

Sumário

Algoritmos para empacotamento
de cadeias de caracteres

- Algoritmo Elementar
- Algoritmo de Rabin-Karp
- Algoritmo baseado em autômatos finitos
- Algoritmo de Knuth-Morris-Pratt

Aulas 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



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

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

if $found$

Print "O padrão ocorre q shift" + s

Complexidade:

$O(n \cdot m)$

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

Input
T: a c a a b c
P: a u b

* $s=1$ T: a c a a b c
 [?]
P: a u b

* $s=2$ T: a c a a b c
 [?]
P: a u b

* $s=3$ T: a c a a b c ✓
 [?]
P: a u b

Estamos a comparar
c com a uma segunda vez
desnecessariamente

Ideia: Usar pré-processamento
para evitar comparações desnecessárias.

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

Input
T: a c a a b c
P: a u b

* $s=1$ T: a **c** a a b c
 [?]
 P: **a** **a** b

* $s=2$ T: a **c** a a b c
 [?]
 P: **a** **a** b

* $s=3$ T: a c **a** a b c ✓
 P: **a** **a** b

Estamos a comparar
c com a uma segunda vez
desnecessariamente

Ideia: Usar pré-processamento
para evitar comparações desnecessárias.

Algoritmo de Rabin-Karp Simplificado

- Associamos um número único a cada símbolo do alfabeto de trabalho

$$a - 1 \quad c - 3$$

$$b - 2 \quad d - 4$$

- Constavímos uma função de hash que mapeia padrões em números

$$h: \text{Pattern} \rightarrow \text{Code}$$

$$h(P[1..m]) = h(P[1]) + \dots + h(P[m])$$

- Exemplo:

$$h(aab) = 1+1+2$$

$$\begin{aligned} \text{Text: } & a a a a a b \\ n = 6 & \end{aligned}$$

$$\text{Pattern: } a a b$$

$$m = 3$$

Algoritmo de Rabin-Karp Simplificado

- Associamos um número único a cada símbolo do alfabeto de trabalho

$$a - 1 \quad c - 3$$

$$b - 2 \quad d - 4$$

- Constavímos uma função de hash que mapeia padrões em números

$$h: \text{Pattern} \rightarrow \text{Code}$$

$$h(P[1..m]) = h(P[1]) + \dots + h(P[m])$$

- Exemplo:

$$h(aab) = 1 + 1 + 2$$

$$= 4$$

Text: a a a a a b
 $n = 6$

Pattern: a a b

$$m = 3$$

① a a a a a b x

② a a a a a b x

③ a a a a a b x

④ a a a a a b ✓

Algoritmo de Rabin-Karp Simplificado

- Função de hash:

$$h(P[1..m]) = h(P[1]) + \dots + h(P[m])$$

- Hash deslizante ("sliding hash")

$$\frac{h(i)}{\hookrightarrow} \text{hash da palavra } T[i+1, \dots, i+m]$$

- Exemplo:

Alfabeto

a - 1

b - 2

c - 3

d - 4

① $\overset{3}{\text{aaa}}$ a a b x

② $\overset{3}{\text{aaa}}$ a a b x

③ $\overset{3}{\text{aaa}}$ a a b x

④ $\overset{4}{\text{aaa}}$ a a b ✓

Algoritmo de Rabin-Karp Simplificado

- Função de hash:

$$h(P[1..m]) = h(P[1]) + \dots + h(P[m])$$

- Hash deslizante ("sliding hash")

$$\frac{h(i)}{h(i) = h(i-1) + c(T[i]) - c(T[i-1])}{\hookrightarrow} \text{hash da palavra } T[i+1, \dots, i+m]$$

- Exemplo:

Alfabeto

a-1

b-2

c-3

d-4

- (1) $\overset{3}{aaa}aab \times$
- (2) $aaa\overset{3=3-1+1}{aab} \times$
- (3) $aaa\overset{3=3-1+1}{aab} \times$
- (4) $aaa\overset{4=5-1+2}{aab} \checkmark$

