Cin-UFPE
String Matching

Rogério dos Santos Rosa

Flávia Roberta Barbosa Araújo

{rsr, frba} @cin.ufpe.br

Recife, Junho de 2008

Roteiro

- Introdução
- Breve histórico
- Algoritmos:
  - **Ingênuo ou Força Bruta**
  - **Rabin-Karp**
  - **Finite automaton**
  - **Knuth-Morris-Pratt**

Introdução

- **Objetivo:**
  - Encontrar uma cadeia de caracteres e geralmente, todas as ocorrências dessa cadeia (conhecida como padrão) em um determinado texto.

- **Utilidade:**
  - Apesar das várias formas de armazenar dados, o texto continua a ser a principal forma de intercâmbio de informações. Isto aplica-se na informática onde uma grande quantidade de dados são armazenados em arquivos lineares. Assim como na biologia molecular, pois muitas vezes as moléculas biológicas podem ser descritas como seqüências de nucleotídeos ou aminoácidos (cadeias de caracteres muito longas).
  - Por esta razão os algoritmos devem ser eficientes para conseguirem tornar essa grande quantidade de informação manipulável, mesmo quando a velocidade e a capacidade de armazenamento dos computadores aumentam regularmente.

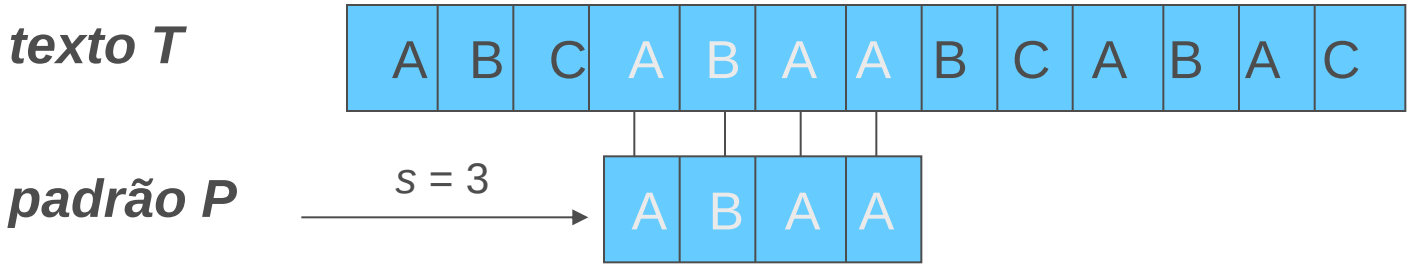
Introdução

- Assumindo então que:
  - **Texto é um array *T*[1..*n*]**
  - **Padrão é um array *P*[1..*m*] , *m* ≤ *n***
  - **Sendo estes arrays de caracteres: *T* e *P* de um mesmo alfabeto finito Σ.**

  Por exemplo Σ = {0, 1} ou Σ = {a, b,..., z}.
  - **|Σ| tamanho do alfabeto.**

Introdução

- Problema *String-matching*:
  - **Diz-se que um padrão ocorre com deslocamento s em um texto** T**.**
  - **Se 0 ≤** s **≤ *n* – *m***
  - **T[s + 1 .. s + *m*] = P[1 .. *m*]**
  - **Se P ocorre em T com deslocamento** s**. Esta é dito como** *deslocamento válido***.**



Breve Histórico

- **Algoritmo Ingênuo ou Força Bruta:**
  - É o algoritmo mais óbvio de busca em cadeia, tem o pior caso de tempo de execução proporcional a *mn.*
  - Embora as cadeias que apareçam em muitas aplicações levam a um tempo de execução que é virtualmente proporcional a *m* + *n.*
- **Reconhecimento por Automato Finito Determinístico:**
  - Em 1970, S. A. Cook provou um resultado teórico sobre um tipo particular de autômato que implicava na existência de um algoritmo de casamento de padrão com tempo proporcional a *M* + *N* no pior caso.

