Técnicas e Análise de Algoritmos

Notação Assintótica - Parte 02

Professor: **Jeremias Moreira Gomes**
E-mail: jeremias.gomes@idp.edu.br

Introdução

Recapitulação

- Complexidade Computacional
- Visualização Numérica da Complexidade
- Notação Assintótica Big-O
- Propriedades da Notação Big-O

Propriedades da Notação Big-O

- **P1 - Transitividade**
  - Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$, então $f(n) = O(h(n))$
- **P2 - Soma de funções de mesma complexidade**
  - Se $f(n) = O(h(n))$ e $g(n) = O(h(n))$, então $f(n) + g(n) = O(h(n))$
- **P3 - Absorção da constante em monômios**
  - A função an^k é $O(n^k)$

Propriedades da Notação Big-O

- **P4 - Cota superior para polinômios**
  - A função n^k é $O(n^{k+j}), \forall j > 0$
- **P5 - Absorção de constantes**
  - Se $f(n) = cg(n)$, então $f(n) = O(g(n))$

Notação Big-O e Polinômios

Se $f(n)$ é um polinômio de grau k então $f(n)$ é $O(n^k)$

Demonstração: Considere os monômios $f_i(n) = a_i x^i, i = 0, 1, \ldots, k$

Temos que

$$f(n) = f_k(n) + f_{k-1}(n) + \cdots + f_0(n)$$

A propriedade 3 nos diz que $f_k(n)$ é $O(n^k)$

A propriedade 4 nos diz que $f_{k-j}(n)$ é $O(n^{(k-j)+j}) = O(n^k))$

Por fim, pela propriedade 2, concluímos que

$$f(n) = \sum_{j=0}^{k} f_{k-j}(n) = O(n^k)$$

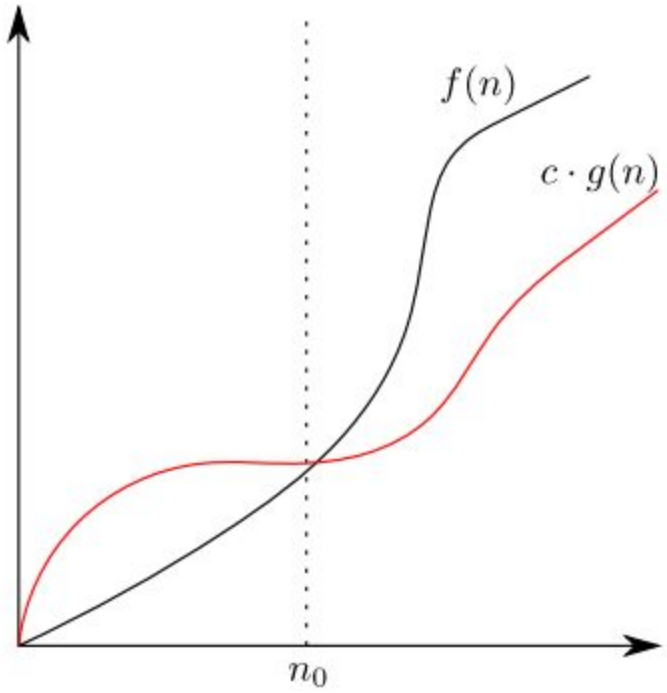
Notação Assintótica Big- Ω

Definição

Dadas duas funções de valores positivos f e g, f(n) é Ω(g(n)) se existem c e N positivos tais que f(n) ≥ cg(n), ∀ n ≥ N

- Big-Ω lê-se Big-Ómega
- Em termos matemáticos, cg(n) é uma cota inferior de f(n)
- Informalmente, f tende a crescer, no mínimo, tão rápida quanto g, a partir de algum determinado ponto

$$\Omega(g(n)) = \{f(n) | 0 \leq c \cdot g(n) \leq f(n), c \in \mathbb{R}^+, \forall n \geq n_0 \in \mathbb{R}^+\}$$



Observações sobre a Notação Big- Ω

- Enquanto Big-O se refere a cotas superiores, Big-Ω se refere a uma cota inferior para f(n)
- Tanto Big-O quanto Big-Ω possuem infinitas possibilidades para a constante c e N
- Equivalência:
  - $f(n)$ é $\Omega(g(n))$ se, e somente se, $g(n)$ é $O(f(n))$

Notação Big- Ω - Exemplos

- Exemplos de utilização

  - $2n^2 \in \Omega(n^2)$
  - $n^2 \in \Omega(n)$
  - $log\ n \in \Omega(log\ log\ n)$

Notação Big- Ω - Exemplos

- Exemplos de utilização

- $2n^2 \in \Omega(n^2)$
- $n^2 \in \Omega(n)$
- $log\ n \in \Omega(log\ log\ n)$
- $n^2 \notin \Omega(n^3)$
- $2n^2 \notin \Omega(2^n)$
- $log\ n \notin \Omega(n)$

