin{matrix} 542 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

$$\uparrow \quad (542 - 500) \times 10 + 1 = 421$$

⑨  $\begin{matrix} 421 \\ c & c & a & c & c & a & a & e & d & b & a \end{matrix}$

## Algoritmo de Rabin-Karp

- Função de hash utilizada no Rabin-Karp

$$h(i+1) = (h(i) - \beta^{m-1} \times \ell(T[i])) \times \beta + \ell(T[i+m])$$



$$h(i+1) = (h(i) - \beta^{m-1} \times \ell(T[i])) \times \beta + \ell(T[i+m]) \quad \text{mod } q$$

} N° primo  $q$  deve ser escolhido  
tendo em conta o tipo de dados  
 $q$  estaremos a usar para guardar  
os hashes.

- Complexidade:

- Melhor caso (Não há spurious hits)

$$\mathcal{O}(n)$$

→ N° de comparações:  $n - m + 1$

- Pior caso (seja todos spurious hits)

$$\mathcal{O}(n \cdot m)$$

→ N° de comparações:  $(n - m + 1) \times m$

## String Matching com Autômatos Finitos

\* Ideia: transformar o padrão num autômato finito determinístico (DFSM) e usar o autômato para calcular os matches.

Revisão: Autômatos finitos determinísticos

$$D = (Q, q_0, F, \Sigma, S)$$

- $Q$  - conjunto de estados
- $q_0$  - estado inicial
- $F$  - conjunto de estados finais
- $\Sigma$  - alfabeto
- $S$  - função de transição

$$S: Q \times \Sigma \rightarrow Q$$

# String Matching com Autômatos Finitos

\* Ideia: transformar o padrão num autômato finito determinístico (DFSM) e usar o autômato para calcular os matches.

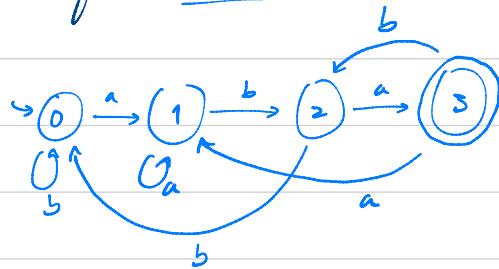
Revisão: Autômatos finitos determinísticos

$$D = (Q, q_0, F, \Sigma, S)$$

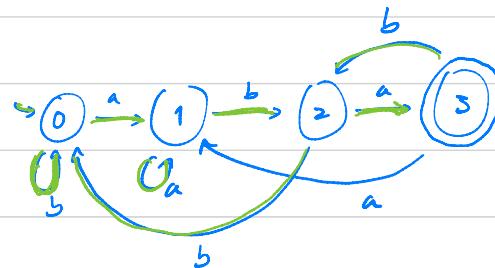
- $Q$  - conjunto de estados
- $q_0$  - estado inicial
- $F$  - conjunto de estados finais
- $\Sigma$  - alfabeto
- $S$  - função de transição

$$S: Q \times \Sigma \rightarrow Q$$

Exemplo: aba



Texto: a bba bbb a ababaa



# String Matching com Autômatos Finitos

\* Ideia: transformar o padrão num autômato finito determinístico (DFSM) e usar o autômato para calcular os matches.

Revisão: Autômatos finitos determinísticos

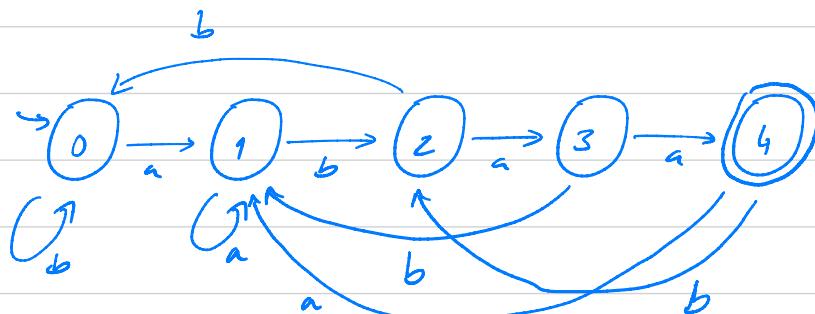
$$D = (Q, q_0, F, \Sigma, S)$$

- $Q$  - conjunto de estados
- $q_0$  - estado inicial
- $F$  - conjunto de estados finais
- $\Sigma$  - alfabeto
- $S$  - função de transição

$$S: Q \times \Sigma \rightarrow Q$$

Exemplo:

Pat: ab a a



## String Matching com Autômatos Finitos

① Ideia: transformar o padrão num autômato finito determinístico (DFSM) e usar o autômato para calcular os matches.

