MAC0338 Análise de Algoritmos

DCC–IME–USP, 29 de março de 2007

Instruções

(i) Esta prova contém cinco questões, cada uma vale dois pontos e meio.

(ii) Serão consideradas questões cujos valores somem até dez pontos.

(iii) Enuncie os teoremas e propriedades usados para justificar suas afirmações.

(iv) Você pode utilizar como subrotina qualquer algoritmo visto em sala de aula sem reescrevê-lo. No entanto, 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* $\mathrm{O}(n^n)$*."*

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

Solução meia-boca da prova

Questão 1 [2,5 pontos]
Foram implementados três algoritmos: A, B e C. As tabelas abaixo mostram o resultado de um estudo experimental dessas implementações. Nas tabelas, para alguns valores do tamanho n da entrada do problema, são indicados os consumos de tempo dos algoritmos em segundos. Apresente funções $T_A(n), T_B(n)$ e $T_C(n)$ para as quais você suspeita que os algoritmos A, B e C consomem tempo $\Theta(T_A(n)), \Theta(T_B(n))$ e $\Theta(T_C(n))$, respectivamente. Justifique a sua resposta.

n	`algoritmo A`
256	0.00
512	0.01
1024	0.03
2048	0.09
4096	0.37
8192	1.70
16384	7.08
32768	28.54
65536	113.55
131072	493.08

n	`algoritmo B`
256	0.00
512	0.00
1024	0.00
2048	0.00
4096	0.01
8192	0.01
16384	0.02
32768	0.04
65536	0.10
131072	0.23
262144	0.54
524288	1.31
1048576	3.06
2097152	7.07
4194304	15.84
8388608	35.85

n	`algoritmo C`
1000000	0.06
2000000	0.11
3000000	0.16
6000000	0.32
8000000	0.43
10000000	0.55
15000000	0.82
20000000	1.08
25000000	1.37
30000000	1.70

Solução: No algoritmo A, a medida que n dobra o consumo de tempo mais ou menos quadruplica. Veja o que acontece quando n vai de 2048 para 4096, de 4096 para 8192 ... O algoritmo tem, portanto, um comportamento aparentemente quadrático. Logo, é plausível chutar que o consumo de tempo é $\mathrm{O}(n^2)$.

No algoritmo C, a medidade que n dobra o consumo de tempo mais ou menos dobra. Veja que acontece quando n vai de 1000000 para 2000000, de 3000000 para 6000000. Quando n vai de 3000000 para 15000000 o tempo vai de 0,16 para 0,82 ($0{,}16 \times 5 = 0{,}80$). O comportamento desse algoritmo é aparentemente linear. Assim, é razoável chutar que o consumo de tempo é $\mathrm{O}(n)$.

Acho que decidir o consumo de tempo do algoritmo algoritmo B é o mais difícil do três. O consumo de tempo parece ser levemente superior a linear. Quando n vai de 32768 para 65536 o tempo vai de 0.04 para 0.10 (um pouco mais que $0{,}4 \times 2$). Quando olhamos para o crescimento no consumo de tempo quando n vai de 32768 para 3,06, percebemos que o algoritmo não é linear: $3.06 > 0{,}04 times 32 = 1{,}28$. Aqui seria razoável qualquer coisa do tipo $\mathrm{O}(f(n))$, onde $f(n)$ é uma função em $\Omega(n)$ e em $o(n^2)$. O consumo de tempo é $\mathrm{O}(n \lg n)$.

Questão 2 [2,5 pontos]
Suponha que, para entradas de tamanho n, você tenha que escolher um dentre três algoritmos A, B e C.

a) Algoritmo A resolve problemas dividindo-os em **cinco** subproblemas de metade do tamanho, recursivamente resolve cada subproblema e então combina as soluções em tempo $\mathrm{O}(n)$.

b) Algoritmo B resolve problemas dividindo-os em **dois** subproblemas de tamanho $n-1$, recursivamente resolve cada subproblema e então combina as soluções em tempo $\mathrm{O}(1)$.

c) Algoritmo C resolve problemas dividindo-os em **nove** subproblemas de tamanho $n/3$, recursivamente resolve cada subproblema e então combina as soluções em tempo $\mathrm{O}(n^2)$.

Qual o consumo de tempo de cada um desses algoritmos? Expresse as suas respostas em termos da notação O, mas procure dar as respostas mais justas possíveis. Qual algoritmo é assintoticamente mais eficiente no pior caso? Justifique as suas respostas.

Solução: Sem fazer as contas:

- algoritmo A consome tempo $\mathrm{O}(n^{\lg 5})$.
- algoritmo B consome tempo $\mathrm{O}(2^n)$.
- algoritmo C consome tempo $\mathrm{O}(n^2 \lg n)$.

