Notação assintótica

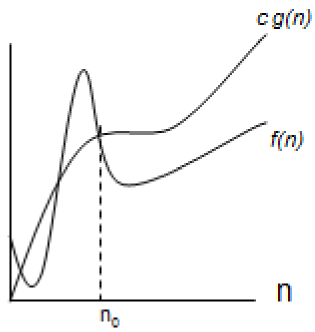
Notação utilizada para classificar a função de tempo de execução de um algoritmo por proximidade (classificação assintótica).

Notação *O*

Define um limite assintótico superior da função de tempo de execução do algoritmo (f(n)).

Dizemos que $f(n) \in O(g(n))$ (ou $f(n) = O(g(n))$) se existe constantes positivas *c* e n_0 tais que

$$0 \leq f(n) \leq c \cdot g(n), \quad \forall n > n_0$$



$$f(n) \in O(g(n))$$

Exercício:

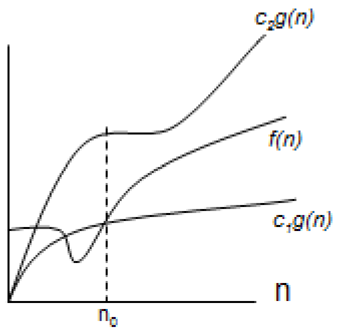
1. $f(n) = 7n + 3$
   $f(n) \in O(n)$?
2. $f(n) = 2n^3 + n^2 + 100$
   $f(n) \in O(n^3)$?
3. $2^n \in O(2^{n+1})$?

Notação θ

Define limites assintóticos superior e inferior (limite "justo") da função de tempo de execução do algoritmo (f(n)).

Dizemos que $f(n) \in \theta(g(n))$ se existe constantes positivas c_1, c_2 e n_0 tais que

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \quad \forall n > n_0$$



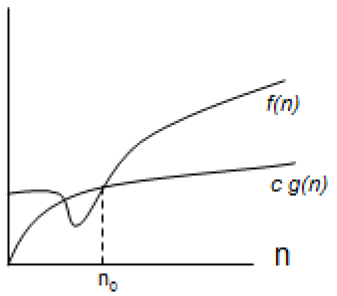
$$f(n) \in \theta(g(n))$$

Notação Ω

Define limite assintótico inferior da função de tempo de execução do algoritmo (f(n)).

Dizemos que $f(n) \in \Omega(g(n))$ se existe constantes positivas c e n_0 tais que

$$0 \leq c \cdot g(n) \leq f(n), \quad \forall n > n_0$$



$$f(n) \in \Omega(g(n))$$

Definição formal de limite

$$\lim_{n\to\infty} f(n) = k \iff \forall \epsilon > 0, \exists n_0 \ tal\ que$$

$$\|f(n) - k\| \leq \epsilon, \qquad \forall n > n_0$$

Se $f(n) - k \geq 0 \ \rightarrow \ \|f(n) - k\| = f(n) - k$

$$f(n) - k \leq \epsilon$$

$$f(n) \leq k + \epsilon.$$

Se $f(n) - k < 0 \ \rightarrow \ \|f(n) - k\| = -(f(n) - k)$

$$k - f(n) \leq \epsilon$$

$$k - \epsilon \leq f(n).$$

Figura

Uso do conceito de limite para a analise assintótica

Proposição 1:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = k \ > 0 \Rightarrow \ O\big(f(n)\big) = O\big(g(n)\big)$$

$$(k < \infty)$$

Prova:

Da definição formal de limite

$$\forall \epsilon > 0, \exists n_0 \ tal\ que \quad k - \epsilon \leq \frac{f(n)}{g(n)} \leq k + \epsilon, \ \ \forall n \geq n_0$$

Da segunda parte da desigualdade temos:

$$f(n) \leq (k + \epsilon) \cdot g(n), \ \ \forall n \geq n_0$$

Fazendo $c = k + \epsilon$ temos que $\exists c, n_0$ tal que

$$f(n) \leq c \cdot g(n), \ \ \forall n \geq n_0$$

Assim da definição de complexidade assintótica:

$$f(n) \in O\big(g(n)\big) \qquad (1)$$

Da primeira parte da desigualdade temos:

$$g(n) \leq \left(\frac{1}{k-\epsilon}\right) f(n), \ \ \forall n \geq n_0 \quad (onde\ \epsilon < k)$$