② Como construir o autômato de forma automática?

- Função de Sufixo [Suffix Function]

$$\sigma(w) = \max \{ i \mid P_i \supseteq w \}, \text{ onde } P_i \text{ o padrão}$$

↳  $\sigma(w)$  é o tamanho do maior prefíxo de  $P$  que é também um sufixo de  $w$

Exemplo:

$$\bullet P = abaa$$

$$\sigma(abab) = 2$$

$$\bullet P = abaa$$

$$\sigma(abaaa) = 1$$

$$\bullet P = abaa$$

$$\sigma(abaab) = 2$$

## String Matching com Autômatos Finitos

\* Ideia: transformar o padrão num autômato finito determinístico (DFSM) e usar o autômato para calcular os matches.

\* Como construir o autômato de forma automática?

- Dado um padrão  $P[1..m]$  construímos um DFSM

da seguinte maneira:

$$\cdot Q = \{0, 1, \dots, m\}$$

$$\cdot f_0 = 0$$

$$\cdot F = \{m\}$$

$$\cdot \Sigma$$

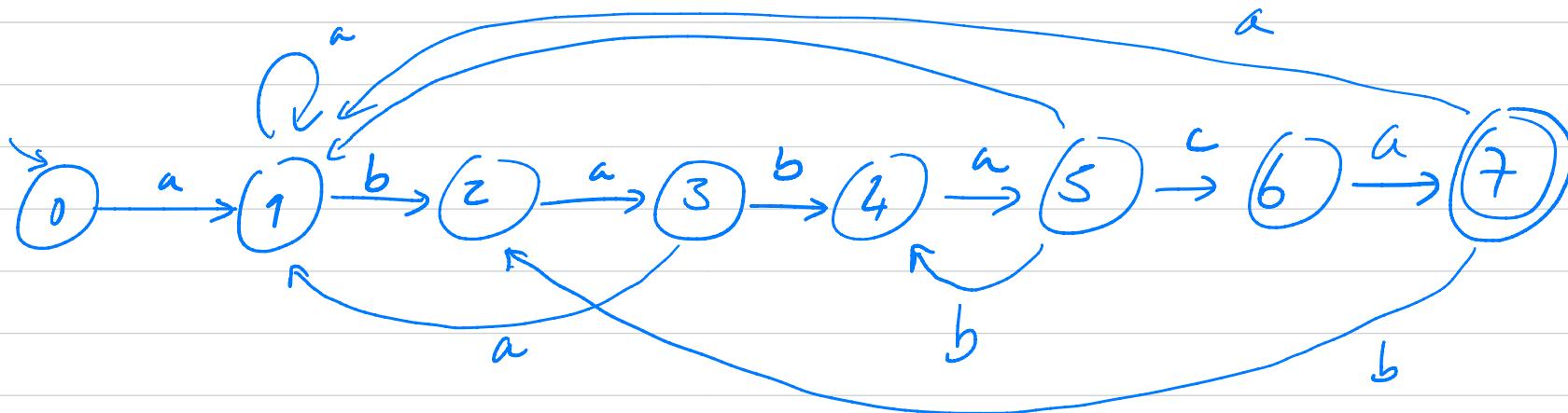
$$\cdot \delta: Q \times \Sigma \rightarrow Q$$

$$\delta(i, x) = \delta(P_i x)$$

# String Matching com Autômatos Finitos

Exemplo:

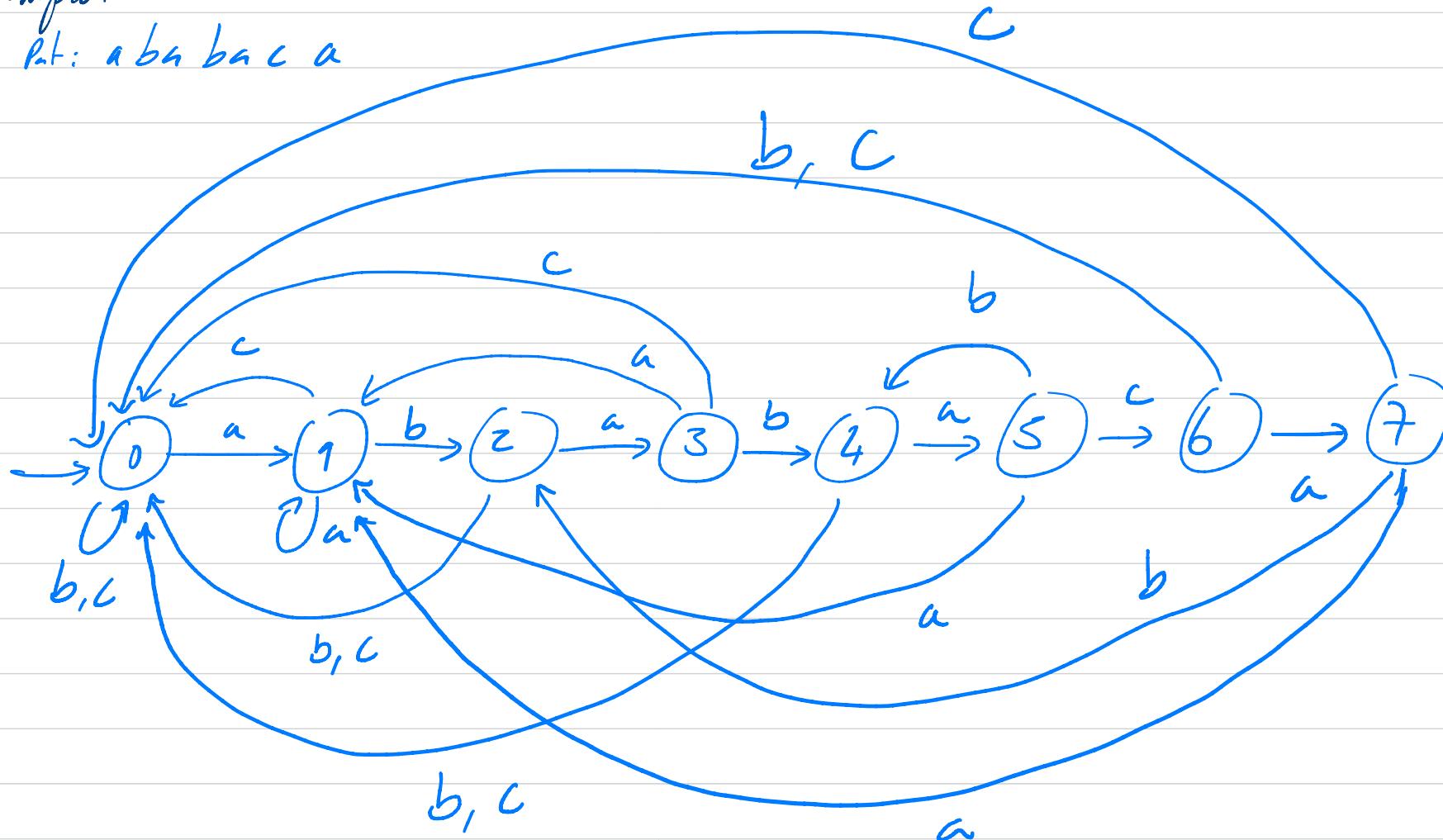
pt: ababaca a



# String Matching com Autômatos Finitos

Exemplo:

