Estrutura de Dados - 1o. período de 2013

Gabarito da Primeira Avaliação à Distância

1. (1,0) Classifique as seguintes funções em ordem crescente de complexidade assintótica: $\sqrt{n}, n\log n, n^2, n^{1/3}+\log n, \log n, (1/3)^n, n, n-n^3+7n^5, n^3, (\log n)^2, n/\log n, (3/2)^n, 2^n, n!\,, \log\log n, 6$.

   Resposta: $(1/3)^n,\ 6,\ \log\log n,\ \log n,\ n^{1/3}+\log n,\ \sqrt{n},\ n/\log n,\ (\log n)^2,\ n,\ n\log n,\ n^2,\ n^3,\ n-n^3+7n^5,\ (3/2)^n,\ 2^n,\ n!$

2. (1,5) 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 existe um algoritmo ótimo para P de complexidade de pior caso $O(n^2)$.

   Resposta: Errado. Se existir também um limite inferior n^3, por exemplo, para P, então, pela definição de limite inferior, não é possível existir um algoritmo para P com complexidade de pior caso $O(n^2)$.

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

   Resposta: Certo. Pela definição de limite inferior, se n^2 é um limite inferior para P, então qualquer algoritmo para P tem complexidade de pior caso $\Omega(n^2)$. Assim, nenhum algoritmo, ótimo ou não, pode ter complexidade de pior caso $O(n\log n)$.

   c. Se um limite inferior para um problema P é n^2, então todo algoritmo ótimo para P tem complexidade de pior caso $\Omega(n^2)$.

   Resposta: Certo. Pela definição de limite inferior, se n^2 é um limite inferior para P, então qualquer algoritmo para P, ótimo ou não, tem complexidade de pior caso $\Omega(n^2)$.

3. (1,5) Determinar a complexidade média de uma busca não ordenada de 10 chaves, em que a probabilidade de busca da chave i é igual a um terço da probabilidade de busca da chave $i-1$, para $i = 2, ...10$. 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 não ordenada, temos 11 entradas distintas. Sejam $E_1, \cdots, E_{10}$ as entradas correspondentes ao sucesso e E_0 a entrada correspondente ao fracasso.

   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 entrada E_1. Logo:

$$p + \frac{1}{3}p + \left(\frac{1}{3}\right)^2 p + \cdots + \left(\frac{1}{3}\right)^9 p = \frac{1}{2}$$

$$p \sum_{i=1}^{10} \left(\frac{1}{3}\right)^{i-1} = \frac{1}{2}$$

$$p \left[\frac{1 - \left(\frac{1}{3}\right)^{10}}{\frac{2}{3}}\right] = \frac{1}{2}$$

$$p = \frac{1}{3} \cdot \frac{1}{1 - \left(\frac{1}{3}\right)^{10}}$$

Considerando a probabilidade de fracasso, temos:
$p(E_0) = \frac{1}{2}$

O número de passos necessários para cada entrada é:

$t(E_i) = i$, $\qquad 1 \le i \le 10$
$t(E_0) = 10$

Logo, a expressão da complexidade média é dada por:

$$\begin{aligned}
C.M. &= \sum_{i=1}^{10} p(E_i)\, t(E_i) + p(E_0)\, t(E_0) \\
&= \sum_{i=1}^{10} \left[\left(\frac{1}{3}\right)^{i-1} p \,.\, i\right] + \frac{1}{2}.10 \\
&= p \left[1.1 + 2.\frac{1}{3} + 3.\left(\frac{1}{3}\right)^2 + \cdots + 10.\left(\frac{1}{3}\right)^9\right] + 5 \\
&= \frac{1}{3} \cdot \frac{1}{1 - \left(\frac{1}{3}\right)^{10}} \left[1.1 + 2.\frac{1}{3} + 3.\left(\frac{1}{3}\right)^2 + \cdots + 10.\left(\frac{1}{3}\right)^9\right] + 5 \\
&= 7
\end{aligned}$$

4. (1,5) 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 exclusivamente a L_1 ou exclusivamente a L_2 (isto é, os elementos que pertencem a ambas as listas devem ser descartados).

   Resposta:

Algoritmo:

$pont1 := ptlista1 \uparrow .prox$ % ponteiro para a lista L_1
$pont2 := ptlista2 \uparrow .prox$ % ponteiro para a lista L_2
$ptaux := ptnovo$ % a lista resultante iniciará em $ptnovo$

```
enquanto pont1 ≠ λ e pont2 ≠ λ faça
    se pont1 ↑ .info < pont2 ↑ .info então
            ocupar(pt)
            pt ↑ .info := pont1 ↑ .info
            pt ↑ .prox := λ
            ptaux ↑ .prox := pt
            ptaux := pt                    % ptaux aponta para o último nó
            pont1 := pont1 ↑ .prox
    senão
        se pont1 ↑ .info > pont2 ↑ .info então
            ocupar(pt)
            pt ↑ .info := pont2 ↑ .info
            pt ↑ .prox := λ
            ptaux ↑ .prox := pt
            ptaux := pt
            pont2 := pont2 ↑ .prox
        senão
            pont1 := pont1 ↑ .prox
            pont2 := pont2 ↑ .prox
enquanto pont1 ≠ λ faça
    ocupar(pt)
    pt ↑ .info := pont1 ↑ .info
    pt ↑ .prox := λ
    ptaux ↑ .prox := pt
    ptaux := pt
    pont1 := pont1 ↑ .prox
enquanto pont2 ≠ λ faça
    ocupar(pt)
    pt ↑ .info := pont2 ↑ .info
    pt ↑ .prox := λ
    ptaux ↑ .prox := pt
    ptaux := pt
    pont2 := pont2 ↑ .prox
```

5. (1,0) Adapte o método iterativo de ordenação por bolha, tornando-o recursivo.

