Estrutura de Dados - 1o. período de 2006

Gabarito da Primeira Avaliação à Distância

1. Escrever as seguintes funções em notação O:
$n^3 + 10$; $n^2 - 2\log n$; $8n^n - 5.4^n$; $(n-1)^n + n^{n-1}$; 51.

   Resposta: $n^3 + 10 = O(n^3)$; $n^2 - 2\log n = O(n^2)$; $8n^n - 5.4^n = O(n^n)$; $(n-1)^n + n^{n-1} = O(n^n)$; $51 = O(1)$.

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

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

   Resposta: errado, pois o pior caso pode corresponder a um número de passos dado por uma função g que não seja $\Theta(f)$.

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

   Resposta: certo, pois qualquer que seja o número de passos, estará limitado superiormente por f, que é a função do pior caso.

   c. A complexidade de pior caso de um algoritmo para um certo problema é necessariamente maior do que qualquer limite inferior para o problema.

   Resposta: Se por "maior"entende-se "estritamente maior", isto é, que a complexidade de pior caso é $\Omega(f)$ mas não $\Theta(f)$, onde f é a função de limite inferior, então a afirmativa está **errada**, pois a complexidade de pior caso pode "empatar"com o limite inferior.

   Mas se por "maior" entende-se apenas que a a complexidade de pior caso é $\Omega(f)$, então a afirmativa está **certa**, porque a definição de limite inferior diz justamente que a complexidade de pior caso de **qualquer** algoritmo para o problema é $\Omega(f)$.

3. 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 $j > 3$. Elabore um algoritmo não recursivo para determinar o elemento f_n da sequência. Calcule sua complexidade em função de n.

   Resposta:

   $f[1] := 0;$
   $f[2] := 1;$
   $f[3] := 1;$ 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. Determinar a expressão da complexidade média de uma busca não ordenada de n chaves, em que a probabilidade de busca da chave i é o dobro da probabilidade de busca da chave $i-1$, para $i = 2, ...n$. Supor, ainda, que a probabilidade de a chave procurada se encontrar na lista é igual a q.

   Resposta: Seja p a probabilidade de busca da chave 1. Temos então que $p + 2p + 4p + \cdots + 2^{n-1}p = q$. Colocando p em evidência e utilizando a fórmula para soma dos termos de uma P.G., temos $p = q/(2^n - 1)$. Logo, a expressão da complexidade média neste caso é dada pela expressão:

$$C.M. = (1.1 + 2.2 + 3.4 + 4.8 + 5.16 + \cdots + n.2^{n-1})\frac{q}{2^n - 1} + n(1-q)$$

onde a parcela final $n(1-q)$ refere-se ao caso em que a informação procurada não se encontra na lista.

5. Comparar algoritmos de busca, inserção e remoção em uma lista não ordenada nas alocações sequencial e encadeada.

   Resposta:

   Busca - $O(n)$ para alocação sequencial, $O(n)$ para encadeada.
   Inserção - $O(1)$ para alocação sequencial (inserir no final), $O(1)$ para alocação encadeada.
   Remoção - $O(1)$ para alocação sequencial (tomar o último elemento e colocá-lo no "buraco"deixado pelo elemento removido), $O(1)$ para alocação encadeada.

   Obs: na prática, a inserção e a remoção exigem uma busca prévia. Portanto, na prática, todos os algoritmos acima se tornam $O(n)$.

6. 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 comuns a L_1 e L_2. (Supor que tanto L_1 como L_2 não contêm elementos repetidos.)

   Resposta:

   $pont1 := ptlista1 \uparrow .prox$ % ponteiro para a lista 1
   $pont2 := ptlista2 \uparrow .prox$ % ponteiro para a lista 2
   $ptaux := ptnovo$ % a lista resultante iniciará em *ptnovo*

```
enquanto pont1 ≠ λ e pont2 ≠ λ faça
      se pont1 ↑ .info < pont2 ↑ .info
            então pont1 := pont1 ↑ .prox
            senão
                  se pont2 ↑ .info < pont1 ↑ .info
                        então pont2 := pont2 ↑ .prox
                        senão    % são elementos iguais !
                              ocupar(pt);
                              pt ↑ .info := pont1 ↑ .info
                              pt ↑ .prox := λ
                              ptaux ↑ .prox := pt
                              ptaux := pt   %ptaux aponta para o último nó
                              pont1 := pont1 ↑ .prox
                              pont2 := pont2 ↑ .prox
fim-enquanto
```

7. Descreva algoritmos de inserção e remoção em pilhas e filas, implementadas utilizando alocação encadeada.

   Resposta:

   Inserção na pilha:
   ocupar(pt)
   $pt \uparrow .info := novo - valor$
   $pt \uparrow .prox := topo$
   $topo := pt$

   Remoção da pilha:
   se $topo \neq \lambda$ então
       $pt := topo$
       $topo := topo \uparrow .prox$
       $valor - recuperado := pt \uparrow .info$
       $desocupar(pt)$
   senão $underflow$

   Inserção na fila:
   $ocupar(pt)$ $pt \uparrow .info := novo - valor$
   $pt \uparrow .prox := \lambda$
   se $fim \neq \lambda$ então
       $fim \uparrow .prox := pt$
   senão $inicio := pt$
   $fim := pt$

   Remoção da fila:
   se $inicio \neq \lambda$ então
       $pt := inicio$
       $inicio := inicio \uparrow .prox$
       se $inicio = \lambda$ então $fim := \lambda$
       $valor - recuperado := pt \uparrow .info$
       $desocupar(pt)$
   senão $underflow$

8. 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 a 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.