Pat: ababac a



## Propriedades de Sufixo e Função de Sufixo

\* Propriedade 1

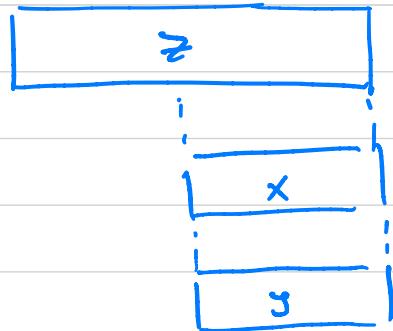
$$x \sqsupseteq z \wedge y \sqsupseteq z \Rightarrow \begin{cases} |x| = |y| \Rightarrow x = y & (a) \\ |x| < |y| \Rightarrow x \sqsubset y & (b) \\ |y| < |x| \Rightarrow y \sqsubset x & (c) \end{cases}$$

## Propriedades de Sufixo e Função de Sufixo

\* Propriedade 1

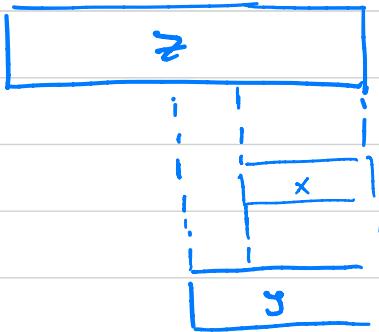
$$x \sqsupseteq z \wedge y \sqsupseteq z \Rightarrow \begin{cases} |x| = |y| \Rightarrow x = y & (a) \\ |x| < |y| \Rightarrow x \sqsubset y & (b) \\ |y| < |x| \Rightarrow y \sqsubset x & (c) \end{cases}$$

a)



$$x = y$$

b)



$$x \sqsubset y$$

## Propriedades de Sufixo e Função de Sufixo

### \* Propriedade 1

$$x \sqsupseteq z \wedge y \sqsupseteq z \Rightarrow \begin{cases} |x| = |y| \Rightarrow x = y & (a) \\ |x| < |y| \Rightarrow x \sqsubset y & (b) \\ |y| < |x| \Rightarrow y \sqsubset x & (c) \end{cases}$$

### \* Propriedade 2

$$\sigma(xa) \leq \sigma(x) + 1$$

$$\sigma(xa) = i$$

## Propriedades de Sufixo e Função de Sufixo

### \* Propriedade 1

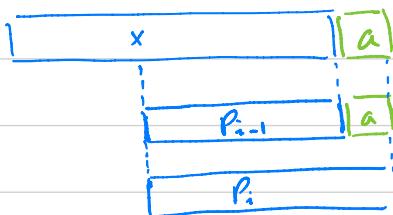
$$x \sqsupseteq z \wedge y \sqsupseteq z \Rightarrow \begin{cases} |x| = |y| \Rightarrow x = y & (a) \\ |x| < |y| \Rightarrow x \sqsubset y & (b) \\ |y| < |x| \Rightarrow y \sqsubset x & (c) \end{cases}$$

### \* Propriedade 2

$$\sigma(xa) \leq \sigma(x) + 1$$

$$\sigma(xa) = i$$

.



- $p_{i-1}$  é um sufixo de  $x$ ; da definição de  $\sigma(x)$  segue que  $i-1 \leq \sigma(x)$ .



$$\sigma(xa) - 1 \leq \sigma(x)$$

$$\Leftrightarrow \sigma(xa) \leq \sigma(x) + 1$$

## Propriedades de Sufixo e Função de Sufixo

### \* Propriedade 1

$$x \sqsupseteq z \wedge y \sqsupseteq z \Rightarrow \begin{cases} |x| = |y| \Rightarrow x = y & (a) \\ |x| < |y| \Rightarrow x \sqsubset y & (b) \\ |y| < |x| \Rightarrow y \sqsubset x & (c) \end{cases}$$

