Análise de Algoritmos – Tópico 3

Prof. Dr. Juliano Henrique Foleiss

1 Notação Assintótica

Embora seja possível determinar de maneira razoavelmente precisa a quantidade de computações necessárias, como fizemos com o `InsertionSort`, o que nos interessa realmente é a taxa de crescimento da quantidade de computações necessárias em função do tamanho do problema, também chamada de ordem de crescimento. A razão é que nossa preocupação é com valores grandes para n. Portanto, a comparação de eficiência entre algoritmos para resolver determinado problema é medida com entradas grandes, uma vez que são mais interessantes na prática.

Assim, podemos desconsiderar os termos de menor ordem no caso do `InsertionSort`, sobrando apenas o $a.n^2$, uma vez que os termos de menor ordem são insignificantes para entradas grandes. Além disso, podemos ignorar a constante do termo de maior ordem uma vez que este é menos significante que a taxa de crescimento quando determinamos eficiência computacional para entradas grandes. No caso do `InsertionSort`, ao ignorar os termos menos significativos e a constante do termo restante da análise do pior caso sobra o fator n^2. Portanto, dizemos que o `InsertionSort` leva tempo proporcional ao quadrado do tamanho da entrada em seu pior caso.

Consideramos que um algoritmo é mais eficiente que o outro se seu tempo no pior caso possui ordem de crescimento mais lenta que o outro. Em função dos fatores constantes e termos de ordem mais baixa, um algoritmo com taxa de crescimento maior pode superar algoritmos com taxas menores de crescimento com entradas pequenas. No entanto, para entradas grandes, um algoritmo n^2 no pior caso sempre vai executar mais rápido que um algoritmo n^3.

Quando olhamos para tamanhos de entrada grandes de forma que somente a ordem de crescimento do tempo de execução é relevante na execução de um algoritmo, estamos estudando o `comportamento assintótico` dos algoritmos. Ou seja, estamos preocupados com como o tempo de execução aumenta com o tamanho da entrada no `limite`, conforme o tamanho da entrada cresce. Normalmente um algoritmo assintoticamente mais eficiente sempre representa a melhor escolha para todas as entradas, exceto para entradas muito pequenas.

Nesta disciplina utilizaremos a notação assintótica para categorizar o tempo de execução de algoritmos. No entanto esta notação pode expressar outras características de algoritmos, como o espaço (memória) que utilizam.

1.1 Notação O

As notações utilizadas para descrever o tempo de execução de um algoritmo são definidas em termos de funções cujo domínio é o conjunto dos números naturais $\mathbb{N}$.

A notação O é usada para representar o limite assintótico superior de um algoritmo. Para dada função $g(n)$, $O(g(n))$ é dado pelo conjunto:

Definição 1 (Notação O)**.**

$O(g(n)) = \{f(n) : \text{existem constantes positivas } n_0 \text{ e } c \text{ tal que } f(n) \leq cg(n), \forall n \,|\, n \geq n_0\}$

Ou seja, a notação O nos dá o limite sperior para uma função, dentro de um fator constante. Assim, para todo n a partir de n_0, $f(n)$ é igual ou menor ao valor de $cg(n)$. A notação $f(n) = O(g(n))$ indica que a função $f(n)$ pertence ao conjunto $O(g(n))$, $f(n) \in O(g(n))$. A Figura 1 mostra a representação gráfica da notação O.

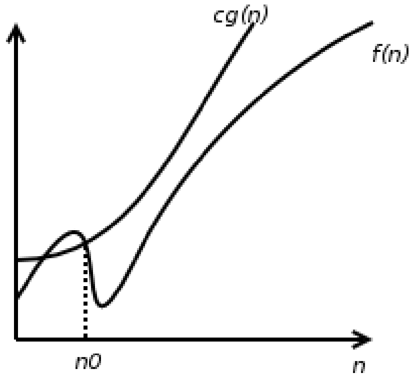

Figura 1: $f(n) = O(g(n))$

Exemplos

I. $T(n) = n^2 + 2n$, $T(n) = O(n^2)$?

Por definição $T(n)$ é $O(n^2)$ se $T(n) \leq cn^2$ para algum $n \geq n_0$.

$$n^2 + 2n \leq cn^2$$
$$1 + \frac{2}{n} \leq c \mid \text{ p/ } n = 1$$
$$1 + 2 \leq c$$
$$c \geq 3$$

Assim, $T(n) = O(n^2)$ considerando $n_0 = 1$ e $c = 3$.