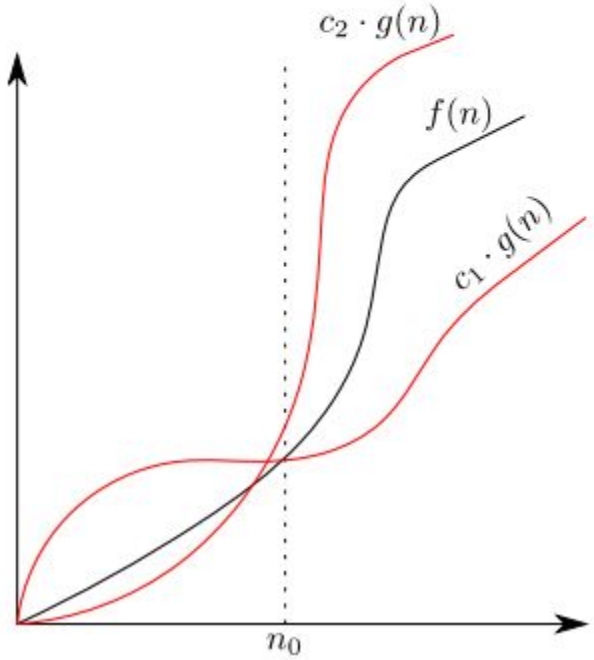
Notação Assintótica Big- Θ

Definição

Dadas duas funções de valores positivos f e g, f(n) é Θ(g(n)) se existem c_1, c_2 e N positivos tais que $c_1 g(n) \leq f(n) \leq c_2 g(n)$, $\forall\, n \geq N$

- Big-Θ lê-se Big-Théta
- A notação Θ é uma cota justa de f(n)
- Informalmente, f tende a crescer tanto quanto quanto g, em termos assintóticos

$$\Theta(g(n)) = \{f(n) | 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \\ c_1, c_2 \in \mathbb{R}^+, \forall n \geq n_0 \in \mathbb{R}^+\}$$



Observações sobre a Notação Big- Θ

- Enquanto Big-O se refere a cotas superiores e Big-Ω se refere a uma cota inferior para f(n), Big-Θ se refere a uma cota justa para f(n)
- Equivalência:

$f(n)$ é $\Theta(g(n))$ se, e somente se, $f(n)$ é $O(g(n))$ e $f(n)$ é $\Omega(g(n))$

Exemplo de Big- Θ

- Já sabemos que $f(n) = 2n^2 + 3n + 1$ é O(n^2)
- Será que ela é Θ(n^2)
  - Falta demonstrar, que ela também é Ω(n^2)

$$2n^2 + 3n + 1 \geq c_2 n^2$$

$$2 + \frac{3}{n} + \frac{1}{n^2} \geq c_2$$

- A inequação é verdadeira para N = 1, c_2 = 2 satisfazem a definição

Notação Big- Ω - Exemplos

- Exemplos de utilização

- $n^2 \in \Theta(n^2)$
- $10^{42}n^2 \in \Theta(n)$
- $\frac{1}{2}log\ n \in \Theta(log\ log\ n)$

Notação Big- Ω - Exemplos

- Exemplos de utilização

- $n^2 \in \Theta(n^2)$
- $10^{42}n^2 \in \Theta(n)$
- $\frac{1}{2}log\ n \in \Theta(log\ log\ n)$

- $n^2 \notin \Theta(n^3)$
- $200n^2 \notin \Theta(n)$
- $\frac{1}{2}log\ n \notin \Theta(\sqrt{n})$

Problemas Possíveis

- O fato de um algoritmo ter ordem de complexidade menor do que outro não implica que ele seja o mais eficaz em todos os casos
- Por exemplo, considere as funções $f(n) = 10^8 n$ e $g(n) = 10n^2$
- Temos que $f(n)$ é $O(n)$ e $g(n)$ é $O(n)$
- Para $n < 10^7$, o tempo de execução de f(n) é maior do que g(n)
- Apenas para valores iguais ou superiores é que a função f(n) se torna mais eficiente que g(n)

N = 10

Classe	Notação	Número de operações	Tempo de execução[1]
constante	$O(1)$	1	$1\mu s$
logarítmica	$O(\log n)$	$2,3$	$2\mu s$
linear	$O(n)$	10	$10\mu s$
$O(n \log n)$	$O(n \log n)$	23	$23\mu s$
quadrática	$O(n^2)$	100	$100\mu s$
cúbica	$O(n^3)$	1000	1ms
exponencial	$O(2^n)$	1024	1ms

[1] Uma instrução por μs

N = 100

Classe	Notação	Número de operações	Tempo de execução[1]
constante	$O(1)$	1	$1\mu s$
logarítmica	$O(\log n)$	$4,6$	$5\mu s$
linear	$O(n)$	100	$100\mu s$
$O(n \log n)$	$O(n \log n)$	460	$460\mu s$
quadrática	$O(n^2)$	10000	10ms
cúbica	$O(n^3)$	10^6	1s
exponencial	$O(2^n)$	10^{30}	10^7a

[1] Uma instrução por μs

N = 1000

Classe	Notação	Número de operações	Tempo de execução[1]
constante	$O(1)$	1	$1\mu s$
logarítmica	$O(\log n)$	$6,9$	$7\mu s$
linear	$O(n)$	1000	1ms
$O(n \log n)$	$O(n \log n)$	6907	7ms
quadrática	$O(n^2)$	10^6	1s
cúbica	$O(n^3)$	10^9	16,7m
exponencial	$O(2^n)$	10^{301}	...

[1] Uma instrução por μs

Conclusão