Análise de Algoritmos
Gabarito da Segunda Prova
Tópicos: Reconhecimento de Padrões e Programação Dinâmica

Instituto de Ciências Exatas, Universidade de Brasília
18 de maio de 2009
Prof. Mauricio Ayala-Rincón

Nome:	Matrícula:

Duração: 100 min.
Inicio: 14:00; Fim: 15:45
Três Páginas, duas questões

1. (4.0 pontos) **Reconhecimento de padrões**

   (a) (1.0) Explique o papel da *função de erros* ou *função de falha* do algoritmo de Knuth-Morris-Pratt e como ela é utilizada na prática. Exemplifique a sua explicação com o padrão

   GGATGGATCAGG

   .

   Em particular, compute a *função de falha* para esse padrão.

R/

A *função de falha* utiliza informação da estrutura do próprio padrão para permitir realizar saltos ou deslizamentos do padrão durante o processo de busca de suas ocorrências num texto. Dessa forma, por exemplo, quando se encontra um casamento dos primeiros oito símbolos do padrão e um *descasamento* do nono símbolo do com o símbolo i-ésimo do texto, t_i, a seguinte comparação será realizada entre esse símbolo do texto e o quinto símbolo do padrão, como ilustrado a seguir.

...	t_{i-8}	t_{i-7}	t_{i-6}	t_{i-5}	t_{i-4}	t_{i-3}	t_{i-2}	t_{i-1}	t_i	t_{i+1}	t_{i+2}	t_{i+3}	t_{i+4}	t_{i+5}	t_{i+6}	t_{i+7}	...
	=	=	=	=	=	=	=	=	≠								
	G	G	A	T	G	G	A	T	C	A	G	G					
					=	=	=	=	≠								
					G	G	A	T	G	G	A	T	C	A	G	G	
	1	2	3	4	5	6	7	8	9	10	11	12					

Dado um padrão $P = p_1 \cdots p_n$, a *função de falha* para a posição j-ésima, $0 < j < n$, do padrão é definida como um mais o comprimento do máximo pré-fixo do pré-fixo

$(j-1)$-ésimo do padrão, denotado P_{j-1} que é também sufixo de P_{j-1} e que precede um símbolo diferente do símbolo p_j. Para os casos em que esse valor não este definido, define-se a *função de falha* como 0. Formalmente:

$$\textit{fun de falha}(j) := \begin{cases} 0, \text{ se } j = 1; \\ \text{máximo}(\{0\} \cup \{1 + |P_k| \mid P_k \sqsubset P_{j-1} \text{ e } P_k \sqsupset P_{j-1} \text{ e } p_{k+1} \neq p_j\}) \text{ se } 1 < j \leq n \end{cases}$$

Para o exemplo da questão, a função de falha é dada por

posição	1	2	3	4	5	6	7	8	9	10	11	12
Padrão	G	G	A	T	G	G	A	T	C	A	G	G
função de falha	0	0	2	1	0	0	2	1	5	1	0	0

(b) (2.0) Explique o papel das funções *matchjump* e *charjump* do algoritmo de Boyer-Moore e como elas são utilizadas na prática. Exemplifique a sua explicação com a busca do padrão

`Neanderthal`

no texto

`These results do not rule out the possibility that Neanderthals contributed other genes to modern humans. However, the results support the hypothesis that modern humans arose in Africa before migrating to Europe and replacing the Neanderthal population with little or no interbreeding.`

Em particular, compute *charjump* e *matchjump* para o alfabeto e padrão envolvidos.

R/

O algoritmo BM realiza uma pesquisa similar à do KMP, mas acrescenta uma função *charjump*, indexada pelos símbolos do alfabeto, que permite desplazar o padrão sobre símbolos no texto que não ocorrem neste. A pesquisa é realizada de esquerda para direita sobre o texto, mas percorrendo os símbolos do padrão de direita para esquerda. Dessa forma, a função *matchjump* desempenha um papel similar ao da *função de erro* de KMP, mas, em contraste, é definida sobre sufixos do padrão. Em cada *descasamento* computa-se o deslizamento do padrão sobre o texto segundo os valores das funções *charjump* e *matchjump*.

