

Análise e Síntese de Algoritmos

Emparelhamento de Cadeias de Caracteres [CLRS, Cap. 32]

2011/2012

Contexto

- Revisão [CLRS, Cap.1-13]
 - Fundamentos; notação; exemplos
- Algoritmos em Grafos [CLRS, Cap.21-26]
 - Algoritmos elementares
 - Árvores abrangentes
 - Caminhos mais curtos
 - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
 - Algoritmos e modelação de problemas com restrições lineares
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
 - Programação dinâmica
 - Algoritmos greedy
- Tópicos Adicionais [CLRS, Cap.32-35]
 - Emparelhamento de Cadeias de Caracteres
 - Complexidade Computacional
 - Algoritmos de Aproximação

Resumo

- 1 Definição do Problema
 - Notação
- 2 Algoritmo Elementar
- 3 Algoritmo Autómatos Finitos
- 4 Algoritmo Knuth-Morris-Pratt
- 5 Algoritmo Rabin-Karp

String Matching

Definição do Problema

Verificar a ocorrência de um padrão P num texto T :

- **Texto**: array com n caracteres, $T[1..n]$
- **Padrão**: array com m caracteres, $P[1..m]$
- Caracteres pertencentes a alfabeto finito Σ
- Padrão P ocorre com deslocamento s em T se:

$$T[s + 1..s + m] = P[1..m] \quad , \quad 0 \leq s \leq n - m$$

String Matching

Definição do Problema

Verificar a ocorrência de um padrão P num texto T :

- **Texto**: array com n caracteres, $T[1..n]$
- **Padrão**: array com m caracteres, $P[1..m]$
- Caracteres pertencentes a alfabeto finito Σ
- Padrão P ocorre com deslocamento s em T se:

$$T[s + 1..s + m] = P[1..m] \quad , \quad 0 \leq s \leq n - m$$

Exemplo

- $T = a**ababab**ab$
- $P = ababab$
- P ocorre em T com deslocamento 1

String Matching

Definição do Problema

Verificar a ocorrência de um padrão P num texto T :

- **Texto**: array com n caracteres, $T[1..n]$
- **Padrão**: array com m caracteres, $P[1..m]$
- Caracteres pertencentes a alfabeto finito Σ
- Padrão P ocorre com deslocamento s em T se:

$$T[s + 1..s + m] = P[1..m] \quad , \quad 0 \leq s \leq n - m$$

Exemplo

- $T = aabababab$
- $P = ababab$
- P ocorre em T com deslocamento 1 e com deslocamento 3

String Matching

Notação

- Conjunto de todas as cadeias de caracteres (strings) de comprimento finito: Σ^*
- String vazia: ϵ
 - $\epsilon \in \Sigma^*$
- Comprimento da string x , $|x|$
- Concatenação de x e y :
 - Representação: xy
 - Comprimento: $|x| + |y|$

String Matching

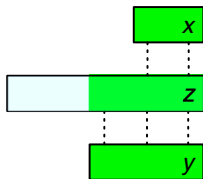
Notação

- w é **prefixo** de x , $w \sqsubset x$, se $x = wy$, com $y \in \Sigma^*$
- w é **sufixo** de x , $w \sqsupset x$, se $x = yw$, com $y \in \Sigma^*$
 - Se $w \sqsubset x$, então $|w| \leq |x|$
- Sejam x , y e z strings tais que $x \sqsubset z$ e $y \sqsubset z$. Então,
 - Se $|x| \leq |y|$, então $x \sqsubset y$
 - Se $|x| \geq |y|$, então $y \sqsubset x$
 - Se $|x| = |y|$, então $x = y$
- Prefixos do padrão pretendido
 - $P_0 = \epsilon$; $P_k = P[1..k]$
 - $T_k = T[1..k]$

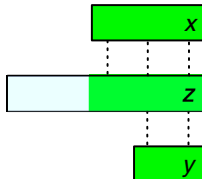
Exemplo

- **abb** é prefixo de **abb**abaabb
- **aaba** é sufixo de **abb**aba**aaba**

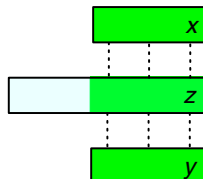
String Matching



$$|x| < |y|$$



$$|x| > |y|$$



$$|x| = |y|$$

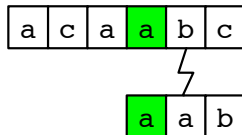
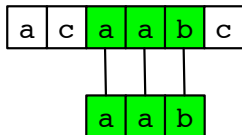
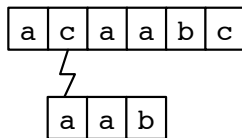
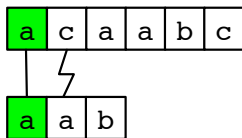
Algoritmo Elementar

Naive-String-Matcher(T, P)

```
1   $n \leftarrow \text{length}[T]$ 
2   $m \leftarrow \text{length}[P]$ 
3  for  $s \leftarrow 0$  to  $n - m$ 
4      do if  $P[1..m] = T[s + 1..s + m]$ 
5          then print "Padrão encontrado com deslocação",  $s$ 
```

Complexidade: $\Theta((n - m + 1)m)$

Algoritmo Elementar



Autómatos Finitos

Definição

- Autómatos finitos $M(Q, q_0, A, \Sigma, \delta)$:
 - Q é um conjunto finito de estados
 - q_0 é o **estado inicial**
 - $A \subseteq Q$ é um **conjunto de estados de aceitação**
 - Σ é o alfabeto de entrada
 - δ é uma função de $Q \times \Sigma$ em Q , designada **função de transição** de M
- Autómatos no estado q , com caracter de entrada a , novo estado é dado por **$\delta(q, a)$**
- Utilizar Autómatos Finitos para aceitar padrão pretendido

