Universidade Federal de Pernambuco
Centro de Informática
Bacharelado em Ciência da Computação

IF672 Algoritmos e Estruturas de Dados — 2013.1
Profs. Katia Guimarães, Paulo Fonseca

PRIMEIRA LISTA DE EXERCÍCIOS DE REVISÃO

1. Um algoritmo A resolve um problema em tempo $\Omega(n^2)$ no pior caso, enquanto um outro algoritmo B resolve o mesmo problema em $O(n^2)$ no melhor caso. Qual dos dois algoritmos é o mais rápido. Explique sucintamente.

2. Os algoritmos A e B gastam exatamente $T_A = c_A n \lg n$ e $T_B = c_B n^2$ microsegundos, respectivamente, para um problema de tamanho n. Pergunta-se:

a) Qual dos dois algoritmos é o mais eficiente do ponto de vista assintótico?
b) Sabendo-se que os algoritmos A gasta 10μs para processar $n = 1024$ itens e B gasta apenas 1μs para processar a mesma entrada, qual o melhor dos dois algoritmos para processar uma entrada de tamanho $n = 220$?

3. Os algoritmos A e B requerem *exatamente* $T_A(n) = 10n \lg n + 20$ e $T_B(n) = 2n^2 + 3n$ operações elementares, respectivamente, para um entrada de tamaho n.

a) Qual o algoritmo mais eficiente do ponto de vista assintótico? Escreva as ordens de complexidade de A e B em notação assintótica.
b) Existe alguma entrada para o qual o algoritmo menos eficiente do ponto de vista assintótico é mais rápido? Se sim, dê um exemplo. Caso contrário, justifique sucintamente.

4. Escreva em pseudo-código um algoritmo que recebe como entrada um vetor V de n inteiros positivos e modifica-o de forma a eliminar os elementos repetidos. O algoritmo deve manter a mesma ordem dos valores originais porém deslocando elementos à esquerda de forma a ocupar o lugar de elementos repetidos eventualmente eliminados. As posições finais eventualmente livres devem ser preenchidas com zeros.

Exemplo: $V = (2, 4, 3, 9, 4, 2, 5, 8, 5) \rightarrow V' = (2, 4, 3, 9, 5, 8, 0, 0, 0)$.

Importante: O algoritmo só poderá usar, além do vetor de entrada, uma quantidade fixa de memória adicional, i.e., não é permitido criar um vetor ou lista auxiliar.

5.

a) O cálculo de algumas funções matemáticas pode ser muito custoso. Assim, se um programa calcula muitas vezes, por exemplo, a função fatorial, pode ser útil manter uma tabela com valores pré-computados. Escreva em pseudo-código uma função *pcfatorial* que recebe um número $n \in \mathbb{N}$ e devolve como saída um vetor $F = (0!, 1!, 2!, \ldots, n!)$.
b) Um exemplo de tal programa seria um programa que calcula o *Triângulo de Pascal*:

$$\begin{matrix}
\binom{0}{0} & & & & & & \\
\binom{1}{0} & \binom{1}{1} & & & & & \\
\binom{2}{0} & \binom{2}{1} & \binom{2}{2} & & & & \\
\vdots & \vdots & \vdots & \ddots & & & \\
\binom{n}{0} & \binom{n}{1} & \binom{n}{2} & \cdots & \binom{n}{n} & & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots &
\end{matrix},$$

onde

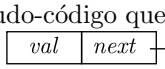
$$\binom{a}{b} = \frac{a!}{b!(a-b)!}.$$

Escreva uma função *tpascal* em pseudo-código que recebe como entrada um valor $n \in \mathbb{N}$ e devolve como saída uma matriz $P_{(n+1)\times(n+1)}$ contendo as $n+1$ primeiras linhas do Triângulo de Pascal, i.e. $P[i, j] = \binom{i-1}{j-1}$, se $i \geq j$. **Importante:** Esta função deve usar a função *pcfatorial* do item (a) e a identidade $\binom{a}{b} = \binom{a}{a-b}$.

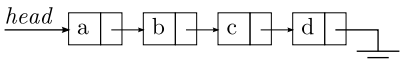
c) Suponha que tenhamos um jogo no qual fazemos $n \in \mathbb{N}$ tentativas independentes, cada uma com a mesma probabilidade de sucesso $p \in [0,1]$. A probabilidade de termos exatamente s sucessos nas n tentativas é dada pela função de probabilidade da chamada *distribuição binomial* como

$$f(s;n,p) = \binom{n}{s} p^s (1-p)^{n-s}.$$

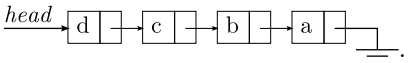
Escreva uma função *binom* que recebe como entrada três vetores $N = (n_1, \ldots, n_r) \in \mathbb{N}^r$, $P = (p_1, \ldots, p_r) \in [0,1]^r$ e $S = (s_1, \ldots, s_r) \in \mathbb{N}^r$ e devolve como resposta o vetor $B = (f(s_1;n_1,p_1), \ldots, f(s_r;n_r,p_r))$. **Importante:** Esta função deve usar a função *tpascal* do item (b) e pode assumir que $s_i \leq n_i$, para $i = 1, \ldots, r$.