Considerando os símbolos do alfabeto, a função *charjump* para o padrão `Neanderthal` é definida por

```
ABCDEFGHIJKLMNOPQRSTUVWXYZa  bcdefghijk l  mnopqrstuvwxyz .,;:    ...
00000000000001000000000010 0056009000 11 04000708000000000000    ...
```

Assim, p.ex., quando se tem um *descasamento* com uma posição do texto onde ocorre um símbolo que não ocorre no padrão, o "0" indica que se deve deslizar o padrão sobre essa posição do texto. Para o caso de um *descasamento* com um símbolo que ocorre no padrão, irá se desplazar o padrão à direita alinhando o símbolo do descasamento do texto com a última ocorrência desse símbolo no padrão. Como exemplo do primeiro, considere a situação embaixo, na qual o símbolo `l` casou e o símbolo `a` *descasou* com um `u` do texto.

```
These results do not rule out the possibility that Neanderthals
        ^=
Neanderthal
```

O padrão irá desplazar-se sobre o símbolo `u` que não ocorre no padrão, atingindo a situação embaixo

```
These results do not rule out the possibility that Neanderthals
                    ^
         Neanderthal
```

Os seguintes passos estão listados embaixo:

```
These results do not rule out the possibility that Neanderthals
                               ^ ^          ^          ^     =
                     Neanderthal ^          ^          ^     =
                       Neanderthal          ^          ^     =
                                  Neanderthal          ^     =
                                                Neanderthal     =
                                                      Neanderthal
```

Observe, em particular que um *descasamento* com o símbolo `e`, unicamente poderia realizar um desplazamento do padrão alinhando esse símbolo com a última ocorrência de `e` no padrão (posição 6), para não ocacionar possíveis desplazamentos sobre ocorrências do padrão no texto.

A função *matchjump* de uma posição j do padrão, similarmente à função de erro de KMP, indica o máximo deslizamento possível, uma vez o sufixo $P^{j+1} = p_{j+1} \cdots p_n$ casou e se tem um *descasamento* na posição j do padrão. Dessa forma, o desplazamento irá alinhar a ocorrência mais à direita de um símbolo p_r que precede uma ocorrência de um sufixo maximal do sufixo $P^{j+1} = p_{j+1} \cdots p_n$, precedido por um símbolo diferente de p_r, se esta posição existe. Caso contrário irá alinhar o máximo prefixo do padrão que é sufixo de P^{j+1}. Para o exemplo `Neanderthal` os cálculos são simples, sempre que o último símbolo `l`, não permite sobreposições com outras subpalavras do padrão; assim, a função *matchjump* está definida por

Posição	1	2	3	4	5	6	7	8	9	10	11
Padrão	N	e	a	n	d	e	r	t	h	a	l
matchjump	21	20	19	18	17	16	15	14	13	12	1

(c) (1.0) Explique porque é mais adequado o uso do algoritmo de Boyer-Moore que o de Knuth-Morris-Pratt no item (b). E explique qual dos dois mais adequado para buscar um padrão como o do item (a)?

R/ Do gabarito da lista:

"O algoritmo KMP tem um bom desempenho para alfabetos com poucos caracteres, como o binário por exemplo, enquanto o algortimo BM se destaca para busca de padrões em linguagem natural onde o padão tem comprimento maior ou igual a 5 (veja 5.3[Baa88]). Essa diferença se deve ao fato da pre-computação do BM não compensar para tais alfabetos, pois a chance de haver um padrão onde não apareça algum desses caracteres é mínima."

Programação Dinâmica

2. (2.0 pontos)**Fundamentos da programação dinâmica**

A sequência de Fibonacci define-se via a função recursiva *fib* como:

$$\mathit{fib}(n) = \begin{cases} 1 & \text{se } n = 0 \text{ ou } n = 1 \\ \mathit{fib}(n-1) + \mathit{fib}(n-2) & \text{se } n \geq 2 \end{cases}$$

Pode-se construir um algoritmo direto dessa definição para computar o k-ésimo número de Fibonacci, para $k \in \mathbb{N}$, como

function `fibo`(k)
  begin
    if $k = 0$ or $k = 1$ **then** 1
    else `fibo`$(k-1)$+ `fibo`$(k-2)$
  end.

O trabalho (número de adições) realizado pelo algoritmo `fibo` está dado pela relação de recurrência

$$W(n) = \begin{cases} 0 & \text{se } n = 0 \text{ ou } n = 1 \\ W(n-1) + W(n-2) + 1 & \text{se } n \geq 2 \end{cases}$$

(a) (1.0 ponto) Demonstre que o trabalho realizado pelo algoritmo `fibo` é exponencial; mais especificamente, demonstre que $W(n) \in \Omega(2^n)$.

