Estrutura de Dados - 2o. período de 2008

Gabarito da Primeira Avaliação à Distância

1. (1,0) De um modo geral, se para um certo problema existem dois algoritmos, o primeiro de complexidade exponencial e o segundo de complexidade polinomial, este último seria o preferido. Justifique a razão desta preferência.

   Resposta: Sendo n o tamanho da entrada, temos que, para valores de n consideravelmente grandes, o algoritmo de complexidade polinomial executa mais rapidamente que o de complexidade exponencial.

2. (1,8) Para cada item abaixo, responda "certo", 'errado" ou "nada se pode concluir". Justifique.

   a. Se um limite inferior para um problema P é n^3, e se A é um algoritmo que resolve P, então a complexidade de A é sempre menor do que $O(n^3)$.

   Resposta: Falso. Sendo n^3 um limite inferior de P, sabemos que qualquer algoritmo ótimo para P tem complexidade de pior caso $\Omega(n^3)$. No entanto, podemos ter para P um algoritmo $\Theta(n^5)$, por exemplo.

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

   Resposta: Falso. Se tivermos, por exemplo, que n^3 também é um limite inferior para P, e existir um algoritmo A cuja complexidade de pior caso seja $O(n^3)$, então A é um algoritmo ótimo para P. Neste caso, a complexidade de melhor caso de A não é necessariamente $O(n^2)$.

   c. Sejam A_1, A_2 dois algoritmos que resolvem um certo problema P, tais que o algoritmo A_1 é ótimo. Então o tempo de execução de A_1 é menor do que o de A_2, para qualquer instância do problema P.

   Resposta: Falso. Como A_1 é ótimo, podemos afirmar que sua complexidade de pior caso é a menor possível. Logo, podemos afirmar que o tempo de execução de A_1 é menor ou igual que o de A_2 apenas no pior caso (e não para qualquer instância).

3. (1,5) Seja $f_1, f_2, \ldots, f_n$ uma sequência de elementos definida do seguinte modo: $f_1 = 0, f_2 = 1, f_3 = 1, f_j = f_{j-1} - f_{j-2} + f_{j-3}$ para $j > 3$. Escrever dois algoritmos para determinar o elemento f_n da sequência, o primeiro recursivo e o segundo não recursivo. Calcule a complexidade de cada um, em função de n.

   Resposta:

Algoritmo recursivo:

função $seq(j)$
    se $j = 1$ então
        retornar 0
    senão se $j = 2$ ou $j = 3$ então
        retornar 1
    senão
        retornar $(seq(j-1) - seq(j-2) + seq(j-3))$;

Chamada externa: $seq(n)$

Complexidade: É dada pela seguinte equação de recorrência:
$T(1) = T(2) = T(3) = 1$
$T(j) = T(j-1) + T(j-2) + T(j-3)$
Resolvendo esta recorrência, verificamos que a complexidade deste algoritmo é $O(3^n)$.

Algoritmo iterativo:

$f[1] := 0$;
$f[2] := 1$;
$f[3] := 1$;
para $j = 4 \ldots n$ faça
    $f[j] := f[j-1] - f[j-2] + f[j-3]$;

A complexidade do algoritmo acima é $O(n)$.

4. (1,5) Determinar a expressão da complexidade média de uma busca ORDENADA de 10 chaves, nas seguintes condições:
(i) As probabilidades de busca das chaves de ordem impar são iguais entre si.
(ii) As probabilidades de busca das chaves de ordem par são iguais entre si.
(iii) A probabilidade de busca de uma chave de ordem impar é o dobro da probabilidade de uma chave de ordem par.
(iv) 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:

$p(E_2) = p(E_4) = \cdots = p(E_{10}) = p$

$p(E_1) = p(E_3) = \cdots = p(E_9) = 2p$

Como a probabilidade de sucesso é de 100%, temos:
$p(E_1) + p(E_2) + \cdots + p(E_{10}) = 1$

Logo:

$$\sum_{i=1}^{10} p(E_i) = 5\ .\ p + 5\ .\ 2p = 1 \qquad p = \frac{1}{15}$$

O número de passos necessários para cada entrada é:

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

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:

$$\begin{aligned} C.M. &= \sum_{i=1}^{10} p(E_i)\, t(E_i) \\ &= \frac{2}{15}(1+3+5+7+9) + \frac{1}{15}(2+4+6+8+10) \;\approx\; 5,3 \end{aligned}$$

5. (CANCELADA) Escrever algoritmos de busca e inserção em LISTAS DUPLAMEMENTE ENCADEADAS NÃO ORDENADAS.

6. (1,5) Sejam L_1 e L_2 duas listas ordenadas, simplesmente encadeadas com nó-cabeça. Apresentar um algoritmo que construa uma lista ordenada L, nas seguintes condições:
(i) Se L_1 contém um elemento x que não pertence a L_2 então colocar x em L.
(ii) Se L_2 contém um elemento x que não pertence a L_1 então não colocar x em L.
(iii) Um elemento comum a L_1 e L_2 será colocado em L, somente se for precedido em L_1 por algum elemento que não pertença a L_2.
(iv) 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 $ptnovo$
$anterior := falso$ % indica se o nó anterior ao apontado por pont1 também está em L_2

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

7. (1,5) Escreva uma versão NÃO RECURSIVA do Algoritmo das Torres de Hanói que se encontra no livro-texto. Sugestão: utilize pilhas (pois uma versão não recursiva do Algoritmo das Torres de Hanói sem o uso de pilhas é um problema de dificuldade bem maior).

   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. (1,2) Seja $1, 2, \ldots, n$ uma seqüência de elementos que serão inseridos e posteriormente retirados de uma pilha P uma vez cada. A ordem de inclusão dos elementos na pilha é $1, 2, \ldots, n$, enquanto que a ordem de remoção depende das operações realizadas. Por exemplo, com $n = 3$, a seqüência de operações "incluir em P, incluir em P, retirar de P, incluir em P, retirar de P, retirar de P" produzirá a permutação $2, 3, 1$ a partir da entrada $1, 2, 3$. Representando por I, R, respectivamente, as operações de inserção e remoção da pilha, a permutação $2, 3, 1$ pode ser denotada por $IIRIRR$. De um modo geral, uma permutação é chamada *admissível* quando ela puder ser obtida mediante uma sucessão de inclusões e remoções em uma pilha a partir da permutação $1, 2, \ldots, n$. Assim, por exemplo, a permutação $2, 3, 1$ é admissível. Pede-se:

   (i) Determinar a permutação correspondente a $IIIRRIRR$, $n = 4$.
   Resposta: $3, 2, 4, 1$.

   (ii) Dê um exemplo de permutação não admissível.
   Resposta: $4, 1, 2, 3$ (para $n = 4$). Motivo: após remover o 4, o 1 se encontra no fundo da pilha e não pode ser o próximo a ser removido.