Capítulo 3

Notação Assintótica

Sabemos que a análise do pior caso de um algoritmo é importante porque fornece uma cota superior para o tempo de execução deste algoritmo. Agora suponha que um algoritmo de ordenação A é tal que $W_A(n) = d.n$ para alguma constante d. Qual dos dois algoritmos é mais rápido, InsertionSort ou A? Isto vai depender das constantes envolvidas, em geral A será mais rápido para algumas entradas, e InsertionSort será mais rápido para outras. No entanto, sabemos que funções quadráticas crescem mais rápido do que funções lineares. Então podemos concluir que *a partir de um certo valor de* n o algoritmo A será **sempre** mais rápido que InsertionSort. Este comentário sugere uma maneira de comparar funções que ignora constantes e argumentos pequenos (valores pequenos para n). Este tipo de comparação é conhecido como "ordem assintótica", ou "taxa de crescimento assintótico"de funções. Como veremos a seguir, a análise assintótica é muito conveniente para analisar algoritmos.

Vejamos algumas definições que nos permitirão maior precisão matemática nas análises a serem feitas:

Definição 3.0.1 (O conjunto $O(g(n))$)**.** *Seja $g(n)$ uma função dos inteiros não-negativos nos reais positivos, isto é, $g : \mathbb{N} \to \mathbb{R}^{>0}$. Então $O(g(n))$ é o conjunto das funções (também dos inteiros não-negativos nos reais positivos) tal que existem uma constante real $c > 0$ e uma constante inteira $n_0 > 0$ satisfazendo a desigualdade $f(n) \leq c \cdot g(n), \forall n \geq n_0$. Alternativamente, $O(g(n)) = \{f(n) : \exists c, n_0$ tais que $0 \leq f(n) \leq c \cdot g(n), \forall n \geq n_0.\}$*

O lema a seguir apresenta uma forma de mostrar que $f(n) \in O(g(n))$ usando limites:

Lema 3.0.2. *Uma função $f(n) \in O(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$, incluindo o caso em que $c = 0$.*

Demonstração. Exercício. □

Aproveitando a análise feita para InsertionSort no pior caso, podemos escrever $W_{IS}(n) \in O(n^2)$, ou simplesmente, $W_{IS}(n) = O(n^2)$. Esta última forma de utilizar a notação assintótica é um abuso de linguagem, já que a igualdade utilizada não é simétrica, mas que será muito conveniente como veremos posteriormente.

Definição 3.0.3 (O conjunto $\Theta(g(n))$)**.** *Seja $g(n)$ uma função dos inteiros não-negativos nos reais positivos. Então $\Theta(g(n))$ é o conjunto das funções (também dos inteiros não-negativos nos reais positivos) tal que existem constantes reais positivas c_1 e c_2, e uma constante inteira $n_0 > 0$ satisfazendo a desigualdade $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \forall n \geq n_0$. Alternativamente, $\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0$ tais que $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \forall n \geq n_0.\}$.*

Exercício 3.0.4. *Mostre que* $\frac{n^2}{2} - 3n \in \Theta(n^2)$.

Exercício 3.0.5. *Mostre que* $6n^3 \notin \Theta(n^2)$.

Exercício 3.0.6. *Mostre que* $W_{IS}(n) \in \Theta(n^2)$.

Intuitivamente, os termos de menor ordem de uma função assintoticamente positiva podem ser ignorados na determinação da cota superior porque são insignificantes para valores grandes do parâmetro n. Assim, quando n é grande qualquer porção ou fração do termo de maior ordem é suficiente para dominar os termos de menor ordem.

Como qualquer constante pode ser vista como um polinômio de grau 0, podemos representar funções constantes como $\Theta(n^0)$, ou simplesmente, $\Theta(1)$.

Observe que $\Theta(g(n)) \subseteq O(g(n))$, para qualquer função g assintoticamente não negativa. Utilizando a notação O, podemos descrever o tempo de execução de um algoritmo a partir da simples inspeção da estrutura geral do algoritmo. Como a notação O descreve uma cota superior, quando utilizada para o tempo de execução no pior caso, obtemos uma cota superior para o algoritmo em todas as entradas. Portanto, a cota $O(n^2)$ para InsertionSort no pior caso, também se aplica para todas as entradas possíveis. No entanto, a cota $\Theta(n^2)$ para o pior caso de InsertionSort não implica em uma cota $\Theta(n^2)$ para o tempo de execução de InsertionSort para todas as entradas. De fato, vimos que para entradas já ordenadas, InsertionSort tem cota $\Theta(n)$.

Definição 3.0.7 (O conjunto $\Omega(g(n))$)**.** *Seja $g(n)$ uma função dos inteiros não-negativos nos reais positivos. Então $\Omega(g(n)$ é o conjunto das funções (também dos inteiros não-negativos nos reais positivos) tal que existem uma constante real $c > 0$ e uma constante inteira $n_0 > 0$ satisfazendo a desigualdade $c \cdot g(n) \leq f(n), \forall n \geq n_0$. Alternativamente, $\Omega(g(n)) = \{f(n) : \exists c, n_0$ tais que $0 \leq c \cdot g(n) \leq f(n), \forall n \geq n_0.\}$.*

Quando dizemos que o tempo de execução de um algoritmo é $\Omega(g(n))$, queremos dizer que independentemente da entrada de tamanho n, o tempo de execução desta entrada é pelo menos uma constante multiplicada por $g(n)$ para n suficientemente grande. Ou seja, estamos fornecendo um cota inferior no melhor caso. Por exemplo, no melhor caso, o algoritmo InsertionSort é $\Omega(n)$, o que implica que o tempo de execução de InsertionSort é $\Omega(n)$. Portanto, o tempo de execução do algoritmo InsertionSort está em $\Omega(n)$ e $O(n^2)$.