II. $T(n) = 3n + 2$, $T(n) = O(n^3)$?

Por definição $T(n)$ é $O(n^3)$ se $T(n) \leq cn^3$ para algum $n \geq n_0$.

$$3n + 2 \leq cn^3$$
$$\frac{3}{n^2} + \frac{2}{n^3} \leq c \mid \text{ p/ } n = 1$$
$$3 + 2 \leq c$$
$$c \geq 5$$

Assim, $T(n) = O(n^3)$ considerando $n_0 = 1$ e $c = 5$.

III. $T(n) = n^3 + 20n + 1$, $T(n) = O(n^2)$?

Por definição $T(n)$ é $O(n^2)$ se $T(n) \leq cn^2$ para algum $n \geq n_0$.

$$n^3 + 20n + 1 \leq cn^2$$
$$n + \frac{20}{n} + \frac{1}{n^2} \leq c$$
$$c \geq n + \frac{20}{n} + \frac{1}{n^2}$$

Como o lado direito da inequação cresce infinitamente, não existe c constante. Portanto, $T(n) \neq O(n^2)$.

Como a notação O descreve o limite superior, quando utilizamos para limitar o tempo de execução do pior caso de um algoritmo, temos um limite no tempo de execução para todas as entradas possíveis.

1.2 Notação Ω

Assim como a notação O provê um limite assintótico superior, a notação Ω provê um limite assintótico inferior. Para dada função $g(n)$, $\Omega(g(n))$ é o conjunto:

Definição 2 (Notação Ω)**.**

$\Omega(g(n)) = \{f :$ existem constantes positivas c e n_0 tal que $cg(n) \leq f(n), \forall n \,|\, n \geq n_0\}$

Ou seja, para todos os valores n iguais ou maiores que n_0 o valor de $f(n)$ é maior ou igual $cg(n)$. A notação $f(n) = \Omega(g(n))$ indica que a função $f(n)$ pertence ao conjunto $\Omega(g(n))$, $f(n) \in \Omega(g(n))$. A Figura 2 mostra a representação gráfica da notação Ω.

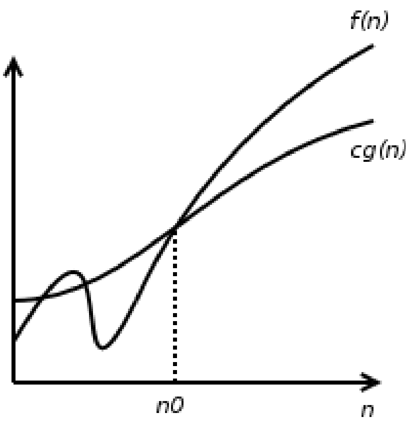

Figura 2: $f(n) = \Omega(g(n))$

Exemplo

I. $T(n) = n^3 + 20n$, $T(n) = \Omega(n^2)$?

Por definição $T(n)$ é $\Omega(n^2)$ se $cn^2 \leq T(n)$ para algum $n \geq n_0$.

$$cn^2 \leq n^3 + 20n$$
$$c \leq n + \frac{20}{n} \mid \text{ p/ } n = 1$$
$$c \leq 1 + \frac{20}{1}$$
$$c \leq 21$$

Assim, $T(n) = \Omega(n^2)$ considerando $n_0 = 1$ e $c = 21$.

1.3 Notação Θ

A notação Θ é usada para representar o limite assintótico ajustado de um algoritmo. Para dada função $g(n)$, $\Theta(g(n))$ é um conjunto dado por:

Definição 3 (Notação Θ)**.**

$\Theta(g(n)) = \{f :$ existem constantes positivas c_1 e c_2 e n_0 tal que $c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \,|\, n \geq n_0\}$

Uma função $f(n)$ pertence ao conjunto $\Theta(g(n))$ se existem contantes positivas c_1 e c_2 tal que $f(n)$ fica entre $c_1g(n)$ e $c_2g(n)$ para n suficientemente grande. $f(n) = \Theta(g(n))$ indica que $f(n)$ pertence ao conjunto $\Theta(g(n))$, $f(n) \in \Theta(g(n))$. A Figura 3 mostra a representação gráfica da notação Θ.

