MAC5711 Análise de Algoritmos

DCC–IME–USP, 10 de novembro de 2004

Instruções:

(i) Esta prova contém cinco questões sendo uma de um ponto, três de dois pontos e uma de três pontos.

(ii) Mencione os teoremas e propriedades usados para justificar suas afirmações.

(iii) Você pode utilizar como subrotina qualquer algoritmo visto em sala de aula sem reescrevê-lo. No entando, você deve descrever clara e sucintamente o que o algoritmo recebe, devolve ou faz e o seu consumo de tempo. Exemplo

> *"O algoritmo* Blá-Blá-Blá *usa como subrotina o algoritmo* Ordenação-Lerda (A, n) *que recebe e rearranja um vetor* $A[1\,..\,n]$ *de modo que ele fique em ordem crescente. O consumo de tempo do algoritmo* Ordenação-Lerda *é* $\mathrm{O}(n^n)$*."*

(iv) Não é permitida a consulta a livros, anotações, colegas, calculadoras, Internet, computadores ...

Duração da prova: 2 horas e 30 minutos

Questão 1 [1 pontos]
Professor Teoria projetou um algoritmo de ordenação baseado em comparações que consome tempo $\mathrm{O}(n \lg \sqrt{n})$. Considerando-se o limite inferior para o consumo de tempo de algoritmos de ordenação, isso é possível? Enuncie a asserção desse "limite inferior para ordenação" para justificar a sua resposta.

Solução: Sim, é possível. Sabe-se que todo algoritmo de ordenação baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no **pior caso**. Como

$$n \lg \sqrt{n} = n \lg n^{\frac{1}{2}} = \frac{1}{2} n \lg n,$$

então o consumo de tempo do algoritmo do Professor Teoria é $\mathrm{O}(n \lg n)$. Isto não contraria em nada o limite inferior de $\Omega(n \lg n)$ comparações no pior caso.

Primos de O e Θ [CLRS 3.1]

Não sei bem por que, mas resolvi aproveitar a oportunidade para apresentar-lhes dois primos de O e Θ. São eles o ("ó pequeno") e ω ("omega pequeno").

Sejam $T(n)$ e $f(n)$ funções dos inteiros no reais.
Dizemos que $T(n)$ **é** $\mathrm{o}(f(n))$ (lê-se $T(n)$ é ó pequeno de $f(n)$) se *para toda* constante positiva c *existe* uma constante inteira n_0 tal que

$$T(n) \ < \ c\, f(n)$$

para todo $n \geq n_0$.

Da mesma forma, dizemos que $T(n)$ é $\omega(f(n))$ (lê-se $T(n)$ é omega pequeno de $f(n)$) se para *para toda* constante positiva c *existe* uma constante inteira n_0 tal que

$$T(n) \ > \ c\, f(n)$$

para todo $n \geq n_0$.

Intuitivamente, $\mathrm{o}(\ldots)$ é assintoticamente análogo a "menor que" e $\omega(\ldots)$ é assintoticamente análogo a "maior que". Note que $\mathrm{O}(\ldots)$ é assintoticamente análogo a "menor ou igual que" e $\Omega(\ldots)$ é assintoticamente análogo a "maior ou igual que".

O limite inferior da ordenação, em termos da notação-o, se traduz em

> **Não existe** a algoritmo de ordenação baseado em comparações de consumo de tempo $\mathrm{o}(n \lg n)$.

Como o consumo de tempo algoritmo do professor Teoria é $\mathrm{O}(n \lg n)$, isto não contraria em nada a inexistência de um algoritmo de consumo de tempo $\mathrm{o}(n \lg n)$. Note que $T(n) = n \lg n$ é $\mathrm{O}(n \lg n)$, mas não é $\mathrm{o}(n \lg n)$.

É evidente que $\mathrm{o}(f(n)) \subset \mathrm{O}(f(n))$, $\omega(f(n)) \subset \Omega(f(n))$ e que

> $T(n)$ está em $\omega(f(n))$ se e somente se $f(n)$ está em $\mathrm{o}(T(n)$.

Questão 2 [2 pontos]
Considere a seguinte variante do algoritmo Quicksort, que recebe e ordena um vetor $A[p\,..\,r]$.

Quicksort2 (A, p, r)
enquanto $p < r$ **faça**
    $q \leftarrow$ Particione (A, p, r)
    Quicksort2 $(A, p, q-1)$
    $p \leftarrow q + 1$

Mostre que a pilha de recursão pode atingir altura $\Omega(n)$, onde $n := r - p + 1$. Modifique o algoritmo de modo que a pilha de recursão tenha altura $\mathrm{O}(\lg n)$. Justifique a sua resposta.

