Estrutura de Dados - 1o. período de 2012

Gabarito da Primeira Avaliação à Distância

1. Para cada item abaixo, responda "certo" ou "errado", justificando:

   (a) (0,25) A função $n^2 + n^2 \log n$ é $O(n^2)$.
   Resposta: Errado. A função é $O(n^2 \log n)$. Podemos também dizer que ela é $\Omega(n^2)$.

   (b) (0,25) A função $n^2 + n^2 \log n$ é $\Theta(n^2 \log n)$.
   Resposta: Certo. Como ela é tanto $O(n^2 \log n)$ quanto $\Omega(n^2 \log n)$, podemos dizer que ela é $\Theta(n^2 \log n)$.

   (c) (0,25) A função $n^2 + n^2 \log n$ é $\Omega(n \log n)$.
   Resposta: Certo. Como ela é $\Omega(n^2 \log n)$, obviamente também é $\Omega(n \log n)$, já que $n^2 \log n$ limita superiormente $n \log n$, considerando valores assintóticos.

   (d) (0,25) Se as complexidades assintóticas de pior caso dos algoritmos A e B são iguais a $\Theta(f)$, então, dada uma entrada E de tamanho n, A e B executam o mesmo número de passos para resolver E.
   Resposta: Errado. Não necessariamente E é uma entrada que represente o pior caso tanto para A quanto para B. Logo, para uma mesma entrada, não podemos afirmar que exista uma função que represente o número de passos executados tanto por A quanto por B. Além disso, mesmo considerando que E seja uma entrada de pior caso para A e para B, podemos ter, por exemplo, o número de passos executados por A no pior caso igual a $2f(n)$, e por B igual a $3f(n)$ (já que as constantes foram desprezadas ao se afirmar que as complexidades de pior caso de A e B são $\Theta(f)$).

2. (1,0) Determinar a expressão da complexidade média de uma busca "ordenada" de 10 chaves, em que a probabilidade de busca da chave i é 10% maior que a probabilidade de busca da chave $i-1$, para $i = 2, \cdots 10$. Supor, ainda, que a probabilidade de a chave procurada se encontrar na lista é igual a 50%.

   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 e $E'_0, \cdots, E'_{10}$ as entradas correspondentes ao fracasso (representando os "espaços" entre as chaves da lista).

   Considerando a probabilidade de sucesso, temos:

$$p(E_1) + p(E_2) + \cdots + p(E_{10}) = \frac{1}{2}$$

   Seja p a probabilidade de busca da chave 1 (entrada E_1). Logo:

$$p + \frac{11}{10}p + \left(\frac{11}{10}\right)^2 p + \cdots + \left(\frac{11}{10}\right)^9 p = \frac{1}{2}$$

$$p \sum_{i=1}^{10} \left(\frac{11}{10}\right)^{i-1} = \frac{1}{2}$$

$$p \left[\frac{\left(\frac{11}{10}\right)^{10} - 1}{\frac{1}{10}}\right] = \frac{1}{2}$$

$$p = \frac{1}{20\left[\left(\frac{11}{10}\right)^{10} - 1\right]}$$

Considerando a probabilidade de fracasso, temos:

$$p(E'_0) + p(E'_1) + \cdots + p(E'_{10}) = \frac{1}{2}$$

Assumindo que as probabilidades de $E'_0, \cdots, E'_{10}$ são iguais entre si, temos:

$$p(E'_i) = \frac{1}{22}, \quad 0 \leq i \leq 10$$

O número de passos necessários para cada entrada é:

$t(E_i) = i \;, \qquad 1 \leq i \leq 10$
$t(E'_i) = i + 1 \;, \quad 0 \leq i \leq 10$

