TÉCNICO LISBOA

Análise e Síntese de Algoritmos

Emparelhamento de Cadeias de Caracteres

CLRS Cap. 32

Instituto Superior Técnico

2022/2023

Contexto

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

Resumo

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:

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

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

$T = aababababab$

$P = ababab$

P ocorre em T com deslocamento 1

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

$T = aababababab$

$P = ababab$

P ocorre em T com deslocamento 1 e com deslocamento 3

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: $|x| + |y|$

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 $w \sqsupset x$, então $|w| \leq |x|$

Sejam x, y e z strings tais que $x \sqsupset z$ e $y \sqsupset z$. Então,

Se $|x| \leq |y|$, então $x \sqsupset y$
Se $|x| \geq |y|$, então $y \sqsupset x$
Se $|x| = |y|$, então $x = y$

Prefixos do padrão pretendido

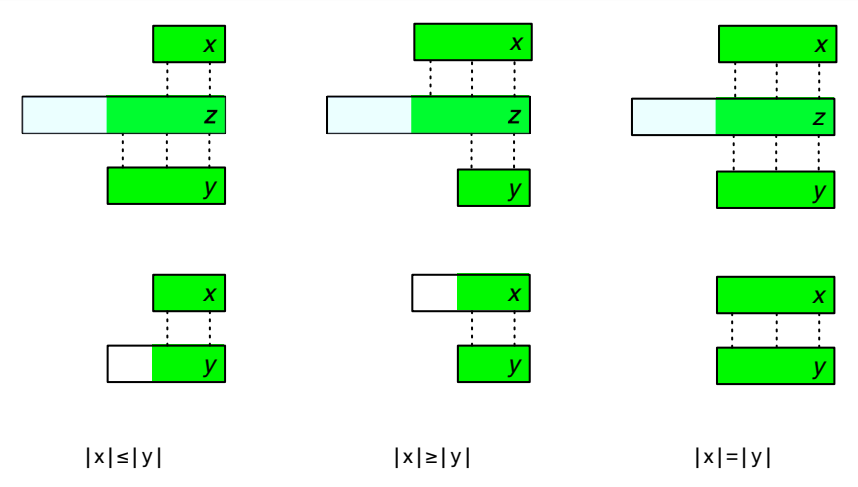
$P_0 = \epsilon; P_k = P[1..k]$
$T_k = T[1..k]$

Exemplo

abb é prefixo de $abbabaabb$

$aaba$ é sufixo de $abbabaaba$

String Matching



Algoritmo Elementar

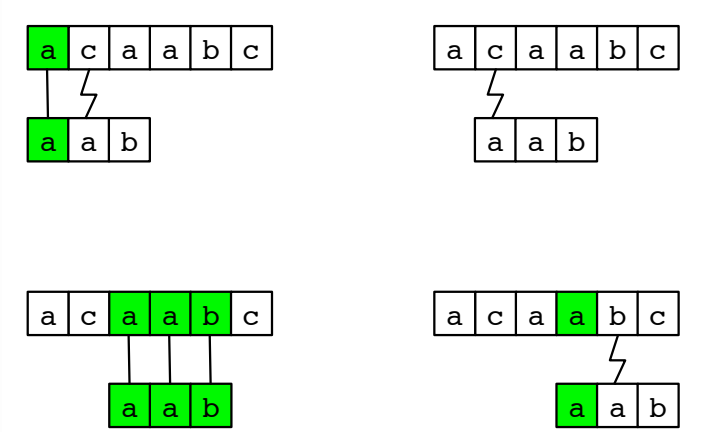
Naive-String-Matcher(T, P)

$n = length[T]$
$m = length[P]$
for $s = 0$ **to** $n - m$ **do**
  if $P[1 \ldots m] = T[s+1 \ldots s+m]$ **then**
    print "Padrão encontrado com deslocação", s
  end if
end for

Complexidade

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

Algoritmo Elementar



Autómatos Finitos

Definição

Autómato finito $M(Q, q_0, A, \Sigma, \delta)$:

- 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

Autómato no estado q, com caracter de entrada a, novo estado é dado por $\delta(q, a)$