Algoritmo de Rabin-Karp - Simplificado

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

Pattern: $\begin{matrix} d & b & a \end{matrix}$
 $n=3$

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

Associamos a cada símbolo do alfabeto
 de trabalho Σ um número único

① $\underline{\text{c c a}} \text{ccaaedba}$

⑦ $\text{c c a} \underline{\text{ccaae}} \text{d ba}$

② $\text{c} \underline{\text{c a c}} \text{ccaaedba}$

⑧ $\text{c c a} \underline{\text{ccaae}} \underline{\text{d}} \text{ba}$

③ $\text{c c a} \underline{\text{cc}} \text{aaedba}$

⑨ $\text{c c a} \text{cc} \underline{\text{aae}} \text{d ba}$

④ $\text{c c a} \text{cc} \underline{\text{aae}} \text{d ba}$

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

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 fivelha Σ um número único

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

① $\underline{c \ c \ a} \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

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

② $\underline{c \ c \ a} \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

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

③ $\underline{c \ c \ a} \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

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

④ $\underline{c \ c \ a} \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

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

⑤ $\underline{c \ c \ a} \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

⑥ $\underline{c \ c \ a} \ c \ c \ a \ a \ e \ d \ b \ a$

$$\uparrow$$

$7-1+4 = 10$
⑦ $\underline{c \ c \ a} \ c \ c \ a \ a \ e \ d \ b \ a \text{ (Not a match)}$

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

$$\uparrow$$

$$11-5+1 = 7$$

⑨ $\underline{c \ c \ a} \ c \ c \ a \ a \ e \ d \ b \ a \text{ (Real match)}$

Algoritmo de Rabin-Karp - Simplificado²

* Função de Hash

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

$$= \sum_{i=1}^m C(P[i]) \cdot \beta^{m-i}$$

$\beta = |\Sigma| \Rightarrow$ tamanho do alfabeto de trabalho

* Exemplo:

Text:

1	2	3	4	5	6	7	8	9	10	11
c	c	a	c	c	a	a	e	d	b	a

$n=6$

Pattern: d b a $\beta=5$
 $n=3$

$$\begin{aligned} h(db a) &= 4 \cdot \beta^2 + 2 \cdot \beta + 1 \\ &= 4 \cdot 5^2 + 2 \cdot 5 + 1 \\ &= 111 \end{aligned}$$

Algoritmo de Rabin-Karp - Simplificado²

* Função de Hash

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

$$= \sum_{i=1}^m C(P[i]) \cdot \beta^{m-i}$$

$\beta = |\Sigma| \Rightarrow$ tamanho do alfabeto de trabalho

* Exemplo:

Text:

1	2	3	4	5	6	7	8	9	10	11
c	c	a	c	c	a	a	e	d	b	a

$n=6$

Pattern: d b a $\beta=5$
 $n=3$

$$\begin{aligned} h(db a) &= 4 \cdot \beta^2 + 2 \cdot \beta + 1 \\ &= 4 \cdot 5^2 + 2 \cdot 5 + 1 \\ &= 111 \end{aligned}$$

Algoritmo de Rabin-Karp - Simplificado²

* Função de Hash

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

$$= \sum_{i=1}^m C(P[i]) \cdot \beta^{m-i}$$

$\beta = |\Sigma| \Rightarrow$ tamanho do alfabeto de trabalho

* Hash Deslizante

$$h(i) =$$

Algoritmo de Rabin-Karp - Simplificado²

* Função de Hash

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

$$= \sum_{i=1}^m C(P[i]) \cdot \beta^{m-i}$$

$\beta = |\Sigma| \Rightarrow$ tamanho do alfabeto de trabalho

* Hash Deslizante

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

↳ hash da palavra: $T[i, \dots, i+m-1]$

Observação: no livro/slides da aula a fórmula é diferente porque se assume que que $h(i)$ é o hash da palavra \bar{q} comeca 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: $\begin{matrix} d & b & a \end{matrix}$
 $n=3$

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

Associamos a cada símbolo do alfabeto
de fórmula Σ um número único

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

↑

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

↑

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

↑

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

↑

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

↑

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

↑

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

↑

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

↑

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: $d \ b \ a$
 $n=3$

$$h(P) = 421$$

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

Associamos a cada símbolo do alfabeto
de fórmula Σ um número único

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

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

③ $\begin{matrix} 33 \\ c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a \end{matrix}$

$$\uparrow \quad (133 - 300) \times 10 + 1 = 33$$

④ $\begin{matrix} 311 \\ c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a \end{matrix}$

$$\uparrow \quad (33 - 300) \times 10 + 1 = 31 \times 10 + 1 = 311$$

⑤ $\begin{matrix} 115 \\ c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a \end{matrix}$