Logo, a expressão da complexidade média é dada por:

$$\begin{aligned}
C.M. \quad &= \sum_{i=1}^{10} p(E_i)\, t(E_i) + \sum_{i=0}^{10} p(E'_i)\, t(E'_i) \\
&= \sum_{i=1}^{10} \left[\left(\frac{11}{10}\right)^{i-1} p\,.\,i\right] + \sum_{i=0}^{10} \left[\frac{1}{22}(i+1)\right] \\
&= p\left[1.1 + 2.\frac{11}{10} + 3.\left(\frac{11}{10}\right)^2 + \cdots + 10.\left(\frac{11}{10}\right)^9\right] + \frac{1}{22}\,.\,66 \\
&= \frac{1}{20\left[\left(\frac{11}{10}\right)^{10} - 1\right]}\left[1.1 + 2.\frac{11}{10} + 3.\left(\frac{11}{10}\right)^2 + \cdots + 10.\left(\frac{11}{10}\right)^9\right] + 3
\end{aligned}$$

3. (1,0) Comparar as complexidades assintóticas de melhor e pior caso dos algoritmos de busca, inserção e remoção em *listas sequenciais não ordenadas* e *listas sequenciais ordenadas.*

   Resposta:

	Listas sequenciais					
	Não ordenadas			Ordenadas		
	Busca	Inserção	Remoção	Busca	Inserção	Remoção
Melhor caso	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)
Pior caso	O(n)	O(1)	O(1)	O(log n)	O(n)	O(n)

Obs: Para as complexidades de inserção e remoção, não foi considerada uma busca prévia.

4. (1,0) Forneça um exemplo de entrada para o algoritmo de busca binária que leva o algoritmo ao pior caso, em relação ao número de comparações efetuadas. Assuma que n (número de elementos) é igual a 20. Mostre todas as comparações que foram efetuadas ao longo da execução do algoritmo.

   Resposta: Seja x o elemento a ser buscado. Cada chamada do algoritmo é da forma *busca-bin*(L, i, n, x), onde L é a lista de entrada, e i e n os índices inicial e final da busca, respectivamente. O algoritmo retorna o índice do elemento procurado, caso ele esteja em L, ou -1, em caso contrário.

   Exemplo: $L = \{10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105\}$ e $x = 110$.
   Chamada inicial: *busca-bin*$(L, 1, 20, 110)$.
   Passos:
   1) $i = 1$, $f = 20$, $L[10] = 55 < 110$. Executamos *busca-bin*$(L, 11, 20, 110)$.
   2) $i = 11$, $f = 20$, $L[15] = 80 < 110$. Executamos *busca-bin*$(L, 16, 20, 110)$.
   3) $i = 16$, $f = 20$, $L[18] = 95 < 110$. Executamos *busca-bin*$(L, 19, 20, 110)$.
   4) $i = 19$, $f = 20$, $L[19] = 100 < 110$. Executamos *busca-bin*$(L, 20, 20, 110)$.
   5) $i = 20$, $f = 20$, $L[20] = 105 < 110$. Executamos *busca-bin*$(L, 21, 20, 110)$.
   6) $(i = 21) > (j = 20)$. Retorna -1.

5. (1,5) Seja L uma lista sequencial ordenada, implementada em um vetor com n elementos. Denote por $L[j]$ o elemento que se encontra na posição j, onde $1 \leq j \leq n$. Elabore um algoritmo que retire de L os elementos repetidos. Calcule sua complexidade.

   Resposta:

```
i := 1
enquanto (i < n) faça
    j := i + 1
    se (L[i] = L[j]) então        // primeiro repetido encontrado
        k := 1        // armazena o total de repetidos de L[i]
        l := j
        m := i + 1
        enquanto (l < n) e (L[i] = L[l + 1]) faça        // conta o total de repetidos
            k := k + 1
            l := l + 1
        enquanto (m ≤ (n − k)) faça        // remove os k repetidos
            L[m] := L[m + k]
            m := m + 1
        n := n − k
    i := i + 1
```

   No pior caso, pode haver $O(n)$ deslocamentos de um mesmo elemento do vetor. Logo, considerando os n elementos, o algoritmo é $O(n^2)$.

6. (1,5) Elabore um algoritmo que resolva o seguinte problema: Dados dois números inteiros positivos m e n, onde $m \geq n$, achar o **mínimo múltiplo comum** de m e n. Calcule a complexidade do seu algoritmo em função de m e n.

