Departamento de Ciência de Computadores — FCUP

Algoritmos (CC4010) — 2016/17

Exame (5.1.2017) — *duração: 3h*

Das quatro questões assinaladas por **(***)**, resolva apenas duas.

1. Averigue a veracidade de cada uma das afirmações seguintes e justifique a resposta partindo da definição das ordens de grandeza.

a) $2^n \in \Theta(3^n)$ **b)** $\log_2 n \in \Theta(\log_3 n)$ **c)** $3^n \in \Omega(2^n)$ **d)** $2^n + 10n \in O(2^n)$

2. Considere a função MyPartition assim definida, sendo x é um *array* de n inteiros $x[1], \ldots, x[n]$, e a e b inteiros tais que $1 \leq a \leq b \leq n$.

```
MyPartition(x, a, b)
  z = x[a]
  j = a
  for i = a + 1 to b do
      if x[i] < z then
          Exchange x[i] with x[j + 1]
          j = j + 1
  Exchange x[a] with x[j + 1]
  return j + 1
```

a) Apresente o valor de retorno e o estado do *array* x depois da execução de MyPartition(x, a, b), para $x = [7, 4, 5, 9, 2, 7, 9, 8, 7, 10, 4]$, com $a = 1$ e $b = 11$.

b) No caso geral, qual é o estado de x e o valor de retorno após a execução MyPartition(x, a, b)? Qual é o invariante de ciclo (para x, j e i e z) que permite justificar essa resposta? Prove que esse invariante é preservado em cada iteração.

c) Usando pseudocódigo e MyPartition, escreva uma versão de *quicksort* que permita ordenar x por ordem crescente (i.e., não decrescente). Justifique a terminação e correção da função que apresentou.

d) Dê exemplo de uma instância genérica $x[1], \ldots, x[n]$ que permita provar que a complexidade temporal assintótica de *quicksort* é $\Theta(n^2)$ no pior caso. Apresente a prova usando custos 1 para operações $\Theta(1)$.

e) Sendo o tempo de execução de *quicksort* no caso médio e o valor esperado do tempo de execução de *randomized quicksort* da mesma ordem de grandeza, porque se optaria pelo último? Pode o tempo de execução de uma chamada de *randomized quicksort* ser $\Theta(n^2)$ para a instância que indicou em **2d)**?

f) Apresente a noção de algoritmo de ordenação estável e diga, justificando, se o algoritmo *quicksort* que apresentou em **2c)** é estável.

3. **(***)** Apresente a prova de que qualquer algoritmo comparativo que resolva o problema da ordenação de um *array* de n inteiros, $x[1], \ldots, x[n]$, tem complexidade assintótica $\Omega(n \log_2 n)$ no pior caso.

4. Suponha que as funções QuickSelect e MediansOfFive implementam os algoritmos de seleção ordinal dados nas aulas, para seleção do k-ésimo menor elemento de $x[a], \ldots, x[b]$, sendo x um *array* de n elementos, e x, a, b e k os parâmetros da função, com $1 \leq a \leq b \leq n$ e $1 \leq k \leq b - a + 1$.

a) No pior caso, qual é a ordem de grandeza do tempo de execução de cada uma das funções quando executada para determinar a mediana de um *array* de n inteiros, $x[1], \ldots, x[n]$, com n ímpar?

b) Como pode usar as funções para calcular o mínimo e o máximo de $x[a], \ldots, x[b]$? Para esse efeito, usaria alguma delas? Se sim, porquê, se não, porquê?

(Continua, v.p.f.)

5. $\boxed{\textbf{(***)}}$ Usando o método contabilístico ou o método do potencial, apresente a prova de que o custo amortizado por operação de uma sequência de n operações (PUSH/POP) sobre uma *stack* de inteiros, suportada por um *array*, com duplicação do espaço se necessário (por realocação) é $O(1)$.

6. No problema BIN PACKING são dados n itens com pesos $p_1, \ldots, p_n$, tais que $0 < p_i \leq 1$, para todo i, e há que os distribuir por latas de capacidade unitária, usando o menor número de latas possível. Considere o algoritmo seguinte, sendo n e p dados do problema, e b e x *arrays* de n elementos e $n \geq 1$.

```
FIRSTFIT(p, b, n, x)
  t = 1;  b[1] = p[1];  x[1] = 1
  for j = 2 to n do b[j] = 0
  for i = 2 to n do
      j = 1
      while (j ≤ t and p[i] + b[j] > 1)  do   j = j + 1
      if (j > t) then t = t + 1
      b[j] = b[j] + p[i];   x[i] = j
  return t
```

a) Para a instância $n = 7$ e $p = [0.9, 0.4, 0.8, 0.3, 0.6, 0.5, 0.2]$, qual é o valor final de x, b e t? Conclua que FIRSTFIT não determina a solução ótima de BIN PACKING.

b) Prove que FIRSTFIT determina em x uma distribuição admissível dos itens pelas latas (i.e., que não excede a capacidade das latas, podendo não ser ótima). Para tal, indique os invariantes de ciclo que nos permitem chegar a essa conclusão e prove que são preservados em cada iteração do ciclo correspondente.

c) Determine a complexidade temporal assintótica de FIRSTFIT no melhor caso e no pior caso.

d) $\boxed{\textbf{(***)}}$ Prove que FIRSTFIT produz uma solução aproximada, com fator de aproximação 2.

