Estrutura de Dados - 1o. período de 2008

Gabarito da Primeira Avaliação à Distância

1. Numa conversa de bar, alguns amigos discutiam complexidades de algoritmos. As complexidades discutidas eram funções dos seguintes tipos: exponencial, logarítmica, quadrática, linear, cúbica, constante. Coloque estas funções em ordem crescente de complexidade.

   Resposta: Constante, logarítmica, linear, quadrática, cúbica, exponencial.

2. Para cada item abaixo, responda "certo", "errado" ou "nada se pode concluir". Justifique.

   a. Se um limite inferior para um problema P é n^2, então nenhum algoritmo ótimo para P pode ter complexidade de pior caso $O(n^3)$.

   Resposta: Nada se pode concluir. Se existir um algoritmo A para P cuja complexidade de pior caso é $\Theta(n^2)$, então A é um algoritmo ótimo e é $O(n^3)$. Neste caso, todos os algoritmos ótimos para P terão complexidade de pior caso $\Theta(n^2)$, e, portanto, $O(n^3)$. No entanto, o fato de um limite inferior conhecido para P ser n^2 não significa que não exista limite inferior maior para P (por exemplo, n^4). Assim, se n^4 também for limite inferior para P, então nenhum algoritmo ótimo para P poderá ter complexidade de pior caso $O(n^3)$.

   b. Se um limite inferior para um problema P é n^2, então qualquer algoritmo ótimo para P tem complexidade de pior caso $O(n^2)$.

   Resposta: Nada se pode concluir. Como no item anterior, se existir um algoritmo para P cuja complexidade de pior caso é $\Theta(n^2)$, então qualquer algoritmo ótimo para P tem complexidade de pior caso $\Theta(n^2)$, e, portanto, $O(n^2)$. Mas, se existir um limite inferior maior para P, como n^4, então nenhum algoritmo ótimo para P poderá ter complexidade de pior caso $O(n^2)$.

   c. Se um limite inferior para um problema P é n^2, e se A é um algoritmo que resolve P com complexidade de pior caso $\Omega(n^2)$, então A é um algoritmo ótimo.

   Resposta: Falso. Pela definição de limite inferior, a complexidade de pior caso de qualquer algoritmo que resolva P é $\Omega(n^2)$.

3. No instante $t = 0$, uma cultura de bactérias contém 8×10^6 indivíduos. No instante $t = i$ (sendo i um inteiro positivo que expressa o número de horas), o número de indivíduos na cultura é o dobro do número de indivíduos no instante anterior menos 7×10^3. Escreva dois algoritmos, um recursivo e outro não-recursivo, que calculem o número de indivíduos presentes na cultura no instante i. Calcule as complexidades dos algoritmos.

   Resposta:

   <u>Algoritmo recursivo:</u>

   função $ind(j)$
       se $j = 0$ então
           $ind(j) := 8 \times 10^6$

senão

$ind(j) := 2\,ind(j-1) - 7 \times 10^3$

Chamada externa: $ind(i)$

Complexidade: Seja $T(j)$ o número de passos do algoritmo no instante $t = j$. Temos:

$T(0) = 1$

$T(j) = T(j-1) + 1$

Resolvendo esta recorrência, verificamos que a complexidade deste algoritmo é $\Theta(i)$.

Algoritmo não recursivo:

$ind[0] := 8 \times 10^6$

para $j = 1, \cdots, i$ faça

$ind[j] := 2 \times ind[j-1] - 7 \times 10^3$

Complexidade: Este algoritmo possui apenas um loop que realiza exatamente i iterações, e em cada uma delas apenas um comando é executado. Logo, sua complexidade é $\Theta(i)$.

4. Determinar a expressão da complexidade média de uma busca ORDENADA de 10 chaves, em que a probabilidade de busca da chave i é igual à da chave $i+1$, $i = 1, ..., 9$. Supor, ainda, que a probabilidade de a chave procurada se encontrar na lista é igual a 100%.

