Algoritmos em Strings

R. Rossetti, A. P. Rocha, J. Pascoal Faria

FEUP, MIEIC, CAL, 2013/2014

Índice

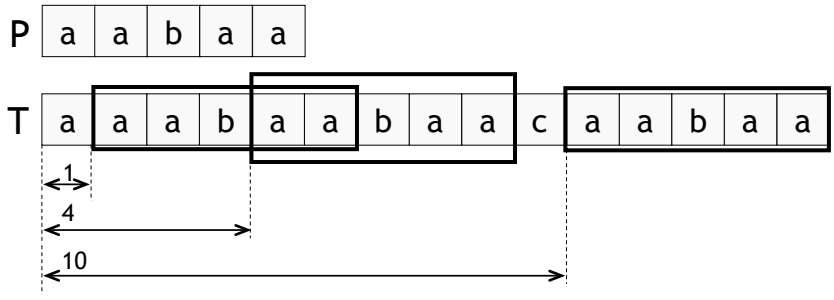
- Pesquisa exata (*string matching*)
- Pesquisa aproximada (*approximate string matching*)
- Outros problemas

Pesquisa exata

(*string matching*)

Problema

- Encontrar todas as ocorrências de um padrão P num texto T
  - P e T são cadeias de caracteres
  - Ocorrências são definidas pela deslocação em relação ao início do texto
  - Ocorrências podem ser sobrepostas



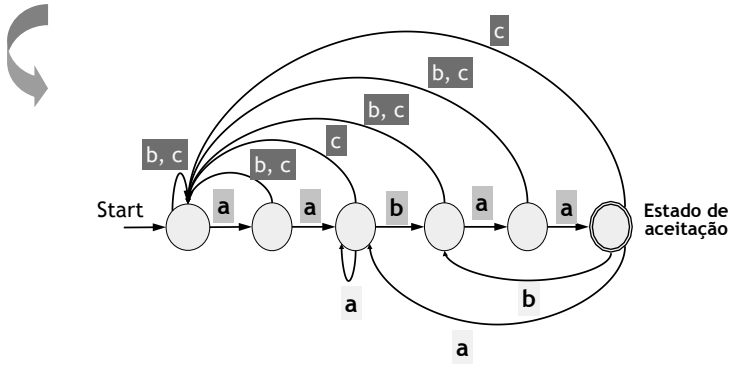
Algoritmos

- Algoritmo *naive*
  - Para cada deslocamento possível, compara desde o início do padrão
  - Ineficiente se o padrão for comprido: O(|P|.|T|)
- Algoritmo baseado em autómato finito
  - Pré-processamento: gerar autómato finito correspondente ao padrão
  - Permite depois analisar o texto em tempo linear O(|T|), pois cada carácter só precisa de ser processado uma vez
  - Mas tempo e espaço requerido pelo pré-processamento pode ser elevado: O(|P|.|Σ|), em que |Σ| é o tamanho do alfabeto
- Algoritmo de Knuth-Morris-Pratt
  - Efetua um pré-processamento do padrão em tempo O(|P|), sem chegar a gerar explicitamente um autómato, seguido de processamento do texto em O(|T|), dando total O(|T|+|P|)

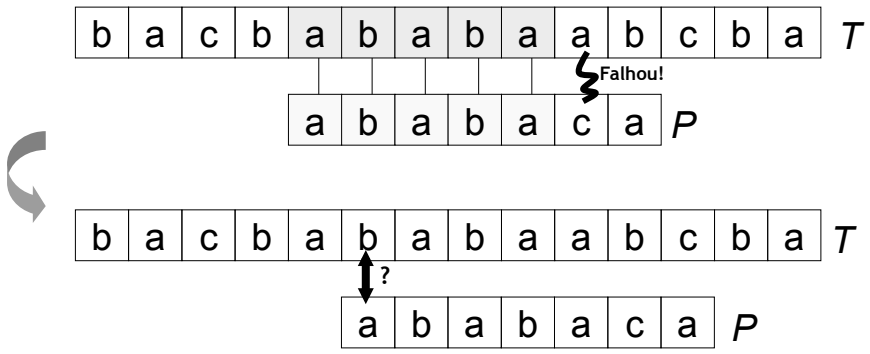
Autómato finito correspondente ao padrão

