Notação Assintótica e algoritmo de ordenação

profº Mauricio Conceição Mario

Análise Matemática

Logaritmo $\rightarrow$ todo expoente cuja base é positiva e diferente de 1.

$$a^c = b \Leftrightarrow \log_a b = c$$

Logaritmo de ***b*** na base ***a*** igual a ***c***.

- Logaritmo = ***c***
- Base = ***a***
- Logaritmando = ***b***

- Exemplo: $\log_{10} 1000 \Rightarrow 10^c = 1000 \Rightarrow c = 3$

(Lemos et al., 1979)

Sejam os algoritmos que têm $n = 1000$ entradas, e cujas funções $f(n)$ determinam o tempo estimado de execução para cada algoritmo. Calcular o tempo (considerando em μs) de execução em cada caso.

	$f(n)$	***tempo estimado de execução***
a) Algoritmo 1	$\log_{10} n$	$t = \log 1000 = 3\ \mu s$
b) Algoritmo 2	$\sqrt{n}$	$t = \sqrt{1000} \cong 31{,}62\ \mu s$
c) Algoritmo 3	n	$t = 1000\ \mu s$
d) Algoritmo 4	$n \log_{10} n$	$t =1000 * \log 1000 = 3000\ \mu s$
e) Algoritmo 5	n^2	$t = 1000^2 = 1000000\ \mu s = 1s$
f) Algoritmo 6	n^3	$t = 1000^3 = 1000000000\ \mu s = 1000s \cong 16{,}67min$
g) Algoritmo 7	2^n	$t = 2^{1000} \rightarrow \infty$
h) Algoritmo 8	$n!$	$t = 1000! \rightarrow \infty$

(Cormen et al., 2012)

Complexidade de Tempo → *seja **A** um algoritmo e* $\{E_1, ..., E_m\}$ *o conjunto de todas as entradas possíveis de **A**. Denote por **ti** o número de passos efetuados por **A**, quando a entrada for* $\mathbf{E}_i$*, então definem-se:*

→ ***Complexidade do pior caso*** $= \; max_{E_i \in E} \; \{t_i\},$ ←

→ ***Complexidade do melhor caso*** $= \; min_{E_i \in E} \; \{t_i\},$

→ ***Complexidade do caso médio*** $= \; \Sigma_{\; 1 \leq i \leq m} \; p_i t_i,$

onde p_i é a probabilidade de ocorrência da entrada E_i.

A **Complexidade do pior caso** corresponde ao número de passos que o algoritmo efetua no seu pior caso de execução, ou seja, para a entrada mais desfavorável.

A **complexidade de pior caso** fornece um limite superior para o número de passos que um algoritmo pode efetuar.

(Szwarcfiter & Markenzon, 2020)

As Complexidades têm por objetivo avaliar a eficiência de ***tempo*** *ou* ***espaço*** (*pode ser compreendido como o número de itens de entrada ou como a quantidade de bits para representar a entrada*).

$\rightarrow$ *A* ***complexidade de tempo de pior caso*** *corresponde ao número de passos que o algoritmo efetua para uma entrada maior ou mais desfavorável* (*por exemplo no caso de ordenação de números, uma entrada favorável seria se os números já estivessem ordenados*). *Neste caso há um limite superior para o número de passos que o algoritmo pode efetuar. O termo complexidade é empregado como significado de pior caso.*

(Szwarcfiter & Markenzon, 2020; Cormen et al., 2012)

Expressão do tempo de execução

Exemplo: algoritmo de ***ordenação por inserção***:

Considerando modelo de computação genérico de máquina de acesso aleatório à memória (*random-access machine*, *RAM*).

