Análise de Complexidade

Fundamentos

Prof. Edson Alves - UnB/FGA

2020

Sumário

Complexidade Computacional e Assintótica

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
- Ela indica o esforço ou custo de um algoritmo
- Critérios para esforço: tempo de desenvolvimento, recursos humanos, viabilidade
- Critérios para custo: tempo de execução e espaço em memória
- Comparações absolutas do tempo de execução entre dois algoritmos distintos devem ser realizadas na mesma máquina, e os algoritmos devem ser escritos na mesma linguagem de programação

Comparação relativa do tempo de execução

- 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
- 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 de medidas lógicas de tempo:

$$t = 10N$$

$$t = \log_2 N$$

$$t = f(N)$$

Complexidade assintótica

- A função que expressa o tempo em função do número de termos tende a ser bastante elaborada e difícil de se explicitar
- Por isso, 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
- Em notação matemática, $t(N)$ caracteriza o comportamento da função $f(N)$ se

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

  onde c é uma constante
- Esta aproximação é denominada complexidade assintótica

Exemplos de complexidade assintótica

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
$g(N) = \text{senh}\, N + N^3$	$\frac{1}{2}e^N$	Exponencial
$h(N) = e^N + N!$	$N!$	Fatorial

Visualização numérica da complexidade assintótica

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

Por exemplo, considere a função $f(N) = N^2 + 100N + \log_{10} N + 1000$

	$N = 1$	$N = 10$	$N = 100$	$N = 1.000$
1000	90,8%	47,6%	4,8%	0,1%
$\log_{10}(N)$	0,0%	0,1%	0,0%	0,0%
$100N$	9,1%	47,6%	47,6%	9,1%
N^2	0,1%	4,7%	47,6%	90,8%

Notações Big-O, Big-Ω e Big-Θ

Notação 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 $f(n) \leq cg(n), \ \forall n \geq N$

- O Big-O é uma notação para complexidade assintótica desenvolvida em 1894 por Paul Bachmann
- Em termos matemáticos, $cg(n)$ é uma cota superior de $f(n)$
- Informalmente, f tende a crescer, no máximo, tão rápido quanto g a partir de um determinado ponto

Determinando c e N

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

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

- Assim, para todo $n \geq 1$, temos que o lado direito é menor ou igual a 6. Portanto, $N = 1, c = 6$ satisfazem a definição da notação Big-O
- A melhor escolha para c e N é aquela que baseada no ponto a partir do qual o termo principal se torna e se mantém o maior termo
- Retornando à função $f(n)$, qual é a solução da inequação $2n^2 > 3n$? Resposta: $n > 1$
- Para $N = 2$, temos $c = \frac{15}{4}$

Observações sobre o Big-O

- Para função anteriormente citada, a afirmação $f(n)$ é $O(n^3)$ também é verdadeira
- De fato, $f(n)$ é $O(g(n))$ para qualquer $g(n) = n^k$, $k \geq 2$
- Na prática, escolhe-se o monômio de menor grau possível
- A aproximação Big-O pode ser refinada se aplicada apenas em parte da função. Ex.:

$$
\begin{aligned}
f(n) &= O(n^2) \\
f(n) &= n^2 + O(n) \\
f(n) &= n^2 + 100n + O(\log_{10} n) \\
f(n) &= n^2 + 100n + \log_{10} n + O(1) \\
f(n) &= n^2 + 100n + \log_{10} n + 1000
\end{aligned}
$$

Propriedades da Notação Big-O

P1. Propriedade Transitiva

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 de 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 constante

Se $f(n) = cg(n)$, então $f(n)$ é $O(g(n))$.

P6. Equivalência entre logaritmos

A função $f(n) = \log_a n$ é $O(\log_b n)$ para quaisquer a e b positivos diferentes de 1.

P7. Normalização da base logarítmica

A função $f(n) = \log_a n$ é $O(\log n)$ para qualquer a positivo diferente de 1, com $\log n = \log_2 n$.

Notação Big-O e polinômios

Proposição.

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

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

A propriedade 3 nos diz que $f_k(n)$ é $O(n^k)$ e a propriedade 4 nos diz que $f_{k-j}(n)$ é $O(n^{(k-j)+j}) = O(n^k)$.

Por fim, pela propriedade 2,

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

$\square$

Notação Big-Ω

Definição

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

- Lê-se "f é Big-Ômega g"
- Em termos matemáticos, $cg(n)$ é uma cota inferior de $f(n)$
- Informalmente, $f(n)$ cresce, no mínimo, tão rápido quanto $g(n)$ a partir de determinado ponto
- Enquanto o Big-O se refere às cotas superiores de $f(n)$, o Big-Ω se refere às cotas inferiores
- Equivalência: $f(n)$ é $\Omega(g(n))$ se, e somente se, $g(n)$ é $O(f(n))$
- Tanto a definição do Big-O quanto do Big-Ω permitem infinitas possibilidades para c e N.
- É possível restringir o conjunto de escolhas para c e N através da notação Big-Θ

Notação Big-Θ

Definição

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

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \ \forall n \geq N$$

- Lê-se "f é Big-Theta g", ou "f tem ordem de magnitude g"
- 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 da notação Big-Θ

- Seja $f(n) = 2n^2 + 3n + 1$. Sabemos que $f(n)$ é $O(n^2)$
- Pergunta: $f(n)$ é $\Omega(n^2)$?

$$2n^2 + 3n + 1 \geq c_1 n^2$$

$$2 + \frac{3}{n} + \frac{1}{n^2} \geq c_1$$

- A inequação é verdadeira para $c_1 = 2, N = 1$
- Logo $f(n)$ é $\Omega(g(n))$ e, portanto, $f(n)$ é $\Theta(g(n))$

Problemas possíveis

- O fato de um algoritmo ter ordem de complexidade menor do que o 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 $O(n^2)$
- Para $n < 10^7$, o tempo de execução de $f(n)$ é maior do que o de $g(n)$
- Apenas para valores maiores ou iguais a 10^7 é que a função $f(n)$ se torna mais eficiente do que a função $g(n)$

Exemplos de complexidade ($n = 10$)

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

[1] Uma instrução por μs

Exemplos de complexidade ($n = 100$)

Classe	Notação	Número de operações	Tempo de execução[1]
constante	$O(1)$	1	1μs
logarítmica	$O(\log n)$	$4,6$	5μs
linear	$O(n)$	100	100μs
$O(n \log n)$	$O(n \log n)$	460	460μs
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

Exemplos de complexidade ($n = 1000$)

Classe	Notação	Número de operações	Tempo de execução[1]
constante	$O(1)$	1	1μs
logarítmica	$O(\log n)$	$6,9$	7μs
linear	$O(n)$	1000	1ms
$O(n \log n)$	$O(n \log n)$	6907	60ms
quadrática	$O(n^2)$	10^6	1s
cúbica	$O(n^3)$	10^9	$16,7$m
exponencial	$O(2^n)$	10^{301}	$\ldots$

[1] Uma instrução por μs

Referências

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.