   Resposta:

   Como a busca se dá em uma lista ordenada, temos 21 entradas distintas (10 entradas em que a chave é encontrada e 11 entradas correspondentes a fracasso). Sejam $E_1, \cdots, E_{10}$ as entradas correspondentes ao sucesso. Temos:

   $p(E_1) + p(E_2) + \cdots + p(E_{10}) = 1$

   $p(E_1) = p(E_2) = \cdots = p(E_{10})$

   Logo: $p(E_i) = \frac{1}{10}, \quad 1 \le i \le 10.$

   O número de passos necessários para cada entrada é:

   $t(E_i) = i, \quad 1 \le i \le 10.$

   Como a probabilidade de fracasso é 0 para qualquer entrada, o somatório referente ao fracasso não contribui para o cálculo da complexidade média.

   A complexidade média é dada por:

$$
\begin{aligned}
C.M. &= \sum_{i=1}^{10} p(E_i)\, t(E_i) \\
&= \frac{1}{10} \sum_{i=1}^{10} i \;=\; 5,5
\end{aligned}
$$

5. Escrever algoritmos de busca, inserção e remoção em LISTAS SIMPLESMENTE ENCADEADAS ORDENADAS.

   Resposta:

   Busca:

   procedimento busca-enc-ord($x, ant, pont$)

   $\quad ant := ptlista$

   $\quad pont := \lambda$

   $\quad ptr := ptlista \uparrow .prox$

   $\quad$ enquanto $ptr \neq \lambda$ faça

   $\qquad$ se $ptr \uparrow .chave < x$ então

   $\qquad\quad ant := ptr$

   $\qquad\quad ptr := ptr \uparrow .prox$

   $\qquad$ senão

   $\qquad\quad$ se $ptr \uparrow .chave = x$ então

   $\qquad\qquad pont := ptr$

   $\qquad\quad ptr := \lambda$

   Inserção:

   busca-enc-ord($x, ant, pont$)

   se $pont \neq \lambda$ então

   $\quad$ *"elemento já existe na lista"*

   senão

   $\quad ocupar(pt)$

   $\quad pt \uparrow .info :=$ novo_valor

   $\quad pt \uparrow .chave := x$

   $\quad pt \uparrow .prox := ant \uparrow .prox$

   $\quad ant \uparrow .prox := pt$

   Remoção:

   busca-enc-ord($x, ant, pont$)

   se $pont = \lambda$ então

   $\quad$ *"elemento não existe na lista"*

   senão

   $\quad ant \uparrow .prox := pont \uparrow .prox$

   $\quad$ valor-recuperado:= $pont \uparrow .info$

   $\quad desocupar(pont)$

6. Sejam L_1 e L_2 duas listas ordenadas, simplesmente encadeadas com nó-cabeça. Apresentar um algoritmo que construa uma lista ordenada contendo os elementos que pertencem simultaneamente a L_1 e L_2. (Isto é, aqueles elementos que se encontram em ambas as listas.) Supor que os elementos em cada lista são todos distintos.

   Resposta:

Algoritmo:

$pont1 := ptlista1 \uparrow .prox$ % ponteiro para a lista 1
$pont2 := ptlista2 \uparrow .prox$ % ponteiro para a lista 2
$ptaux := ptnovo$ % a lista resultante iniciará em $ptnovo$

enquanto $pont1 \neq \lambda$ e $pont2 \neq \lambda$ faça
    se $pont1 \uparrow .info < pont2 \uparrow .info$ então
        $pont1 := pont1 \uparrow .prox$
    senão
        se $pont2 \uparrow .info < pont1 \uparrow .info$ então
            $pont2 := pont2 \uparrow .prox$
        senão % são elementos iguais!
            $ocupar(pt)$
            $pt \uparrow .info := pont1 \uparrow .info$
            $pt \uparrow .prox := \lambda$
            $ptaux \uparrow .prox := pt$
            $ptaux := pt$ % ptaux aponta para o último nó
            $pont1 := pont1 \uparrow .prox$
            $pont2 := pont2 \uparrow .prox$

7. Escreva uma versão NÃO RECURSIVA do Algoritmo das Torres de Hanói que se encontra no livro-texto. Sugestão: utilize pilhas!