Autómatos Finitos

Função de Estado Final

- $\phi(w)$: estado de M após ter lido string w
- M aceita w se e só se $\phi(w) \in A$
- Definição:
 - $\phi(\epsilon) = q_0$
 - $\phi(wa) = \delta(\phi(w), a)$, para $w \in \Sigma^*$, $a \in \Sigma$

Autômatos Finitos

Função de Sufixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de x

$$\sigma : \Sigma^* \rightarrow \{0, 1, \dots, m\}$$

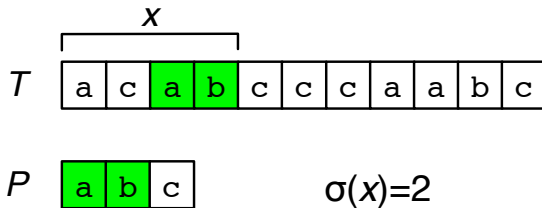
$$\sigma(x) = \max\{k : P_k \sqsupseteq x\}$$

- Saber em que estado o autômato deve estar dada a string x

Autômato para emparelhamento de strings

- $Q = \{0, 1, \dots, m\}$
- Função de transição: $\delta(q, a) = \sigma(P_q a)$
- Novo estado $\delta(q, a)$ corresponde ao prefixo de P com o maior comprimento que é também sufixo de $P_q a$

Autómatos Finitos



Autômatos Finitos

Finite-Automaton-Matcher(T, δ, m)

```
1   $n \leftarrow \text{length}[T]$ 
2   $q \leftarrow 0$ 
3  for  $i \leftarrow 1$  to  $n$ 
4      do  $q \leftarrow \delta(q, T[i])$ 
5          if  $q = m$ 
6              then  $s \leftarrow i - m$ 
7              Print "Padrão encontrado com deslocação",  $s$ 
```

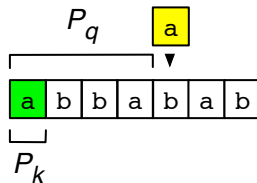
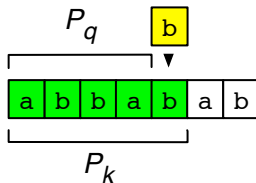

Autômatos Finitos

Compute-Transition-Function(P, Σ)

```

1   $m \leftarrow \text{length}[P]$ 
2  for  $q \leftarrow 0$  to  $m$ 
3      do for each  $a \in \Sigma$ 
4          do  $k \leftarrow \min(m + 1, q + 2)$ 
5              repeat  $k \leftarrow k - 1$ 
6                  until  $P_k \sqsupseteq P_q a$ 
7                   $\delta(q, a) \leftarrow k$ 
8  return  $\delta$ 

```



Autômatos Finitos

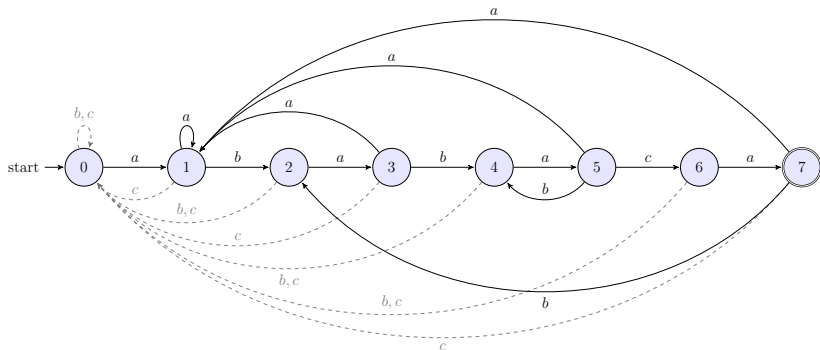
Compute-Transition-Function(P, Σ)

```
1   $m \leftarrow \text{length}[P]$ 
2  for  $q \leftarrow 0$  to  $m$ 
3      do for each  $a \in \Sigma$ 
4          do  $k \leftarrow \min(m + 1, q + 2)$ 
5              repeat  $k \leftarrow k - 1$ 
6                  until  $P_k \sqsupseteq P_q a$ 
7                   $\delta(q, a) \leftarrow k$ 
8  return  $\delta$ 
```

Análise de Complexidade

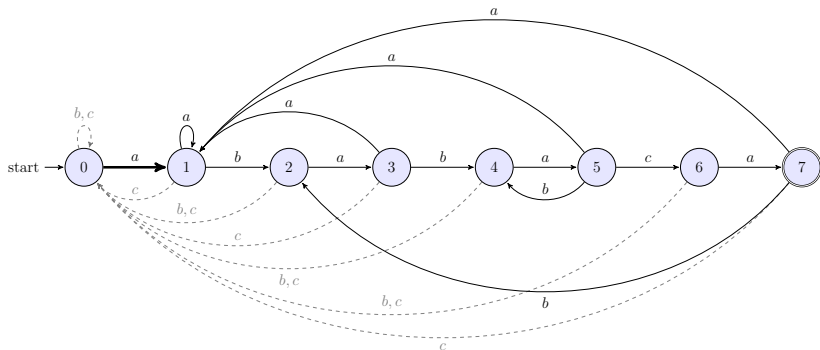
- Complexidade do cálculo de δ : $O(m^3|\Sigma|)$
- Complexidade do algoritmo: $O(n + m^3|\Sigma|)$
 - É possível obter $O(n + m|\Sigma|)$