Solução: Se os elementos do vetor $A[p..r]$ estão em ordem crescente, então a seqüência de chamadas recursivas feitas pelo algoritmo é

Quicksort2 (A, p, r)
  Quicksort2 $(A, p, r-1)$
    Quicksort2 $(A, p, r-2)$
      Quicksort2 $(A, p, r-3)$
        $\ddots$
          Quicksort2 (A, p, p)

Assim, vemos que a altura da pilha de execução chega a ser $r - p + 1 = n$.

Considere a seguinte modificação do Quicksort2.

Quicksort3 (A, p, r)
enquanto $p < r$ **faça**
    $q \leftarrow$ Particione (A, p, r)
    se $q - p < r - q$
        então Quicksort3 $(A, p, q-1)$
            $p \leftarrow q + 1$
        senão Quicksort3 $(A, q+1, r)$
            $r \leftarrow q - 1$

Considere agora uma seqüência

Quicksort3 (A, p, r)
  Quicksort3 (A, p_1, r_1)
    Quicksort3 (A, p_2, r_2)
      Quicksort3 (A, p_3, r_3)
        $\ddots$
          Quicksort3 (A, p_k, r_k)

de chamadas recursivas do algoritmo. Nesta situação a pilha de recursão tem altura $k+1$. Devido a condição da linha 3 tem-se que

$$
\begin{array}{ccccccc}
n & = & p-r+1 & & & & \\
n_1 & = & p_1-r_1+1 & \leq & n/2 & & \\
n_2 & = & p_2-r_2+1 & \leq & n_1/2 & \leq & n/2^2 \\
n_3 & = & p_3-r_3+1 & \leq & n_2/2 & \leq & n/2^3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
n_k & = & p_k-r_k+1 & \leq & n_{k-1}/2 & \leq & n/2^k.
\end{array}
$$

De onde se conclui que $k \leq \lceil \lg n \rceil$ e a altura da pilha de recursão nunca ultrapassa $\lceil \lg n \rceil + 1$.

Questão 3 [2 ponto]
Descreva um algoritmo ORDENE (A, n) que recebe e ordena um vetor $A[1\,..\,n]$ em que todos os elementos pertencem a $\{0, 1, \ldots, n^2 - 1\}$. O consumo de tempo do algoritmo deve ser $\mathrm{O}(n)$. Justifique a sua resposta.

Solução: Estamos habituados a representar números naturais na base decimal que utiliza os algarismos $0, 1, \ldots, 9$. Na solução consideraremos a representação de cada elemento em $A[1\,..\,n]$ na base n, que utiliza os algarismos $0, 1, \ldots, n - 1$. Desta forma, para $j = 1, \ldots, n$, temos que

$$A[j] = a_1 \times n + a_0,$$

onde a_0 e a_1 são algarismos em $\{0, 1, \ldots, n - 1\}$. Diremos que a_0 é o algarismo **menos significativo** de $A[j]$ e que a_1 é o algarismo **mais significativo** de $A[j]$.

O algoritmo COUNTING-SORT recebe e ordena um vetor $X[1\,..\,n]$ em que todos os elementos estão em $\{0, 1, \ldots, k\}$, para um dado k. O consumo de tempo do algoritmo é $\Theta(n + k)$. Em particular, se k é $\Theta(n)$, então o consumo de tempo do COUNTING-SORT é $\Theta(n)$.

O algoritmo ORDENE (A, n) é uma mera adaptação do RADIX-SORT.

ORDENE (A, n)
1 ordene $A[1\,..\,n]$ usando como chaves seus algarismos **menos significativos**.
2 ordene $A[1\,..\,n]$ usando como chaves seus algarismos **mais significativos**.

Utilizando uma adaptação do COUNTING-SORT para $k = n - 1$ nas ordenações das linhas 1 e 2 obtemos um algoritmo de consumo de tempo $\Theta(n + n) = \Theta(n)$. A estabilidade do COUNTING-SORT é fundamental para a correção do algoritmo ORDENE.

Questão 4 [2 ponto]
Considere o algoritmo abaixo que recebe vetores $A[1\,..\,n]$ e $B[1\,..\,n]$ e devolve 1 se $A[j] = 1 = B[j]$ para algum j e devolve 0 em caso contrário.

```
ELEMENTO-COMUM (A, B, n)
1  j ← 1
2  enquanto j ≤ n e (A[j] ≠ 1 ou B[j] ≠ 1) faça
3      j ← j + 1
4  se j ≤ n
5      então devolva 1
6      senão devolva 0
```

Suponha que o valor de cada componente de $A[1\,..\,n]$ e de $B[1\,..\,n]$ é escolhido ao acaso e independentemente para ser 0 ou 1 com probabilidade 1/2. Qual o consumo de tempo esperado do algoritmo? Dê a resposta em termos da notação O e justifique-a.