Utilizar Autómato Finito para aceitar padrão pretendido

Função de Estado Final

$\phi(w)$: estado de M após ter lido string w

M aceita w se e só se $\phi(w) \in A$

$\phi(\epsilon) = q_0$

$\phi(wa) = \delta(\phi(w), a)$, para $w \in \Sigma^*$, $a \in \Sigma$

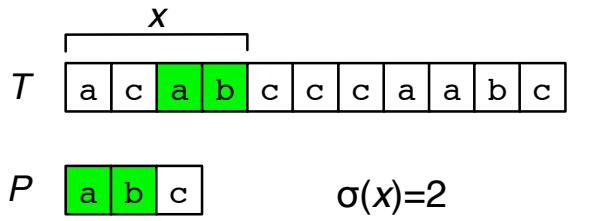
Função de Sufixo

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

$$\sigma : \Sigma^* \to \{0, 1, \ldots, m\}$$

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

Saber em que estado o autómato deve estar dada a string x

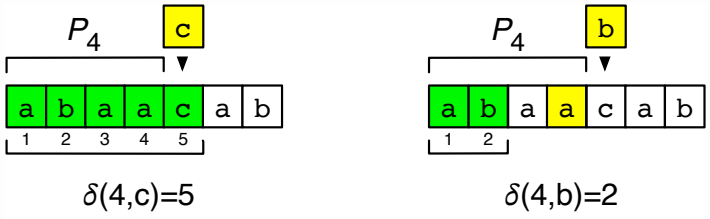


Autómato para Emparelhamento de Strings

Conjunto de estados: $Q = \{0, 1, \ldots, m\}$

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 $P_q a$

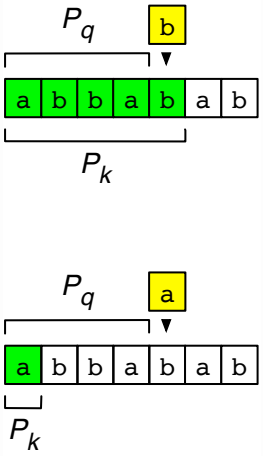


Finite-Automaton-Matcher(T, δ, m)

```
n = length[T]
q = 0
for i = 1 to n do
    q = δ(q, T[i])
    if q == m then
        print "Padrão encontrado com deslocação", i − m
    end if
end for
```

Compute-Transition-Function(P, Σ)

$m = length[P]$
for $q = 0$ **to** m **do**
  for each $a \in \Sigma$ **do**
    $k = \min(m+1, q+2)$
    repeat
      $k = k - 1$
    until $P_k \sqsupset P_q a$
    $\delta(q, a) = k$
  end for
end for
return δ



Compute-Transition-Function(P, Σ)

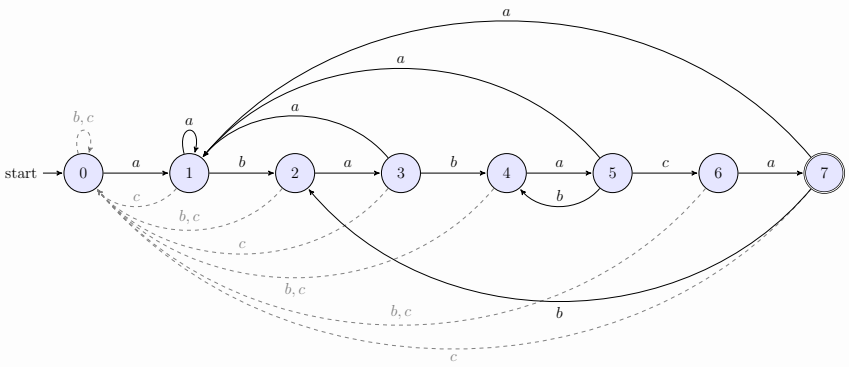
$m = length[P]$
for $q = 0$ **to** m **do**
  for each $a \in \Sigma$ **do**
    $k = \min(m+1, q+2)$
    repeat
      $k = k - 1$
    until $P_k \sqsupset P_q a$
    $\delta(q, a) = k$
  end for
end for
return δ

Análise de Complexidade

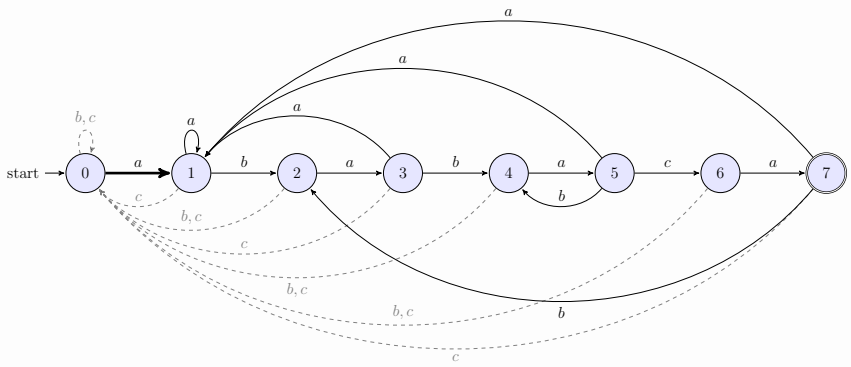
Complexidade do cálculo de δ: $O(m^3|\Sigma|)$

