PCC104 - Projeto e Análise de Algoritmos

Marco Antonio M. Carvalho

Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto

UFOP
Universidade Federal de Ouro Preto

decom
departamento de computação

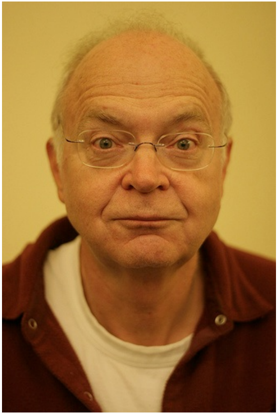
Conteúdo

Fonte

Este material é baseado nos livros

- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- S. Halim. *Competitive Programming*. 3rd Edition, 2013.
- Ian Parberry and William Gasarch. *Problems on Algorithms*. Second Edition, 2002.
- Ian Parberry *Lecture Notes on Algorithm Analysis and Complexity Theory*. Fourth Edition, 2001.

Licença

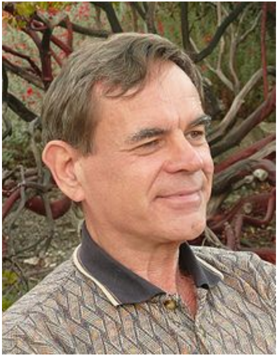


Donald Knuth (*1938)

- Cientista da Computação americano;
- Prêmio Turing em 1974;
- John von Neumann Theory Prize 1995;
- National Medal of Science 1979;
- Kyoto Prize 1996;
- Famoso por:
  - TeX;
  - Concrete Mathematics: A Foundation for Computer Science;
  - The Art of Computer Programming;
  - Cheques de US$2,56;
  - etc...

James H. Morris (*1941)

- Cientista da Computação americano;
- Dean da *Carnegie Mellon School of Computer Science* e da *Carnegie Mellon Silicon Valley*;
- Famoso por:
  - Inter-module protection;
  - Lazy evaluation;
  - Xerox Alto System;
  - Andrew Project;
  - Maya Design;
  - KMP;
  - etc...



Vaughn Pratt (*1944)

- Cientista da Computação americano;
- Professor emérito de Stanford;
- Famoso por:
  - Vários algoritmos;
  - Contribuições em algoritmos de busca;
  - Contribuições em algoritmos de ordenação;
  - Contribuições em algoritmos de teste de primalidade;
  - etc...

Introdução

O algoritmo de busca de padrões **Knuth-Morris-Pratt** (ou **KMP**) foi proposto por Donald Knuth e Vaughan Pratt em 1974 e também, independentemente, por James H. Morris;

Os três, conjuntamente, publicaram o algoritmo em 1977;

Este algoritmo utiliza uma função prefixo para o padrão, denominada **função** π para evitar testar deslocamentos inválidos como no algoritmo ingênuo de *matching*;

A idéia básica é, quando há uma diferença entre $P[j]$ e $T[i]$, realizar um deslocamento maior de P à direita, de maneira a evitar comparações redundantes.

```
T=...abaab|bba
P=   abaab|
P=     --ab|aab
```

Princípio

Consideremos a operação do algoritmo de *matching* ingênuo;

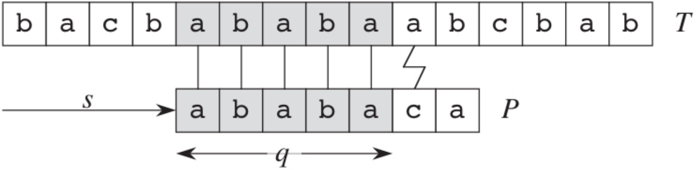
A figura a seguir apresenta um deslocamento s de um gabarito contendo o padrão $P=$ ababaca em relação a um texto T;

No exemplo, q=5 dos caracteres coincidiram com o texto, mas o sexto caractere do padrão não coincidiu;

A informação de que q caracteres coincidiram com sucesso determina quais são os caracteres correspondentes no texto;

Conhecendo estes q caracteres do texto, podemos determinar imediatamente que determinados deslocamentos são inválidos;

No mesmo exemplo, o deslocamento $s+1$ é inválido, porque o primeiro caractere do padrão (a) estaria alinhado com um caractere que sabemos não ser coincidente (b).