Solução: Seja X a variável aleatória representando o número de execuções da linha 2 do algoritmo. O consumo de tempo do algoritmo é $\Theta(X)$. Determinaremos $\Theta(\mathrm{E}[X])$.

Para $i = 1, 2, \ldots, n+1$, seja X_i a variável aleatória que indica se a linha 2 é executada (*pelo menos*) i vezes, ou seja,

$$X_i \;=\; \begin{cases} 1 & \text{se a linha 2 é executado } i \text{ vezes,} \\ 0 & \text{caso contrário.} \end{cases}$$

Portanto,

$$X = X_1 + X_2 + \cdots + X_{n+1} \tag{1}$$

Temos que

$$\mathrm{E}[X_i] = 1 \ \Pr\{X_i = 1\} + 0 \ \Pr\{X_i = 0\} = \Pr\{X_i = 1\} = \Pr\{X \geq i\}.$$

Em palavras, $\mathrm{E}[X_i]$ é igual a probabilidade da linha 2 ser executada i vezes. A linha 2 é executada i vezes se e somente se

$$(A[1] = 0 \text{ ou } B[1] = 0) \text{ e } (A[2] = 0 \text{ ou } B[2] = 0) \text{ e } \ldots \text{ e } (A[i-1] = 0 \text{ ou } B[i-1] = 0)$$

Logo, para $i = 1, 2, \ldots, n+1$,

$$\begin{aligned} \Pr\{X \geq i\} &= \Pr\{(A[1] = 0 \text{ ou } B[1] = 0) \text{ e } \ldots \text{ e } (A[i-1] = 0 \text{ ou } B[i-1] = 0)\} \\ &= \Pr\{A[1] = 0 \text{ ou } B[1] = 0\} \times \cdots \times \Pr\{A[i-1] = 0 \text{ ou } B[i-1] = 0\} \\ &= \frac{3}{4} \times \cdots \times \frac{3}{4} \\ &= \frac{3^{i-1}}{4^{i-1}}\,. \end{aligned}$$

Agora podemos calcular $\mathrm{E}[X]$. De (1) obtêm-se que

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}[X_1 + X_2 + \cdots + X_{n+1}] \\
&= \mathrm{E}[X_1] + \mathrm{E}[X_2] + \cdots + \mathrm{E}[X_{n+1}] \\
&= 1 + \frac{3}{4} + \cdots + \frac{3^n}{4^n} \\
&< 1 + \frac{3}{4} + \cdots + \frac{3^n}{4^n} + \frac{3^{n+1}}{4^{n+1}} + \cdots \\
&= \frac{1}{1 - 3/4} \\
&= 4.
\end{aligned}$$

Portanto, $\mathrm{E}[X]$ é $\Theta(1)$.

Observação: Há ainda uma outra maneira de mostrar-se que $\mathrm{E}[X] = 4$, utilizando-se a equação

$$\mathrm{E}[X] = 1\ \Pr\{X = 1\} + 2\ \Pr\{X = 2\} + \cdots + (n+1)\ \Pr\{X = n+1\}.$$

Conclusão: O algoritmo Elemento-Comum decide se dois subconjuntos de um conjunto com n elementos têm um elemento em comum em tempo esperado constante. Os subconjuntos são dados através do seus vetores de incidência $A[1 \mathinner{.\,.} n]$ e $B[1 \mathinner{.\,.} n]$.

Questão 5 [3 pontos]
Uma subseqüência $Z[1\,..\,k]$ de um vetor de números inteiros $A[1\,..\,n]$ é **crescente** se

$$Z[1] \leq \cdots \leq Z[k].$$

Uma subseqüência crescente de $A[1\,..\,n]$ é **máxima** se não existe outra subseqüência crescente mais longa. Considere o problema de encontrar uma subseqüência crescente máxima de um vetor $A[1\,..\,n]$ dado.

a) Qual a subestrutura ótima para este problema?

b) Escreva um algoritmo Compr-Subseq-Máxima (A, n) que recebe um vetor $A[1\,..\,n]$ e devolve o **comprimento** de uma subseqüência crescente máxima. O consumo de tempo do algoritmo deve ser $\mathrm{O}(n^2)$. Justifique a correção e o consumo de tempo do algoritmo.

c) Modifique o algoritmo do item anterior para que ele devolva uma **subseqüência** crescente máxima $Z[1\,..\,k]$ de $A[1\,..\,n]$. Alternativamente, escreva um algoritmo Subseq-Máxima $(A, n, \ldots)$ que recebe um vetor $A[1\,..\,n]$ e alguma estrutura "..." calculada pelo algoritmo do item anterior e devolve uma **subseqüência** crescente máxima $Z[1\,..\,k]$ de $A[1\,..\,n]$. O consumo de tempo do trecho da modificação ou do algoritmo deve ser $\mathrm{O}(n)$. Justifique a correção e o consumo de tempo da modificação ou do algoritmo.