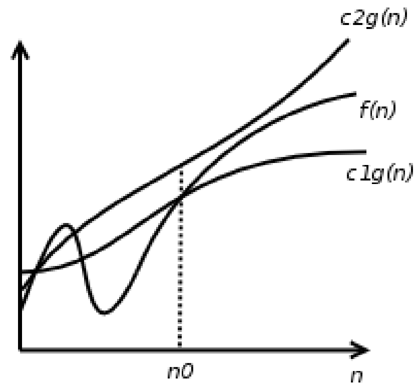

Figura 3: $f(n) = \Theta(g(n))$

Exemplo

I. $T(n) = \frac{1}{2}n^2 - 3n$, $T(n) = \Theta(n^2)$?

Por definição, $\frac{1}{2}n^2 - 3n = \Theta(n^2)$ se existem c_1, c_2 e n_0 tais que $c_1n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2n^2$ para todo n a partir de n_0. Assim,

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

$$\begin{gathered} \text{p/ o lado direito} \\ \frac{1}{2} - \frac{3}{n} \leq c_2 \mid \text{ p/ } n = 1 \\ \frac{1}{2} - \frac{3}{1} \leq c_2 \\ c_2 \geq \frac{5}{2} \end{gathered}$$

$$\begin{gathered} \text{p/ o lado esquerdo} \\ c_1 \leq \frac{1}{2} - \frac{3}{n} \mid \text{ p/ } n = 7 \\ c_1 \leq \frac{1}{2} - \frac{3}{7} \\ c_1 \leq \frac{1}{14} \end{gathered}$$

Assim, considerando $n_0 = 7$, $c_1 = \frac{1}{14}$ e $c_2 = \frac{5}{2}$ é possível concluir que $\frac{1}{2}n^2 - 3n = \Theta(n^2)$.

Note que a definição da notação Θ é uma combinação direta das definições da notação Ω e O. Como consequência, temos:

Teorema 1. *Para duas funções quaisquer $f(n)$ e $g(n)$, $f(n) = \Theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.*

Intuitivamente, os termos de baixa ordem podem ser ignorados ao determinar limites assintóticos ajustados pois são insignificantes para n grande. Quando n é grande, mesmo uma pequena fração do termo de ordem maior é suficiente para dominar os termos de ordem menor. Assim, atribuindo c_1 para um valor ligeiramente menor que o coeficiente do termo de maior ordem e atribuindo c_2 para um valor ligeiramente maior permite que as inequações da definição de Θ sejam satisfeitas. Além disto, o coeficiente do termo de maior ordem também pode ser ignorado, já que mudam c_1 e c_2 por um fator constante igual o coeficiente. O Teorema 2 assegura que qualquer polinômio tem seu limite assintótico ajustado ao termo de maior ordem.

Teorema 2. *Para qualquer polinômio $p(n) = \sum_i^d a_i n^i$ onde a_i são constantes e $a_d > 0$, $p(n) = \Theta(n^d)$.*

Uma consequência significativa é que como qualquer constante é polinômio de grau 0, funções constantes são indicadas por $\Theta(n^0)$, ou $\Theta(1)$. $\Theta(1)$ é considerado abuso de notação uma vez que a expressão não especifica qual variável está tendendo ao infinito.

Em vários casos o custo de um algoritmo é a soma dos custos de duas seções em sequência. Seja o custo da primeira seção $f(n)$ e o custo da segunda $g(n)$. O Teorema 3 a seguir assegura que o custo final da sequência é limitado assintoticamente pela função com maior custo entre $f(n)$ e $g(n)$.

Teorema 3. *Para duas funções quaisquer $f(n)$ e $g(n)$, $\Theta(f(n) + g(n)) = \max(f(n), g(n))$.*

Como exercício, use o Teorema 3 para provar o Teorema 2.

1.4 Teste do Limite

Sejam duas funções $f(n)$ e $g(n)$ positivas e monótonas crescentes. Para saber a relação assintótica entre $f(n)$ e $g(n)$ podemos usar o teste do limite.

O limite:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$

tem uma entre três soluções possíveis. Cada uma nos diz algo importante sobre a relação assintótica entre $f(n)$ e $g(n)$:

1. Se o limite acima é **0** então a função $g(n)$ assume valores muito maiores que $f(n)$ quando n é grande. Intuitivamente, isso indica que $f(n)$ é limitada superiormente por $g(n)$. Portanto, podemos afirmar que $f(n) = O(g(n))$.

2. Se o limite acima é ∞ então a função $f(n)$ assume valores muito maiores que $g(n)$ quando n é grande. Intuitivamente, isso indica que $f(n)$ é limitada inferiormente por $g(n)$. Portanto, podemos afirmar que $f(n) = \Omega(g(n))$.

3. Se o limite acima é uma **constante**, então a função $f(n)$ assume valores relativamente próximos de $g(n)$, mesmo para n grande, que diferem apenas no máximo por um coeficiente que não depende de n. Intuitivamente, isso indica que o comportamento assintótico de $f(n)$ e $g(n)$ é o mesmo. Portanto, podemos afirmar que $f(n) = \Theta(g(n))$.

Exemplos

I. Você desenvolveu dois algoritmos para resolver o mesmo problema e avaliou seu custo computacional. O algoritmo 1 tem custo $f(n) = n^2 + 20n + 10$ e o algoritmo 2 tem custo $g(n) = 2n^2 + 5n$. Qual dos dois tem o menor custo computacional em termos de análise assintótica?