R/

$$
\begin{aligned}
W(n) &= W(n-1) + W(n-2) + 1 \\
&> 2\,W(n-2) + 1 \\
&> 2^2\,W(n-4) + 2^1 + 2^0 \\
&> \vdots \\
&> \begin{cases} 2^{\lfloor n/2 \rfloor}\,W(1) + \displaystyle\sum_{i=0}^{\lfloor n/2 \rfloor - 1} 2^i, & n \text{ ímpar} \\ 2^{n/2}\,W(0) + \displaystyle\sum_{i=0}^{n/2-1} 2^i, & n \text{ par} \end{cases} \\
&= \sum_{i=0}^{\lfloor n/2 \rfloor - 1} 2^i \\
&= 2^{\lfloor n/2 \rfloor} \\
&\in \Theta(2^n)
\end{aligned}
$$

Logo, $W(n) \in \Omega(2^n)$.

(b) (1.0 ponto) Desenvolva um algoritmo (`fiboPD`), em no máximo 5 linhas, baseado em técnicas de programação dinâmica para computar a função de Fibonacci *fib*, e calcule a sua complexidade em tempo e em espaço.

R/

O algoritmo computa todos os valores precedentes da sequência de Fibonacci em um arranjo dinâmico F para computar o k-ésimo número de Fibonacci.

```
function fiboPD(k)
    F[0] ← 1; F[1] ← 1;
    for i = 2 to k do F[i] ← F[i − 1] + F[i − 2];
    return F[k];
end function
```

O tempo e espaço para computar o n-ésimo número de Fibonacci com este algoritmo está em $\Theta(n)$, mas a tabela F pode ser eliminada, obtendo tempo em $\Theta(n)$ e espaço em $\Theta(1)$.

3. (4.0 pontos) **Reconhecimento Aproximado de Padrões**

O *reconhecimento aproximado de padrões em palavras* é uma generalização do problema de reconhecimento de padrões em palavras. O problema pode-se estabelecer da seguinte maneira:

> dadas palavras **padrão** e **texto**, p e t, respectivamente, sobre um alfabeto Σ, com $|p| = m \geq 0$ e $|t| = n \geq 0$ e dado um inteiro $k \geq 0$, encontrar subpalavras s de t tais que a **distância de edição** entre s e p, $d(s, p)$, é menor ou igual que k.

A distância de edição entre duas palavras x e y em Σ^* é o custo total mínimo possível de uma sequência de *passos de edição* para converter x em y. Denotando com ϵ a palavra vazia, em cada passo de edição aplica-se uma *regra de reescrita* da forma:

- $a \rightarrow \epsilon$, $a \in \Sigma$ (eliminação);
- $\epsilon \rightarrow a$, $a \in \Sigma$ (inserção);
- $a \rightarrow b$, $a, b \in \Sigma$ (substituição).

Diz-se que a palavra x é uma *k-aproximação* de y, se se pode converter x em y em k passos de edição. Por exemplo, existe uma ocorrência 1-aproximada do padrão *abb* na primeira posição do texto *abcdbdb* e uma ocorrência 2-aproximada na quarta posição, como se ilustra no seguinte esquema.

$$\begin{array}{cccc} a & b & b & \\ \| & \| & \updownarrow & \\ a & b & \epsilon & c \end{array} \qquad \begin{array}{cccc} a & b & \epsilon & b \\ \not\| & \| & \updownarrow & \| \\ d & b & d & b \end{array}$$

O mecanismo de solução mais "popular" do problema de reconhecimento aproximado de padrões é baseado em técnicas de *programação dinâmica* e tem complexidade $O(nm)$. Sejam $p = p_1 \ldots p_m$ e $t = t_1 \ldots t_n$ um padrão e um texto, respectivamente. Basicamente o método consiste em construir uma $m + 1 \times n + 1$-tabela D tal que para $0 \leq i \leq m$, $0 \leq j \leq n$, $D[i, j]$ é a mínima distância de edição do prefixo $p_1 \ldots p_i$ com um sufixo do prefixo $t_1 \ldots t_j$ do texto. Existem aproximações do padrão com distância de edição $\leq k$ e um sufixo do prefixo $t_1 \ldots t_j$ se, e soamente se $D[m, j] \leq k$. A tabela se construi segundo a seguinte relação recursiva:

$$D[i,j] = \begin{cases} 0, \text{ se } i = 0; \\ i, \text{ se } j = 0; \\ min \left\{ \begin{array}{l} D[i-1,j]+1, \\ D[i,j-1]+1, \\ D[i-1,j-1]+ \text{ se } p_i = t_j \text{ então 0 senão 1} \end{array} \right\}, \begin{array}{l} \text{se} \\ 1 \leq i \leq m \text{ e} \\ 1 \leq j \leq n \end{array} \end{cases}$$

