MAC5711 Análise de Algoritmos

DCC–IME–USP, 29 de setembro de 2004

Solução de prova

Questão 1 [2 pontos]
Diga se são verdadeiras ou falsas as seguintes afirmações e justifique sucintamente as suas respostas. Sejam f e g funções dos números inteiros não-negativos nos números inteiros não-negativos.

a) Se $f(n) = \mathrm{O}(n)$ e $g(n) = \mathrm{O}(f(n))$, então $g(n) = \mathrm{O}(n)$.

Solução: Verdadeira.
Sabemos que existem números reais positivos c_f e c_g e números inteiros não-negativos n_f e n_g tais que:

$$\begin{aligned} f(n) &\leq c_f\, n \quad \text{para todo } n \geq n_f\text{; e} \\ g(n) &\leq c_g\, f(n) \quad \text{para todo } n \geq n_g. \end{aligned}$$

Logo, para $c := c_g c_f$, $n_0 := \max\{n_f, n_g\}$ e todo $n \geq n_0$ vale que

$$\begin{aligned} g(n) &\leq c_g\, f(n) \\ &\leq c_g c_f\, n \\ &= c\, n. \end{aligned}$$

b) Se $f(n) = \Omega(n)$ e $g(n) = \mathrm{O}(f(n))$, então $g(n) = \mathrm{O}(n)$.

Solução: Falsa.
Para $f(n) = 2^n$ e $g(n) = 2^n$ tem-se que $f(n)$ é $\Omega(n)$ e $g(n)$ é $\mathrm{O}(f(n))$, entretanto $g(n) = 2^n$ **não** é $\mathrm{O}(n)$.

c) Se $f(n) = \Omega(n)$ e $g(n) = \mathrm{O}(f(n))$, então $g(n) = \Omega(n)$.

Solução: Falsa.
Para $f(n) = 2^n$ e $g(n) = 1$ tem-se que $f(n)$ é $\Omega(n)$ e $g(n)$ é $\mathrm{O}(f(n))$, entretanto $g(n) = 1$ **não** é $\Omega(n)$.

d) Se $f(n) = \mathrm{O}(g(n))$, então $2^{f(n)} = \mathrm{O}(2^{g(n)})$.

Solução: Falsa.
Para $f(n) = 2n$ e $g(n) = n$ tem-se que $f(n)$ é $\mathrm{O}(g(n))$, entretanto $2^{f(n)} = 2^{2n} = 4^n$ não é $\mathrm{O}(2^n)$.
Ou ainda , para $f(n) = \lg n^2$ e $g(n) = \lg n$ tem-se que $f(n) = \lg n^2 = 2 \lg n$ é $\mathrm{O}(g(n)) = \mathrm{O}(\lg n)$, entretanto $2^{f(n)} = 2^{\lg n^2} = n^2$ não é $\mathrm{O}(2^{\lg n}) = \mathrm{O}(n)$.

Questão 2 [3 pontos]
Considere o seguinte algoritmo, cujo argumento n é uma potência de 2. O algoritmo não faz nada de útil.

Algo(n)
se $n = 1$ **então devolva** 1
para $i \leftarrow 1$ **até** 8 **faça** $z \leftarrow$ Algo($n/2$)
para $i \leftarrow 1$ **até** n^3 **faça** $z \leftarrow 0$

a) Seja $T(n)$ o número de vezes que a atribuição "$z \leftarrow 0$" é executada. Escreva uma recorrência que defina $T(n)$.

Solução:

$$\begin{aligned} T(1) &= 0 \\ T(n) &= 8\,T(n/2) + n^3 \qquad \text{para } n = 2, 4, 8, \ldots \end{aligned}$$

b) A que classe Θ pertence $T(n)$? Justifique a sua resposta (sem usar o *Master Theorem*, seja lá o que for isto).