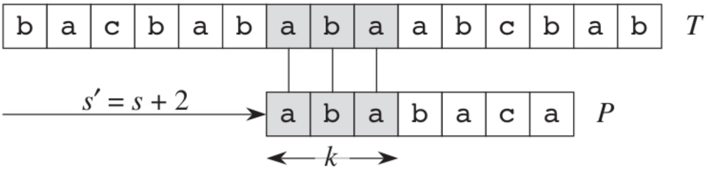
O padrão $P =$ ababaca está alinhado com o texto T tal que os primeiros $q = 5$ caracteres coincidem.

Princípio

Ainda no exemplo anterior, o deslocamento $s' = s + 2$, entretanto, alinha os três primeiros caracteres do padrão com três caracteres do texto, que necessariamente coincidem;

A figura a seguir ilustra esta situação.

KMP



Utilizando a informação dos 5 caracteres coincidentes, podemos deduzir que o deslocamento $s+1$ é inválido, mas que o deslocamento $s' = s+2$ é consistente com a parte analisada do texto e potencialmente é válido.

Princípio

Em geral, é útil sabermos a resposta da seguinte pergunta:

"Dados os caracteres do padrão $P[1..q]$ que casam o caracteres do texto $T[s+1..s+q]$, qual é o menor deslocamento $s' > s$, tal que para algum $k < q$,

$$P[1..k] = T[s'+1..s'+k], \tag{1}$$

em que $s' + k = s + q$?

Tal deslocamento s' é o primeiro deslocamento maior que s potencialmente válido, dado que "conhecemos" apenas $T[s+1..s+q]$;

Em outras palavras, sabendo que $P_q \sqsupset T_{s+q}$, queremos o prefixo próprio mais longo P_k de P_q que também é um sufixo de T_{s+q};

Uma vez que $s' + k = s + q$, e são dados s e q, determinar o menor deslocamento s' é equivalente a determinar o maior comprimento k de um prefixo de P_q.

KMP

Princípio

Adicionamos a diferença dos comprimentos destes prefixos de P (ou seja, $q - k$) ao deslocamento s para determinarmos o novo deslocamento s', tal que $s' = s + (q - k)$;

No melhor caso, $k = 0$, tal que $s' = s + q$, e imediatamente eliminamos os deslocamentos $s + 1, s + 2, \ldots, s + q - 1$;

De qualquer maneira não será necessário, a cada novo deslocamento s', comparar os k primeiros caracteres de P com os caracteres correspondentes em T, uma vez que a equação 1 garante que eles coincidem;

Podemos pré-computar a informação necessária comparando o padrão contra si próprio por meio de uma **Função Prefixo**.

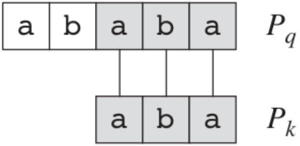
Função Prefixo

Considerando que $T[s'+1..s'+k]$ é parte da parcela conhecida do texto, esta cadeia de caracteres é um sufixo da cadeia de caracteres P_q;

Portanto, podemos interpretar a equação 1 como se ela pedisse o maior $k < q$ tal que $P_k \sqsupset P_q$, assim, sendo o deslocamento $s' = s + (q - k)$ o próximo deslocamento potencialmente válido;

É mais conveniente armazenar, para cada valor de q, o valor k de caracteres coincidentes no novo deslocamento s', ao invés de armazenar $s' - s$;

Formalmente, dado um padrão $P[1..m]$, a **função prefixo** de P é a função $\pi : \{1, 2, \ldots, m\} \rightarrow \{0, 1, \ldots, m-1\}$ tal que
$\pi[q] = max\{k : k < q \text{ e } P_k \sqsupset P_q\}$.