Complexidade do algoritmo: $O(n + m^3|\Sigma|)$

É possível obter $O(n + m|\Sigma|)$

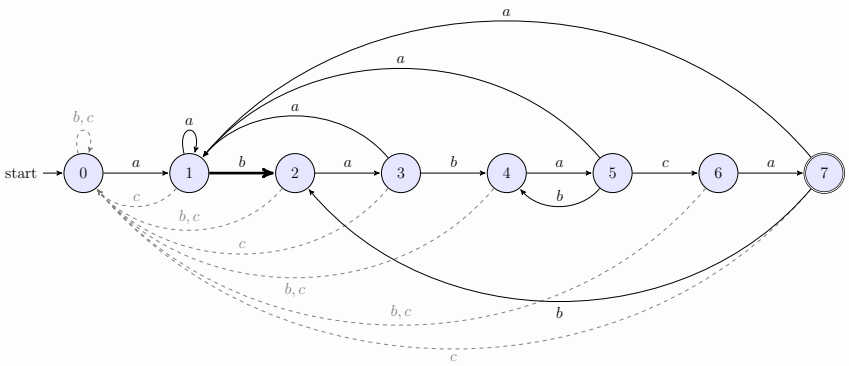


P = ababaca T = abcababacababacaabacab



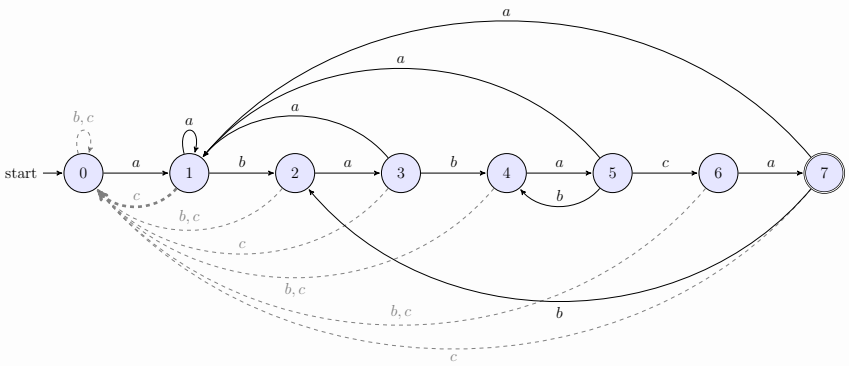
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



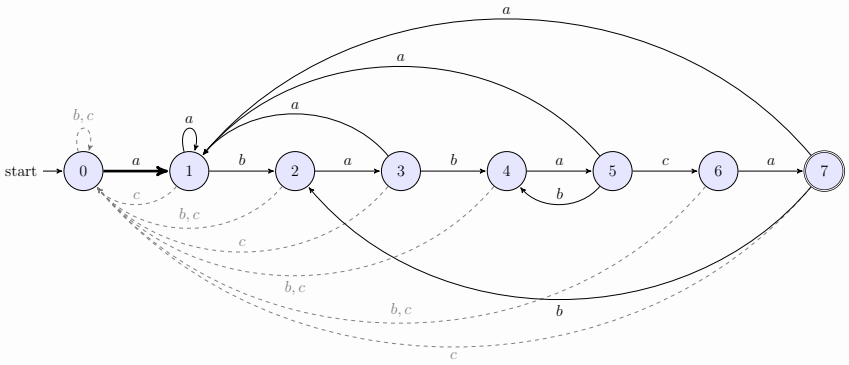
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



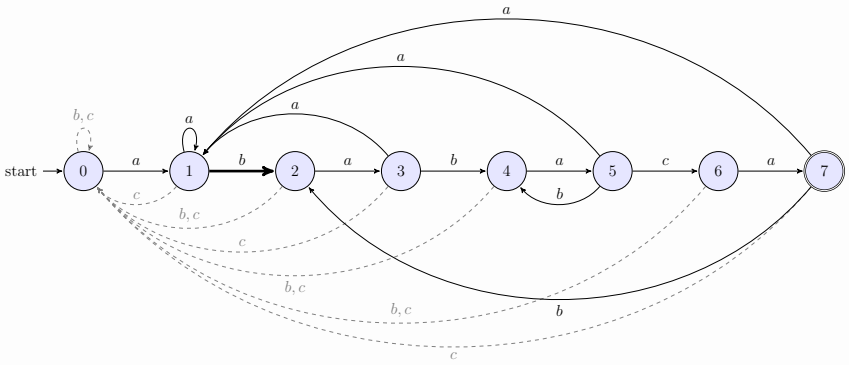
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



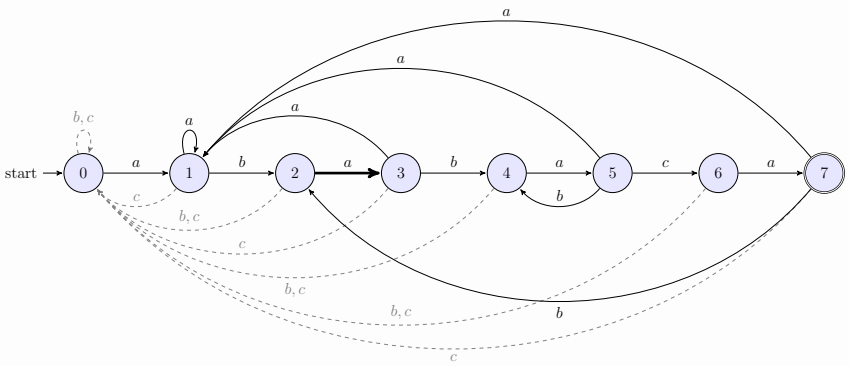
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



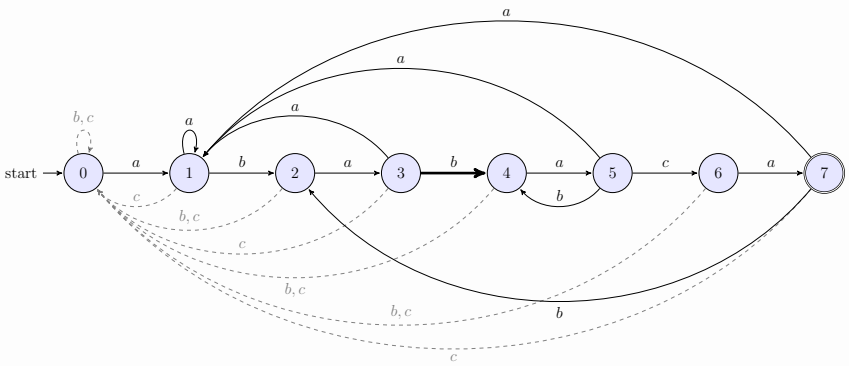
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