O algoritmo assintoticamente mais eficiente no pior caso é o algoritmo C.

Questão 3 [2 pontos]
O algoritmo abaixo recebe um vetor de números inteiros $A[1\,..\,n]$ e rearranja os seus elementos.

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 $\qquad i \leftarrow \lfloor i/2 \rfloor$

O algoritmo B-H faz o mesmo que o BUILD-HEAP. Enuncie uma relação invariante que torne esse fato evidente. Não é necessário demonstrar que a relação é de fato invariante.

Qual o consumo de tempo do algoritmo BUILD-HEAP? Não é necessário demonstrar sua afirmação.

Qual o consumo de tempo do algoritmo B-H? Use a notação O, mas procure dar a resposta mais justa possível e justifique-a.

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,

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

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

$$\begin{aligned} S &\ge \frac{n}{2} \lg \frac{n}{2} \qquad (1) \\ &= \frac{n}{2} \lg n - \frac{n}{2} \\ &\ge \frac{n}{4} \lg n \end{aligned}$$

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 demonstrar isto?

Questão 4 [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

$$
\begin{array}{ccccccc}
n_0 & := & p - r + 1 & = & n & & \\
n_1 & := & p_1 - r_1 + 1 & \leq & n/2 & & \\
n_2 & := & p_2 - r_2 + 1 & \leq & n_1/2 & \leq & n/2^2 \\
n_3 & := & p_3 - r_3 + 1 & \leq & n_2/2 & \leq & n/2^3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
n_k & := & p_k - r_k + 1 & \leq & n_{k-1}/2 & \leq & n/2^k.
\end{array}
$$

De onde se conclui que $k \leq \lfloor \lg n \rfloor$ e a altura da pilha de recursão nunca ultrapassa $\lfloor \lg n \rfloor + 1 = \mathrm{O}(\lg n)$.

Questão 5 [2 pontos]
Escreva um algoritmo BUSCA(A, n, x) que recebe um vetor crescente $A[1\,..\,n]$ e um elemento x e devolve um índice i em $\{1, \ldots, n\}$ tal que $A[i] = x$; se x não está em $A[1\,..\,n]$ o algoritmo devolve 0. Seu algoritmo deve consumir tempo $\mathrm{O}(\lg n)$. 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 ← 1        r ← n
2  enquanto p ≤ r faça
3       q ← ⌊(p + r)/2⌋
4       se A[q] = x
5            então devolva q
6       se A[q] < x
7            então p ← q + 1
8            senão r ← q − 1
9  devolva 0
```

Correção

Na linha 2 vale que:

$$\text{(i0) } A[1\,..\,p-1] < x < A[r+1\,..\,n].$$

Se o algoritmo pára após devolver o índice q na linha 5, então é evidente que a resposta está correta. Por outro lado, devido a relação invariante (i0) e a condição do **enquanto** da linha 2, é claro **se** o algoritmo pára após devolver 0 na linha 9, então $p = r + 1$ e x, de fato, não está no vetor $A[1\,..\,n]$.

Demonstração de (i0). A relação invariante (i0) vale no início da primeira iteração, já que como $p = 1$ e $r = n$ então $A[1\,..\,p-1]$ e $A[r+1\,..\,n]$ são vetores vazios. Considere o início de uma iteração que não seja a última. Temos que o valor de q calculado na linha 3 é tal que $p \leq q \leq r$. Logo, como o vetor é crescente e da maneira que p ou r é alterado pela linha 7 ou 8, vemos que (i0) também vale no início da próxima iteração.

Consumo de tempo

O consumo de tempo de cada iteração das linhas 3-6 é $\Theta(1)$. Logo, 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 a escolha do valor de q na linha 3 e a atualização de p e r nas linhas 7 e 8, respectivamente, temos que

$$
\begin{aligned}
& n_0 := r_0 - p_0 + 1 = n \\
& n_1 := r_1 - p_1 + 1 \leq \left\lfloor \frac{n_0}{2} \right\rfloor = \left\lfloor \frac{n}{2} \right\rfloor \\
& n_2 := r_2 - p_2 + 1 \leq \left\lfloor \frac{n_1}{2} \right\rfloor = \left\lfloor \frac{n}{2^2} \right\rfloor \\
& n_3 := r_3 - p_3 + 1 \leq \left\lfloor \frac{n_2}{2} \right\rfloor = \left\lfloor \frac{n}{2^3} \right\rfloor \\
& \quad \ldots \\
& n_k := r_k - p_k + 1 \leq \left\lfloor \frac{n_{k-1}}{2} \right\rfloor = \left\lfloor \frac{n}{2^k} \right\rfloor
\end{aligned}
$$

Como

$$1 \leq n_k \leq \left\lfloor \frac{n}{2^k} \right\rfloor \leq \frac{n}{2^k} \ ,$$

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