EXERCÍCIOS: 0

RAUL H.C. LOPES

1. Introdução

A lista a seguir apresenta uma série de exercícios que você pode considerar como "treino coletivo" para os grandes jogos que se aproximam. Você encontrará aqui exercícios sobre os fundamentos de análise e correção de algoritmos vistos até agora. Importante: não existe (como não poderia haver) a pretensão de cobrir tudo o que foi apresentado.

1.1. **Notação.** Assumindo que P é um predicado

$$[P] = \begin{cases} 1, \text{ if } P \\ 0, \text{ if } \neg P \end{cases}$$

2. Crescimento de funções

(1) O que você entende por
   - $f(n) \in \Theta(g(n))$
   - $f(n) \in O(g(n))$
   - $f(n) \in \Omega(g(n))$

(2) Por que escrever

$$f(n) \in O(g(n))$$

é mais adequado do que

$$f(n) = O(f(n))$$

(3) Mostre as seguintes relações são transitivas e reflexivas:
   - $f(n) \in \Theta(g(n))$
   - $f(n) \in O(g(n))$
   - $f(n) \in \Omega(g(n))$

(4) Prove a simetria de $f(n) \in \Theta(g(n))$

(5) Dada soma

$$s(n) = \sum_{0 \leq i \leq d a_i n^i}$$

prove:

(a)

$$s(n) \in O(n^k), \text{ when } k \geq d$$

(b)
$$s(n) \in \Omega(n^k), \text{ when } k \leq d$$
(c)
$$s(n) \in \Theta(n^k), \text{ when } k = d$$

(6) Prove que
$$\log n! \in O(n \lg n).$$

3. Somas e recorrências

(1) Para cada uma das seguintes recorrências, faça:
- Escreva, se possível, a recorrência como somatória.
- Resolva a recorrência, encontrando uma fórmula fechada. Não use indução neste passo, nem métodos trivialoides como: *guess and prove*, olhar em uma referência, usar o *Master method*, etc. O objetivo é exercitar o desenvolvimento da recorrência. Lembre-se: aplicar uma fórmula pronta até um colegial consegue. Melhor que isso é desenvolver a fórmula.
- estabeleça claramente a restrições que devem ser satisfeitas para sua fórmual ser válida;
- use indução para provar a correção de sua solução.

(a)
$$\begin{aligned} U_0 &= a \\ U_n &= b + U_{n-1} \end{aligned}$$
(b)
$$\begin{aligned} U_0 &= a \\ U_n &= b^n + U_{n-1} \end{aligned}$$
(c)
$$\begin{aligned} U_0 &= a \\ U_n &= cb^n + U_{n-1} \end{aligned}$$
(d)
$$\begin{aligned} U_0 &= 0 \\ U_n &= 2U_{n-1} + 1 \end{aligned}$$
(e)
$$\begin{aligned} U_0 &= 0 \\ U_n &= 3U_{n-1} + 2 \end{aligned}$$

(f)

$$U_0 = 0$$
$$U_n = U_{n-1} + \log n$$

(2) Resolva as seguintes somas:
(a)

$$\sum_{k=4}^{0} a^k$$

(b)

$$\sum_{i \leq k < n} f(k), \text{ when } i = n$$

(c)

$$\sum_{0 \leq i < n} i[P(i)], \text{ onde } P(i) \text{ significa } i \text{ é par}$$

(d)

$$\sum_{1 \leq k \leq n} k^2$$

(e)

$$\sum_{k=1}^{n} k2^k$$

(f)

4. Complexidade em árvores

4.1. Árvores binárias. Dada uma árvore binária t:

(1) Defina o grau de qualquer nó de t.
(2) Estabeleça o máximo de nós no nível l.
(3) Defina recorrência (e resolva-a) para o máximo de nós em uma árvore com k níveis.
(4) Estabeleça os números máximo e mínimo de níveis em uma árvore com n nós.
(5) Mostre que a busca pela versão ordenada de uma seqüência (ordenação por comparação) pode ser vista como uma busca em uma árvore binária de decisão.
(6) Quantas folhas teria a árvore de decisão do exercício anterior?
(7) Qual seria o número mínimo de níveis dessa árvores?
(8) Use essa árvore de decisão para achar o número mínimo de comparações que qualquer algoritmo de ordenação por comparação de chaves deve realizar.

4.2. **Árvores B.** Dada uma árvore B de ordem m, encontre (e prove):

(1) o número máximo de nós;
(2) o número mínimo de nós;
(3) o número máximo de níveis;
(4) o número mínimio de níveis.

5. Correção de algoritmos

Os exercícios desta seção são pre-requisitos para quem pretende resolver qualquer trabalho em **ACL2**: se você não consegue resolvê-los, implemente seus trabalhos em C/C++/Ada.

(1) Defina as invariantes das seguintes estruturas de dados:
   (a) Stack.
   (b) Fila (FIFO).
   (c) Árvore binária.
   (d) Árvore binária de pesquisa.
   (e) Heap.
   (f) Árvore B.
(2) Defina algoritmos recursivos para operar sobre naturais que usem apenas operações de incremento/decremento de 1 e comparação com 0.
   - Soma.
   - Subtração.
   - Multiplicação.
   - Divisão.
(3) Defina invariantes que relacionam soma com subtração e divisão com multiplicação, prove-as usando ACL2.
(4) Defina propriedades invariantes de uma concatenação de stacks.