### \* Propriedade 2

$$\sigma(xa) \leq \sigma(x) + 1$$

b) Seja  $i = \sigma(xa)$

$$\cdot p_i \sqsupseteq xa$$

$$\cdot p_i \sqsupseteq x \Rightarrow p_i.a \sqsupseteq xa$$

$$\cdot i \leq q+1 \quad (\text{Propriedade } 2)$$

$$\cdot |p_i| \leq |p_{q+1}|$$

$$\cdot p_i \sqsupseteq p_{q+1}a \quad (\text{Propriedade 1})$$

$$\sigma(p_i) \leq \sigma(p_{q+1}a)$$

$$i = \sigma(xa) \leq \sigma(p_{q+1}a)$$

### \* Propriedade 3

$$q = \sigma(x) \Rightarrow \sigma(xa) = \sigma(p_q a)$$

$$a) \sigma(p_q a) \geq \sigma(xa)$$

$$b) \sigma(xa) \leq \sigma(p_q a)$$

a)  $q = \sigma(x) \Rightarrow p_q \sqsupseteq x$

$$\Rightarrow p_q.a \sqsupseteq x.a$$

$$\Rightarrow \sigma(p_q a) \leq \sigma(xa)$$

# String Matching com Automatos Finitos

Finite Automaton Matcher ( $T, \delta, m$ )

let  $n = T.length$

$j = 0$

for  $i = 1$  to  $n$

$j = \delta(j, T[i])$

if  $j == m$

Print "Match with shift" (i-m)

Complexidade:

$\Theta(m)$

Completa Transição Funcional ( $P, \Sigma$ )

$m = P.length$

for  $j = 0$  to  $m$

for each  $a \in \Sigma$

$k = \min(m+1, j+2)$

repeat

$k = k - 1$

until  $p_k \neq p_j.a$

$\delta(j, a) = k$

return  $j$

Complexidade:

$O(m^3 |\Sigma|)$

=

↓ Algoritmos mais eficientes

$O(m |\Sigma|)$

# Algoritmo de Knuth-Morris-Pratt

## • Fazendo de Prefixo

- Dado um padrão  $P[1..n]$

$$\pi: \{1, 2, \dots, m\} \rightarrow \{0, 1, \dots, m-1\}$$

$$\pi[i] = \max \left\{ k : \underline{k < i} \wedge P_k \sqsupseteq P_i \right\}$$

Tamanho do maior prefixo de  $P$  que também é sufixo de  $P_i$

Observação: No fundo a função de prefixo é uma resposta à seguinte pergunta:

- Em quais posições é que o inicio do padrão volta a aparecer dentro do próprio padrão?

## Exemplos:

I) abcdeabefabc  
0000120120

III) aabcaadababe  
0100101230

IV) aaaabaaacd  
012301200

II) abcdeabfabca  
00000120123

V) ababababca  
0012345601

# Algoritmo de Knuth-Morris-Pratt

## • Fazendo de Prefixo

- Dado um padrão  $P[1..n]$

$$\pi : \{1, 2, \dots, m\} \rightarrow \{0, 1, \dots, m-1\}$$

$$\pi[i] = \max \left\{ k : k < i \wedge P_k \sqsupseteq P_i \right\}$$

Tamanho do maior prefixo de  $P$  que também é sufixo de  $P_i$

Observação: No fundo a função de prefixo é uma resposta à seguinte pergunta:

- Em quais posições é que o inicio do padrão volta a aparecer dentro do próprio padrão?

## Exemplos:

I) abc<sub>0</sub>dabc<sub>1</sub>eab<sub>2</sub>f<sub>3</sub>

0 0 0 1 2 0 1 2 0

III) aabc<sub>0</sub>c<sub>1</sub>da<sub>2</sub>a<sub>3</sub>abe<sub>4</sub>  
0 1 0 0 1 0 1 1 2 0

IV) aa<sub>0</sub>a<sub>1</sub>b<sub>2</sub>a<sub>3</sub>a<sub>4</sub>c<sub>5</sub>d<sub>6</sub>  
0 1 2 3 0 1 2 0 0

II) abcdeabfabc

0 0 0 0 0 1 2 0 1 2 3

V) ababababca  
0 0 1 2 3 4 5 6 0 1

## Algoritmo de Knuth-Morris-Pratt

$$\pi[i] = \max \left\{ k : k < i \wedge P_k \sqsupseteq P_i \right\}$$

Complexidade

$O(m)$

Compute Prefix Function ( $\rho$ )

let  $m = P.length$

let  $\pi[1..m]$  be a new array

$\pi[1] = 0$

$k = 0$

for  $i = 2$  to  $m$

  while ( $k > 0$  and  $P[k+1] \neq P[i]$ )