Podemos pré-computar informações úteis para computação do próximo deslocamento pela comparação do padrão com ele próprio;

No exemplo, temos que o prefixo mais longo de P que também é um sufixo de P_5 é P_3. Representamos esta informação no arranjo π, tal que $\pi[5] = 3$, e o próximo deslocamento potencialmente válido é $s' = s + (q - \pi[q])$, dado que q caracteres coincidiram no deslocamento s.

Função Prefixo

Em outras palavras, $\pi[q]$ é o comprimento do prefixo mais comprido de P que também é um sufixo próprio de P_q;

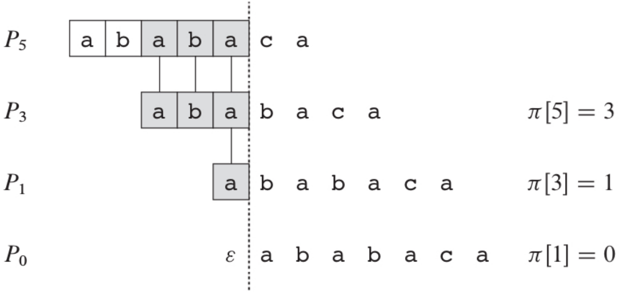
A figura seguinte apresenta a função prefixo π completa para o padrão `ababaca`

KMP

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

Primeiro exemplo: função π para o padrão $P =$ ababaca.

KMP



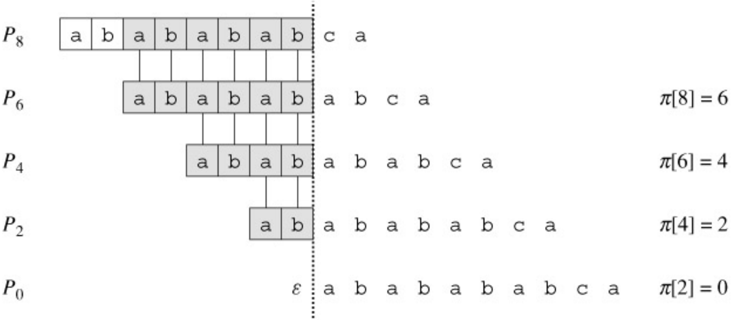
Deslizamos o gabarito que contém P para a direita e observamos quando algum prefixo P_k de P coincide com algum sufixo próprio de P_5; Temos coincidências quando $k = 3, 1, 0$;

Na figura, a primeira linha fornece P e a linha vertical pontilhada é desenhada logo após P_5. As linhas seguintes mostram todos os deslocamentos de P que levam um prefixo P_k de P a coincidir com algum sufixo de P_5;

Caracteres coincidentes são apresentados e cinza; linhas verticais conectam caracteres alinhados coincidentes, logo, $\{k : k < 5 \text{ e } P_k \sqsupset P_5\} = \{3, 1, 0\}$.

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)

Segundo exemplo de cálculo da função prefixo, desta vez, para $P =$ ababababca.

Algoritmo de *matching*

A seguir é apresentado o procedimento de *matching* do KMP, referido como **KMP-Matcher**;

Este procedimento, por sua vez, utiliza o procedimento auxiliar **ComputaFuncaoPrefixo**, para o cálculo de π, apresentado primeiro;

Estes dois procedimentos possuem muito em comum, dado que ambos alinham uma cadeia de caracteres ao padrão P: o **KMP-Matcher** alinha o texto T ao padrão P, ao passo que o **ComputaFuncaoPrefixo** alinha o padrão P consigo mesmo;

Dada esta coincidência entre os procedimentos, a análise de complexidade de ambos é semelhante.

```
ComputaFuncaoPrefixo(P)
Entrada: Cadeia de caracteres P
m ← |P|;
Crie π[1..m]; é um arranjo completamente novo
π[1] ← 0;
k ← 0;
// considere q ≤ m
para q ← 2 até m faça
    enquanto k > 0 e P[k + 1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k + 1] = P[q] então
        k ← k + 1;
    fim
    π[q] ← k;
fim
retorna (π);
```