Autômatos Finitos



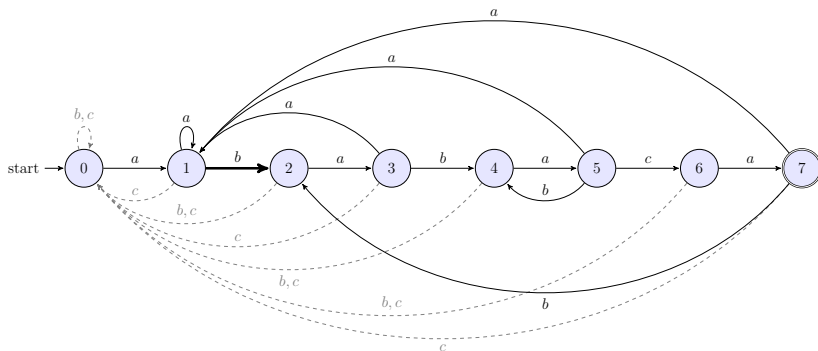
P = ababaca T = abcababacababacaabacab

Autômatos Finitos



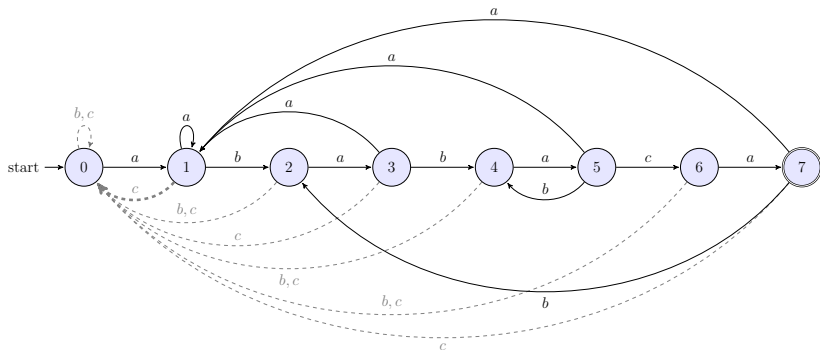
P = ababaca T = abcababacababacaabacab

Autômatos Finitos



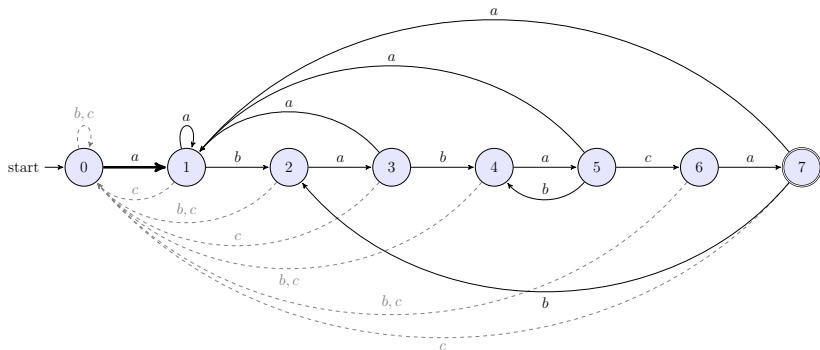
P = ababaca T = **a**bcababacababacaabacab