Solução: Eu acho que $T(n) = n^3 \lg n$ para $n = 1, 2, 4, 8 \ldots$. Vou verificar isto por indução.
Se $n = 1$ então $T(n) = 0 = 1^3 \lg 1 = n^3 \lg n$.
Seja n uma potência de 2, $n \geq 2$ e suponha que a fórmula está certa para $n/2$:

$$\begin{aligned} T(n) &= 8\,T(n/2) + n^3 \\ &\overset{\text{hi}}{=} 8\,\tfrac{n^3}{2^3} \lg \tfrac{n}{2} + n^3 \\ &= n^3(\lg n - 1) + n^3 = n^3 \lg n\,. \end{aligned}$$

Logo, $T(n)$ é $\Theta(n^3 \lg n)$.

c) Troque "8" por "7" no algoritmo e seja $S(n)$ o número de vezes que a atribuição "$z \leftarrow 0$" é executada neste novo algoritmo. A que classe O pertence $S(n)$? Procure dar a resposta mais justa possível e justifique a sua resposta (sem usar o *Master Theorem*).
Solução: Temos que

$$\begin{aligned} S(1) &= 0 \\ S(n) &= 7\,S(n/2) + n^3 \qquad \text{para } n = 2, 4, 8, \ldots \end{aligned}$$

Eu acho que $S(n) < 8\,n^3$ para $n = 1, 2, 4, 8 \ldots$. Vou verificar isto por indução.
Se $n = 1$ então $S(n) = 0 < 8 = 8 \cdot 1^3 = 8\,n^3$.
Seja n uma potência de 2, $n \geq 2$ e suponha que a desigualdade vale para $n/2$:

$$\begin{aligned} S(n) &= 7\,S(n/2) + n^3 \\ &\overset{\text{hi}}{<} 7 \cdot 8\,\tfrac{n^3}{2^3} + n^3 \\ &= 7\,n^3 + n^3 = 8\,n^3\,. \end{aligned}$$

Logo, $S(n)$ é $\mathrm{O}(n^3)$. (Não é difícil mostrar que $S(n)$ é $\Theta(n^3)$.)

Questão 3 [2,5 pontos]
Um vetor $A[1\,..\,n]$ de inteiros é dito **compacto** se $|A[i] - A[i+1]| \leq 1$ para $i = 1, \ldots, n-1$. Escreva um algoritmo Busca(A, n, x) que, recebe um vetor compacto $A[1\,..\,n]$ e um inteiro x tais que $A[1] \leq x \leq A[n]$ e devolve um índice i em $\{1, \ldots, n\}$ tal que $A[i] = x$. Seu algoritmo deve consumir tempo $O(\lg n)$. Explique sucintamente porque seu algoritmo está correto e tem o consumo de tempo pedido.

Solução: Eis uma versão iterativa de solução.

Busca(A, n, x)
1 $p \leftarrow 1$ $\quad r \leftarrow n$
2 **enquanto** $A[p] < z$ **e** $z < A[r]$ **faça**
3 $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4 $\quad$ **se** $A[q] < z$
5 $\quad\quad$ **então** $p \leftarrow q + 1$
6 $\quad\quad$ **senão** $r \leftarrow q$
7 **se** $A[p] = z$
8 $\quad$ **então devolva** p
9 $\quad$ **senão devolva** r

Correção

Na linha 2 vale que:

$$\text{(i0) } 1 \leq p \leq r \leq n; \quad \text{e} \quad \text{(i1) } A[p] \leq z \leq A[r].$$

Devido a estas relações invariantes e a condição do **enquanto** da linha 2, é evidente que **se** o algoritmo pára então ele devolve um índice i em $\{1, \ldots, n\}$ tal que $A[i] = z$.

Demonstração de (i0). A invariante (i0) vale no início da primeira iteração. Considere o início de uma iteração em que $A[p] < z < A[r]$. Temos que o valor de q calculado na linha 3 na iteração é tal que $p \leq q < r$. (Hmmm, alguém me explica esse "$q < r$"? Não é possível que tenhamos $q = r$?). Logo, da maneira que p ou r é alterado pela linha 5 ou 6, vê-se que (i0) vale no início da próxima iteração.

Demonstração de (i1). A invariante (i1) vale no início da primeira iteração. Considere o início de uma iteração em que $A[p] < z < A[r]$. Se a linha 5 é executada, então tem-se que

$$A[q+1] \leq A[q] + 1 < z + 1 \leq z, \tag{1}$$

onde o primeiro "”$\leq$" vale pois A é compacto (Ufa, tinha que usar isto em algum lugar!). Se a linha 6 é executada, então vale que

$$z \leq A[q]. \tag{2}$$

De (1) e (2) concluí-se que (i1) vale no início da próxima iteração.