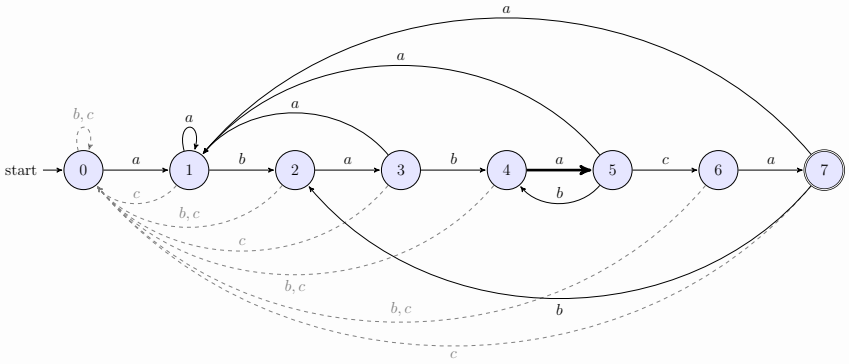
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



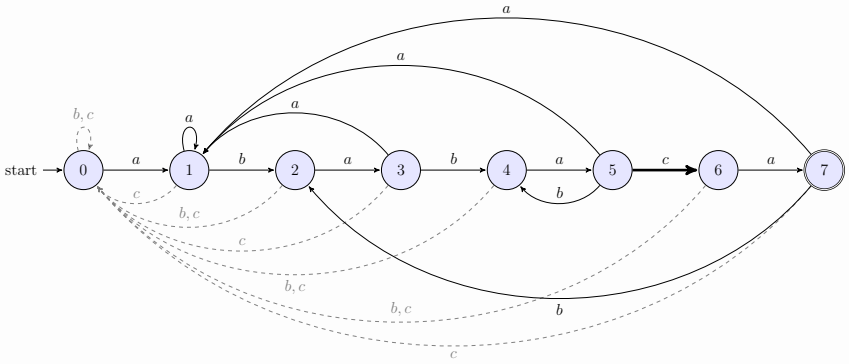
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



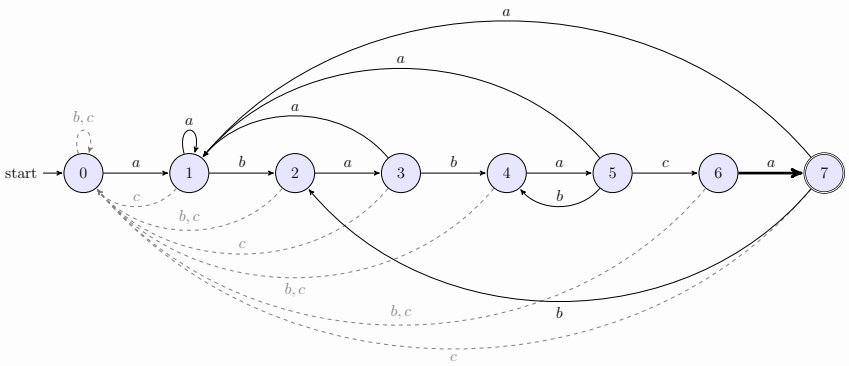
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



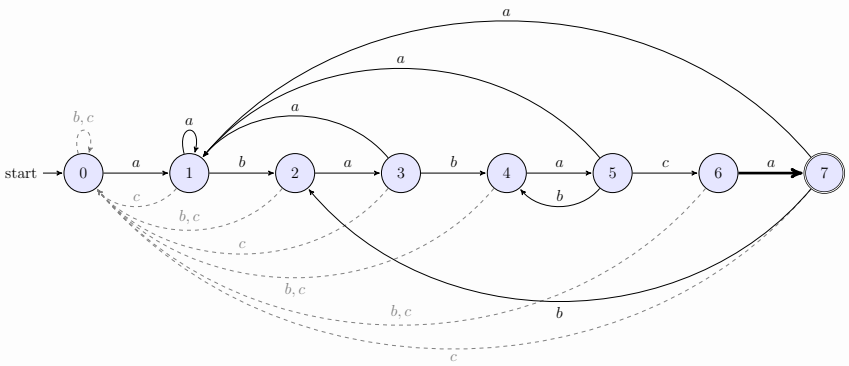
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