   Resposta:

$mmc := m$
$x := 1$
enquanto $(mmc \mod n) \neq 0$ faça
    $x := x + 1$
    $mmc := m * x$
imprimir('O mmc é:', mmc)

Complexidade: No pior caso, temos que o mmc entre m e n é $m.n$. Neste caso, o loop enquanto seria executado $n-1$ vezes. Logo, o algoritmo é $O(n)$.

7. (1,5) Elabore um algoritmo que utilize uma *pilha* para resolver o seguinte problema de contagem. Em um processo de votação, existem apenas dois tipos de votos ("a favor" e "contra"). É dada uma lista sequencial não ordenada V com n elementos que representa a votação, onde n é o número de votantes, $V[i] = 1$ significa um voto favorável e $V[i] = 0$ significa um voto contrário $(1 \leq i \leq n)$. Elabore um algoritmo que utiliza uma pilha auxiliar para decidir que tipo de voto é majoritário, ou se houve empate.

   Resposta: Sejam P a pilha utilizada e *topo* a variável que aponta para o topo de P. A cada voto lido de V, o algoritmo compara este voto com o armazenado no topo de P (caso P contenha algum voto). Se estes votos forem diferentes, então ambos são desconsiderados na decisão do vencedor. Logo, o voto lido de V não é empilhado e o voto do topo de P é removido. Se o voto lido de V for igual ao de P, então o voto de V também é empilhado. Ao final da análise de todos os votos, basta analisar o topo da pilha para saber se houve empate ou qual voto é vencedor.

$topo := 0$
para $i := 1$ até n faça
    se $(topo = 0)$ ou $(P[topo] = V[i])$ então
        $topo := topo + 1$
        $P[topo] := V[i]$
    senão
        $topo := topo - 1$
    $i := i + 1$
se $(topo = 0)$ então
    imprimir ("Houve *empate*")
senão
    se $(P[topo] = 1)$ então
        imprimir ("A maioria dos votos foi *a favor*")
    senão
        imprimir ("A maioria dos votos foi *contra*")

8. (1,5) Elabore um algoritmo que utilize uma *fila* para resolver o seguinte problema. É dada uma sequência B contendo apenas dois tipos de elementos, c e s. O elemento c indica a chegada de um novo cliente ao caixa, e o elemento s a saída de um cliente. Os clientes são atendidos por ordem de chegada. Existe apenas um caixa, e 6 cadeiras para a fila de espera. O cliente que está sendo atendido no momento é denotado por y, e os clientes que estão sentados aguardando sua vez são denotados por x's. Faça um algoritmo que leia a sequência B de entrada e imprima o estado atual da fila F que representa o atendimento. Exemplo: para uma sequência $B = c\ c\ c\ c\ s\ c\ s\ s\ c\ c\ c$,

a saída deve ser $F = x\ -\ -\ y\ x\ x\ x$, onde "$-$" representa uma posição vazia. (Lembre-se que a fila é implementada circularmente.) O algoritmo deve prever *overflow*, como no exemplo $B = c\ c\ c\ c\ c\ c\ c\ c$. Suponha sempre que o número de c's é sempre maior ou igual que o número de s's.

Resposta: Sejam n o número de elementos do vetor B e F uma fila circular com sete posições, representando as seis cadeiras de espera e o caixa. Sejam f e r os índices do início e do fim da fila, respectivamente.

```
f := 0
r := 0
para i := 1 até 7 faça        // inicializando F
        F[i] = −
para i := 1 até n faça        // percorrendo B
        se B[i] = s então
                F[f] := −
                se f = r então        // a fila ficará vazia
                        f := 0
                        r := 0
                senão
                        f := (f  mod  7) + 1
                        F[f] := y        // o próximo vai para o caixa
        senão
                r := (r  mod  7) + 1
                se f = r então
                        imprimir ("overflow")
                        i := n + 1
                senão
                        se f = 0 então        // primeiro elemento inserido em F
                                F[r] := y        // vai para o caixa
                                f = r
                        senão
                                F[r] := x
para i := 1 até 7 faça        // imprimindo F
        imprimir (F[i])
```