Autômatos Finitos



P = ababaca T = ab**c**ababacababacaabacab

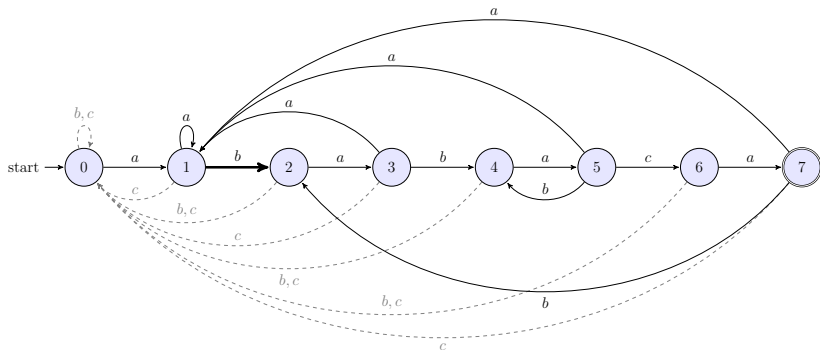
Autômatos Finitos



P = ababaca

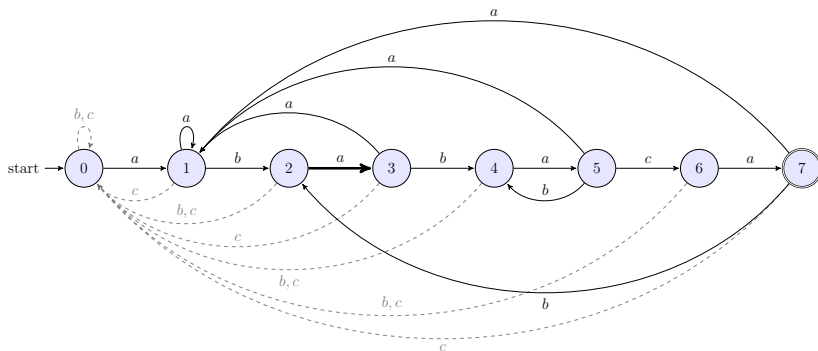
T = abcababacababacaabacab

Autômatos Finitos



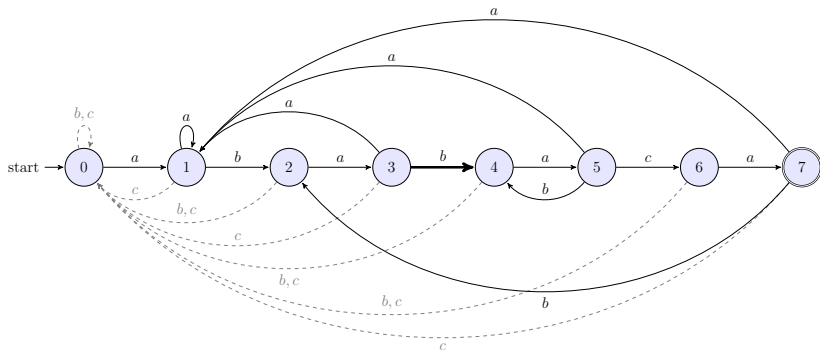
P = ababaca T = abcababacababacaabacab

Autômatos Finitos



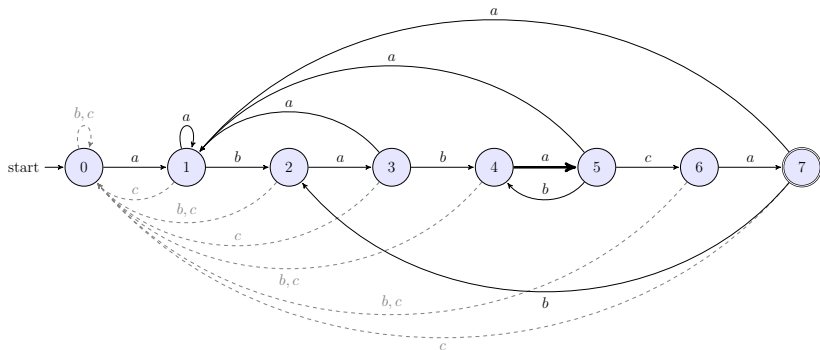
P = ababaca T = abcab**a**bacababacaabacab

Autômatos Finitos



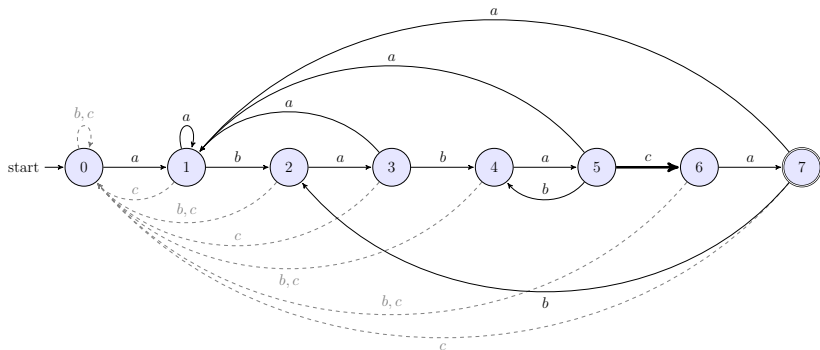
P = ababaca T = abcababacababacaabacab

Autômatos Finitos



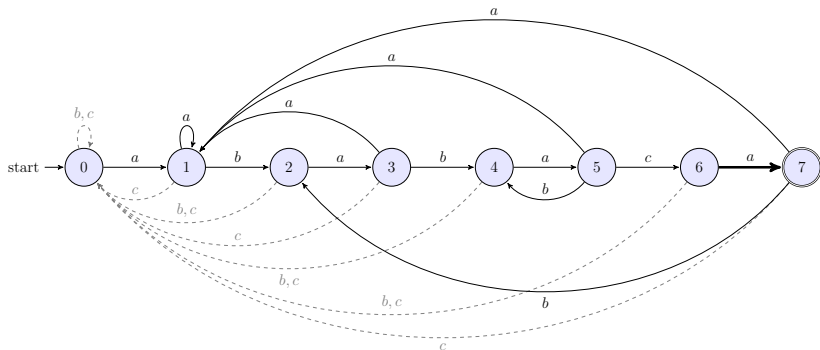
P = ababaca T = abcababacababacaabacab

Autômatos Finitos



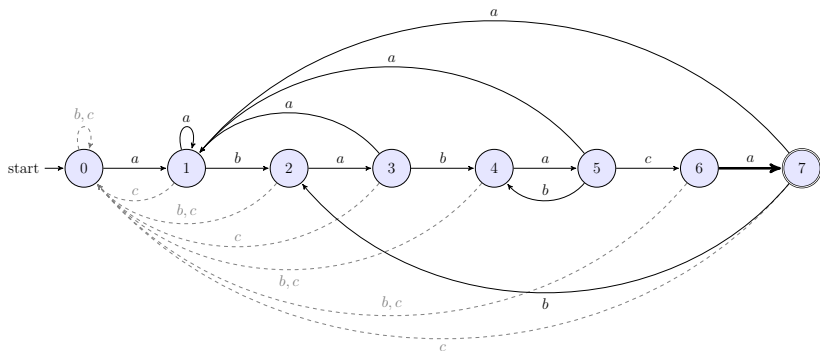
P = ababaca T = abcababacababacaabacab

Autômatos Finitos



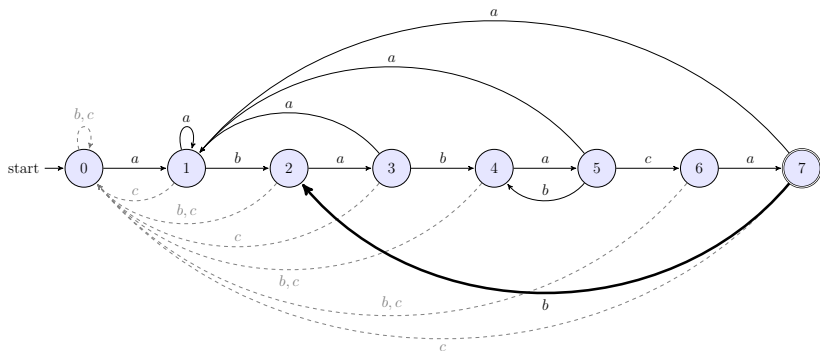
P = ababaca T = abcababac**ab**abacaabacab

Autômatos Finitos



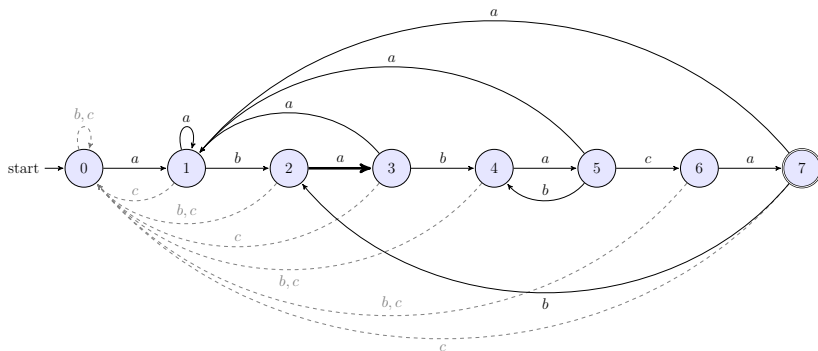
P = ababaca T = abc**ababaca**babacaabacab