Breve Histórico

- **Algoritmo Knuth-Pratt-Morris:**
  - D. E. Knuth e V. R. Pratt seguindo a construção que Cook usaram na demonstração do seu teorema e obtiveram um algoritmo relativamente simples e prático.
  - Ocorreu também que J. H. Morris descobriu praticamente o mesmo algoritmo como solução de um problema de edição de texto.
  - Os três cientistas, Knuth, Morris e Pratt, publicaram conjuntamente o algoritmo em 1977.

Breve Histórico

- **Algoritmo Rabin-Karp:**
  - Em 1980, M. O. Rabin e R. M. Karp desenvolveram um algoritmo tão simples quanto o de força bruta que roda virtualmente sempre em tempo proporcional a $m + n$.
  - Além disso, o algoritmo deles estende-se facilmente a padrões bidimensionais que o torna mais útil que os outros para processamento de figuras.

Tempo de processamento dos dados

- Com exceção do algoritmo de Força Bruta, todos os outros que serão apresentados têm uma etapa anterior ao matching de pré-processamento do padrão.
- Sendo o tempo total do algoritmo o tempo de processamento mais o tempo de matching.

Algoritmo	Tempo de PP	Tempo de Matching
Ingênuo	0	$O((n - m + 1)m)$
Rabin-Karp	$\Theta(m)$	$O((n - m + 1)m)$
Autômato Finito	$O(m\|\Sigma\|)$	$\Theta(n)$
KMP	$\Theta(m)$	$\Theta(n)$

Ingênuo ou Força Bruta

- O algoritmo procura por todos os deslocamentos **s** válidos, usando um ciclo para checar a seguinte condição:

  P[1 .. *m*] = T[s + 1 .. s + *m*] para cada

  n – *m* + 1 possível valor de **s**.

Ingênuo ou Força Bruta

Algoritmo

```
NAIVE-STRING-MATCHER(T, P)
1 n ← length[T]
2 m ← length[P]
3 for s ← 0 to n - m
4     do if P[1 □ m] = T[s + 1 □ s + m]
5           then print "Pattern occurs with shift" s
```

O tempo de complexidade do algoritmo no pior caso é O((n – m + 1)m). Serão feitas comparações para cada deslocamento s, de acordo com o tamanho do padrão *m*.

http://www-igm.univ-mlv.fr/~lecroq/string/node3.html#SECTION0030

Rabin Karp

- **Princípio:** tratar o texto como dados numéricos e não realizar comparações diretamente entre os caracteres

  - O padrão *P* ocorre no texto *T* se o valor calculado para *P* for igual ao valor calculado para qualquer substring *X* de *T, de* tamanho *m,* tal que | *X* | = | *P* |
  - Os valores calculados para cada substring de *T* não precisam ser previamente calculados
  - Valores gerados são normalmente muito longos, necessitando de uma estratégia
  - Realiza pré-processamento do padrão *P* em tempo $O(m)$
  - Realiza o *matching* de *P* em *T,* no pior caso em tempo $O((n-m+1)m)$
  - No caso médio o tempo de *matching* é linear $O(n)$

Rabin Karp

Pré-Processamento do Padrão

$\sum = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $|\sum| = 10$

P	1	9	9	1

temos:

$(P[1] * 10 + P[2]) = 19$
$(19 * 10) + P[3] = 199$
$(199 * 10) + P[4] = 1991$

Generalizando:

$$P[m] + |\sum| (P[m-1] + |\sum| (P[m-2] + \ldots + |\sum| (P[2] + |\sum| P[1])))$$

- $\sum$ = alfabeto
- $|\sum|$ = tamanho de $\sum$
- Dado um caractere, a representação numérica deste será sua posição no alfabeto $\sum$
- Complexidade $O(m)$

Rabin Karp

Processamento do Texto

Padrão P

1	9	9	1

Texto T

1	8	8	7	1	9	9	1	2	0	0	0	5

1	8	8	7

= 1887 tempo ***O*(*m*)**, para ***s* = 0**

8	8	7	1

= 8871 **não usaremos tempo *O*(*m*) para *s* > 0**, pois temos que:

$(1887 - |\sum|^{m-1} * P[1]) * |\sum| + P[s+m] = 8871$,
onde $|\sum|^{m-1}$ foi previamente calculado