Fazendo $c' = \frac{1}{k-\epsilon} > 0$ temos

$$\exists c1, n_0 \ \ tal\ que \ \ g(n) \leq c' \cdot f(n), \quad \forall n \geq n_0$$

Agora da definição de complexidade assintótica temos que:

$$g(n) \in O\big(f(n)\big) \qquad (2)$$

Assim da proposição 1 e de (1) e (2) temos:

$$f(n) \in O\big(g(n)\big)\ e \qquad \Rightarrow \ \ O(f(n) = O(g(n))$$

$$g(n) \in O\big(f(n)\big)$$

Logo

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

Pergunta:

1. $O(\log n) = O(\ln n)$?

$$\lim_{n\to\infty} \frac{\log n}{\ln n} \ = \ \lim_{n\to\infty} \frac{\ln n/\ln 2}{\ln n}$$

$$= \lim_{n\to\infty} \frac{1}{\ln 2} > 0 \quad (constante)$$

Logo $O(\log n) = O(\ln n)$.

2. $O\big(\sqrt{n}\big) = O\big(\sqrt[3]{n}\big)$?

Proposição 2:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0 \quad \Rightarrow \quad O\big(f(n)\big) \subset O\big(g(n)\big)$$

Prova:

$$\forall \epsilon > 0, \exists n_0 \; tal\ que \quad -\epsilon \leq \frac{f(n)}{g(n)} \leq +\epsilon, \;\; \forall n \geq n_0$$

Da segunda parte da desigualdade temos:

$$f(n) \leq (+\epsilon) \cdot g(n), \;\; \forall n \geq n_0$$

Fazendo $c = +\epsilon$ temos que $\exists c, n_0$ tal que

$$f(n) \leq c \cdot g(n), \;\; \forall n \geq n_0$$

Logo $\quad f(n) \in O\big(g(n)\big) \quad$ (1)

Da primeira parte da desigualdade temos:

$$g(n) \geq \left(\frac{1}{\epsilon}\right) f(n), \;\; \forall n \geq n_0 \quad (\forall \epsilon > 0) \quad (*)$$

Suponha por absurdo que $g(n) \in O\big(f(n)\big)$. Assim, da definição de complexidade assintótica temos $\exists c'$, $n_0{}'$ tal que:

$$g(n) \leq c' \cdot f(n), \;\; \forall n \geq n_0' \qquad (**)$$

Fazendo $\overline{n_0} = \max\{n_0\ , n_0'\}$ temos de $(*)$ e $(**)$ que

$$\left(\frac{1}{\epsilon}\right) \cdot f(n) \leq g(n) \leq c' \cdot f(n), \;\; \forall \epsilon > 0, \forall n \geq \overline{n_0}$$

$$\left(\frac{1}{\epsilon}\right) \leq c' \quad (+)$$

Chegamos portanto em um absurdo, pois para $\epsilon < \frac{1}{c\prime}$ a desigualdade (+) não se aplica.

Portanto $g(n) \notin O(f(n))$

Pela proposição 2 temos:

$$f(n) \in O\big(g(n)\big)\ e \qquad \Rightarrow \quad O(f(n) \subset O(g(n))$$

$$g(n) \notin O\big(f(n)\big)$$

Logo

$$O(f(n) \subset O(g(n))$$

Pergunta:

a) $O(2^n) \subset O(3^n)$?

$$\lim \frac{2^n}{3^n} \quad = \quad \lim \left(\frac{2}{3}\right)^n \ = \ 0$$

Logo $\quad O(2^n) \subset O(3^n)$

b) $O\big(2^{n/2}\big) \subset O(2^n)$?

Algoritmo de Hanoi

Hanoi (n, A, C, B)

Se n=1 então

C←A; (move de A para C)

Senão

Hanoi(n-1, A, B, C);

C←A;

Hanoi(n-1, B, C, A)

Fim

Figura Hanoi

Complexidade algoritmo de Hanoi

$$\begin{cases} T(1) = 1 \\ T(n) = 1 + 2 \cdot (T(n-1)) \quad para \quad n \geq 2 \end{cases}$$

$T(n) = 2(\ 2(T(n-2) + 1)) + 1$

$= 2^2\big(T(n-2)\big) + 2 + 1)$ (passo 2)

$= 2^2(2(T(n-3) + 1)) + 2 + 1)$

$= 2^3\big(T(n-3)\big) + 4 + 2 + 1)$ (passo 3)

Após k passos:

$= 2^k\big(T(n-k)\big) + 2^{(k-1)} + .. + 2^2 + 2 + 1)$ (passo k)

e $n - k = 1$ (passos para atingir a base)