P | a | a | b | a | a |

Σ = {a, b, c}

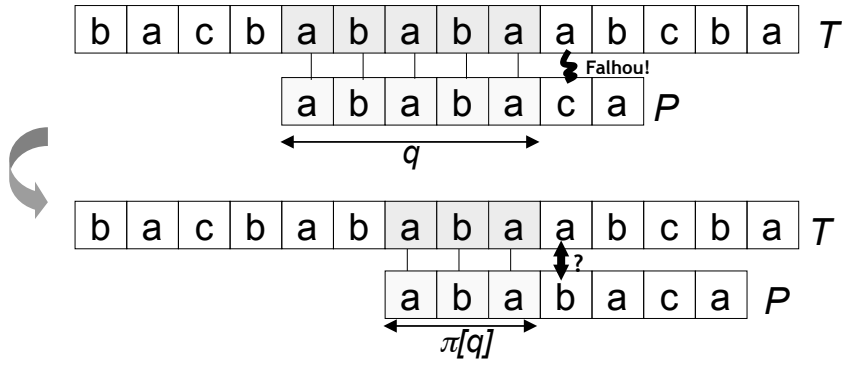


Algoritmo *naive*



Desloca-se o padrão uma casa para a direita e recomeça-se a comparação do início do padrão! Ineficiente: O(|P| |T|)

Algoritmo de Knuth-Morris-Pratt



Desloca-se o padrão para a direita de uma forma que permite continuar a comparação na mesma posição do texto! Evita comparações inúteis!

Deslocamento é determinado por uma função π[q] calculada numa fase de pré-processamento do padrão!

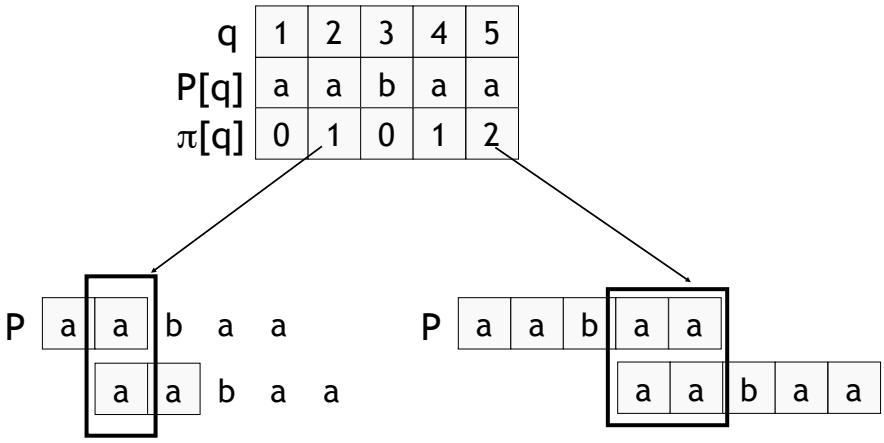
Pré-processamento do padrão

- Compara-se o padrão com deslocações do mesmo, para determinar a **função prefixo**

$$\pi[q] = \max \{k:\ 0 \leq k < q \text{ e } P[1..k] = P[(q-k+1)..q] \}$$

- q = 1, …, |P|
- P[i…j] – substring entre índices i e j
- Índices a começar em 1
- π[q] é o comprimento do maior prefixo de P que é um sufixo próprio do prefixo de P de comprimento q

Pré-processamento do padrão



Pseudo-código

```
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.
```


Pseudo-código

```
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
```

Eficiência do algoritmo Knuth-Morris-Pratt *

- KMP-MATCHER (sem incluir COMPUTE-PREFIX-FUNCTION) *
  - Eficiência depende do nº de iterações do ciclo "while" interno
  - Dado que $0 \leq \pi[q] < q$, cada vez que a instrução 7 é executada, o valor de q é decrementado de pelo menos *1*, sem nunca chegar a ser negativo
  - Dado que o valor de q começa em 0 e só é incrementado no máximo n vezes (+1 de cada vez, na linha 9), o nº máximo de vezes que pode ser decrementado (nas linhas 7 e 12) é também n

  ⇒ Nº máximo de iterações do ciclo "while" interno (no conjunto de todas as iterações do ciclo "for" externo) é n

  ⇒ Tempo de execução da rotina é $O(n)$, i.e., $O(|T|)$
- COMPUTE-PREFIX-FUNCTION
  - Seguindo o mesmo raciocínio, tempo de execução é $O(m)$, i.e., $O(|P|)$
- Total: O(n+m),isto é, **O(|T| + |P|)**