Gabarito da Primeira Avaliação à Distância
Segundo Semestre de 2011 - Estrutura de Dados e Algoritmos

1) Responda V ou F, justificando cuidadosamente (0,5 ponto cada item).

a) Um algoritmo é considerado ótimo quando sua complexidade de pior caso é a melhor possível, dentre todos os algoritmos existentes que resolvem a mesma tarefa.

Resposta: Falso. Mesmo que a complexidade de pior caso de um algoritmo seja a melhor (menor) dentre todos os algoritmos existentes, não necessariamente esta complexidade é igual ao limite inferior do problema. Neste caso, ainda não existe um algoritmo ótimo desenvolvido para este problema (tarefa).

b) Se a complexidade de melhor caso de um algoritmo é $\Theta(n)$, então a complexidade de caso médio é necessariamente $\Omega(n)$.

Resposta: Verdadeiro. Sendo a complexidade de melhor caso $\Theta(n)$, podemos afirmar que o algoritmo é $\Omega(n)$. Logo, todas as entradas, inclusive as correspondentes ao caso médio, são $\Omega(n)$.

2) (2 pontos) Descrever um algoritmo recursivo para resolver o problema da Torre de Hanói com 4 pinos e n discos, sendo o primeiro o pino de origem, o segundo o pino de destino, e os dois últimos os pinos de trabalho. Determine a complexidade do algoritmo (em termos do número de movimentos de discos efetuados no processo).

Resposta: Sejam A o pino de origem, B o pino destino, e X e Y os pinos de trabalho.

procedimento $hanoi4pinos(n, A, B, X, Y)$
  se $(n > 1)$ então
    $hanoi4pinos(n-2, A, X, Y, B)$
    mover o disco do topo de A para Y
    mover o disco do topo de A para B
    mover o disco do topo de Y para B
    $hanoi4pinos(n-2, X, B, Y, A)$
  senão se $(n = 1)$ então
      mover o disco do topo de A para B

O número de movimentos de discos do algoritmo corresponde à seguinte equação de recorrência:
$T(1) = 1$
$T(n) = 2T(n-2) + 3$, $n > 1$.

Temos então que $T(n) = 3(2^{\lceil \frac{n}{2} \rceil} - 1)$. Logo, o número de passos do algoritmo é $O(2^n)$.

3) (1 ponto) Determinar a expressão da complexidade média de uma busca sequencial não ordenada, com n chaves, supondo que a probabilidade de busca de qualquer chave, exceto a última, é igual a um terço da probabilidade de busca da chave seguinte. Supor também que a probabilidade de a chave procurada se encontrar na lista é igual a 50%.

Resposta:
Seja p a probabilidade de busca da chave 1. Temos então que $p + 3p + 9p + \cdots + 3^{n-1}p = 0,5$. Logo,

$$p \sum_{i=1}^{n} 3^{i-1} = \frac{1}{2}$$
$$p \left(\frac{3^n - 1}{2} \right) = \frac{1}{2}$$

$$p = \frac{1}{3^n - 1}$$

A expressão da complexidade média neste caso é dada por:

$$\begin{aligned} C.M. &= \sum_{i=1}^{n}(t_i \,.\, p_i) + 0,5n \\ &= \sum_{i=1}^{n}\left(i \,.\, \frac{3^{i-1}}{3^n - 1}\right) + 0,5n \\ &= \frac{1}{3^n - 1}\sum_{i=1}^{n}(i \,.\, 3^{i-1}) + 0,5n \end{aligned}$$

4) (1 ponto) Escreva um algoritmo para obter o segundo menor elemento de uma lista com n elementos. (Pode haver elementos repetidos.) Tente elaborar seu algoritmo de modo a minimizar a complexidade de pior caso. Determine esta complexidade.

Resposta:
Seja L uma lista com n elementos. Ao final da execução do algoritmo abaixo, a variável $menor1$ conterá o menor elemento de L e a variável $menor2$, o segundo menor.

```
se n ≥ 2 então
        menor1 := L[1]
        i := 2, j := 2          // i armazena o índice do 1o. elem. diferente de L[1], se existir
        enquanto j <= n faça
                se (L[i] = menor1) então
                        i := i + 1
                        j = j + 1
                senão j := n + 1          // encontrou um elemento diferente
se i > n então
        imprimir (Todos os elementos são iguais.)
senão
        se L[i] > menor1 então
                menor2 := L[i]
        senão
                menor2 := menor1
                menor1 := L[i]
                para j = i + 1 ··· n faça
                        se L[i] < menor1 então
                                menor2 := menor1
                                menor1 := L[i]
                        senão se L[i] ≠ menor1 e L[i] < menor2 então
                                   menor2 := L[i]
```

A complexidade do algoritmo é $\Theta(n)$.

5) (2 pontos) Sejam L_1 e L_2 duas listas sequenciais, ordenadas. (Pode haver elementos repetidos.) Escreva um algoritmo que gere uma lista sequencial ordenada, que corresponda à união das listas L_1 e L_2, mas *sem elementos repetidos*.

Resposta: Sejam $|L_1| = m$, $|L_2| = n$ e L_3 a lista resultante, de tamanho suficiente para armazenar o resultado.