→ *Considerando que a execução de cada i-ésima instrução de código leve um tempo c_i onde c_i é uma constante. Para escrever a expressão do tempo de execução do algoritmo são utilizados todos os "custos" de tempo de instrução c_i assim como o número de* ***vezes*** *que cada instrução é executada, sendo n = A.dimensão.*

	custo	vezes
for *j* = 2 **to** *A.dimensão*	c_1	n
temporário = A[j]	c_2	$n - 1$
//inserir A[j] na seq ordenada A[1 ... j - 1]	0	$n - 1$
i = j -1	c_4	$n - 1$
while *i > 0* ***e*** *A[i] > temporário*	c_5	$\sum_{j=2}^{n} t_j$
A[i + 1] = A[j]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
i = i -1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
A[i + 1] = temporário	c_8	$n - 1$

t_j = número de vezes que o laço while é executado para aquele valor de j.

(Cormen et al., 2012)

Expressão do tempo de execução

Algoritmo de ***ordenação por inserção***:

→ *O tempo total de execução é a soma de todos os tempos de execução para cada instrução.*

$$T_{(n)} = c_1.n + c_2.(n-1) + c_4.(n-1) + c_5.\sum_{j=2}^{n} t_j + c_6.\sum_{j=2}^{n}(t_j - 1) + c_7.\sum_{j=2}^{n}(t_j - 1) + c_8.(n-1)$$

	custo	vezes
for *j* = 2 **to** *A.dimensão*	c_1	n
temporário = *A*[*j*]	c_2	$n-1$
//*inserir A*[*j*] *na seq ordenada A*[1 ... *j* - 1]	0	$n - 1$
i = *j* -1	c_4	$n - 1$
while *i* > 0 ***e*** *A*[*i*] > *temporário*	c_5	$\sum_{j=2}^{n} t_j$
A[*i* + 1] = *A*[*j*]	c_6	$\sum_{j=2}^{n}(t_j - 1)$
i = *i* -1	c_7	$\sum_{j=2}^{n}(t_j - 1)$
A[*i* + 1] = *temporário*	c_8	$n - 1$

(Cormen et al., 2012)

Expressão do tempo de execução Algoritmo de ***ordenação por inserção***:

$$T_{(n)} = c_1.n + c_2.(n-1) + c_4.(n-1) + c_5.\sum_{j=2}^{n} t_j + c_6.\sum_{j=2}^{n}(t_j - 1) + c_7.\sum_{j=2}^{n}(t_j - 1) + c_8.(n-1)$$

→ Se o arranjo já estiver ordenado (**melhor caso**) pode-se expressar o tempo de execução como $T_{(n)} = a.n + b$ para constantes ***a*** e ***b*** que dependam dos custos de instrução c_i e desse modo o tempo de execução se torna uma função linear de ***n***.

→ Para o **pior caso**, ou seja, se o arranjo estiver ordenado em ordem inversa, o tempo de execução do algoritmo de ordenação por inserção pode ser expresso por: $T_{(n)} = a.n^2 + b.n + c$ onde a, b e c dependem dos custos de instrução c_i.

→ *O tempo de execução é portanto uma função quadrática de n.*

→ *Os custos de instrução representados pelas constantes a, b e c, podem ser desprezados assim como os termos de ordem mais baixa (n). Taxa de crescimento ou ordem de crescimento do tempo de execução → Seria considerada então apenas a ordem de crescimento mais alta, no caso* $\boldsymbol{n^2}$. *Então o tempo de execução para o pior caso seria simplificado para* $\boldsymbol{\Theta(n^2)}$ [“*teta de n ao quadrado*”].

(Szwarcfiter & Markenzon, 2020; Cormen et al., 2012)

Notação Assintótica

Quando a dimensão da entrada para um algoritmo for grande o suficiente para tornar relevante apenas a ordem de crescimento do tempo de execução $\mathbf{n^m}$*, está se estudando a* ***eficiência assintótica do algoritmo***.

A ***Notação Assintótica*** *é utilizada para descrever o tempo de execução assintótico de um algoritmo e são definidas em termos de funções cujos domínios são o conjunto dos números naturais* $\mathbb{N} = \{0, 1, 2, ...\}$.