Observação. Dado que $A[1] \leq z \leq A[z]$, para que o algoritmo funcione basta que

$$A[i] - A[i+1] \leq 1$$

para $i = 1, \ldots, n-1$.

Consumo de tempo

O consumo de tempo do algoritmo é proporcional ao número de execuções da linha 2. Seja

$$\langle (p_0, r_0), (p_1, r_1), \ldots, (p_k, r_k) \rangle$$

A seqüência dos valores de p e q no início de cada execução da linha 2. Desta forma, por exemplo, $(p_0, r_0) = (1, n)$. Devido as linhas 4 e 5 temos que

$$\begin{aligned}
n_0 &:= r_0 - p_0 + 1 = n \\
n_1 &:= r_1 - p_1 + 1 \leq \left\lceil \frac{n_0}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil \\
n_2 &:= r_2 - p_2 + 1 \leq \left\lceil \frac{n_1}{2} \right\rceil = \left\lceil \frac{n}{2^2} \right\rceil \\
n_3 &:= r_3 - p_3 + 1 \leq \left\lceil \frac{n_2}{2} \right\rceil = \left\lceil \frac{n}{2^3} \right\rceil \\
&\cdots \\
n_i &:= r_i - p_i + 1 \leq \left\lceil \frac{n_{i-1}}{2} \right\rceil = \left\lceil \frac{n}{2^i} \right\rceil \\
&\cdots
\end{aligned}$$

Para cada i tem-se que

$$1 \leq n_i \leq \left\lceil \frac{n}{2^i} \right\rceil \leq \frac{n+1}{2^i}. \tag{3}$$

Logo, o número máximo de execuções da linha 2 é não superior a $\lfloor \lg(n+1) \rfloor$ e o consumo de tempo do algoritmo é $\mathrm{O}(\lg n)$. .

Questão 4 [2,5 pontos]
O algoritmo abaixo recebe um vetor de números inteiros $A[1\,..\,n]$ e rearranja os seus elementos. O algoritmo faz o mesmo que o BUILD-HEAP?

B-H (A, n)
1 **para** $m \leftarrow 2$ **até** n **faça**
2 $\quad i \leftarrow m$
3 $\quad$ **enquanto** $i > 1$ **e** $A[\lfloor i/2 \rfloor] < A[i]$ **faça**
4 $\qquad A[\lfloor i/2 \rfloor] \leftrightarrow A[i]$
5 $\qquad i \leftarrow \lfloor i/2 \rfloor$

Qual a relação invariante no início de cada iteração do bloco de linhas 2–5?
Qual o consumo de tempo do algoritmo BUILD-HEAP?
Qual o consumo de tempo do algoritmo B-H? Use a notação O, mas procure dar a resposta mais justa possível.

Solução: O algoritmo B-H faz o mesmo serviço que o BUILD-HEAP. A relação invariante que vale na linha 1 é

$$A[1\,..\,m{-}1] \text{ é um max-heap.}$$

Como na última vez que a linha 1 é executada temos que $m = n+1$, então ao final do algoritmo $A[1\,..\,n]$ é uma max-heap.

O consumo de tempo do algoritmo no pior caso é proporcional a

$$S := \lceil \lg 2 \rceil + \lceil \lg 3 \rceil + \cdots + \lceil \lg n \rceil,$$

que é o número de execuções das linha 4 e 5 no pior caso. (O número de execuções da linha 3 é não superior a $S + n$.) É fácil mostra que S é $\mathrm{O}(n \lg n)$. De fato,

$$\begin{aligned} S &\leq \lceil \lg n \rceil + \lceil \lg n \rceil + \cdots + \lceil \lg n \rceil, \\ &= (n-1)\lceil \lg n \rceil. \end{aligned}$$

Portanto, o consumo de tempo do algoritmo é $\mathrm{O}(n \lg n)$.

Bem, na realidade, o consumo de tempo no **pior caso** é $\Theta(n \lg n)$, já que

$$\begin{aligned} S &\geq \frac{n}{2} \lg \frac{n}{2} \\ &= \frac{n}{2} \lg n - \frac{n}{2} \\ &\geq \frac{n}{4} \lg n \end{aligned} \tag{4}$$

para n suficientemente grande.

Hmmm, um passarinho me contou que

$$\sum_{k=2}^{n} \lceil \lg k \rceil = n \lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1.$$

Alguém sabe demostrar isto?