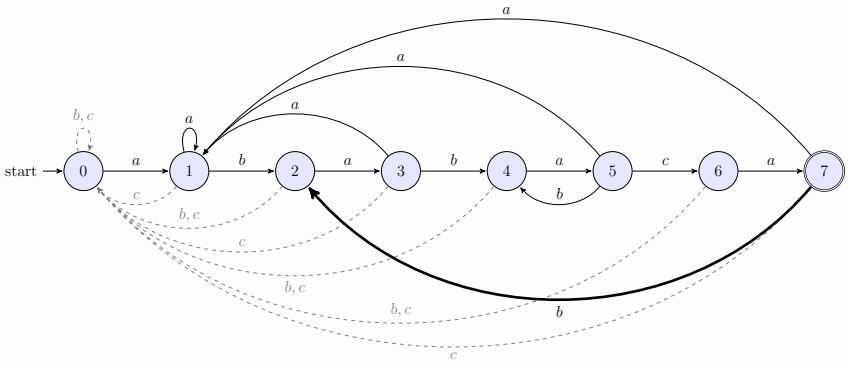
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



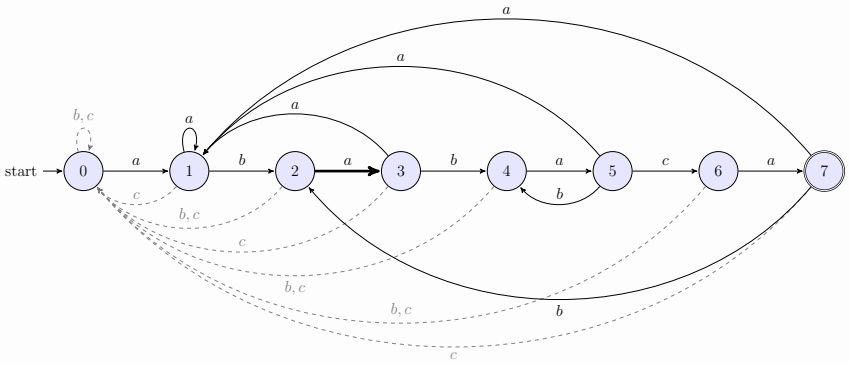
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



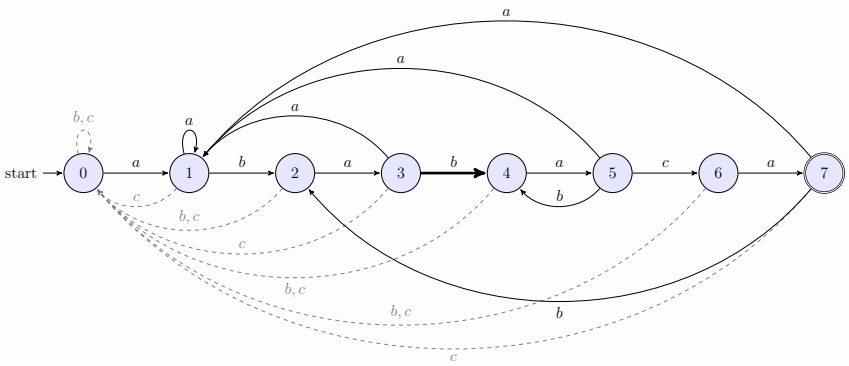
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



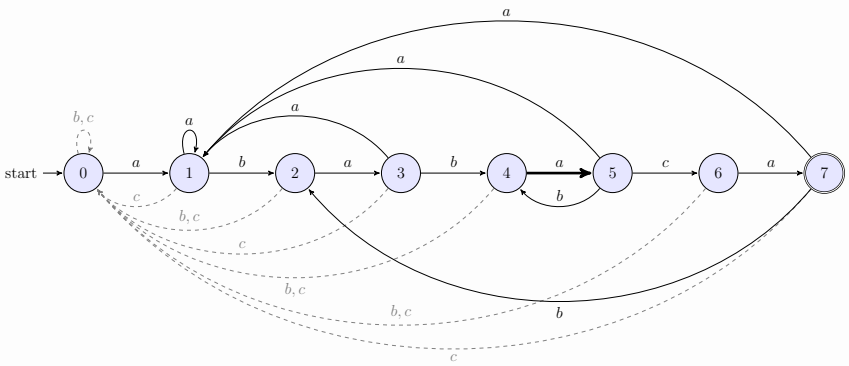
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



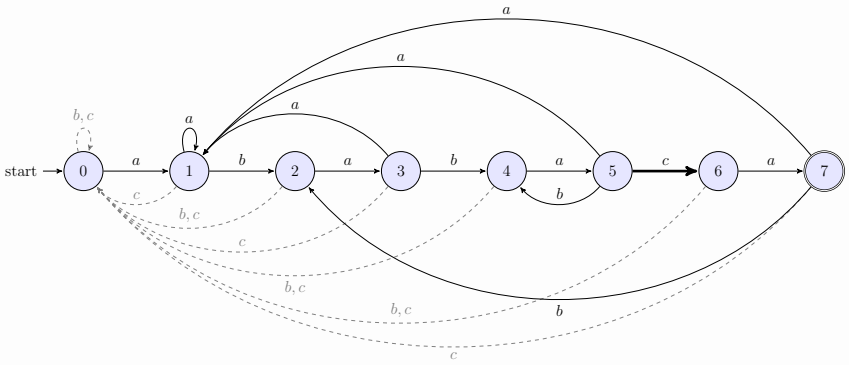
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