$i := 1$ // percorre L_1
$j := 1$ // percorre L_2
$k := 1$ // percorre L_3

se $L_1[1] < L_2[1]$ então
    $L_3[1] := L_1[1]$
    $i := i + 1$
senão
    $L_3[1] := L_2[1]$
    $j := j + 1$
$k := k + 1$

enquanto $i \leq m$ e $j \leq n$ faça // percorrendo L_1 e L_2
    se $L_1[i] < L_2[j]$ então
        se $L_1[i] \neq L_3[k-1]$ então
            $L_3[k] := L_1[i]$
            $k := k + 1$
        $i = i + 1$
    senão
        se $L_2[j] \neq L_3[k-1]$ então
            $L_3[k] := L_2[j]$
            $k := k + 1$
        $j = j + 1$

se $i > m$ então // L[2] não foi percorrido até o final
    enquanto $j \leq n$ faça
        $L_3[k] := L_2[j]$
        $j = j + 1$
        $k := k + 1$
senão // L[1] não foi percorrido até o final
    enquanto $i \leq m$ faça
        $L_3[k] := L_1[i]$
        $i = i + 1$
        $k := k + 1$

6) (1 ponto) Considere duas pilhas implementadas em um único vetor V de posições 1 a n, onde a primeira pilha P_1 utiliza posições do vetor com índices crescentes $1, 2, 3, \cdots$ à medida que nela são realizadas inserções, e a segunda pilha P_2 utiliza posições do vetor com índices decrescentes $n, n-1, n-2, \cdots$, similarmente.

a) Escreva algoritmos de inserção para P_1 e P_2 (cuidado com as condições de overflow!).

Resposta: Sejam *topo*1 e *topo*2 as variáveis que indicam os topos das pilhas P_1 e P_2, respectivamente. Inicialmente, temos $topo1 = 0$ e $topo2 = n + 1$, indicando que P_1 e P_2 estão vazias.

Inserção em P_1:

se $topo2 = (topo1 + 1)$ então *overflow*
senão $topo1 := topo1 + 1$
    $V[topo1] :=$ *novo-valor*

Inserção em P_2:

se $topo2 = (topo1 + 1)$ então *overflow*
senão $topo2 := topo2 - 1$
    $V[topo2] :=$ *novo-valor*

b) Escreva algoritmos de remoção para P_1 e P_2 (cuidado com as condições de underflow!).

Remoção em P_1:

se $topo1 = 0$ então *underflow*
senão $valor := V[topo1]$
    $topo1 := topo1 - 1$

Remoção em P_2:

se $topo2 = n + 1$ então *underflow*
senão $valor := V[topo2]$
    $topo2 := topo2 + 1$

7) (2 pontos) Um micro-processador executa vários programas, da seguinte forma: enquanto um deles está sendo executado, os outros ficam numa fila de espera. A cada 50 microssegundos, o programa em execução pode voltar para o último lugar na fila de espera (caso ainda reste processamento para ele) ou ser encerrado (caso contrário); imediatamente, o primeiro programa na fila de espera passa a ocupar o processador pelos próximos 50 microssegundos. Suponha que cada novo programa que entra no sistema para ser executado é colocado sempre no último lugar da fila de espera. Suponha também que, quando um programa encerra seu processamento antes dos 50 microssegundos regulamentares, o primeiro programa da fila de espera passa a ocupar o processador imediatamente.

Escreva um algoritmo que, inicialmente, leia uma sequencia de n triplas $(a_1, t_1, p_1), (a_2, t_2, p_2), \cdots, (a_n, t_n, p_n)$, onde a_i representa a identificação de um programa, t_i o instante de tempo em que ele entra no sistema pela primeira vez, e p_i a duração total do programa. (Suponha que t_i e p_i são dados em microssegundos, e que $t_1 < t_2 < \cdots < t_n$.) A seguir, o programa lê um número t de microssegundos e exibe o estado atual do sistema: programa em execução, programas na fila de espera (em ordem), e tempo restante de processamento para cada um deles.

Resposta: Seja F uma fila circular, implementada com um vetor V de tamanho n. Cada posição de V é formada por uma tripla. Assume-se que, durante a execução do sistema, uma vez que uma tripla foi reinserida no final da fila em um instante t, por necessitar de mais de 50 microssegundos, todas as demais triplas possuem $t_i < t$, ou seja, já entraram no sistema (para evitar inconsistência na fila, relativa a triplas que ainda nem entraram no sistema estarem antes de triplas que já foram executadas).

Algoritmo:

```
// construção da lista
para i := 1 ··· n faça
        V[i].a := a_i
        V[i].t := t_i
        V[i].p := p_i
ini := 1
fim := n

// simulação do sistema

tempo := 0            // o sistema inicia no tempo 0
leia(tfinal)          // tempo t lido, para o qual será exibido o estado do sistema
enquanto tfinal > tempo faça
        se V[ini].t ≤ tempo então          // programa do início da fila já está no sistema
                se (tfinal − tempo) ≥ 50 então
                        se (V[ini].p) ≤ 50 então          // o processo sai do sistema
                                tempo := tempo + V[ini].p
                                ini := (ini mod n) + 1
                        senão          // o processo volta ao fim da fila
                                V[ini].p := V[ini].p − 50
                                tempo := tempo + 50
                                fim := (fim mod n) + 1
                                V[fim] := V[ini]
                                ini := (ini mod n) + 1
                senão
                        se (tfinal − tempo) ≥ V[ini].p então
                                tempo := tempo + V[ini].p
                                ini := (ini mod n) + 1
                        senão
                                V[ini].p := V[ini].p − (tfinal − tempo)
                                tempo := tfinal          // força o fim da simulação
        senão
                tempo := V[ini].t          // a simulação aguarda até a hora em que o processo entra no sistema

// impressão do estado do sistema

imprimir(Programa em execução: V[ini].a, V[ini].t, V[ini].p)
ini := (ini mod n) + 1
imprimir(Fila de espera: )
enquanto ini ≠ (fim mod n) + 1
        imprimir(V[ini].a, V[ini].t, V[ini].p)
        ini := (ini mod n) + 1
```