Portanto temos tempo ***O*(1)**, para cada deslocamento ***s* > 0**

s = 0, temos ***O*(*m*)**
s > 0, temos ***O*(1)**
s variando de 0 à *n* – *m*
para calcular $|\sum|^{m-1}$ temos ***O*(lg *m*)**
= *O*(*m*) + (*n* – *m*)*O*(1) + *O*(lg *m*)
= *O*(*n*)

Rabin Karp
Problemas

Os valores das transformações de *P* e das substrings de T são muito grande, quando *m* e |∑| são muito longos

Solução 1: reduzir esses valores a uma faixa controlada, utilizando módulo de um número *q*, por exemplo.

Novo problema: um mesmo valor pode representar substrings distintas.

Solução 2: ocorrendo um provável casamento de *P* com uma substring *X* de *T*, cada caractere de *P* deve ser comparado a cada caractere de *X*, para verificar se o casamento realmente acontece.

Rabin Karp
Algoritmo

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n ← length[T]
 2  m ← length[P]
 3  h ← d^(m-1) mod q
 4  p ← 0
 5  t_0 ← 0
 6  for i ← 1 to m                    ▹ Preprocessing.
 7      do p ← (dp + P[i]) mod q
 8         t_0 ← (dt_0 + T[i]) mod q
 9  for s ← 0 to n - m                ▹ Matching.
10      do if p = t_s
11            then if P[1 □ m] = T [s + 1 □ s + m]
12                    then print "Pattern occurs with shift" s
13         if s < n - m
14            then t_s+1 ← (d(t_s - T[s + 1]h) + T[s + m + 1]) mod q
```

http://www-igm.univ-mlv.fr/~lecroq/string/node5.html#SECTION0050

Rabin Karp
Análise

Já vimos que:

- custo para pré-processamento do padrão *P* é ***O*(*m*)**
- custo para processamento do texto *T* é ***O*(*n*)**
- número máximo de deslocamentos *s* válidos é $n - m + 1$

Agora suponha que, no pior caso:

- Todas as substrings *X* de *T* casam com *P*

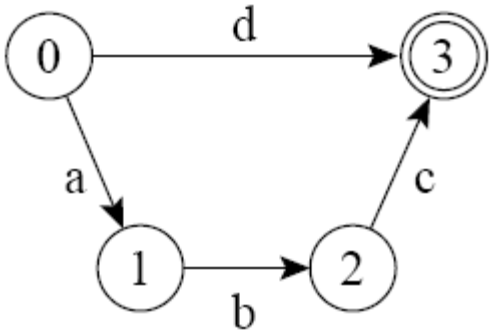
Sabemos que o número de deslocamentos *s* válidos é $n - m + 1$, então temos *s* possíveis *X,* sabemos também que $|X| = |P| = m$, é possível concluir então que para cada *s* faremos *m* comparações, então: ***O*((*n*-*m*+1)*m*).**

Reconhecimento por Autômato

Autômato é um modelo de computação simples

A *finite automaton* M is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$, where

- Q is a finite set of ***states***,
- $q_0 \in Q$ is the ***start state***,
- $A \subseteq Q$ is a distinguished set of ***accepting states***,
- Σ is a finite ***input alphabet***,
- δ is a function from $Q \times \Sigma$ into Q, called the ***transition function*** of M.



Autômato determinístico

Reconhecimento por Autômato

Funções

Função de transição entre estados

δ(q, a) dado o estado atual *q* e o caractere lido “a”, a função retorna o próximo estado;

Função de estado final

φ(w) terminada de ler toda a string *w,* a função retorna o estado do autômato, ao final da string *w.*

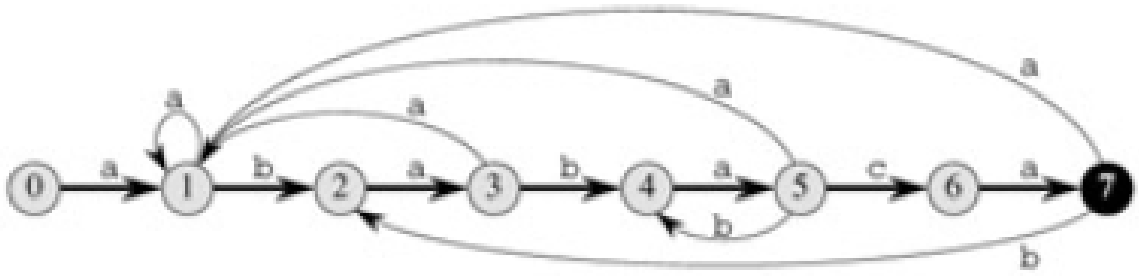
Função de sufixo

$P_k \sqsupset x$ sufixo

σ(x) = max $\{k : P_k \sqsupset x\}$, tamanho do maior prefixo de *P* que é sufixo de *x.*

Reconhecimento por Autômato

Construção do Autômato



(a)

state	input a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

(b)

Padrão P

a	b	a	b	a	c	a

i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

(c)

Reconhecimento por Autômato

Definição

Dado um Padrão P[1..*m*]

• Conjunto de estados *Q* {0, 1, ..., *m*}, sendo estado inicial = q_0 e estado aceito = *m*

• A função de transição *δ* é definida pela equação abaixo, para qualquer estado *q* e caractere *a.*

$$\delta(q,a) = \sigma(P_q a)$$

Isto significa que depois de lido os primeiros *i* caracteres de *T:*

• o autômato está no estado $\varphi(T_i) = q$
• onde $q = \sigma(T_i)$
• o próximo caractere é $T[i + 1] = a$
• então a transição será $\sigma(T_{i+1}) = \sigma(T_i a)$
• em cada estado o autômato conhecer somente o tamanho do maior prefixo de *P* que é sufixo da substring lida até o momento, então, temos que $\delta(q, a) = \sigma(Pqa)$

Reconhecimento por Autômato

Algoritmo