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

⑥ $\begin{matrix} 421 \\ c \ c \ a \ c \ c \ a \ a \ e \ d \ b \ a \end{matrix}$

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

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

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

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

$$\uparrow \quad (542 - 500) \times 10 + 1 = 42$$

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

Algoritmo de Rabin-Karp

- * Função de hash usada 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)

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

\rightarrow N° de comparações: $n - m + 1$

- Pior caso (São todos spurious hits)

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

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

Algoritmo de Rabin-Karp

- * Função de hash usada 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)

- Pior caso (São todos spurious hits)

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

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

- Q - conjunto de estados

- q_0 - estado inicial

- F - conjunto de estados finais

- Σ - alfabeto

- δ - função de transição

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

Exemplo: aba

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

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

- Q - conjunto de estados

- q_0 - estado inicial

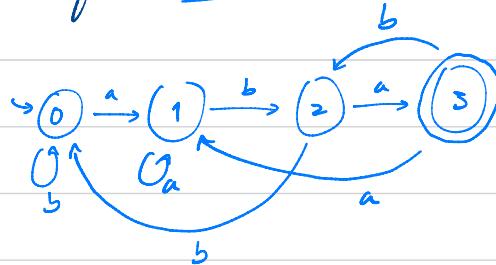
- F - conjunto de estados finais

- Σ - alfabeto

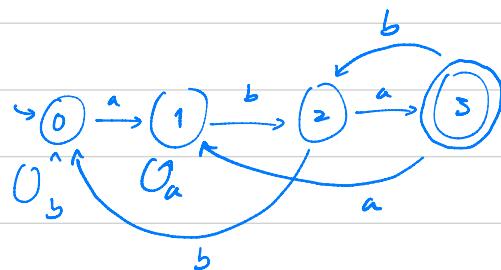
- δ - função de transição

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

Exemplo: aba



Texto: abbbabbaababaa



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 Sufíxo [Suffix Function]

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

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

Exemplo:

$$\bullet P = abaa$$

$$\sigma(abab) =$$

$$\bullet P = abaa$$

$$\sigma(abaaa) =$$

$$\bullet P = abaa$$

$$\sigma(abaab) =$$

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 Sufíxo [Suffix Function]

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

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

Exemplo:

$$\bullet P = abaa \quad \sigma(abab) = 2$$

$$\bullet P = abaa \quad \sigma(abaaa) = 1$$

$$\bullet P = abaa \quad \sigma(abaab) = 2$$

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:

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

$$\cdot f_0 = 0$$

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

$$\cdot \Sigma$$

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

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

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:

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

$$\cdot f_0 = 0$$

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

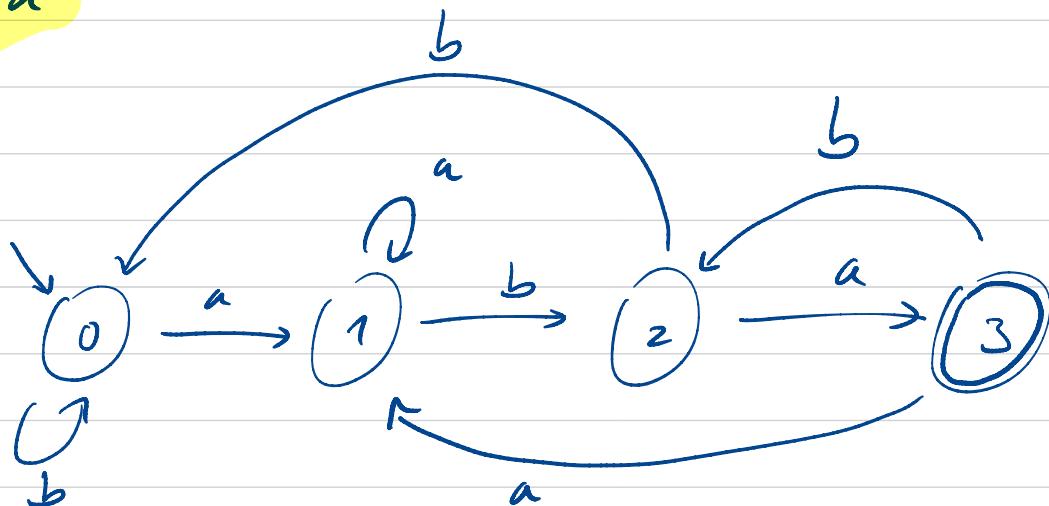
$$\cdot \Sigma$$

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

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

Exemplo 1

aba



$$\delta(1, a) = \sigma(aa) = 1$$

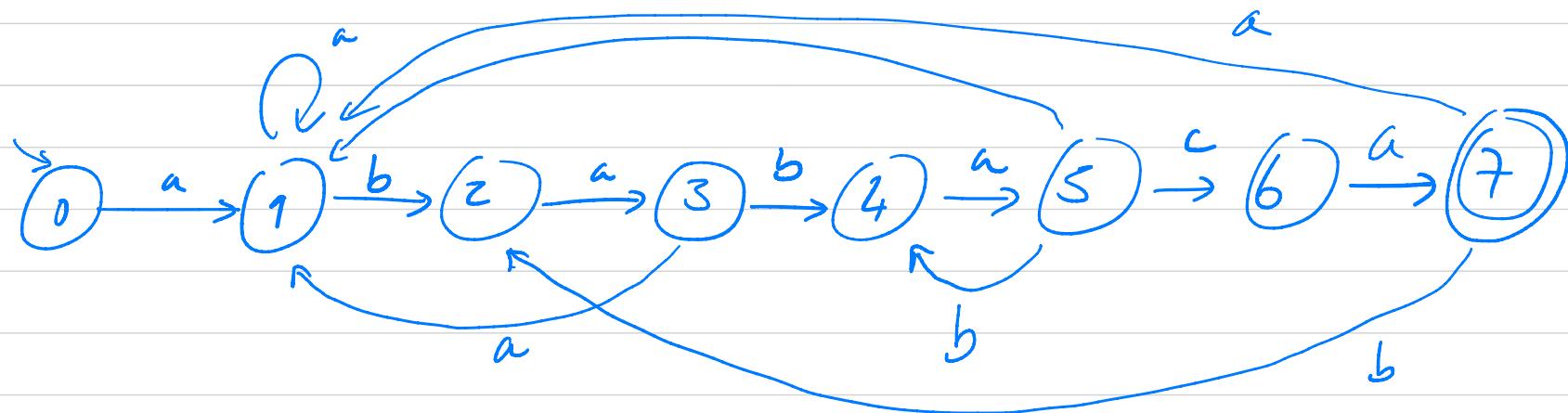
$$\delta(2, b) = \sigma(abb) = 0$$

$$\delta(3, a) = \sigma(abaaa) = 1$$

$$\begin{aligned}\delta(3, b) &= \sigma(abab) \\ &= 2\end{aligned}$$

Exemplo 2

Pat: aba bac a



String Matching com Automatos Finitos

Finite Automaton Matcher (T, δ, m)

let $n = T.length$

$j = 0$

for $i = 1$ to n

Complexidade:

Complete Transition Function (P, Σ)

$m = P.length$

for $j = 0$ to m

for each $a \in \Sigma$

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

repeat

$k = k - 1$

until $P_k \sqsupseteq p_j.a$

$\delta(j, a) = k$

return δ

Complexidade:

String Matching com Automatos Finitos

Finite Automaton Matcher (T, δ, m)

let $n = T.length$

$j = 0$

for $i=1$ to n

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

if $j == m$

Print "Match with shift" ($i-m$)

Complete Transition Function (P, Σ)

$m = P.length$

for $j=0$ to m

for each $a \in \Sigma$

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

repeat

$k = k - 1$

until $p_k \sqsupseteq p_j \cdot a$

$\delta(j, a) = k$

return δ

Complexidade:

$O(m)$

Complexidade:

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

=

↓ Algoritmos mais eficientes

$O(m \cdot |\Sigma|)$

Algoritmo de Knuth-Morris-Pratt

• Fngô de Prefixo

- Dado um padrão $P[1..n]$

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

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

Tamanho do maior prefixo de P que também é sufixo de P :

Exemplos:

(III) a b c a d a a b e

(I) a b c d a b e a b f

(IV) a a a a b a a c d

(II) a b c d e a b f a b c

(V) a b a b a b a b c a

Algoritmo de Knuth-Morris-Pratt

Função 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 .