Utilizando essa relação diretamente, obtem-se um algoritmo recursivo `appx_spm` para computar a melhor aproximação entre o prefixo i-ésimo de um padrão p e um sufixo do prefixo j-ésimo de um texto t como

```
function appx_spm(p, i, t, j)
  begin
    if i = 0 then aux ← 0
    else
      if j = 0 then aux ← i
      else
        aux ← (if p_i = t_j then 1 else 0);
        aux ← aux + appx_spm(p, i − 1, t, j − 1);
        if appx_spm(p, i − 1, t, j) + 1 < aux then
                    aux ← appx_spm(p, i − 1, t, j) + 1;
        if appx_spm(p, i, t, j − 1) + 1 < aux then
                    aux ← appx_spm(p, i, t, j − 1) + 1;
       end if;
     end if;
     return aux;
  end.
```

O trabalho realizado por `appx_spm` expressa-se na seguinte relação de recurrência

$$W(i,j) = \begin{cases} 0, \text{se } i = 0 \text{ ou } j = 0 \\ W(i-1,j-1) + W(i-1,j) + W(i,j-1) + \Theta(1), \text{se } i,j > 0 \end{cases}$$

(a) (2.0 pontos) Demonstre que o algoritmo `appx_spm` tem complexidade tempo exponencial; mais especificamente demonstre que $W(n,n) \in \Omega(3^n)$.

R/

$$\begin{aligned} W(n,n) &= W(n-1,n) + W(n-1,n-1) + W(n,n-1) + \Theta(1) \\ &> 3\,W(n-1,n-1) + \Theta(1) \\ &> 3^2\,W(n-2,n-2) + 2^1\Theta(1) + 2^0\Theta(1) \\ &> \vdots \\ &> 3^n\,W(0,0) + \Theta(1)\sum_{i=0}^{n-1} 3^i \\ &= \Theta(1)\sum_{i=0}^{n-1} 3^i \\ &= \Theta(1)\,\frac{3^n - 1}{3-1} \\ &\in \Theta(3^n) \end{aligned}$$

Logo, $W(n,n) \in \Omega(3^n)$.

(b) (2.0 pontos) Construa uma solução para achar todas as approximações de um padrão em um texto baseada em técnicas de programação dinâmica e demonstre que o trabalho por esta realizada está em $\Theta(m\ n)$.

R/

Construi-se uma tabela dinâmica, $TD[i, j]$ na qual são computados os valores de $D[i, j]$ ascendentemente, coluna por coluna, fila por fila.

```
function appx_spm_PD(p, m, t, n)
  begin
    for j = 0 to  n do TD[0, j] ← 0;
    for i = 1 to  m do TD[i, 0] ← i;
    for j = 1 to  n do
      for i = 1 to  m do
        TD[i, j] ← (if p_i = t_j then 1 else 0);
        TD[i, j] ← TD[i, j] + TD[i − 1, j − 1];
        if TD[i − 1, j] + 1 < TD[i, j] then
                TD[i, j] ← TD[i − 1, j] + 1;
        if TD[i, j − 1] + 1 < TD[i, j] then
                TD[i, j] ← TD[i, j − 1] + 1;
       end if;
     end if;
  end.
```

Na fila m-ésima da tabela TD computada com a função `appx_spm_PD` temos os valores da melhor aproximação entre o padrão p e um sufixo do prefixo j-ésimo do texto ($TD[m, j]$).

Calcula-se o trabalho para cada um dos **for**'s da função:

- O primeiro **for** realiza $n + 1$ iterações;
- O segundo **for** realiza m iterações;
- Os **for**'s aninhados realizam $m\,n$ iterações para computar cada um dos componentes da tabela TD, $1 \leq i \leq m; 1 \leq n \leq n$.

Dessa forma realizam-se $\Theta(m\,n)$ operações.