```
FINITE-AUTOMATON-MATCHER(T, δ, m)
1   n ← length[T]
2   q ← 0
3   for i ← 1 to n
4       do q ← δ(q, T[i])
5          if q = m
6             then print "Pattern occurs with shift" i - m
```

Reconhecimento por Autômato

Função de Transição

Função de Transição

```
COMPUTE-TRANSITION-FUNCTION(P, Σ)
1 m ← length[P]
2 for q ← 0 to m
3     do for each character a ∈ Σ
4            do k ← min(m + 1, q + 2)
5               repeat k ← k - 1
6                 until P_k ⊐ P_q a
7               δ(q, a) ← k
8 return δ
```

A complexidade dessa função é $O(m^3|\sum|)$, entretanto o código das linhas 5 e 6 pode ser alterado resultando em uma complexidade final $O(m|\sum|)$.

Algoritmo Knuth-Morris-Pratt

Definição

Algoritmo de tempo linear

• O KMP é baseado no algoritmo de reconhecimento por autômato, simplificando a função de transição (***δ***).

•O tempo de *matching* é ***Θ*(*n*)** usando apenas uma função auxiliar **π[1..*m*]** que é pre-computada a partir do padrão no tempo ***Θ*(*m*)**.

• De grosso modo, para qualquer estado $q = 0, 1, \ldots, m$ e qualquer caracter $a \subset \Sigma$, o valor π[*q*] contém a informação que é independente de *a* e é necessária para calcular **δ(*q*, *a*)**.