Autômatos Finitos



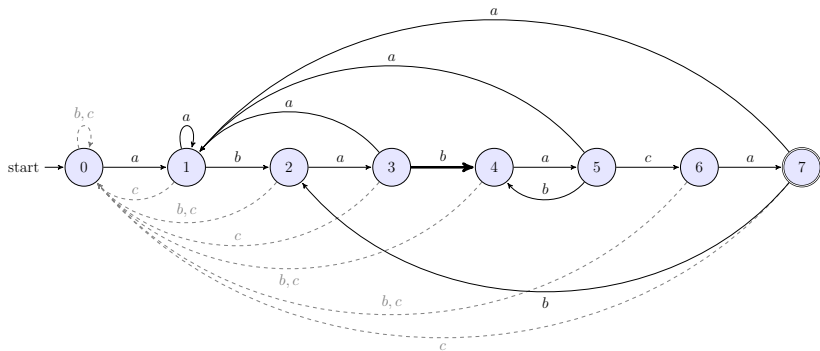
P = ababaca T = abcababacababacaabacab

Autômatos Finitos



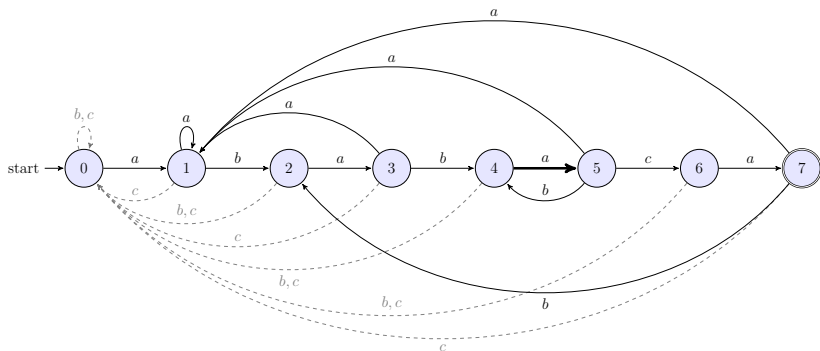
P = ababaca T = abcababacab**a**acaabacab

Autômatos Finitos



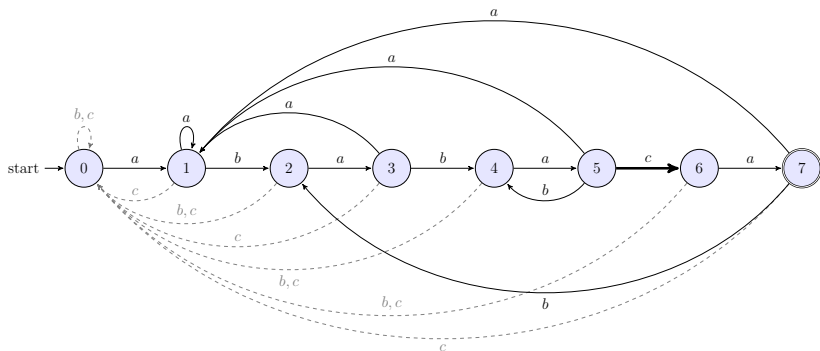
P = ababaca T = abcababacab**a**caabacab

Autômatos Finitos



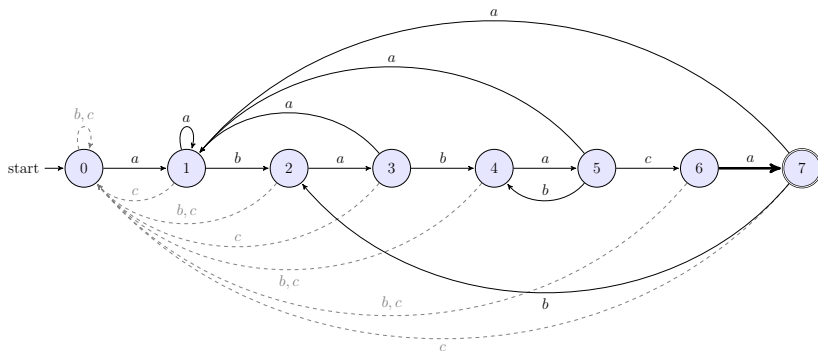
P = ababaca T = abcababacabab**a**caabacab

Autômatos Finitos



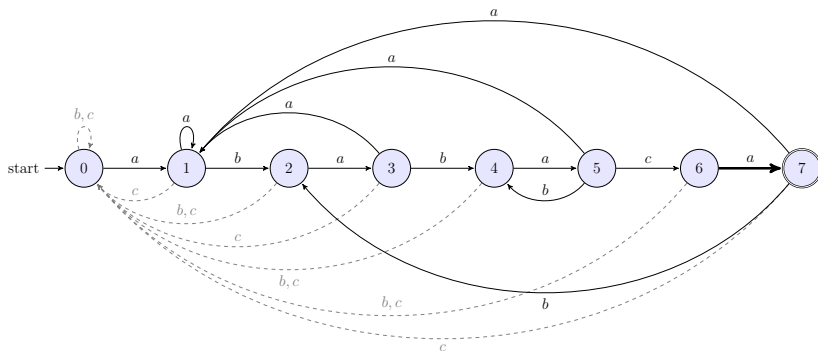
P = ababaca T = abcababacababab**a**aabacab