Normalmente escreveremos $f(n) = \Theta(g(n))$ ao invés de $f(n) \in \Theta(g(n))$. Isto será conveniente porque podemos usar a notação assintótica em (des)igualdades. Por exemplo, podemos escrever $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$. Como devemos interpretar esta equação? Neste caso, $\Theta(n)$ deve ser vista como uma função anônima. Assim, a equação $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ significa $2n^2 + 3n + 1 = 2n^2 + f(n)$, onde $f(n)$ é alguma função em $\Theta(n)$.

O número de funções anônimas em uma expressão é igual ao número de vezes que a notação assintótica aparece: por exemplo, na expressão $\sum_{i=1}^{n} O(i)$ contém apenas uma função anônima (a função que tem parâmetro i), e portanto esta expressão não é o mesmo que $O(1) + O(2) + \ldots + O(n)$ (que não possui uma interpretação clara).

3.1 Propriedades da Notação Assintótica

A notação assintótica também pode aparecer do lado esquerdo de uma equação: $2n^2 + \Theta(n) = \Theta(n^2)$. Neste caso, independentemente da forma como as funções anônimas são escolhidas do lado esquerdo da equação, existe uma forma de escolher funções anônimas do lado direito da equação de forma que a equação se verifique. No caso do exemplo acima, temos que para qualquer $f(n) \in \Theta(n)$, existe uma função $g(n) \in \Theta(n^2)$ tal que $2n^2 + f(n) = g(n), \forall n$.

Equações também podem ser encadeadas como em $2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$, e podem ser interpretadas separadamente de acordo com as regras anteriores. Assim, a primeira equação nos diz que existe alguma função $f(n) \in \Theta(n)$ para a qual a equação se verifica para todo n. A segunda equação nos diz que para toda função $g(n) \in \Theta(n)$, existe uma função $h(n) \in \Theta(n^2)$ tal que a equação se verifica para todo n. Este encadeamento é transitivo, ou seja, podemos concluir que $2n^2 + 3n + 1 = \Theta(n^2)$.

Teorema 3.1.1. *Dadas funções $f(n)$ e $g(n)$, temos que $f(n) = \Theta(g(n))$ se, e somente se, $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.*

Lema 3.1.2. *Uma função $f(n) = \Omega(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$, incluindo o caso em que o limite é igual a ∞.*

Lema 3.1.3. *Uma função $f(n) = \Theta(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, para alguma constante $0 < c < \infty$.*

Definição 3.1.4 (O conjunto $o(g(n))$)**.** *Seja $g(n)$ uma função dos inteiros não-negativos nos reais positivos. Definimos, $o(g(n)) = \{f(n)$: para qualquer constante positiva c, existe uma constante positiva n_0 tal que $0 \leq f(n) < c \cdot g(n), \forall n \geq n_0.\}$*

Lema 3.1.5. *Uma função $f(n) = o(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.*

Definição 3.1.6 (O conjunto $\omega(g(n))$)**.** *Seja $g(n)$ uma função dos inteiros não-negativos nos reais positivos. Definimos, $\omega(g(n)) = \{f(n)$: para qualquer constante positiva c, existe uma constante positiva n_0 tal que $0 \leq c \cdot g(n) < f(n), \forall n \geq n_0.\}$*

Lema 3.1.7. *Uma função $f(n) = \omega(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, se este limite existir.*

Lema 3.1.8. *Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$ então $f(n) = O(h(n))$, ou seja O é transitiva. Também são transitivos Ω, Θ, o e ω.*

Lema 3.1.9.

1. *$f(n) = O(g(n))$ se, e somente se $g(n) = \Omega(f(n))$.*
2. *Se $f(n) = \Theta(g(n))$ então $g(n) = \Theta(f(n))$.*
3. *Θ define uma relação de equivalência sobre as funções. Cada conjunto $\Theta(f(n))$ é uma classe de equivalência que chamamos de classe de complexidade.*
4. *$O(f+g(n)) = O(\max\{f(n), g(n)\})$. Equações análogas valem para Ω e Θ. Estas equações são úteis na análise de algoritmos complexos onde $f(n)$ e $g(n)$ podem descrever o trabalho feito em diferentes partes do algoritmo.*

Teorema 3.1.10. $\lg n = o(n^\alpha), \forall \alpha > 0$. *Ou seja, a função logaritmo cresce mais lentamente do que qualquer potência de n (incluindo potências fracionárias)*

Teorema 3.1.11. $n^k = o(2^n), \forall k > 0$. *Ou seja, potências de n crescem mais lentamente que a função exponencial 2^n. Mais ainda, potências de n crescem mais lentamente do que qualquer função exponencial $c^n, c > 1$.*

Classes básicas de eficiência assintótica:

Classe	**Nome**
1	constante
$\log n$	logarítmica
n	linear
$n.\log n$	linearítmica
n^2	quadrática
n^3	cúbica
n^k	polinomial ($k \geq 1$ e finito)
a^n	exponencial ($a \geq 2$)
$n!$	fatorial