$k = \pi[k]$

    if  $P[k+1] == P[i]$

$k = k + 1$

$\pi[i] = k$

## Algoritmo de Knuth-Morris-Pratt

$$\pi[i] = \max \left\{ k : k < i \wedge P_k \sqsupseteq P_i \right\}$$

Compute Prefix Function ( $\rho$ )

let  $m = P.length$

let  $\pi[1..m]$  be a new array

$\pi[1] = 0$

$k = 0$

for  $i = 2$  to  $m$

. while ( $k > 0$  and  $P[k+1] \neq P[i]$ )

. .  $k = \pi[k]$

. if  $P[k+1] == P[i]$

. .  $k = k + 1$

.  $\pi[i] = k$

### Complexidade

Observação 1:  $k$  é incrementado no máximo  $(m-1)$  vezes.

Observação 2:

$\pi[i] < i \quad \forall 1 \leq i \leq m$ .

↳ Note-se que no início do loop  $k \leq i-1$

Observação 3:  $k$  nunca fica negativo

Conclusão: O while interno é executado no máximo  $m-1$  vezes. De onde segue que a complexidade do algoritmo é  $\Theta(m)$ .

## Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: ab a b c a b c a b a b a b d  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern: a b a b d  
0 0 1 2 0

i i i i i j j j j j j  
0 1 2 3 4 5  
a b a b d      Match  
π: 0 0 1 2 0    //

→  
i i i i i j i j i j i j i j i j i j i j i j i j i j  
ab a b c a b c a b a b a b a b d  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  
→

## Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern: ababd  
0 0 1 2 0

i↓ j↓ j↓ j↓ j↓  
0 1 2 3 4 5  
ababd  
0 0 1 2 0

What to do now?

i↓ j↓ j↓ i↓ j↓  
ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

## Algortimo de Knuth-Morris-Pratt - Exemplo

Text: ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern: ababd  
0 0 1 2 0

i↓ i↓ j↓ j↓ j↓  
0 1 2 3 4 5  
a b a b d  
0 0 1 2 0

What to do now?

- Look at the  $P_i$

i↓ i↓ j↓ i↓ j↓ j↓  $\rightarrow$  skip!  
ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

## Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern: ababd  
0 0 1 2 0

i j i j j  
0 1 2 3 4 5  
ababd  
0 0 1 2 0

i j i j i j i  
ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

## Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern: ababd  
0 0 1 2 0

i  
↓

0 1 2 3 4 5  
ababd  
- 0 0 1 2 0

j  
↓

Não há mais

o rito para

andar para trás!

i j j i j i

ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

→ Esta na altura do  $\underline{i}$  andar para a frente!

## Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern: ababd  
0 0 1 2 0

i j j j  
0 1 2 3 4 5  
ababd  
0 0 1 2 0

i j j i j j i j i j i  
ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

## Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern: ababd  
0 0 1 2 0

i j j j j j j  
0 1 2 3 4 5  
ababd  
0 0 1 2 0

i j i j i j i j i j i j i j i j i j i j i j i j i  
ababcabcabababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

## Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: abababcabcababababd  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattan: a b a b d  
          0 0 1 2 0

Handwritten musical notation on lined paper. The top staff consists of five vertical stems with downward arrows at the top. Below this are five numbers: 0, 1, 2, 3, 4, 5. The bottom staff shows the letters P, a, b, a, b, d. Below this are the numbers 0, 0, 1, 2, 0. A horizontal line with an arrow points to the right under the last two numbers.

## Algoritmo de Knuth-Morris-Pratt

KMP-Match( $T, P$ )

let  $n = T.length$

let  $m = P.length$

let  $\pi = \text{Compute Prefix Function}(P)$

let  $j = 0$

for  $i = 1$  to  $n$

    while  $j > 0$  &  $T[i] \neq P[j+1]$

$j = \pi[j]$

    if  $P[j+1] == T[i]$

$j = j + 1$

    if  $(j == m)$

        print "Match at" "(i - m)"

$j = \pi[j]$

Complexidade:

$\Theta(m)$