Autômatos Finitos



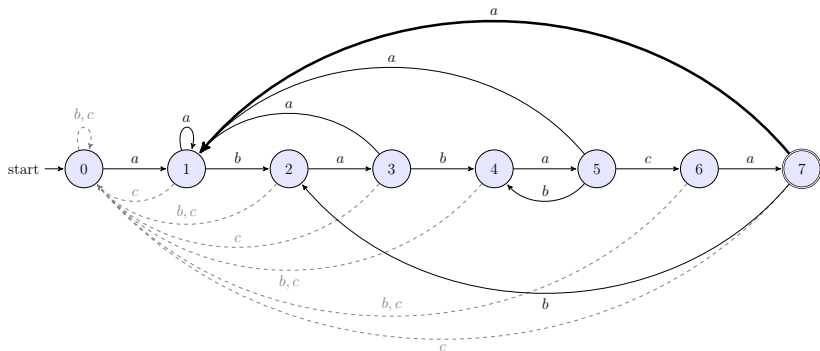
P = ababaca T = abcababacababacab

Autômatos Finitos



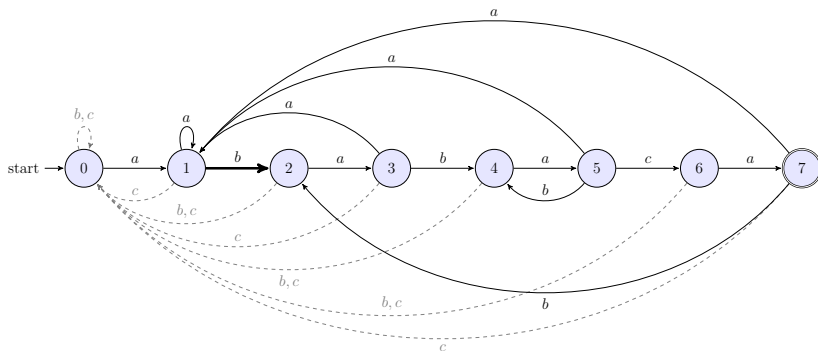
P = ababaca T = abcababac**ababaca**abacab

Autômatos Finitos



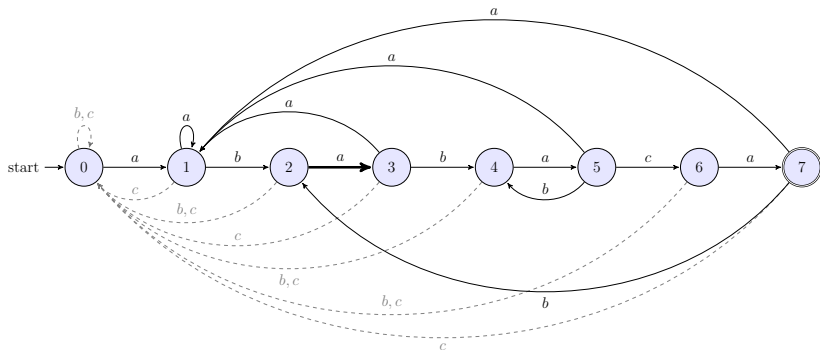
P = ababaca T = abcababacababacab

Autômatos Finitos



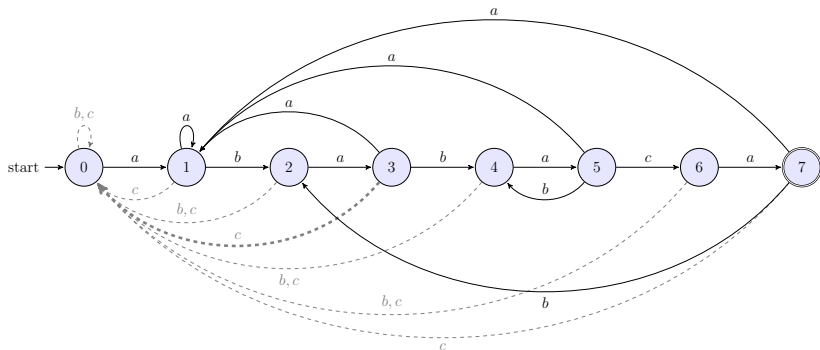
P = ababaca T = abcababacababacaa**b**acab

Autômatos Finitos



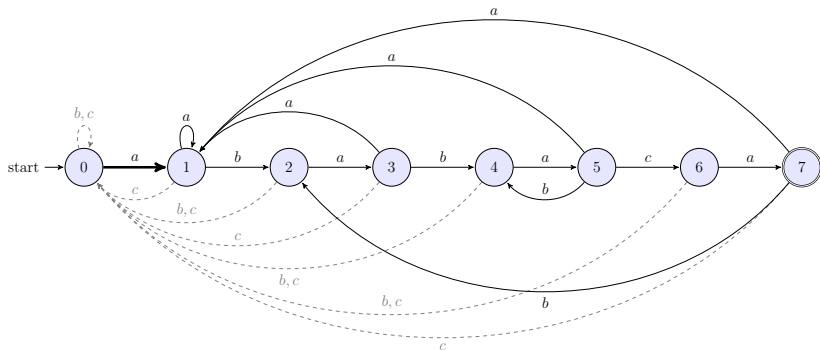
P = ababaca T = abcababacababacaab**a**cab

Autômatos Finitos



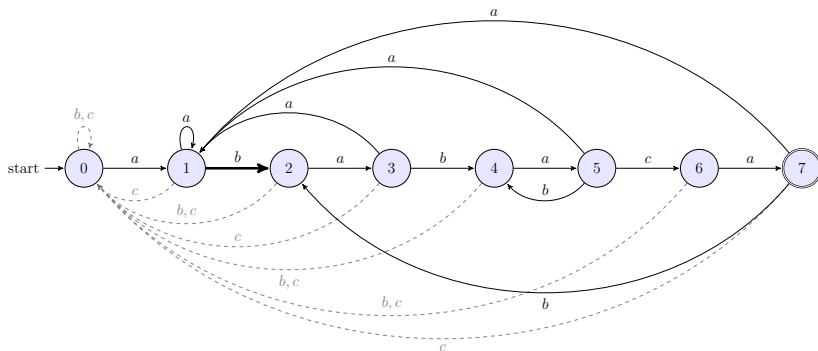
P = ababaca T = abcababacababacaabacab

Autômatos Finitos



P = ababaca T = abcababacababacaabac**ab**

Autômatos Finitos



P = ababaca

T = abcababacababacaabac**b**

Algoritmo Knuth-Morris-Pratt

Vantagens

- Complexidade: $O(n + m)$
- Evita cálculo da função de transição δ !