6. Escreva um algoritmo em pseudo-código que recebe como entrada uma lista simplesmente encadeada L que utiliza a estrutura de nó habitual 

e que a modifica de forma a invertê-la. Por exmplo, a lista



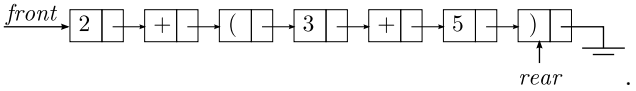
deve ser convertida em



Importante: O algoritmo só poderá usar, além da lista de entrada, uma quantidade fixa de memória adicional, i.e., não é permitido copiar a lista para uma lista ou vetor auxiliar.

7. Expressões aritméticas podem ser escritas com o auxílio de parênteses de forma a eliminar ambiguidades e especificar uma ordem precisa de execução das operações. Por exemplo, a expressão $10+3\times5-2$ pode ser avaliada de várias maneiras se não assurmirmos uma ordem de precedência. Porém, ao escrevermos $(10 + ((3 \times 5) - 2))$ a expressão deve ser inequivocamente avaliada como 23. Entretanto, para que esteja sintaticamente correta, a expressão deve ter *parênteses balanceados*, ou seja, cada "(" dever ter seu correspondente ")", e cada ")" deve sempre fechar um "(" anterior. Escreva em pseudocódigo um algoritmo que recebe como entrada uma expressão aritmética e devolve como resposta o valor verdadeiro (**V**), se a expressão estiver corretamente balanceada, ou falso (**F**), caso contrário.

Importante: A expressão de entrada vai estar codificada como uma lista de strings na qual cada elemento contém um número, um operador aritmético, ou um parênteses. Por exemplo, a expressão $2 + (3 + 5)$ será codificada como



Além dessa lista de entrada, o algoritmo só poderá usar mais uma pilha e mais uma quantidade fixa de variáveis que armazenam valores retirados da fila ou da pilha. Pode-se assumir como dadas apenas as funções *enfileirar*/*desenfileirar* e *empilhar*/*desempilhar*, sendo que não é permitido desenfileirar/desempilhar elementos de uma fila/pilha vazia.

8. A *altura* de uma árvore binária pode ser definida como

$$h(T) = \begin{cases} 0, & \text{se } T = \varnothing \\ 1 + \max\{h(T_L), h(T_R)\}, & \text{se } T = \begin{array}{ccc} & r & \\ \swarrow & & \searrow \\ T_L & & T_R \end{array} \end{cases},$$

onde r representa um nó e T_L, T_R as sub-árvores binárias à esquerda e à direita de r.

A altura de um nó de uma árvore binária é então definida como a altura da sub-árvore nele enraizada.

Escreva um procedimento recursivo que recebe um ponteiro para a raiz de uma árvore binária e imprime a altura dos nós desta árvore *em pós-ordem*.

9. Seja n um nó de uma árvore binária e $D(n)$ o número de *descendentes* de n (i.e. filhos, filhos dos filhos, filhos dos filhos dos filhos, etc.)

a) Escreva uma definição *recursiva* de $D(n)$.
b) Escreva um algoritmo em pseudo-código para imprimir os valores de $D(n)$ *em pós-ordem* para cada n de uma árvore binária fornecida como entrada.

10. Escreva um algoritmo que recebe um apontador *root* para a raiz de uma árvore binária e visita os valores dos seus nós*por ordem de profundidade*. Nesta ordem, primeiro é visitada a raiz (nível 0), depois os filhos da raiz (nível 1), depois os filhos dos filhos da raiz (nível 2) etc., sempre da esquerda para a direita. *Dica:* um percurso em pré-ordem faz uso da pilha de chamadas recursivas. Considere usar explicitamente outra ED linear.

11. Escreva em pseudo-código um algoritmo recursivo *CheckBST* que recebe como entrada um apontador *root* para a raiz de uma árvore binária T e devolve uma tripla (bst, min, max), onde *bst* é um booleano que indica se T é uma árvore de busca binária e min, max indicam, respectivamente, os valores do menor e maior elemento de T.

12. Árvores de busca binária são úteis para busca de valores dentro de intervalos. Escreva em pseudo-código um procedimento recursivo que recebe um apontador *root* para a raiz de uma árvore de busca binária, dois inteiros $\ell, r, \ell \leq r$, e imprime, em ordem crescente, todos os elementos da árvore cujo valor situa-se no intervalo $[\ell, r]$.

13. A remoção de um nó de uma árvore AVL faz-se como numa árvore de busca binária, seguida por rotações eventualmente necessárias para a restauração do seu balanceamento. Escreva em pseudo-código o procedimento de remoção de um nó de uma árvore AVL.

14. Considere a árvore AVL T resultante da inserção dos valores $(5, 3, 10, 2, 4, 7, 11, 1, 6, 9, 12, 8)$ nesta ordem. Pede-se:

a) Desenhar T
b) Desenhar T logo após a remoção do valor 10 (antes das eventuais rotações)
c) Desenhar T após cada rotação necessária

15.

a) Represente a árvore AVL cujos nós enumerados em pré-ordem são

$$3, 2, 1, 5, 4, 7, 6, 8.$$

b) Represente a remoção do elemento '1' da árvore do item (a), desenhando a árvore imediatamente após a remoção (antes das rotações) e também após cada rotação que se fizer necessária.