Complexidade – ComputaFuncaoPrefixo

O tempo de execução de **ComputaFuncaoPrefixo** é $\Theta(m)$. O detalhe está na análise amortizada do laço **enquanto**, executado $O(m)$ vezes:

- k inicia de zero e só é incrementado na linha 11, instrução executada no máximo uma vez por iteração do laço **para** (incremento máximo de m-1, portanto);
- Considerando que $k < q$ antes do laço **para** e que cada iteração deste laço incrementa q, temos sempre que $k < q$;
- Desta maneira, as atribuições das linhas 4 e 13 garantem que $\pi[q] < q$ para todo $q = 1, 2, \ldots, m$, o que significa que a cada iteração, o laço **enquanto** decrementa k;
- k nunca se torna negativo;
- Juntando todas as informações, temos que o decremento total em k no laço **enquanto** é limitado superiormente pelo incremento total de k no laço **para**, que é $m - 1$.

Complexidade – **ComputaFuncaoPrefixo**

Considerando todas as observações sobre k, o laço **enquanto** é repetido por $m-1$ vezes, e o procedimento **ComputaFuncaoPrefixo** é executado em tempo $\Theta(m)$.

KMP

KMP-Matcher(T, P)
Entrada: Cadeias de caracteres T e P
$n \leftarrow |T|$;
$m \leftarrow |P|$;
$\pi \leftarrow ComputaFuncaoPrefixo(P)$;
$q \leftarrow 0$;
// considere $i \leq n$
para $i \leftarrow 1$ ***até*** n **faça**
    enquanto $q > 0$ ***e*** $P[q+1] \neq T[i]$ **faça**
        $q \leftarrow \pi[q]$;
    fim
    se $P[q+1] = T[i]$ **então**
        $q \leftarrow q + 1$;
    fim
    se $q = m$ **então**
        Imprima "O padrão ocorre com deslocamento $i - m$";
        $q \leftarrow \pi[q]$;
    fim
fim

Complexidade – **KMP-Matcher**

De maneira análoga à analise amortizada realizada para o procedimento **ComputaFuncaoPrefixo**, podemos analisar a complexidade do procedimento **KMP-Matcher**;

Com efeito, basta substituir as observações sobre k por q, o limite m por n e o índice do laço **para** q por i.

Complexidade – KMP-Matcher

O tempo de execução de **KMP-Matcher** é $\Theta(n)$. O detalhe está na análise amortizada do laço **enquanto**, executado $O(n)$ vezes:

- q inicia de zero e só é incrementado na linha 11, instrução executada no máximo uma vez por iteração do laço **para** (incremento máximo de n-1, portanto);
- Considerando que $q < i$ antes do laço **para** e que cada iteração deste laço incrementa i, temos sempre que $q < i$;
- Desta maneira, as atribuições das linhas 4 e 15 garantem que $q < i$ para todo $i = 1, 2, \ldots, n$, o que significa que a cada iteração, o laço **enquanto** decrementa q;
- q nunca se torna negativo;
- Juntando todas as informações, temos que o decremento total em q no laço **enquanto** é limitado superiormente pelo incremento total de q no laço **para**, que é $n - 1$.

Complexidade – KMP

O algoritmo KMP possui tempo de pré-processamento $\Theta(m)$ e tempo de *matching* $\Theta(n)$, logo, a complexidade do algoritmo é $\Theta(m+n)$;

Sabemos que $m < n$ na prática, portanto, a complexidade esperada do algoritmo na prática é $\Theta(n)$.

Exercícios

1. Compute a função prefixo para P= `abarba`;
2. Use o algoritmo KMP para buscar P em T=abacaabaccabacabaabb.

Dúvidas?