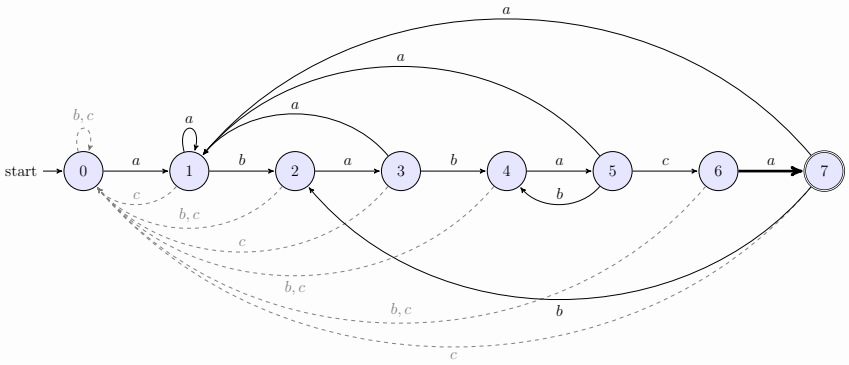
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



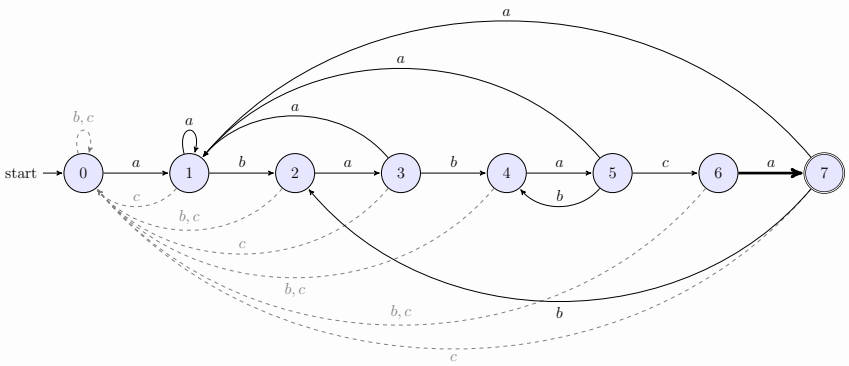
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



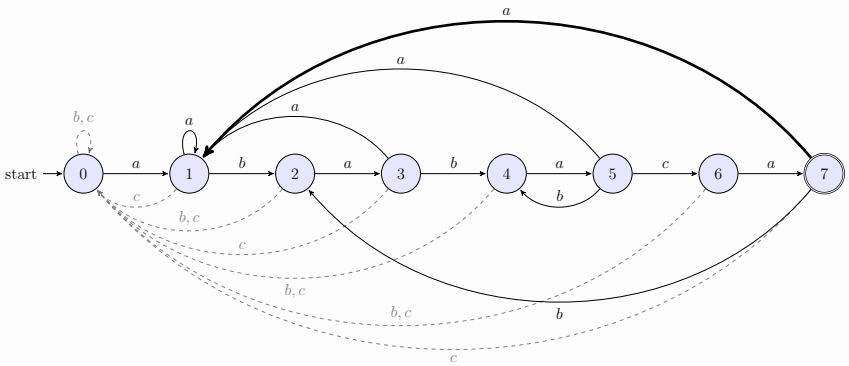
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