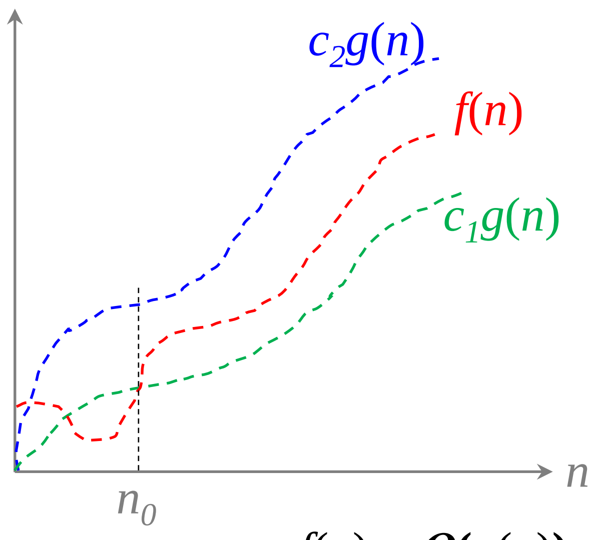
(Szwarcfiter & Markenzon, 2020; Cormen et al., 2012)

Notação Θ (téta)

$T_{(n)} = \boldsymbol{\Theta}(\boldsymbol{n^2})$ é o tempo de execução para o pior caso do algoritmo de ordenação por inserção. Para uma dada função $g(n)$, denota-se por $\boldsymbol{\Theta}(g(n))$ o conjunto de funções:

$\boldsymbol{\Theta}(g(n)) = f(n)$: (tal que) { existem constantes positivas c_1, c_2 e n_0 tais que

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ para todo } n \geq n_0 \}$$



$f(n) = \boldsymbol{\Theta}(g(n))$

Para todos os valores de n em n_0 ou à direita de n_0, o valor de $f(n)$ encontra-se em $c_1g(n)$ ou acima de seus valores, e em $c_2g(n)$ ou abaixo de seus valores. Ou seja, para todo $n \geq n_0$ a função $f(n)$ é igual a $g(n)$ dentro de um fator constante. Diz-se, então, que $g(n)$ é um *limite assintoticamente restrito* para $f(n)$.

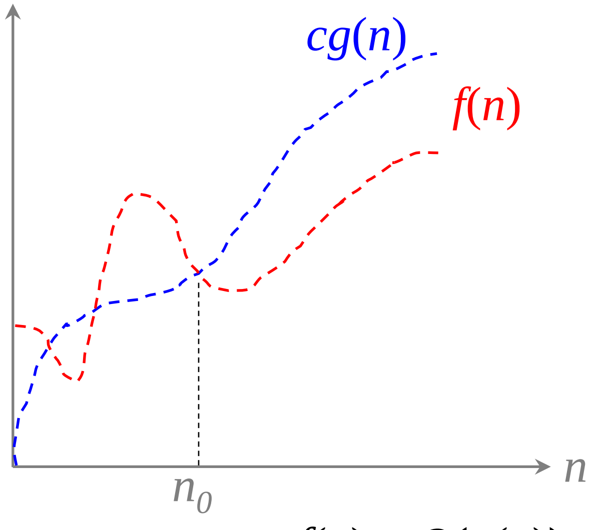
(Szwarcfiter & Markenzon, 2020; Cormen et al., 2012)

Notação O (ó grande)

A notação Θ limita assintoticamente uma função acima e abaixo. Quando se tem apenas um ***limite assintótico superior***, utiliza-se a notação ***O***. Para uma dada função $g(n)$, denota-se por $\boldsymbol{O}(g(n))$ [lê-se "*Ó grande de g de n*"] o conjunto de funções:

$\boldsymbol{O}(g(n)) = f(n)$: (tal que) { existem constantes positivas c e n_0 tais que

$$0 \leq f(n) \leq cg(n) \text{ para todo } n \geq n_0 \}$$



$f(n) = \boldsymbol{O}(g(n))$

Para todos os valores de n em n_0 ou à direita de n_0, o valor de $f(n)$ encontra-se abaixo de $cg(n)$ ou em $cg(n)$.

