DCA0214.0 ESTRUTURAS DE DADOS - TEORIA: Lista de Exercícios sobre Análise de Complexidade de Algoritmos

Professor: Islame Felipe da Costa Fernandes

08 de Agosto de 2019

1. Uma forma de se obter a raiz quadrada de um número qualquer x seria através de busca binária. Assuma que a raiz quadrada de x está entre 0 e x (Se o número for negativo, retorne 0). Para sabermos se um palpite y é a raiz quadrada de x, basta testar se $y * y$ é próximo o suficiente de x ou, em outras palavras, se o módulo da diferença entre eles está dentro de uma tolerância definida. Caso contrário, podemos restringir a busca entre 0 e y ou entre y e x. Escreva um algoritmo (pseudo-código) para calcular a raiz quadrada de um inteiro x, considerando 10^{-6} como tolerância para o cálculo do resultado.

2. Prove que o tempo de execução de um algoritmo é $\Theta(g(n))$ se, e somente se, seu pior caso é $O(g(n))$ e seu melhor caso é $\Omega(g(n))$. Utilize a definição formal de notação assintótica.

3. Prove (usando a definição formal de notação assintótica) ou refute (apresentando contra-exemplo) as proposições abaixo.

   (a) $f(n) = O(g(n))$ implica $g(n) = O(f(n))$

   (b) $f(n) + g(n) = \Theta(min(f(n), g(n)))$

   (c) $f(n) = O(g(n))$ implica $2^{f(n)} = O(2^{g(n)})$

   (d) $f(n) = O(g(n))$ implica $g(n) = \Omega(f(n))$

4. Seja

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

   um polinômio de grau d, com $a_d > 0$. Seja ainda k uma constante. Utilizando a definição formal de notação assintótica, prove as seguintes propriedades:

   (a) Se $k \geq d$, então $p(n) = O(n^k)$

(b) Se $k \leq d$, então $p(n) = \Omega(n^k)$

(c) Se $k = d$, então $p(n) = \Theta(n^k)$.

5. Seja $S = \{s_1, ..., s_n\}$ uma sequência com n valores numéricos. Faça o que se pede:

   (a) Escreva um algoritmo iterativo que encontre o segundo maior elemento de S. Seu algoritmo deve ser baseado em comparações.

   (b) Quantas comparações seu algoritmo efetua em função de n? Qual a complexidade no melhor e no pior caso?

6. Seja P um problema de tamanho n. Para cada algoritmo abaixo, responda:

   (a) Quantos passos o algoritmo efetua no prior caso? Deduza uma fórmula para quantidade de passos em função de n.

   (b) Dada a quantidade de passos em função de n, qual a complexidade assintótica no pior caso?

Algoritmo 1:

Procedimento `algoI`(n)
 para $i \leftarrow 1, ..., n$ **faça**
  para $j \leftarrow 1, ..., n$ **faça**
   operações constantes
  fim
 fim

Algoritmo 2:

Procedimento `algoII`(n)
 para $i \leftarrow 1, ..., n$ **faça**
  para $j \leftarrow 1, ..., 2^i$ **faça**
   operações constantes
  fim
 fim

Algoritmo 3:

Procedimento `algoIII`(n)
 para $i \leftarrow 1, ..., n$ **faça**
  $j \leftarrow 1$
  enquanto $j \leq i$ **faça**
   operações com complexidade $O(2^j)$
   $j \leftarrow j + 1$
  fim
 fim

Algoritmo 4:

```
Procedimento algoIV(n)
    se n > 1 então
        operações contantes
        algoIV (n-1)
    fim
```

Algoritmo 5:

```
Procedimento algoV(n)
    se n == 0 então
        operações contantes
    senão
        c chamadas recursivas algoV (n-1), onde c é constante
    fim
```

7. A sequência de Fibonacci pode ser definida recursivamente da seguinte forma: o primeiro termo é 0 e o segundo termo é 1. O n-ésimo termo é definido recursivamente com base na soma dos dois termos anteriores. Formalmente:

$$fibo(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fibo(n-1) + fibo(n-2) & \text{case contrário} \end{cases} \tag{1}$$

   (a) Com base na definição acima, formule um algoritmo recursivo (pseudo-código) para encontrar o n-ésimo, $fibo(n)$, da sequência de Fibonacci. Qual a complexidade do seu algoritmo recursivo?

   (b) Escreva um algoritmo iterativo (pseudo-código), **utilizando apenas três variáveis auxiliares**, para encontrar o n-ésimo, $fibo(n)$, da sequência de Fibonacci. Qual a complexidade do seu algoritmo iterativo?