Técnicas e Análise de Algoritmos

Notação Assintótica - Parte 01

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

Introdução

Complexidade Assintótica e Computacional

Complexidade Computacional

- Um mesmo problema pode ser resolvido por algoritmos diferentes com eficiências distintas
- A complexidade computacional é uma medida de comparação da eficiência entre diferentes algoritmos
- Foi desenvolvida por Juris Hartmanis e Richard E. Stearns

Complexidade Computacional

- Complexidade indica o esforço ou custo de um algoritmo
  - **Critérios para esforço**
    - Tempo de desenvolvimento
    - Recursos humanos
    - Viabilidade
  - **Critérios para custo**
    - Tempo de execução
    - Espaço em memória

Complexidade Computacional

- Comparações absolutas do tempo de execução entre dois algoritmos distintos devem ser realizadas na mesma máquina
  - Os algoritmos devem ser escritos na mesma linguagem de programação
- Comparações absolutas do tempo de execução são difíceis de realizar e de pouca utilidade prática
- Sendo assim, uma alternativa é comparar relativamente os tempos de execução entre algoritmos distintos, abstraindo fatores concretos

Complexidade Computacional

- Neste contexto, o tempo deve ser expresso não em unidades de medidas físicas (segundos, milissegundos, etc), e sim em unidades de medidas lógicas
  - Relação entre o número de elementos N a serem processados e o tempo t necessário para o processamento dos mesmos
  - Exemplos:

$$t = 10N$$
$$t = N^2$$
$$t = n * log_2(n)$$
$$t = f(n)$$

Complexidade Computacional

- A função que expressa o tempo em função do tamanho da entrada tende a ser bastante elaborada e difícil de se explicitar
- Por causa disso, geralmente considera-se apenas os termos que afetam a ordem de magnitude da função
- A ordem de magnitude é determinada pelo termo que caracteriza o comportamento da função quando o número de elementos N tende ao infinito

Complexidade Computacional

- De uma maneira mais formal, $t(N)$ caracteriza o comportamento da função $f(N)$ se

$$\lim_{N \to \infty} \frac{f(N)}{t(N)} = c$$

onde c é uma constante

- Essa aproximação é denominada **complexidade assintótica**

Complexidade Computacional - Exemplos

Função	Termo Dominante	Ordem de Magnitude
$a(N) = 123$	123	Constante
$b(N) = \log N$	$\log N$	Logarítmica
$c(N) = N + 7\log_3 N^2$	N	Linear
$d(N) = N^2 + 50N + 250$	N^2	Quadrática
$e(N) = N^2 + N^3$	N^3	Cúbica
$f(N) = N^4 + \sqrt[5]{N^{21}}$	$\sqrt[5]{N^{21}}$	Polinomial
$g(N) = \operatorname{senh} N + N^3$	$\frac{1}{2}e^N$	Exponencial
$h(N) = e^N + N!$	$N!$	Fatorial

Visualização Numérica da Complexidade

- A complexidade assintótica pode ser aproximada numericamente através da observação da contribuição de cada termo da função f(N) a medida que N cresce
  - Exemplo a seguinte função:

$$f(N) = N^2 + 100N + log_{10}N + 1000$$

Visualização Numérica da Complexidade

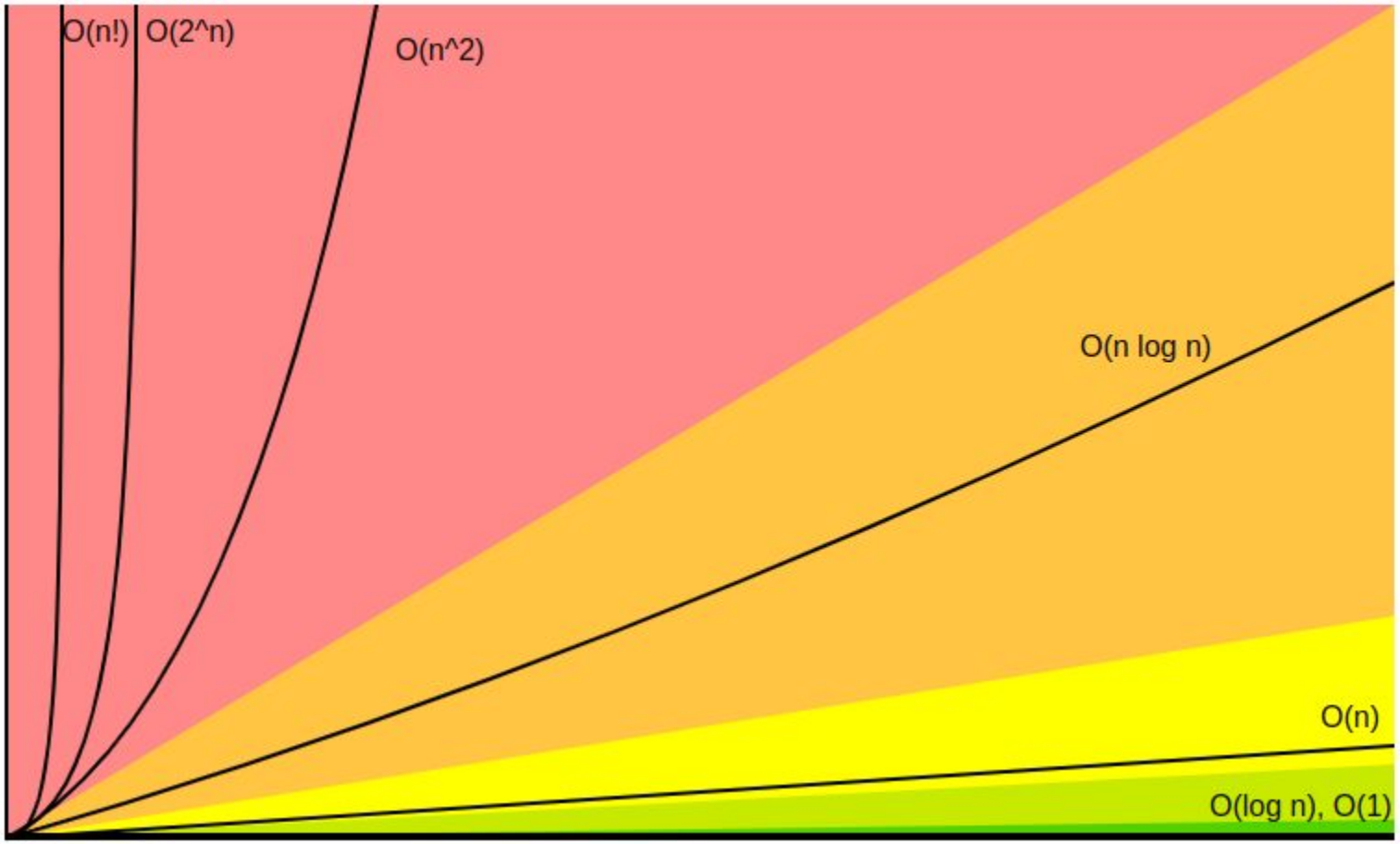
$$f(N) = N^2 + 100N + log_{10}N + 1000$$

qde	N^2	$100N$	$log_{10}N$	1000
1	1	100	0	1000
10	100	1000	1	1000
100	10000	10000	2	1000
1000	1000000	100000	3	1000
10000	100000000	1000000	4	1000
100000	10000000000	10000000	5	1000

Visualização Numérica da Complexidade

$$f(N) = N^2 + 100N + log_{10}N + 1000$$

%	N^2	$100N$	$log_{10}N$	1000
1	0,001	0,091	0,000	0,908
10	0,048	0,476	0,000	0,476
100	0,476	0,476	0,000	0,048
1000	0,908	0,091	0,000	0,001
10000	0,990	0,010	0,000	0,000
100000	0,999	0,001	0,000	0,000


