

Folha de Exercícios IV

Programação e Algoritmos Complementos de Informática Programação II

Universidade da Beira Interior
Departamento de Informática

6 Análise de Complexidade e Recursividade

Exercício 46:

Resolva os exercícios 9, 10 e 14 da folha de exercícios I e determine a complexidade (para o pior caso) de cada programa.

Exercício 47: (*Horner*)

Podemos representar um polinómio P de grau n por um vector p de reais em que uma célula de índice i representa o coeficiente associado à potência de grau i . Escreva um programa que peça o valor de n (com a restrição que $n \geq 0$), inicialize p e que dado um x , calcule $P(x)$ usando o método de Horner, i.e.

$$P_n(x) = (\cdots((p_n x + p_{n-1})x + p_{n-2})x + \cdots + p_1)x + p_0$$

de forma recursiva e de forma iterativa.

Determine a complexidade da solução iterativa para o pior dos casos.

Exercício 48: (*Fibonacci*)

Escreva um programa que permita calcular para um dado $n \in \mathbb{N}$ o valor de $Fib(n)$, **com e sem recursividade**, tendo em conta que:

$$\begin{cases} Fib(0) = 1 \\ Fib(1) = 1 \\ Fib(n) = Fib(n-2) + Fib(n-1), \text{ para } n > 1 \end{cases}$$

Determine a complexidade, no pior dos casos, da solução iterativa.

Exercício 49: (*Euclides*)

Escreva duas funções C que, usando o algoritmo de Euclides, permitam calcular o máximo divisor comum de dois números **de forma recursiva e de forma iterativa**.

Relembra-se que o método é o seguinte:

$$\begin{cases} MDC(x, y) = MDC(y, modulo(x, y)) \\ MDC(x, 0) = x \end{cases}$$

Determine a complexidade da solução iterativa para o pior caso.

Exercício 50: (*Leitura recursiva de um vector*)

Modifique a implementação do método de Horner de tal forma que a leitura do vector de coeficientes do polinómio seja implementada sob a forma duma função recursiva.

Exercício 51: (*Teste de “primalidade”*)

1. Escreva uma função recursiva e uma função iterativa para testar se um dado inteiro é primo ou não. Relembra-se que um inteiro $n > 1$ é primo se, e só se, é apenas divisível por ele mesmo e por 1.
2. Qual é a complexidade no pior caso para a função iterativa?
3. Consegue reduzir a complexidade?

Exercício 52: (*Iteração, recursividade, recursividade terminal e uso de acumuladores*)

1. Escreva as versões iterativas das funções *factorial* e *fibonacci*.
2. Simule a execução de *factorial*(6) e de *fibonacci*(6) com ambas as versões. Comente os resultados em termos de ocupação de memória.
3. Modifique as versões recursivas de forma a otimizar a respectiva ocupação de memória.

Sugestão:

- No caso do cálculo do factorial, escreva a função
`int factorial_acc (int acc, int n);` onde `acc` é um parâmetro que acumula os resultados dos cálculos temporários.
- no caso do cálculo da série de Fibonacci, escreva a função
`int fib_acc (int acc1, int acc2, int n);` onde `acc1` e `acc2` são parâmetros que acumulam os resultados dos cálculos temporários.

Nota: Diz-se de tais funções recursivas que são *terminais* e que os parâmetros `acc`, `acc1` e `acc2` são *acumuladores*.

Exercício 53: (*Avaliação da boa formação de funções recursivas*)

Diga se as seguintes definições recursivas terminam com valores iniciais de $n > 0$. Justifique e implemente. Considere para este efeito que n é um inteiro ($n \in \mathbb{Z}$) e que as operações sobre n devolvem valores inteiros.

$$a(n) = \begin{cases} 1 & \text{se } n \leq 0 \\ a(n+1) + a(n-1) & \text{outros valores} \end{cases}$$

$$b(n) = \begin{cases} 1 & n \leq 1 \\ 2 \times b(\frac{n}{2}) & \text{se } n \text{ par} \\ 1 + b(n+1) & \text{outros valores} \end{cases}$$

$$c(n) = \begin{cases} 1 & \text{se } n = 0 \\ \frac{c(n+1)}{n+1} & \text{outros valores} \end{cases}$$

$$d(n) = \begin{cases} 1 & \text{se } n = 0 \\ d(3) & \text{se } n = 1 \\ d(n-2) & \text{outros valores} \end{cases}$$

$$e(n) = \begin{cases} 5 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ e(n-3) + e(n-1) & \text{outros valores} \end{cases}$$

$$f(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ f(\lfloor \frac{n}{n+1} \rfloor) & \text{outros valores} \end{cases}$$

$$McCarthy(n) = \begin{cases} n - 10 & \text{se } n > 100 \\ McCarthy(McCarthy(n + 11)) & \text{outros valores} \end{cases}$$

$$Morris(n, p) = \begin{cases} 1 & \text{se } n \leq 0 \\ Morris((n-1), Morris(n, p)) & \text{outros valores} \end{cases}$$

$$Collatz(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ Collatz(\frac{n}{2}) & \text{se } n \text{ par} \\ Collatz(3 \times n + 1) & \text{se } n \text{ ímpar e } n \geq 1 \end{cases}$$

Exercício 54: (*Funções mutuamente recursivas*)

Seja $par(n)$ a função que devolve 1 se n for par e 0 no caso contrário. E seja $ímpar(n)$ a função que devolve 1 se n for ímpar e 0 no caso contrário.

1. Defina as funções par e $ímpar$ de forma mutuamente recursiva.
2. Apresente uma definição sem recursividade mútua.

Exercício 55: (*Binómio de Pascal*)

Para $(n, p) \in \mathbb{N}^2$ tal que $(0 \leq p \leq n)$, a função \mathcal{C}_p^n pode ser calculada usando a definição seguinte:

$$\begin{cases} \mathcal{C}_0^n = 1 \\ \mathcal{C}_n^n = 1 \\ \mathcal{C}_p^n = \mathcal{C}_p^{n-1} + \mathcal{C}_{p-1}^{n-1} & \text{se } (0 < p < n) \end{cases}$$

Escreva uma função correspondente em C.