• Dado que o array π tem apenas *m* entradas, considerando que δ tem Θ(m|Σ|) entradas, uma fração de |Σ| é usada no pré-processamento para computar π em vez de δ.

Algoritmo Knuth-Morris-Pratt

Função prefixo para o padrão

Função Prefixo

• A função prefixo π para um padrão encapsula conhecimento sobre o modo como o padrão casa contra os deslocamentos de si próprio.

• Esta informação pode ser usada para evitar testes de deslocamentos desnecessários como no algoritmo ingênuo ou para evitar a pre-processamento de δ para um autômato string-matching.

Algoritmo Knuth-Morris-Pratt

Função prefixo para o padrão

Função Prefixo

• Tendo P = ababaca contra um texto *T.*

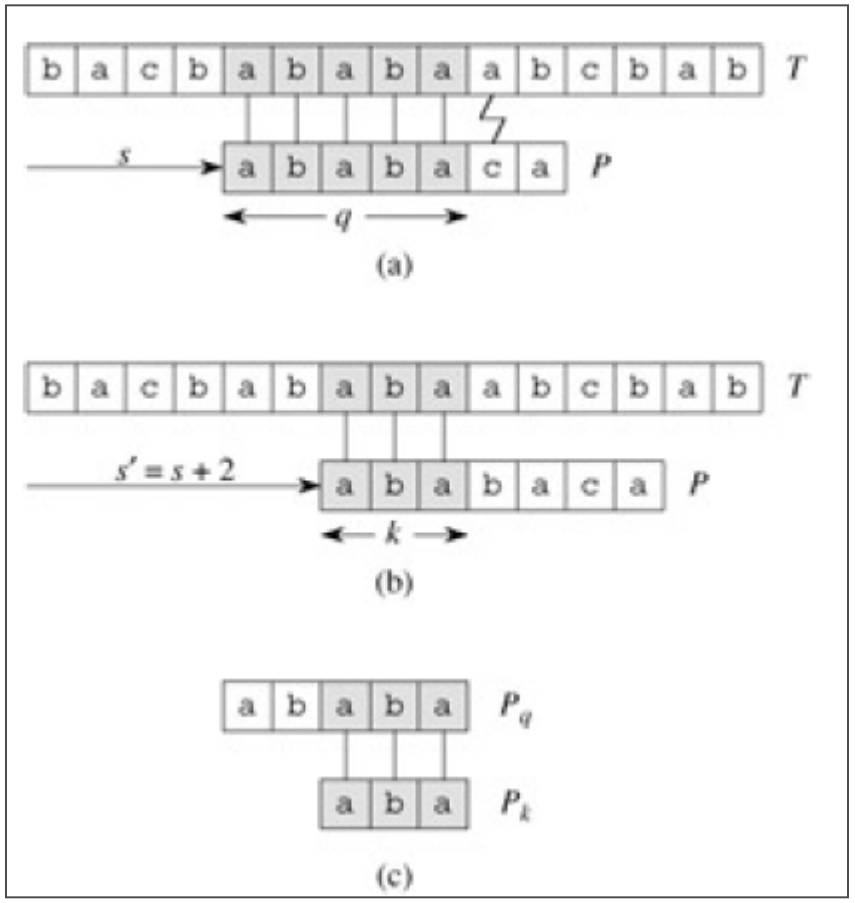
• Em (a) sendo, *q* = 5, de caracteres que pareram com *T*.

• Conhecendo estes q caracteres do texto é possível determinar que alguns deslocamentos *s* são inválidos (não precisam ser testados).

• O deslocamento s’ = s + 1 é inválido, mas o deslocamento s’ = s + 2 é potencialmente válido pelo que conhecemos do texto.

• Dado que q caracteres tiveram comparações com sucesso no deslocamento s, o próximo potencial deslocamento válido será:

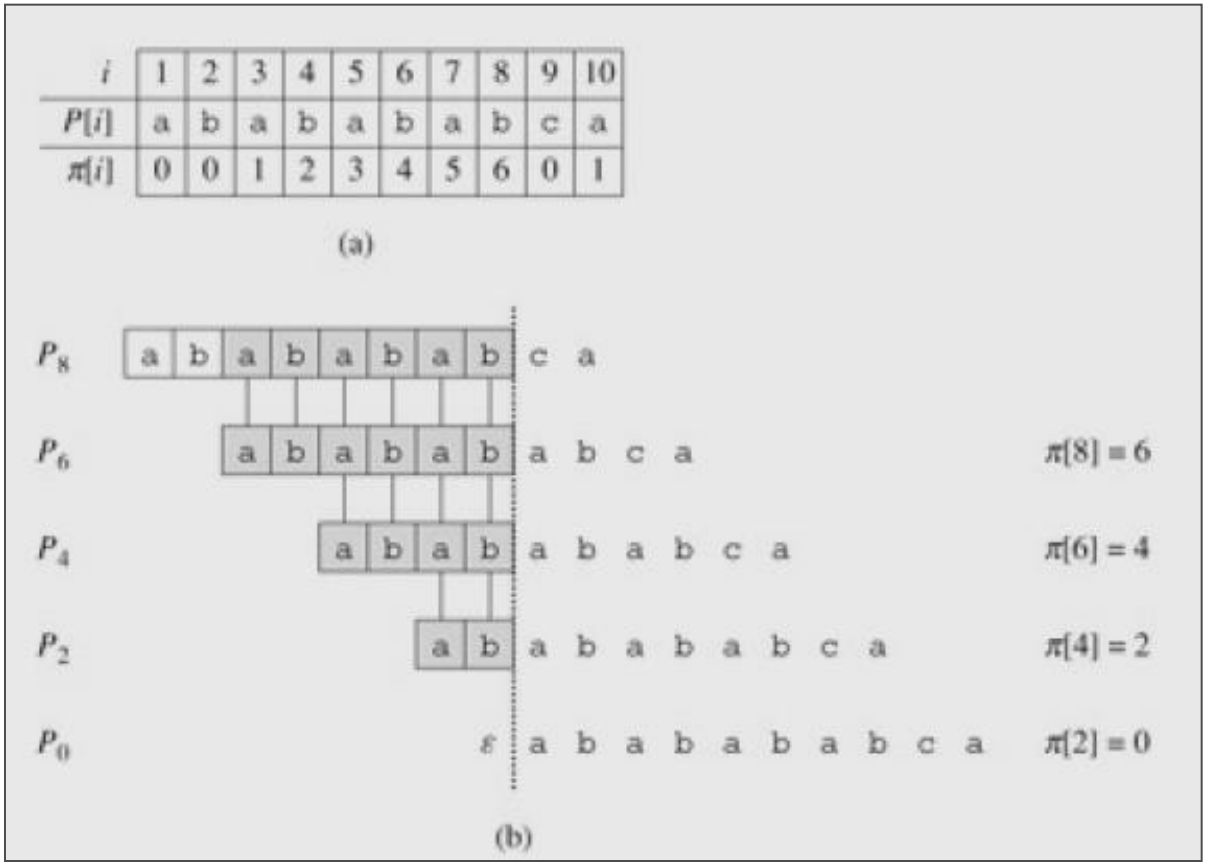
s’ = s + (q – π[q])



Algoritmo Knuth-Morris-Pratt

Função prefixo para o padrão