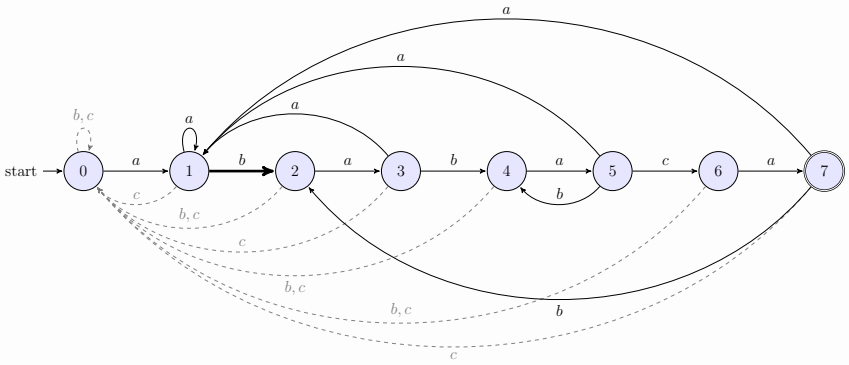
P = ababaca T = abcababacababacaabacab

Autómatos Finitos



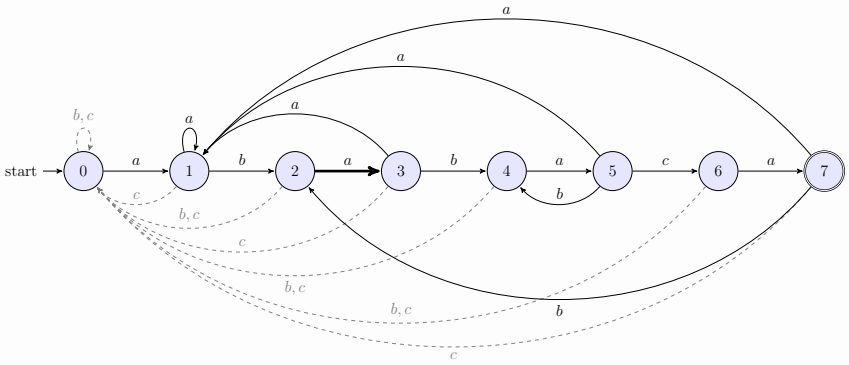
P = ababaca T = abcababacababacaabacab

Autómatos Finitos

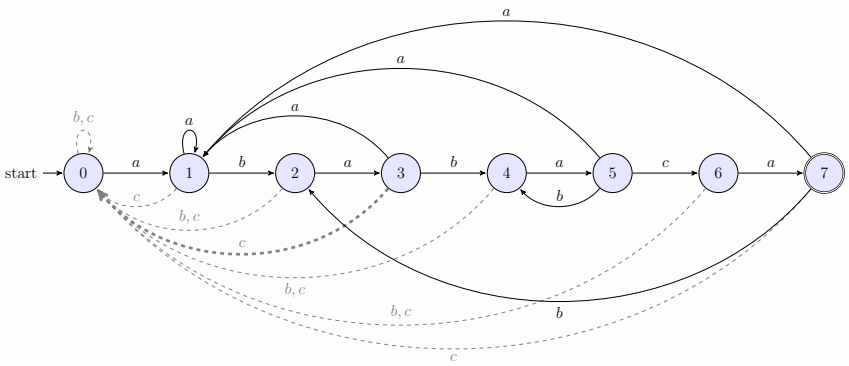


P = ababaca T = abcababacababacaabacab

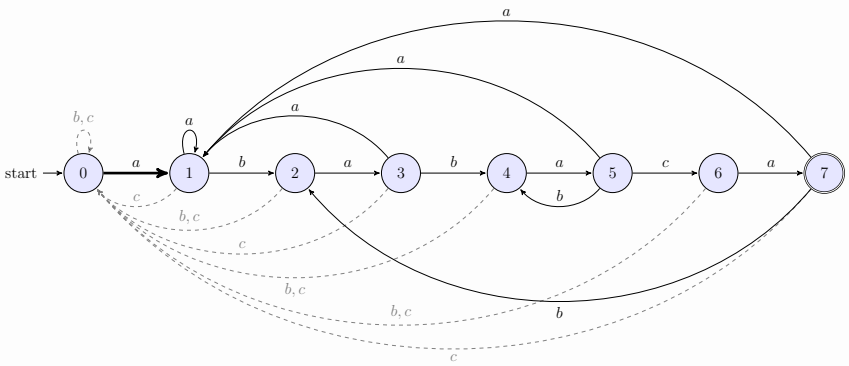
Autómatos Finitos



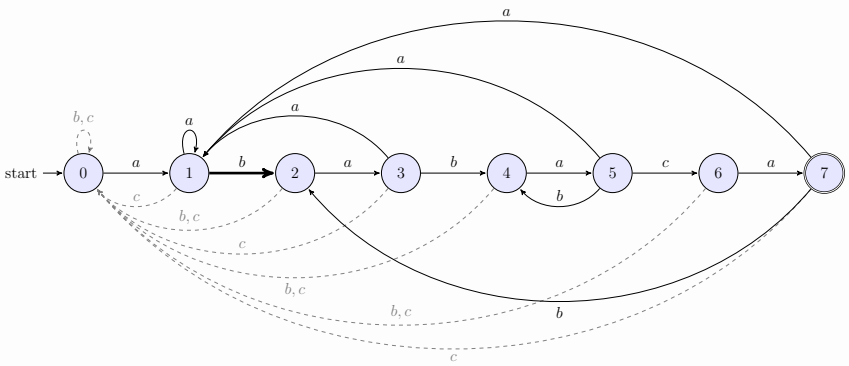
P = ababaca T = abcababacababacaabacab



P = ababaca T = abcababacababacaabacab



P = ababaca T = abcababacababacaabacab



P = ababaca T = abcababacababacaabacab

Vantagens

Complexidade: $O(n + m)$

Evita os deslocamentos consecutivos
do Algoritmo Elementar: $O((n - m + 1)\, m)$

Evita cálculo da função de transição δ
do Algoritmo de Autómatos Finitos: $O(n + m|\Sigma|)$

Função de Prefixo

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

$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

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
$\pi[i]$										

Função de Prefixo

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

$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

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									

Função de Prefixo

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

$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

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								

Função de Prefixo

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

$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

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							

Função de Prefixo

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

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

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

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

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

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

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					

Análise e Síntese de Algoritmos - 2022/2023 42/59

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

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

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

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

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				

Análise e Síntese de Algoritmos - 2022/2023 42/59

Algoritmo Knuth-Morris-Pratt

Função de Prefixo

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

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

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

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			

Análise e Síntese de Algoritmos - 2022/2023 42/59

Função de Prefixo

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

$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

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		

Função de Prefixo

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

$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

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	

Função de Prefixo

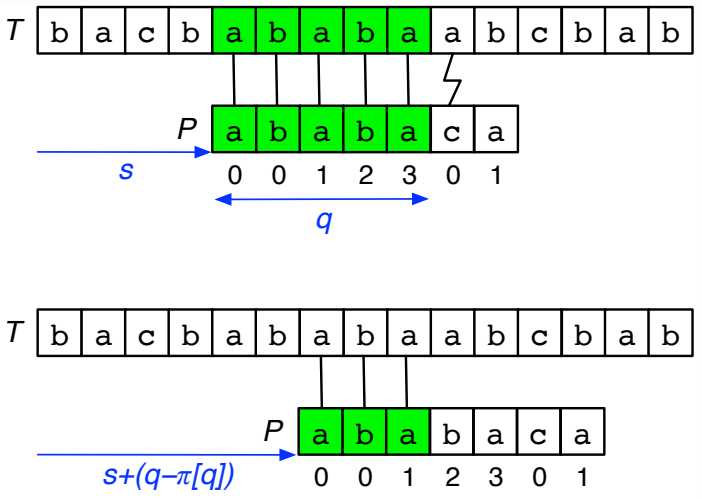
Comprimento do maior prefixo de $P[1..m]$ que é sufixo de P_q