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$										

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0									

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0								

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1							

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2						

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3					

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4				

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5			

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6		

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

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

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

- Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

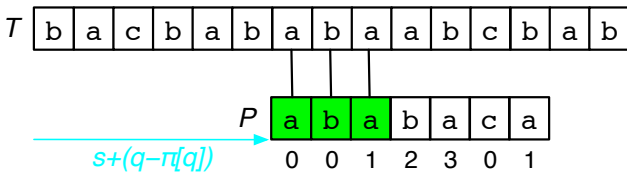
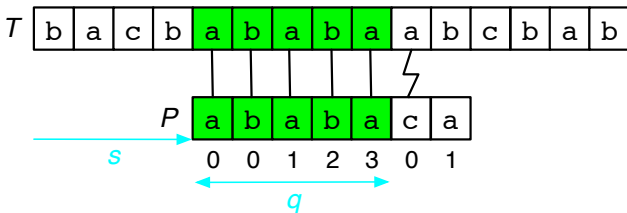
$$\pi : \{1, 2, \dots, m\} \rightarrow \{0, 1, \dots, m-1\}$$

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupseteq P_q\}$$

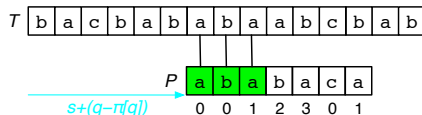
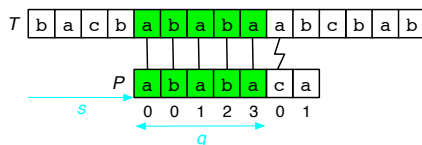
- Codifica conhecimento acerca de similaridades entre partes do padrão

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

Algoritmo Knuth-Morris-Pratt



Algoritmo Knuth-Morris-Pratt



- Caso o próximo caracter de T não emparelhe, $\pi[q]$ indica qual o maior prefixo de P que é possível obter como sufixo de P_q sabendo que em T existe neste momento (i.e. em T_i) o sufixo P_q (e que $P[q+1] \neq T[i+1]$)

Algoritmo Knuth-Morris-Pratt

Algoritmo Knuth-Morris-Pratt

- Se $P[k+1] \neq T[i]$, então P_{k+1} não é sufixo de T_i
 - Encontrar j tal que P_j seja sufixo de T_i
- Observar que:
 - $P_k \sqsupset T_{i-1}$
 - $P_{\pi[k]} \sqsupset P_k$
 - maior sufixo de P_k e T_{i-1} que é menor que P_k
 - $P_{\pi[\pi[k]]} \sqsupset P_{\pi[k]}$
 - maior sufixo de $P_{\pi[k]}$, P_k e T_{i-1} que é menor que $P_{\pi[k]}$

Algoritmo Knuth-Morris-Pratt

KMP-Matcher(T, P)

```
1   $n \leftarrow \text{length}[T]$ 
2   $m \leftarrow \text{length}[P]$ 
3   $\pi \leftarrow \text{Compute-Prefix-Function}(P, \Sigma)$ 
4   $q \leftarrow 0$ 
5  for  $i \leftarrow 1$  to  $n$ 
6      do while ( $q > 0 \wedge P[q + 1] \neq T[i]$ )
7          do  $q \leftarrow \pi[q]$ 
8      if  $P[q + 1] = T[i]$ 
9          then  $q \leftarrow q + 1$ 
10     if  $q = m$ 
11         then Print "Padrão encontrado com deslocação",  $i - m$ 
12          $q \leftarrow \pi[q]$ 
```

Algoritmo Knuth-Morris-Pratt

Compute-Prefix-Function(P, Σ)

```
1   $m \leftarrow \text{length}[P]$ 
2   $\pi[1] \leftarrow 0$ 
3   $k \leftarrow 0$ 
4  for  $q \leftarrow 2$  to  $m$ 
5      do while  $k > 0 \wedge P[k+1] \neq P[q]$ 
6          do  $k \leftarrow \pi[k]$ 
7          if  $P[k+1] = P[q]$ 
8              then  $k \leftarrow k + 1$ 
9           $\pi[q] \leftarrow k$ 
10 return  $\pi$ 
```

Algoritmo Knuth-Morris-Pratt

Análise de Complexidade

- Compute-Prefix-Function: $O(m)$
 - k é incrementado de 1 unidade não mais do que uma vez por cada valor de q , com número total de incrementos limitado superiormente por m
 - Valor de k decrementado devido a atribuição $k = \pi[k]$, mas $k > 0$ e valor acumulado de decremento limitado a um total de m unidades
- KMP-Matcher: $O(n + m)$
 - Análise semelhante à anterior permite obter resultado

Algoritmo Rabin-Karp

Definições

- Vamos assumir que $\Sigma = \{0, 1, 2, \dots, 9\}$, tal que cada caracter é um dígito decimal
 - No caso geral podemos assumir que cada caracter é um dígito numa notação base d , onde $d = |\Sigma|$
- Dado o padrão $P[1..m]$, designamos por p o valor decimal correspondente
- Dado o texto $T[1..n]$, designamos por t_s o valor decimal da sub-string de T com dimensão m e deslocamento s , $T[s + 1..s + m]$, onde $s = 0, 1, \dots, n - m$