$$\pi[q] = \max \{k : k < q \text{ and } P_k \sqsupset P_q\}.$$



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

(a)

(b)

Algoritmo Knuth-Morris-Pratt

Algoritmo

```
KMP-MATCHER(T, P)
 1 n ← length[T]
 2 m ← length[P]
 3 π ← COMPUTE-PREFIX-FUNCTION(P)
 4 q ← 0                              ▹Number of characters matched.
 5 for i ← 1 to n                     ▹Scan the text from left to right.
 6      do while q > 0 and P[q + 1] ≠ T[i]
 7           do q ← π[q]              ▹Next character does not match.
 8         if P[q + 1] = T[i]
 9           then q ← q + 1           ▹Next character matches.
10         if q = m                        ▹Is all of P matched?
11           then print "Pattern occurs with shift" i - m
12                q ← π[q]              ▹Look for the next match.
COMPUTE-PREFIX-FUNCTION(P)
 1 m ← length[P]
 2 π[1] ← 0
 3 k ← 0
 4 for q ← 2 to m
 5      do while k > 0 and P[k + 1] ≠ P[q]
 6           do k ← π[k]
 7         if P[k + 1] = P[q]
 8           then k ← k + 1
 9         π[q] ← k
10 return π
```

Bibliografia

- http://www-igm.univ-mlv.fr/~lecroq/string/
- http://en.wikipedia.org/wiki/String_searching_algorithm
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms, second edition, MIT Press and McGraw-Hill.