O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)

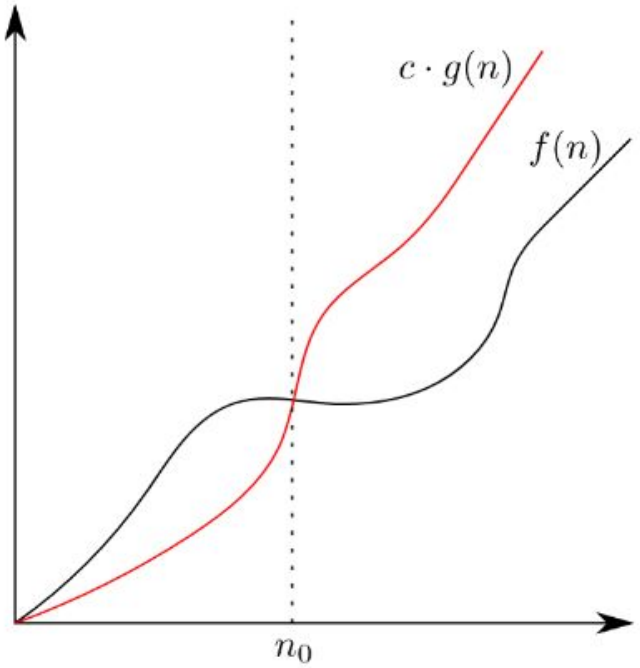
Notação Assintótica Big-O

Definição

Dadas duas funções de valores positivos f e g, f(n) é O(g(n)) se existem c e N positivos tais que $f(n) \leq cg(n), \quad \forall n \geq N$

- Big-O é uma notação para complexidade assintótica desenvolvida por Paul Bachmann
- Em termos matemáticos, cg(n) é uma cota superior de f(n)
- Informalmente, f tende a crescer, no máximo, tão rápida quanto g, a partir de algum determinado ponto

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



Definindo c e N

- Dados f e g, como determinar c e N?
- Por exemplo, a função $f(n) = 2n^2 + 3n + 1$ é O(n^2), porque

$$2n^2 + 3n + 1 \leq cn^2$$
$$2 + \frac{3}{n} + \frac{1}{n^2} \leq c$$

- Assim, para todo n ≥ 1, temos que o lado direito é menor ou igual a 6
  - Portanto, N = 1, c = 6 satisfazem a definição de Big-O

Definindo c e N

- Pode-se reparar que c e N possuem múltiplas soluções válidas
- A melhor escolha para c e N é aquela baseada no ponto a partir do qual o termo principal se torna e se mantém o maior termo, observando-se o termo dominante
  - Dessa forma, no caso do exemplo anterior ($f(n) = 2n^2 + 3n + 1$), pode-se refinar a solução calculando-se a solução da inequação $2n^2 > 3n$
    - Resposta: n > 1
    - Então, para o exemplo anterior N = 2 e c = 15/4

Observações sobre a Notação Big-O

- Para o exemplo apresentado, a afirmação f(n) é $O(n^3)$ também é verdadeira
  - É uma observação verdadeira, mas não representa uma "realidade aproximada" do comportamento real da função
  - Parando-se para pensar, f(n) é O(g(n)) para qualquer

$$g(n) = n^k, k \geq 2$$

Observações sobre a Notação Big-O

- Na prática, escolhe-se o monômio de menor grau possível, onde a notação Big-O seja válida
- Além disso, a notação também costuma ser aplicada somente ao termo dominante de f(n), pelo seguinte:

$$f(n) = n^2 + 100n + log_{10}n + 1000$$
$$f(n) = n^2 + 100n + log_{10}n + O(1)$$
$$f(n) = n^2 + 100n + O(log_{10}n)$$
$$f(n) = n^2 + O(n)$$
$$f(n) = O(n^2)$$

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

Conclusão