Solução

Pelo teste do limite,

$$\begin{aligned}\lim_{n\to\infty}\frac{n^2+20n+10}{2n^2+5n} &=\\ \lim_{n\to\infty}\frac{2n+20}{4n+5} &= \text{(pela regra de L'Hôpital)}\\ \lim_{n\to\infty}\frac{2}{4} &= \text{(pela regra de L'Hôpital)}\\ \lim_{n\to\infty}\frac{2}{4} &= \frac{2}{4} = 0.5\end{aligned}$$

Como o limite tende a uma constante, o teste do limite indica que $f(n) = \Theta(g(n))$. Portanto, os dois algoritmos tem o mesmo custo computacional em termos de análise assintótica.

II. Você desenvolveu dois algoritmos para resolver o mesmo problema e avaliou seu custo computacional. O algoritmo 1 tem custo $f(n) = 2n$ e o algoritmo 2 tem custo $g(n) = 5\lg(n)$. Qual dos dois tem o menor custo computacional em termos de análise assintótica?

Solução

Pelo teste do limite,

$$\begin{aligned}\lim_{n\to\infty}\frac{2n}{5\lg(n)} &=\\ \lim_{n\to\infty}\frac{2}{\frac{5}{n}} &= \text{(pela regra de L'Hôpital)}\\ \lim_{n\to\infty}\frac{2}{5}n &= \frac{2}{5}\lim_{n\to\infty} n = \infty\end{aligned}$$

Como o limite tende a ∞, o teste do limite indica que $f(n) = \Omega(g(n))$. Portanto o custo algoritmo 2 pra n grande é menor que o custo do algoritmo 1. Assim, o algoritmo 2 é mais eficiente.

1.5 Notação assintótica em equações em inequações

Como já visto anteriormente, $f(n) = \Theta(f(n))$, ou seja, a notação assintótica é usada do lado direito e o lado esquerdo apenas nomeia uma função qualquer que tenha comportamento $f(n)$.

A notação assintótica também pode ser usada em equações e inequações como termos que representam uma função anônima que não nos preocupamos em especificar. Neste caso a notação assintótica representa uma função qualquer do conjunto de funções representado na notação. Por exemplo, a equação $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ representa $2n^2 + 3n + 1 = 2n^2 + f(n)$ onde $f(n)$ é alguma função do conjunto $\Theta(n)$. Neste caso, $f(n) = 3n + 1$, que é $\Theta(n)$.

Como já vimos anteriormente, a notação assintótica é usada para eliminar detalhes desnecessários à interpretação da taxa de crescimento de uma função. Por exemplo, $T(n) = 2n^2 + 3n$ pode ser escrita apenas como $T(n) = \Theta(n^2)$. Como estamos preocupados com o comportamento assintótico de $T(n)$ não é necessário especificar todos os termos de baixa ordem de maneira exata, pois todos estão implícitos no termo $\Theta(n)$.

Em alguns casos a notação assintótica aparece no lado esquerdo da equação como em $2n^2 + \Theta(n) = \Theta(n^2)$. Neste caso, utilizamos a seguinte regra: não importa como as funções anônimas são escolhidas no lado esquerdo da equação, existe uma maneira de escolher funções do lado direito da equação de maneira que a equação é válida. Em outras palavras, o lado direito da equação provê um nível mais amplo de detalhe que o lado esquerdo.

$$\begin{aligned}2n^2 + 3n + 1 &= 2n^2 + \Theta(n)\\ &= \Theta(n^2)\end{aligned}$$

1.6 Comparando funções assintóticas

Transitividade

Se $f(n) = \Theta(g(n))$ e $g(n) = \Theta(h(n))$ então $f(n) = \Theta(h(n))$
Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$ então $f(n) = O(h(n))$
Se $f(n) = \Omega(g(n))$ e $g(n) = \Omega(h(n))$ então $f(n) = \Omega(h(n))$

Reflexão

$f(n) = \Theta(f(n))$
$f(n) = O(f(n))$
$f(n) = \Omega(f(n))$

Simetria

$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$

Simetria Transposta

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

Comparando

$f(n) = O(g(n))$ é análogo a $a \leq b$
$f(n) = \Omega(g(n))$ é análogo a $a \geq b$
$f(n) = \Theta(g(n))$ é análogo a $a = b$

Bibliografia

[CRLS] CORMEN, T. H. et al. Algoritmos: Teoria e Prática. Elsevier, 2012. 3a Ed. Capítulo 3 (Crescimento de Funções), Seção 3.1