Respostas à Lista 1 da disciplina de Introdução à Análise de Algoritmos

> Feita por Guilherme de Abreu Barreto[1]

Exercício 1

```
m ← 0                              // c_2
for i ← 1 até n                    // c_2 + c_5 + (n - 1) c_1
    do for j ← 1 até n             // c_2 + c_5 + (n - 1)² c_1
        do for k ← 1 até n         // c_2 + c_5 + (n - 1)³ c_1
            if a_i + b_j + c_k = 0 // n³ (2 c_3 + c_5)
                then m ← m + 1     // 0 (c_2 + c_3) até n³ (c_2 + c_3)
return m                           // c_4
```

> **Obs:** Admite-se

$$c_1(\text{iteração}) = c_3(\text{soma}) + c_2(\text{atribuição}) + c_5(\text{comparação} - \text{com o valor } n)$$

Assim o sendo, temos que este algoritmo, na ausência de valores capazes de satisfazer a condição de acréscimo da variável m (melhor caso) tem um tempo de execução equivalente à:

$$(n^3 - 2n^2 + n - 1)c_1 + 4c_2 + 2n^3c_3 + c_4 + (n^3 + 3)c_5$$

Enquanto quando este encontra apenas valores compatíveis (pior caso), este tem um tempo de execução equivalente à:

$$(n^3 - 2n^2 + n - 1)c_1 + (n^3 + 4)c_2 + 3n^3c_3 + c_4 + (n^3 + 3)c_5 \blacksquare$$

Exercício 2

Segundo Márcio Ribeiro (2021, p. 32), temos que:

> [...] funções crescem de maneira similar uma vez que abstraímos as constantes.
>
> A notação Θ formaliza matematicamente essa ideia. [...] $\Theta(g(n))$ é o conjunto de todas as funções que crescem de maneira parecida com g, que são *assintoticamente equivalentes* a g.

O autor então exprime este conceito na seguinte fórmula matemática:

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

Assim sendo, temos que em nosso caso $g(n) = n^3$, enquanto a função equivalente ao pior caso, igualando as constantes à 1, e substituindo c_1 por $c_2 + c_3 + c_5$ é $f(n) = 8n^3 - 6n^2 + 3n + 5$, para qualquer $n \geq 1$. Assim sendo, tem-se:

$$0 \leq c_1 n^3 \leq 8n^3 - 6n^2 + 3n + 5 \leq c_2 n^3$$

Utilizando-se de uma calculadora eletrônica, obtemos que $0 \leq c_1 \lessapprox 2.16$ e $10 \leq c_2$.

De fato, vemos que este resultado se concretiza para valores de $n \geq 1$:

> Linha azul: $f(n)$; linha verde: $2.16\,g(n)$; e linha vermelha: $10\,g(n)$

Portanto, o tempo de processamento do algoritmo 3Soma é $\Theta(n^3)$ no pior caso. ■

Exercício 3

```
insertionSort (A, n)
    if (n ≤ 1)
        then return
    insertionSort (A, n - 1)
    tmp ← A[n]
    i ← n - 1
    while i > 0 and A[i] > tmp do
        A[i + 1] ← A[i]
        i ← i - 1
    end
    A[i + 1] ← tmp
end

binarySearch (A, n, key)
    i ← ⌊n + 1 / 2⌋

    if (A[i] = key)
        then return true
    if (n ≤ 1)
        then return false
    if (A[i] < key)
        then return binarySearch(A[i + 1], n - i, key)
    return binarySearch(A, i - 1, key)
end

3Soma (A, B, C)
    m ← 0

    insertionSort (C, sizeof(C))
    for i ← 1 to sizeof(A)
        do for j ← 1 to sizeof(B)
            do if binarySearch (C, sizeof(C), -(A[i] + B[j]))
                then m ← m + 1
    return m
end
```

> **Obs:** Os algoritmos `insertionSort` e `binarySearch` acima expostos admitem listas cujos índices encontram-se numerados $1, 2, \ldots, n$.

Conforme demonstra Ribeiro (Ibid., p. 45), o tempo de execução T do algoritmo *Insertion Sort* é

$$T(n) \in \Theta(n^2)$$

Enquanto o tempo de execução T do algoritmo de busca binária (Ibid. p. 35) é

$$T(n) \in \Theta(\log(n))$$

Com isso e a modificação na função `3Soma` que reduz o número de repetições do tipo `for` de três para duas ($T(n) \in \Theta(n^2)$), é possível afirmar que o tempo de execução T no pior caso para a 32ª linha do código (`do if binarySearch (C, sizeof(C), -(A[i] + B[j]))`), é tal que

$$T(n) \in \Theta(n^2 \log n)$$

Sendo esta a linha com maior número de iterações no algoritmo `3Soma`, podemos, por extensão, afirmar que a notação Θ anteriormente descrita é representativa do tempo de execução do algoritmo como um todo. ■

Exercício 4

Conforme a hipótese, o algoritmo proposto é correto se este retorna para qualquer sequência $A[1], \ldots, A[n]$ a mesma ordenada de forma crescente.

Ora, ao longo de toda sua execução, é invariável que os elementos em $A[1], \ldots, A[i]$ encontram-se ordenados em ordem crescente.

- Na inicialização, quando $A[i] = A[1]$, a base da hipótese;
- Nos passos seguintes, seja quando$A[1]\ e\ A[2]$ encontram-se ordenados e $A[3]$ é relocado na linha 6 se avaliado necessário na linha 5, seja para qualquer valor $i+1$ que se segue, o que constitui a manutenção da propriedade invariável;
- Seja ao término do programa, quando $i+1=n$ e o programa finalmente retorna a sequência ordem crescente.

O programa descrito é portanto, correto. Ainda que bastante ineficiente, mas isso não cabe aqui avaliar. ■

Exercício 5

O algoritmo proposto é correto se

- havendo um ou mais valores $a_i + b_j + c_k = 0$, onde $a_i \in \{a_1, ..., a_n\}$, $b_j \in \{b_1, ..., b_n\}$ e $c_k \in \{a_1, ..., a_n\}$, este retorna um valor $m \geq 1$;

- senão este retorna um valor $m = 0$.

Ao longo de sua execução, é invariável que m equivale ao número de somas possíveis iguais a 0 entre todos os números em $\{a_1, ..., a_i\}$, com $\{b_1, ..., b_j\}$ e $\{c_1, ..., c_k\}$ desde a inicialização (a_1, b_1, c_1) (base da hipótese), para cada valor (a_i, b_j, c_k) (passo da indução), e ao ser alcançada a condição de término onde (a_n, b_n, c_n). Assim, ao término deste programa todas as combinações possíveis foram avaliadas e este é, portanto, correto. ■

Referências

RIBEIRO, M. **Introdução à Análise de Algoritmos**. Disponível em: https://github.com/marciomr/apostila-iaa/blob/master/apostila-iaa.pdf. Acesso em: 13 out. 2021.

---

1. nUSP: 12543033; Turma 04 ↩