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:

(III) a a b c a d a a b e

(I) a b c d a b e a b f

(IV) a a a a b a a c d

(II) a b c d e a b f a b c

(V) a b a b a b a b c a

Algoritmo de Knuth-Morris-Pratt

Função 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) $a b c d a b e a b f$
0 0 0 0 1 2 0 1 2 0

(II) $a a b c a d a a b e$
0 1 0 0 1 0 1 2 3 0

(III) $a b c d e a b f a b c$
0 0 0 0 0 1 2 0 1 2 3

(IV) $a a a a b a a c d$
0 1 2 3 0 1 2 0 0

(V) $a b a b a b a b c a$
0 0 1 2 3 4 5 6 0 1

Algoritmo de Knuth-Morris-Pratt

$$\pi[i] = \max \{ k : k < i \wedge p_k \sqsupseteq p_i \}$$

Compute Prefix Function (π)

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

$O(m)$

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

Observação 2:

$\pi[i] < i \quad (1 \leq i \leq m)$

↳ 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 $O(m)$.

Algoritmo de Knuth-Morris-Pratt - Exemplo

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

Pattern: abab d
0 0 1 2 0

0 1 2 3 4 5
abab d
π: 0 0 1 2 0



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



Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: a b 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 j i j i j i
0 1 2 3 4 5
a b a b d
0 0 1 2 0

What to do now?

i j i j i j i
↓ ↓ ↓ ↓ ↓ ↓
a b 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

Algoritmo de Knuth-Morris-Pratt - Exemplo

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

Pattern: abab d
0 0 1 2 0

i i j j j j j j
0 1 2 3 4 5
abab d
0 0 1 2 0

What to do now?

- Look at the P_i

i i j i l i → skip!
ababcbabcaba 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: ababcbabcababababd
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
ababcbabcababababd
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Algoritmo de Knuth-Morris-Pratt - Exemplo

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

Pattern: abababd
0 0 1 2 0

i
j
0 1 2 3 4 5
- abababd
- 0 0 1 2 0
↓
Não há mais
sitio para
andar para trás!

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

→ Está na altura do i andar para a frente!

Algoritmo de Knuth-Morris-Pratt - Exemplo

Text: ababcbabcaba b abd
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
ababbd
0 0 1 2 0

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

Algoritmo de Knuth-Morris-Pratt

KMP-Matcher (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:

$\underline{\mathcal{O}}(m)$

Propriedades da Função de Sufixo

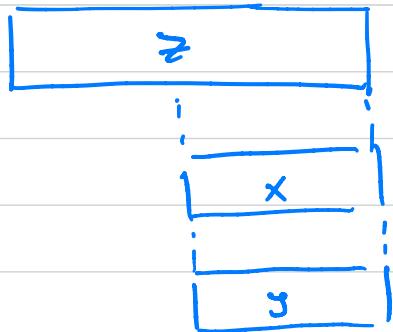
(Notas para estudo individual)

Propriedades de Sufixo e Função de Sufixo

* Propriedade 1

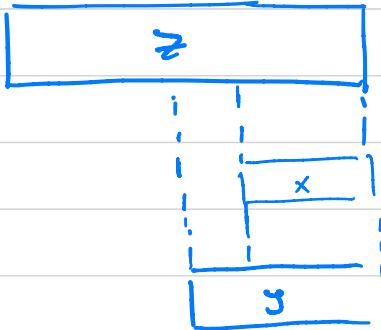
$$x \sqsupseteq z \wedge y \sqsupseteq z \Rightarrow \begin{cases} |x| = |y| \Rightarrow x = y & (a) \\ |x| < |y| \Rightarrow x \sqsubset y & (b) \\ |y| < |x| \Rightarrow y \sqsubset x & (c) \end{cases}$$

a)



$$x = y$$

b)



$$x \sqsubset y$$

Propriedades de Sufixo e Fungão 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 $i = \sigma(xa)$

$$\cdot p_i \sqsupseteq xa$$

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

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

$$\cdot |p_i| \leq |p_q.a|$$

$$\cdot p_i \sqsupseteq p_q.a \quad (\text{Propriedade 1})$$

$$\sigma(p_i) \leq \sigma(p_q.a)$$

$$i = \sigma(xa) \leq \sigma(p_q.a)$$

a) $i = \sigma(x) \Rightarrow p_i \sqsupseteq x$

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

$$\Rightarrow \sigma(p_i.a) \leq \sigma(xa)$$