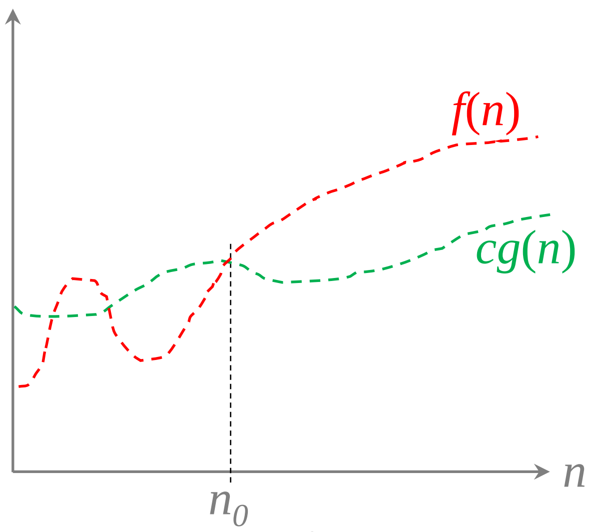
(Szwarcfiter & Markenzon, 2020; Cormen et al., 2012)

Notação Ω (ômega grande)

A notação Ω fornece um ***limite assintótico inferior***. Para uma dada função $g(n)$, denota-se por $\Omega(g(n))$ [lê-se "*ômega grande de g de n*"] o conjunto de funções:

$\Omega(g(n)) = f(n)$: (tal que) { existem constantes positivas c e n_0 tais que

$$0 \leq cg(n) \leq f(n) \quad \text{para todo } n \geq n_0 \ \}$$



$f(n) = \Omega(g(n))$

Para todos os valores de n em n_0 ou à direita de n_0, o valor de $f(n)$ encontra-se em $cg(n)$ ou acima de $cg(n)$.

(Szwarcfiter & Markenzon, 2020; Cormen et al., 2012)

Tempo de execução do algoritmo de ordenação por inserção

→ Variável = dimensão da entrada

C:\Users\cmari\.spyder-py3\ordenação_inserção.py

history_internal.py | função_enumerate.py | iteradores.py | inverte_sequência.py | datetime.py | soma_matricial.py | ordenação_inserção.py

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Jul  7 10:00:53 2022

@author: cmari
"""

import time
#from datetime import datetime

A = [5, 2, 4, 6, 1, 3]

print("MATRIZ A:", A)
print("dimensão da matriz A = ", len(A))

tempo_inicial = time.time()
for p in range(1):
   for j in range(1, len(A)):
        temporário = A[j]
        i = j - 1
        while i >= 0 and A[i] > temporário:
           A[i + 1] = A[i]
           i = i - 1
        A[i + 1] = temporário
tempo_final = time.time()
print("MATRIZ A ordenada  = ", A)
print("tempo de execução = ", tempo_final - tempo_inicial)
```

Console 1/A

```
In [8]: runfile('C:/Users/cmari/.spyder-py3/ordenação_inserção.py', wdir='C:/Users/cmari/.spyder-py3')
MATRIZ A: [5, 2, 4, 6, 1, 3]
dimensão da matriz A =  6
MATRIZ A ordenada  =  [1, 2, 3, 4, 5, 6]
tempo de execução =  0.0
```

j = 1

temporário = A[1] = 2

i = j – 1 = 1 – 1 = 0

enquanto i ≥ 0 e A[i] > temporário

  (i = 0 e A[0] = 5 > temporário = 2)

  A[i+1] = A[i]

  A[1] ← A[i = 0] = 5

  i = i – 1

  i = 0 – 1 = -1

A[-1 + 1] = temporário

A[0] = 2

∴ A = [2, 5, 4, 6, 1, 3]

Referências Bibliográficas

- Estruturas de Dados e Seus Algoritmos
  Jayme L. Szwarcfiter & Lilian Markenzon
  3ª edição – editora *gen* LTC – 2010 - 2020

- Matemática Avançada para Engenharia
  Dennis G. Zill & Michael R. Cullen
  Álgebra Linear e Cálculo Vetorial (2) – 3ª edição, editora Bookman – 2009

- Algoritmos – Teoria e Prática
  Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
  3ª edição – editora Elsevier - *gen* LTC – 2012