$= 2^{n-1} \cdot T(1) + 2^{(n-2)} + .. + 2^2 + 2 + 1)$

$T(n) = \sum_{i=0}^{n-1} 2^{\,i} \quad = \quad 2^n - 1$

Resolvendo a série:

$$S = \sum_{i=0}^{n} a^i * (a-1) \qquad para \;\; a \neq 1$$

$$S = 1 + a + a^2 + .. + a^n * (a-1)$$

$$(a-1)S = a + a^2 + a^3 + .. + a^n + a^{n+1}$$

$$-a - a^2 - a^3 - .. - a^n - 1$$

$$(a-1)S = a^{n+1} - 1$$

$$S = \frac{a^{n+1}-1}{a-1}$$

Logo

$$T(n) = \sum_{i=0}^{n-1} 2^{\,i} = \frac{2^n - 1}{2-1} \;\Rightarrow\; 2^n - 1$$

Complexidade MergeSort

$$\begin{cases} T(1) = 0 \\ T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \quad + \quad \underbrace{c \cdot n}_{merge} \qquad para \quad n \geq 2 \end{cases}$$

1. Para n potência de 2:

   $T(n) = 2(T(\frac{n}{2})) + c \cdot n$ (passo 1)

   $= 2(2(T(\frac{n}{4})) + c \cdot n + c \cdot n$

   $= 2^2 \left(T(\frac{n}{2^2})\right) + 2c \cdot n$ (passo 2)

   $= 2\left(2^2 \left(T(\frac{n}{2^3})\right)\right) + 2c \cdot n + c \cdot n$

   $= 2^3 \left(T(\frac{n}{2^3})\right) + 3c \cdot n$ (passo 3)

   Após k passos:

   $= 2^k \left(T(\frac{n}{2^k})\right) + kc \cdot n$

   Ao final esperamos que:

   $\frac{n}{2^k} = 1 \quad \Rightarrow \quad n = 2^k \quad \Rightarrow \quad k = \log n$ (passos para atingir a base)

   Logo:

   $= 2^{\log n} \cdot T(1) + c \cdot n \log n$

   $= n + c \cdot n \log n$

   $T(n) = c \cdot n \log n$

2. Para n qualquer:

   Sabemos que n está entre duas potências de 2, logo:

   $2^{k-1} \leq n \leq 2^k$ (*)

   Da primeira desigualdade de (*) temos:

   $2^{k-1} \leq n \quad \Rightarrow \quad k \leq \log n + 1$

   Da segunda parte de (*) temos:

   $n \leq 2^k$

   $T(n) \leq T(2^k) = c \cdot 2^k \cdot \log 2^k = c \cdot k \cdot 2^k$

   Majorando o valor de k ($k = \log n + 1$) temos:

   $T(n) \leq c \cdot (\log n + 1) \cdot 2^{\log n} \cdot 2$

   $T(n) \leq 2c \cdot n(\log n + 1)$

$$T(n) \leq c' \cdot n \log n + c' \cdot n$$

$$T(n) \in O(n \log n)$$

Complexidade QuickSort

<u>Pior caso:</u>

$$\begin{cases} T(0) = 0 \\ T(1) = 0 \\ T(2) = 1 \\ T(n) = T(0) + T(n-1) \quad + \quad \underbrace{c \cdot n}_{partition} \qquad para \quad n \geq 3 \end{cases}$$

$T(n) = T(n-1) + \ c \cdot n$ (passo 1)

$T(n) = T(n-2) + \ c \cdot n + c(n-1)$ (passo 2)

$T(n) = T(n-3) + \ c \cdot n + c(n-1) \ + \ c(n-2)$ (passo 3)

Após k passos:

$T(n) = T(n-k) + \ c \cdot \sum_{i=(n-k+1)}^{n} i$

Ao final do processo esperamos obter:

$n - k = 2 \ \Rightarrow \ k = n - 2$ (base)

$T(n) = T(2) + \ c \cdot \sum_{i=3}^{n} i = 1 + n \cdot \frac{n+1}{2} - 3$

$$T(n) \in O(n^2)$$

<u>Melhor caso:</u>

$$\begin{cases} T(0) = 0 \\ T(1) = 0 \\ T(2) = 1 \\ T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \quad + \quad \underbrace{c \cdot n}_{partiton} \qquad para \quad n \geq 3 \end{cases}$$

A prova é análoga ao MergeSort