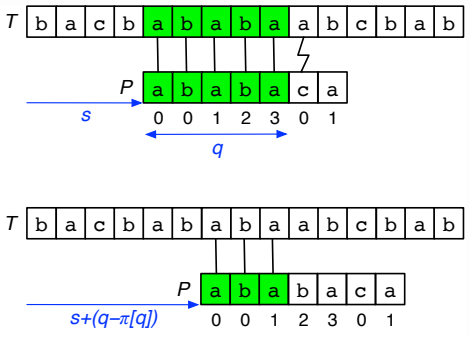
$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

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





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 $P[q+1] \neq T[i+1]$)

Funcionamento

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 $P_{\pi[k]}$

KMP-Matcher(T,P)

```
n = length[T]
m = length[P]
π = Compute-Prefix-Function(P)
q = 0
for i = 1 to n do
    while q > 0 ∧ P[q+1] ≠ T[i] do
        q = π[q]
    end while
    if P[q+1] == T[i] then
        q = q+1
    end if
    if q == m then
        print "Padrão encontrado com deslocação", i − m
        q = π[q]
    end if
end for
```


Compute-Prefix-Function(P)

```
m = length[P]
π[1] = 0
k = 0
for q = 2 to m do
    while k > 0 ∧ P[k+1] ≠ P[q] do
        k = π[k]
    end while
    if P[k+1] == P[q] then
        k = k+1
    end if
    π[q] = k
end for
return π
```

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

Intuição

Uso de uma rolling hash function

Comparar a função de hash do padrão contra o texto

Se for match, comparar os caracteres individuais

Se for match, padrão encontrado
Se não, match espúrio

Avançar no texto e voltar ao início

Algoritmo Rabin-Karp

Definições

Vamos assumir que $\Sigma = \{0, 1, 2, ..., 9\}$, 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 $s = 0, 1, ..., n - m$

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 $t_4 = 41567$

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 os 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 deslocamento s válidos em tempo $\Theta(m) + \Theta(n-m+1) = \Theta(n)$, comparando p com cada um dos valores de t_s

Aplicação da Regra de Horner

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

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 $t_1, t_2, ..., t_{n-m}$, em tempo $\Theta(n-m)$, basta observar que t_{s+1} pode ser calculado a partir de t_s, em tempo constante:

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

Exemplo (cont.)

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

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

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 $t_4 = 41567 = 10(t_3 - 10000 \times 1)) + 7$

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

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

Utilização do Módulo

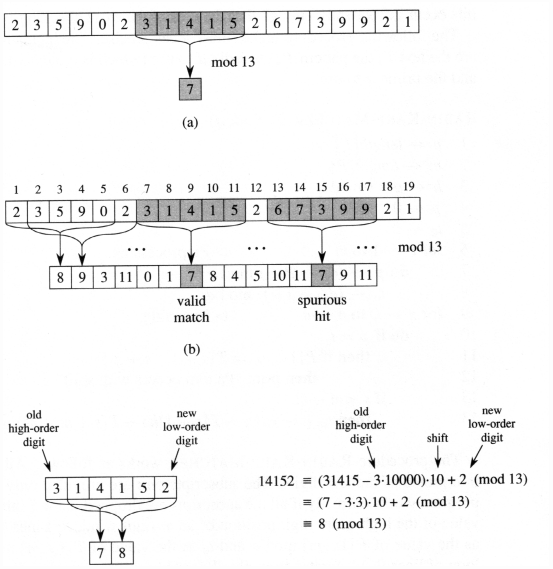
$t_s = p \mod q$ não implica necessariamente que $t_s = p$

$t_s \neq p \mod q$ implica que $t_s \neq p$

Podemos utilizar o teste $t_s = p \mod q$ como uma heurística rápida para invalidar certos deslocamentos s, quando $t_s \neq p \mod q$

Nos casos em que se verifica $t_s = p \mod q$, 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



Rabin-Karp-Matcher(T, P, d, q)

```
n = T.length
m = P.length
h = d^{m-1} mod q
p = 0
t_0 = 0
for i = 1 to m do
    p = (dp + P[i]) mod q
    t_0 = (dt_0 + T[i]) mod q
end for
for s = 0 to n − m do
    if p == t_s then
        if P[1..m] == T[s+1..s+m] then
            print "Pattern occurs with shift", s
        end if
    end if
    if s < n − m then
        t_{s+1} = (d(t_s − T[s+1]h) + T[s+m+1]) mod q
    end if
end for
```


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 $\Theta(m)$