Algoritmo Rabin-Karp

Exemplo

- Ao padrão $P = 31415$ corresponde o valor decimal $p = 31415$
- Seja $T = 123141567$, então:
 - $T[1..5] = 12314$ e $t_0 = 12314$
 - $T[2..6] = 23141$ e $t_1 = 23141$
 - $T[3..7] = 31415$ e $t_2 = 31415$
 - $T[4..8] = 14156$ e $t_3 = 14156$
 - $T[5..9] = 41567$ e $t_4 = 41567$

Algoritmo Rabin-Karp

Observações

- $t_s = p$ apenas quando $T[s + 1..s + m] = P[1..m]$, logo apenas neste caso s é um deslocamento válido
- Se:
 - Conseguirmos calcular o valor de p em tempo $\Theta(m)$
 - Conseguirmos calcular cada um dos valores de t_s em tempo $\Theta(n - m + 1)$
- Então:
 - Conseguimos determinar todos os deslocamentos s válidos em tempo $\Theta(m) + \Theta(n - m + 1) = \Theta(n)$, comparando p com cada um dos valores de t_s

Algoritmo Rabin-Karp

Aplicação da Regra de Horner

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \dots + 10(P[2] + 10P[1])\dots))$$

- Conseguimos calcular p a partir de $P[1..m]$, em tempo $\Theta(m)$
- Conseguimos calcular t_0 a partir de $T[1..m]$, em tempo $\Theta(m)$
- Para calcular os restantes valores t_1, t_2, \dots, t_{n-m} , em tempo $\Theta(n-m)$, basta observar que t_{s+1} pode ser calculado a partir de t_s , em tempo constante:

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$$

Algoritmo Rabin-Karp

Exemplo

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \dots + 10(P[2] + 10P[1])\dots))$$

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$$

- Ao padrão $P=31415$ corresponde o valor decimal $p = 31415$
 - $p = 5 + 10 \times (1 + 10 \times (4 + 10 \times (1 + 10 \times 3)))$
- Seja $T=123141567$, então:
 - $T[1..5]=12314$ e $t_0 = 12314 = 4 + 10(1 + 10(3 + 10(2 + 10 \times 1)))$
 - $T[2..6]=23141$ e $t_1 = 23141 = 10(t_0 - 10000 \times 1) + 1$
 - $T[3..7]=31415$ e $t_2 = 31415 = 10(t_1 - 10000 \times 2) + 5$
 - $T[4..8]=14156$ e $t_3 = 14156 = 10(t_2 - 10000 \times 3) + 6$
 - $T[5..9]=41567$ e $t_4 = 41567 = 10(t_3 - 10000 \times 1) + 7$

Algoritmo Rabin-Karp

Utilização do Módulo

- p e t_s podem ser demasiado grandes
 - Se P contém m caracteres, para m grande não é razoável assumir que cada operação aritmética em p (que tem m dígitos) será efectuada em tempo constante
- Solução: utilizar o módulo (resto da divisão inteira) por q
 - Calcular p e t_s módulo um determinado valor q
 - q é normalmente escolhido como um número primo tal que $10q$ cabe numa *word*, permitindo efectuar todas as operações em aritmética de precisão simples

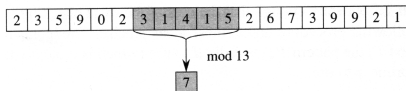
$$t_{s+1} = (10(t_s - T[s+1] \times 10^{m-1} \bmod q) + T[s+m+1]) \bmod q$$

Algoritmo Rabin-Karp

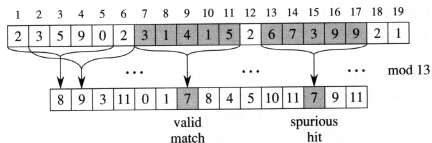
Utilização do Módulo

- $t_s = p \bmod q$ não implica necessariamente que $t_s = p$
- $t_s \neq p \bmod q$ implica que $t_s \neq p$
- Podemos utilizar o teste $t_s = p \bmod q$ como uma heurística rápida para invalidar certos deslocamentos s , quando $t_s \neq p \bmod q$
- Nos casos em que se verifica $t_s = p \bmod q$, temos que verificar explicitamente se $P[1..m] = T[s + 1..s + m]$, ou se se trata de um *spurious hit*
- Se q for suficientemente grande, podemos assumir que os *spurious hits* ocorrem com pouca frequência e logo que o custo de nesse caso ter de verificar se $P[1..m] = T[s + 1..s + m]$ será baixo

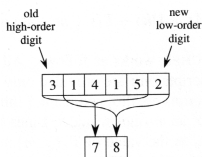
Algoritmo Rabin-Karp



(a)



(b)



$$\begin{aligned}
 14152 &\equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13} \\
 &\equiv (7 - 3 \cdot 3) \cdot 10 + 2 \pmod{13} \\
 &\equiv 8 \pmod{13}
 \end{aligned}$$

Diagram illustrating the update of a rolling hash using the formula:
$$14152 \equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13}$$

$$\equiv (7 - 3 \cdot 3) \cdot 10 + 2 \pmod{13}$$

$$\equiv 8 \pmod{13}$$

Algoritmo Rabin-Karp

Complexidade

- Pré-processamento: $\Theta(m)$
- Procura:
 - $\Theta((n - m + 1)m)$, no pior caso, dado que é necessário verificar todos os deslocamentos válidos
 - $O(n) + O(m(v + n/q))$, onde v é o número de deslocamento válidos e se assume que existem $O(n/q)$ *spurious hits*
 - $O(n)$, porque no pior caso o teste $p = t_s$ falha n vezes
 - $O(m(v + n/q))$, porque por cada deslocamento válido ou *spurious hit* temos que efectuar a comparação $P[1..m] = T[s + 1..s + m]$, que tem um custo $\Theta(m)$