Estrutura de Dados - 1o. período de 2015

Gabarito da Primeira Avaliação à Distância

1. Para cada item abaixo, responda "certo" ou "errado", justificando (meio ponto cada):

   a. Se a complexidade de caso médio de um algoritmo for $\Theta(f)$, então o número de passos que o algoritmo efetua no melhor caso é $O(f)$.

   Resposta: Certo. Se a complexidade de caso médio de um algoritmo for $\Theta(f)$, então no melhor caso o número de passos não será superior à funçao f, caso contrário, a complexidade de caso médio também seria superior a f.

   b. Se a complexidade de melhor caso de um algoritmo for $\Theta(f)$, então o número de passos que o algoritmo efetua, qualquer que seja a entrada, é $O(f)$.

   Resposta: Errado. Se a complexidade de melhor caso de um algoritmo for $\Theta(f)$, então o número de passos que o algoritmo efetua, qualquer que seja a entrada, é $\Omega(f)$.

   c. Se a complexidade de pior caso de um algoritmo for $\Theta(f)$, então o número de passos que o algoritmo efetua, qualquer que seja a entrada, é $O(f)$.

   Resposta: Certo. Se a complexidade de pior caso de um algoritmo for $\Theta(f)$, então para qualquer entrada o número de passos não será superior a funçao f que corresponde ao pior caso.

   d. Se um algoritmo A para um problema P não é comprovadamente ótimo, então certamente existe um algoritmo A' para P cuja complexidade de pior caso é inferior à complexidade de pior caso de A.

   Resposta: Errado. Não existirá necessariamente o algoritmo A'. Se for comprovado que A é ótimo, certamente não existirá o A', pois A terá complexidade igual ao limite inferior para P. Se for comprovado que A não é ótimo, pode existir ou não o algoritmo A'.

2. (2,0) Considere uma sequência de elementos $f_1, f_2, \ldots, f_n$, definida do seguinte modo: $f_1 = 0, f_2 = 1, f_3 = 2, f_j = f_{j-1} + f_{j-2} - f_{j-3}$ para $j > 3$. Compare as complexidades dos algoritmos recursivo e não recursivo para determinar o elemento f_n da sequência.

   Resposta:

   Algoritmos iterativo:

   $f[1] := 0;$

   $f[2] := 1;$

   $f[3] := 2;$

   para $j := 4, ..., n$ faça

   $f[j] := f[j-1] + f[j-2] - f[j-3]$

   Complexidade: $O(n)$.

Algoritmo recursivo:

procedimento *rec(j)*

se $j < 4$ então

retorne $j - 1$

senão

retorne $rec(j-1) + rec(j-2) - rec(j-3)$

Chamada externa: *rec(n)*.

A complexidade deste algoritmo pode ser obtida pela seguinte equação de recorrência:

T(1)=0

T(2)=1

T(3)=2

$T(j) = T(j-1) + T(j-2) - T(j-3)$

Resolvendo esta equação concluímos que a complexidade é $O(3^n)$.

Comparação das complexidades: o algoritmo iterativo tem complexidade linear, enquanto que o recursivo tem complexidade exponencial, portanto o primeiro é mais eficiente.

3. (2,0) Escreva um algoritmo que duplica os nós de uma lista simplesmente encadeada com nó cabeça. Exemplo: se a lista contém os elementos $1, 3, 5, 9, 2$, nesta ordem, então a lista resultante contém os elementos $1, 1, 3, 3, 5, 5, 9, 9, 2, 2$.

Resposta:

Algoritmo:

$pont1 := ptlista \uparrow .prox$ % ponteiro para a lista original

$ptaux := ptnovo$ % ponteiro para a lista resultante

enquanto $pont1 \neq \lambda$ faça

inserir(pont1, ptaux)

inserir(pont1, ptaux)

$pont1 := pont1 \uparrow .prox$

procedimento *inserir(pont1, ptaux)*

ocupar(pt)

$pt \uparrow .info := pont1 \uparrow .info$

$pt \uparrow .prox := \lambda$

$ptaux \uparrow .prox := pt$

$ptaux := pt$

4. (2,0) Sejam duas pilhas P_1 e P_2. Apresentar um algoritmo que intercale as duas pilhas de forma que a pilha resultante P contenha elementos de P_1 e P_2 alternadamente, da seguinte forma: o topo de P é o topo de P_1, o segundo elemento de P é o topo de P_2, o terceiro elemento de P é o segundo elemento de P_1, e assim por diante. Obs: Os elementos excedentes da pilha mais longa entre P_1 e P_2 devem ser colocados diretamente no fundo da pilha P, obedecendo a ordem em que aparecem originalmente. Exemplo: se P_1, a partir do topo, tem configuração $(3, 5, 7, 8)$ e, da mesma forma, P_2 tem configuração $(4, 6)$, então a configuração da pilha resultante P será $(3, 4, 5, 6, 7, 8)$.

   Resposta:

   Seja P_3 a pilha resultante. Sejam $topo_1$, $topo_2$ e $topo_3$ os topos de P_1, P_2 e P_3, respectivamente.

   Algoritmo:

   $topo_3 := 0$

   enquanto $topo_1 \neq 0$ *e* $topo_2 \neq 0$ *faça*

   inserir$(P_1, topo_1)$

   inserir$(P_2, topo_2)$

   enquanto $topo_1 \neq 0$ *faça*

   inserir$(P_1, topo_1)$

   enquanto $topo_2 \neq 0$ *faça*

   inserir$(P_2, topo_2)$

   procedimento *inserir(P, topo)*

   $topo_3 := topo_3 + 1$

   $P_3[topo_3] := P[topo]$

   $topo := topo - 1$

5. (2,0) Seja T uma árvore binária com altura $h > 0$. Calcule o número de nós de T em função de h nos seguintes casos:

   (a) T é uma árvore zigue-zague;

   Resposta: A árvore zigue-zague T tem apenas um nó em cada nível. Como T tem h níveis, o número de nós de T é h.

   (b) T é uma árvore binária cheia;

   Resposta: Seja T' uma árvore binária não-vazia com k níveis. O primeiro nível de T' tem apenas 1 nó. O segundo nível tem 2 nós. O terceiro nível tem 4 nós. Assim sucessivamente até o k-ésimo nível que tem no máximo 2^{k-1} nós.

   Uma árvore binária cheia T tem exatamente 2^{k-1} nós em cada nível, $1 \leq k \leq h$. Portanto, o número de nós em T é a soma do número de nós em todos os níveis: $2^0 + 2^1 + 2^2 + ... + 2^{h-1} = 2^h - 1$.

(c) T é uma árvore estritamente binária com exatamente 2 nós em cada nível (exceto, é claro, no nível da raiz);

Resposta: Como cada nível de T tem exatamente 2 nós, exceto o primeiro nível, podemos concluir que T tem $2(h-1)+1$ nós.

(d) T é uma árvore binária completa com o menor número possível de nós.

Resposta: Uma árvore binária completa é cheia até o penúltimo nível, então até este nível há $2^{h-1}-1$ nós. Se T é uma árvore binária completa com o menor número possível de nós, então há apenas um nó no último nível. Portanto, T tem 2^{h-1} nós.