   Resposta: Considere uma pilha P, que armazena uma estrutura de dados da seguinte forma: (X, Y, Z, n), tal que X, Y, Z indicam pinos e n armazena um inteiro positivo. Sejam A, B, C os pinos de origem, trabalho e destino, respectivamente, e n o número de discos do problema.

   Algoritmo:

   $topo := 1$
   $P[topo] := (A, B, C, n)$
   enquanto $topo > 0$ faça
       $(X, Y, Z, n) := P[topo]$
       $topo := topo - 1$
       se $n = 1$ então $mover(X, Z)$ % move o disco do topo de X para Z
       senão
           $topo := topo + 1$
           $P[topo] := (Y, X, Z, n - 1)$
           $topo := topo + 1$
           $P[topo] := (X, Y, Z, 1)$
           $topo := topo + 1$
           $P[topo] := (X, Z, Y, n - 1)$

8. Um lava-rápido tem capacidade máxima de atendimento de 5 carros (um que está sendo lavado, e quatro em espera). Cada lavagem leva 3 minutos. Ao chegar um novo cliente, o sistema ou o atende imediatamente (caso esteja completamente livre), ou o coloca em espera, ou o rejeita (caso já existam 5 carros sendo atendidos). Escreva um algoritmo que leia um inteiro $n \geq 1$ e um vetor binário (por exemplo, 001011100111000111), onde o i-ésimo bit da esquerda para a direita vale "1" se um novo cliente chega no i-ésimo minuto, e "0" se nenhum novo cliente chega no i-ésimo minuto. A seguir, o algoritmo deve calcular quantos carros foram lavados pelo sistema até o n-ésimo minuto. (Suponha que o vetor tem pelo menos n bits.) Use uma fila para representar o sistema a cada minuto que passa.

   Resposta: O algoritmo proposto recebe um inteiro n e um vetor binário S com pelo menos n bits, e utiliza uma fila circular F com 5 posições, que representa o sistema a cada minuto. F está vazia quando os ponteiros ini e fim, que apontam para o início e o final de F, respectivamente, valem 0. Se $ini \neq 0$, então há algum carro sendo lavado. Ao final, a variável *ncarros* armazena o total de carros que foram lavados até o n-ésimo minuto.

```
procedimento lava-carros(n, S)
    ini := 0
    fim := 0
    ncarros := 0
    inicio := 0                         % armazena o minuto em que um carro começou a ser lavado
    para i = 1 ··· n faça
        se S[i] = 1 então
            se ini = 0 então                  % não há nenhum carro sendo lavado
                ini := 1
                fim := 1
                F[fim] := 1
                inicio := i                   % a lavagem do carro é iniciada
            senão
                se ini ≠ (fim mod 5) + 1 então          % F não está cheia
                    fim := (fim mod 5) + 1
                    F[fim] := 1
                senão "carro rejeitado"                 % F está cheia
        se ini ≠ 0 então                        % algum carro está sendo lavado
            se i = inicio + 2 então             % sua lavagem termina no fim do minuto i
                F[ini] := 0                     % termina a lavagem do carro atual
                ncarros := ncarros + 1
                se ini ≠ fim então              % há carros na fila
                    inicio := i+1       % a lavagem do próximo carro se inicia no minuto seguinte
                    ini := (ini mod 5) + 1      % ini aponta para o carro que será lavado
                senão                           % não há carros na fila
                    ini := 0
                    fim := 0
    imprimir ("Total de carros lavados" + ncarros )
```