Solução de a). Suponha que $Z[1\,..\,k]$ é uma subseqüência crescente máxima de de $A[1\,..\,n]$.

Se $A[n] = Z[k]$, então $Z[1\,..\,k-1]$ é uma subseqüência crescente máxima de $A[1\,..\,n-1]$, senão $Z[1\,..\,k]$ é uma subseqüência crescente máxima de $A[1\,..\,n-1]$.

Solução de b). Seja $t[i]$ o comprimento de uma subseqüêcia crescente máxima de $A[1\mathinner{..}i]$ que tem $A[i]$ como último elemento. Temos que,

$$
\begin{aligned}
t[0] &= 0 \\
t[i] &= 1 + \max\{t[k] : 0 \leq k \leq i-1 \ \text{ e } \ A[k] \leq A[i]\}
\end{aligned}
$$

para $i = 1, 2, \ldots$. Suponha aqui que $A[0] = -\infty$.

O algoritmo a seguir baseia-se na recorrência acima.

Compr-Subseq-Máxima (A, n)
0 $\quad$ *compr* $\leftarrow 0$
1 $\quad$ **para** $i \leftarrow 1$ **até** n **faça**
2 $\qquad$ $t[i] \leftarrow 1$
3 $\qquad$ **para** $k \leftarrow 1$ **até** $i-1$ **faça**
4 $\qquad\quad$ **se** $A[k] \leq A[i]$ **e** $t[k] \geq t[i]$
5 $\qquad\qquad$ **então** $t[i] \leftarrow 1 + t[k]$
6 $\qquad$ **se** $t[i] > \mathit{compr}$
7 $\qquad\quad$ **então** *compr* $\leftarrow t[i]$
8 $\quad$ **devolva** *compr*

Consumo de tempo

linha	consumo de **todas** as execuções da linha
0	$\Theta(1)$
1	$\Theta(n)$
2	$\Theta(n)$
3	$\Theta(1 + 2 + \cdots + n) = \Theta(n^2)$
4	$\Theta(1 + 2 + \cdots + (n-1)) = \Theta(n^2)$
5	$\mathrm{O}(1 + 2 + \cdots + (n-1)) = \mathrm{O}(n^2)$
6	$\Theta(n)$
7	$\mathrm{O}(n)$
8	$\Theta(1)$
total	$2\Theta(n^2) + \mathrm{O}(n^2) + 3\Theta(n) + \mathrm{O}(n) + 2\Theta(1) = \Theta(n^2)$

Solução: de c). Eis a solução que usa um algoritmo.

```
Subseq-Máxima (A, n, t, compr)
1 k ← compr
2 Z[k + 1] ← ∞
3 i ← n
4 enquanto k > 0 faça     ▷ aqui vale que i > 0
5     se t[i] = k e A[i] ≤ Z[k + 1]
6         então Z[k] ← A[i]
7               k ← k − 1
8     i ← i − 1
9 devolva Z[1 . . compr]
```

Consumo de tempo

linha	consumo de **todas** as execuções da linha
1	$\Theta(1)$
2	$\Theta(1)$
3	$\Theta(1)$
4	$\mathrm{O}(n)$
5	$\mathrm{O}(n)$
6	$\Theta(k)$
7	$\Theta(k)$
8	$\mathrm{O}(n)$
9	$\Theta(k)$
total	$\mathrm{O}(3n) + \Theta(3k + 3) = \mathrm{O}(n) + \Theta(k) = \mathrm{O}(n + k) = \mathrm{O}(n)$

É possível fazermos uma versão do algoritmo acima que consome tempo $\Theta(k)$. Para isto é necessário que, além do vetor $t[1\,..\,n]$, o algoritmo Compr-Subseq-Máxima construa um vetor $b[1\,..\,n]$ em que $b[i] = k$ se $A[i] < A[k]$ e $t[i] = t[k] + 1$ para $i = 1, \ldots, n$. Suponha aqui que $A[0] = -\infty$.