7. No problema PARTITION são dados n inteiros positivos $a_1, a_2, \ldots, a_n$, e há que decidir se existe uma partição $\{S, T\}$ do conjunto de índices $\{1, 2, \ldots, n\}$ tal que $\sum_{i \in S} a_i = \sum_{i \in T} a_i$. Sabe-se que PARTITION é um problema **NP-completo**.

a) Prove que as instâncias de PARTITION com $a_j > \sum_{j \neq i} a_i$, para algum j, são trivialmente decidíveis.

b) $\boxed{\textbf{(***)}}$ Dada uma instância de PARTITION, com $a_j \leq \sum_{j \neq i} a_i$, para todo j, definimos uma instância de BIN PACKING com $p_j = 2a_j / \sum_{i=1}^{n} a_i$, para todo j. Justifique que se trata de uma redução polinomial que permite decidir PARTITION em tempo polinomial se algum dos algoritmos seguintes existir e conclua que não podem existir a menos que P=NP: (i) um algoritmo polinomial que calcule uma solução ótima para BIN PACKING; (ii) um algoritmo de aproximação polinomial de razão c para BIN PACKING, com $c < 3/2$.

8. Recorde o problema *interval scheduling*, em que são dadas n tarefas e as suas datas de início e de conclusão, e há que selecionar as tarefas que serão realizadas, de modo a efetuar o maior número possível. Em cada instante, estará a decorrer no máximo uma tarefa. Seja $[s_i, f_i[$ o intervalo em que a tarefa t_i teria de ser realizada, para $1 \leq i \leq n$, sendo $s_i < f_i$.

a) Assuma que são dados dois *arrays* $\mathcal{A}$ e $\mathcal{B}$ que definem uma ordenação das tarefas por ordem crescente de data de início e de conclusão ($s_{\mathcal{A}[i]} \leq s_{\mathcal{A}[j]}$ se $i < j$ e, analogamente, $f_{\mathcal{B}[i]} \leq f_{\mathcal{B}[j]}$ se $i < j$). Apresente em pseudocódigo um algoritmo *greedy*, com complexidade $O(n)$, que resolva o problema *interval scheduling*. Justifique a correção do algoritmo que apresentou e a sua complexidade.

b) Assuma que as tarefas estão ordenadas por ordem crescente de data de conclusão. Para cada par (i, j), com $i < j$, seja $N(i, j)$ o número máximo de tarefas que conseguiria realizar após o fim de t_i e antes do início de t_j, e seja C_{ij} o conjunto dos k's tais que se tem $f_i \leq s_k$ e $f_k \leq s_j$ para t_k.

1. Justifique que $N(i, j) = 0$, se $C_{ij} = \{\}$ e, caso contrário, $N(i, j) = \max_{k \in C_{ij}} (N(i, k) + N(k, j) + 1)$.
2. Como usar tal facto para obter uma solução para *interval scheduling*, em tempo polinomial, por programação dinâmica (com memoização), com indicação das tarefas a realizar? **(FIM)**

Master theorem:

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$, for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Stirling's approximation:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + \Theta(1/n)) = \sqrt{2\pi n} \left(\frac{n}{e}\right)^{\alpha_n}, \quad \text{with } 1/(12n+1) < \alpha_n < 1/(12n)$$

Some useful results:

$$\log(\prod_{k=1}^{n} a_k) = \sum_{k=1}^{n} \log a_k \qquad\qquad \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}, \quad \text{for } |x| < 1$$

If $(u_k)_k$ is an arithmetic progression (i.e., $u_{k+1} = r + u_k$, for some constant $r \neq 0$), then $\sum_{k=1}^{n} u_k = \frac{(u_1+u_n)n}{2}$.

If $(u_k)_k$ is a geometric progression (i.e., $u_{k+1} = ru_k$, for some constant $r \neq 1$), then $\sum_{k=1}^{n} u_k = \frac{u_{n+1}-u_1}{r-1}$.

If $f \geq 0$ is continuous and a monotonically increasing function, then

$$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$$