   Resposta: Seja V o vetor com n elementos, indexado de 1 a n.

   Chamada externa: $bolha\_rec(V, 2)$

procedimento $bolha\_rec(V, i)$
    $bolha = i$
    para $j = i - 1$ até 1 faça
        se $V[j] > V[bolha]$ então
            $temp = V[j]$
            $V[j] = V[bolha]$
            $V[bolha] = temp$
            $bolha = j$
    se $i < n$ faça
        $bolha\_rec(V, i + 1)$

6. (1,5) Escreva um algoritmo eficiente que verifique se uma cadeia de caracteres é da forma xCx, onde x é uma cadeia qualquer formada por caracteres A ou B. Dica: utilize uma fila auxiliar.

   Resposta: Seja F a fila com n elementos, indexados de 1 a n, que contém a cadeia de caracteres. Seja H a fila auxiliar utilizada. Os índices i e j serão usados para percorrer as filas F e H, respectivamente. A variável h armazena o total de caracteres de H.

   Algoritmo:

   $i = 1$
   enquanto $F[i] \neq$ 'C' faça
       $H[i] = F[i]$
       $i = i + 1$
   $h = i - 1$
   se $h \neq (n - i)$ então   % as subcadeias separadas por 'C' tem tamanhos diferentes
       imprimir ("A cadeia não é da forma xCx")
   senão
       $i = i + 1$ % posiciona i no 1º caractere depois de 'C' em F
       $j = 1$ % posiciona j no 1º caractere de H
       enquanto $j \leq h$ faça
           se $F[i] \neq H[j]$ então
               $j = h + 2$ % força o fim das comparações
           senão
               $i = i + 1$
               $j = j + 1$
       se $j > h + 1$ então
           imprimir ("A cadeia não é da forma xCx")
       senão
           imprimir ("A cadeia é da forma xCx")

7. (1,0) O *percurso em altura* de uma árvore binária é aquele em que os nós são dispostos em ordem não decrescente de suas alturas. Descrever um algoritmo para efetuar um percurso em altura de uma árvore binária.

Resposta: Seja h a altura da árvore dada. O algoritmo a seguir percorre a árvore $h-1$ vezes, imprimindo e removendo suas folhas (enquanto a raiz não for uma folha). No procedimento busca-folhas, é utilizada uma variável chamada *lado*, configurada com 0 quando o nó é filho esquerdo de seu pai, ou com 1, quando é filho direito.

Algoritmo:

enquanto ($ptraiz \uparrow .esq \neq \lambda$) ou ($ptraiz \uparrow .dir \neq \lambda$) faça
    se ($ptraiz \uparrow .esq \neq \lambda$) então
        busca-folhas($ptraiz \uparrow .esq, ptraiz, 0$)
    se ($ptraiz \uparrow .dir \neq \lambda$) então
        busca-folhas($ptraiz \uparrow .dir, ptraiz, 1$)
$imprimir(ptraiz \uparrow .info)$

procedimento busca-folhas($no, pai, lado$)
    se ($no \uparrow .esq = \lambda$) e ($no \uparrow .dir = \lambda$) então
        se ($lado = 0$) então $pai \uparrow .esq = \lambda$
                senão $pai \uparrow .dir = \lambda$
        $imprimir(no)$
        $desocupar(no)$
    senão
        se ($no \uparrow .esq \neq \lambda$) então busca-folhas($no \uparrow .esq, no, 0$)
        se ($no \uparrow .dir \neq \lambda$) então busca-folhas($no \uparrow .dir, no, 1$)

8. (1,0) Prove ou desprove a seguinte afirmação: Em toda árvore estritamente binária com n folhas ($n \geq 2$), o número de nós internos (que não são folhas) é exatamente igual a $n-1$.

   Resposta: A afirmação é verdadeira, e a prova será dada por indução. Como casos-base, temos que a afirmação é válida para $n=1$ (a árvore contém apenas a raiz, que é uma folha, e não há nó interno) e $n=3$ (temos uma raiz, que é um nó interno, e dois filhos da raiz, que são folhas). No passo indutivo, seja a afirmação válida para $n-1$ folhas. Logo, uma árvore estritamente binária com $n-1$ folhas contém $n-2$ nós internos. Para aumentarmos o número de nós desta árvore com um número mínimo de inserções, mantendo-a estritamente binária, precisamos inserir dois novos nós x_1 e x_2. Estes são necessariamente inseridos como filhos de uma mesma folha f (considerando apenas o "desenho" da árvore, e não os valores contidos nos nós). Após estas inserções, f passa a ser nó interno e x_1 e x_2 novas folhas. Logo, passamos a ter $n-1-1+2=n$ folhas e $n-2+1=n-1$ nós internos.