Resumo Teórico - ICC II

Por: João Paulo Domingues dos Santos - Universidade de São Paulo

(1) Identidades Matemáticas:

(1.1) Logaritmos:

$$\log_b a = c \Leftrightarrow b^c = a$$
$$\log_b ac = \log_b a + \log_b c$$
$$\log_b (a/c) = \log_b a - \log_b c$$
$$\log_b a = \frac{\log_c a}{\log_c b}$$
$$\log_b a^n = n \log_b a$$
$$b^{\log_c a} = a^{\log_c b} \Rightarrow b^{\log_b a} = a$$

(1.2) Somas:

$$\sum_{i=1}^{n} i = (n+1)\frac{n}{2} \qquad (a_1 + a_n)\frac{n}{2}, \text{ caso geral da soma da pa.}$$

$$\sum_{i=0}^{n} a^i = \frac{a^n.a - 1}{a - 1} = \frac{1 - a^{n+1}}{1 - a} \qquad \left(\frac{a_n q - a_1}{q - 1}\right), \text{ caso geral da soma da pg.}$$

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1 - a} \Leftrightarrow |a| < 1 \qquad \left(\frac{a_1}{1 - q}\right), \text{ caso geral da soma infinita da pg.}$$

(1.3) Indução Finita:

Seja $P(n)$, $n \in \mathbb{N}$, uma proposição qualquer. Para provar $P(n)$, procede-se como segue:

(1.3.1) Fraca:

(i) Provar que vale $P(n_0)$, $n_0 < n$; (Caso base)
(ii) Supor que $P(k)$ vale, para um certo k; (Hipótese de indução)
(iii) Provar que vale $P(k+1)$.

(1.3.2) Forte:
(i) Provar que vale $P(n_0)$, $n_0 < n$; (Caso base)
(ii) Supor vale $P(n) \quad \forall \quad n \in \mathbb{I} = [n_0, k]$, $\mathbb{I} \subset \mathbb{N}$
(iii) Provar que vale $P(k+1)$.

---

(2) Complexidade Assintótica:

(2.1)Relacionamento Assintótico:
Dizemos que $g(n)$ domina assintoticamente $f(n)$ se existem c e n_0 tais que:
$|f(n)| \leq c|g(n)| \quad , \quad \forall \quad n \geq n_0 \quad e \quad c, n_0 > 0$

---

obs:
$\wedge$ = conectivo "e"lógico.
$\exists|$ = existe um único.
$\forall$ = para todo
:= tal que

(2.2) Notação *O*:
$g(n)$ cresce no máximo como $f(n) \quad \Rightarrow \quad g(n) \in O(f(n))$

$$O(f(n)) = \{g(n) : \exists \quad c, n_0 > 0 \quad \wedge \quad 0 \leq g(n) \leq cf(n) \quad , \quad \forall \quad n \geq n_0\}$$

Propriedades (2.2.1)

$u(n) \in O(f(n)) \Rightarrow au(n) \in O(f(n)), a$ constante
$u(n) \in O(f(n)) \quad e \quad v(n) \in O(g(n)) \Rightarrow [u(n) + v(n)] \in O(f(n) + g(n))$
$u(n) \in O(f(n)) \quad e \quad v(n) \in O(g(n)) \Rightarrow [u(n).v(n)] \in O(f(n).g(n))$
$u(n) \in O(f(n)) \quad e \quad f(n) \in O(g(n)) \Rightarrow u(n) \in O(g(n))$
Se $f(n)$ é um polinômio de grau d, então $f(n) \in O(n^d)$
$n^k \in O(a^n) \quad \forall\, k > 0\, e\, a > 1$
$log_2\, n^k \in O(log_2\, n) \quad , \forall k > 0$
$(log_2\, n)^{k_1} \in O(n^{k_2}) \quad , \forall\, k_1, k_2 > 0$

(2.3) Notação Ω:
$g(n)$ cresce no mínimo tão lentamente quanto $f(n) \quad \Rightarrow \quad g(n) \in \Omega(f(n))$

$$\Omega(f(n)) = \{g(n) : \exists \quad c, n_0 > 0 \quad \wedge \quad 0 \leq cf(n) \leq g(n) \quad , \quad \forall \quad n \geq n_0\}$$

(2.4) Notação Θ:
$g(n)$ cresce tão rapidamente quanto $f(n)$ (não são necessariamente iguais) $\Rightarrow$ $g(n) \in \Theta(f(n))$

$$\Theta(f(n)) = \{g(n) : \exists \quad c_1, c_2, n_0 > 0 \quad \wedge \quad 0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n) \quad , \quad \forall \quad n \geq n_0\}$$

Ou, de forma equivalente:

$$\Theta(f(n)) = \{g(n) : \ g(n) \in O(f(n)) \quad \wedge \quad g(n) \in \Omega(f(n))\}$$

(2.5) Notação *o*:
$g(n)$ cresce mais lentamente que $f(n)$ $\Rightarrow$ $g(n) \in o(f(n))$

$$o(f(n)) = \{g(n) : \quad \forall c > 0 \quad \exists | \quad n_0 > 0 \quad : \quad 0 \leq g(n) < cf(n) \quad , \quad \forall \quad n > n_0\}$$

Ou, ainda:

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

(2.6) Notação ω:
$g(n)$ cresce mais rapidamente que $f(n)$ $\Rightarrow$ $g(n) \in \omega(f(n))$

$$\omega(f(n)) = \{g(n) : \quad \forall c > 0 \quad \exists | \quad n_0 > 0 \quad : \quad 0 \leq cf(n) < g(n) \quad , \quad \forall \quad n > n_0\}$$

Ou, ainda:

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

---

(3) Recursão:

Técnica de programação na qual uma função pode chamar a si mesma. Pode ser expressa como:

$$\begin{cases} \text{1. Já terminamos? Se sim, mostre os resultados (Caso(s) base)} \\ \text{2. Se não, simplifique o problema em termos de problemas} \\ \text{menores e encaixe os resultados.} \end{cases}$$

---

(4) Divisão e Conquista:

Paradigma de projeto de algoritmos baseada em indução forte, isto é, a hipótese de indução é que sabemos resolver o problema para qualquer valor de entrada entre n_0 e $n-1$. Diferentemente da construção incremental, que supõe que sabemos resolver apenas para n-1.

Dividir em subproblemas $\rightarrow$ $\underbrace{\textit{Conquistar} \text{ os subproblemas}}_{\text{recursivamente}}$ $\rightarrow$ *Combinar* as soluções para compor a solução original.

---

(5) Equações de recorrência:

Função condicional definida em termos de si mesma. Exemplo:

$$f(n) = \begin{cases} c_1 & \text{se } n = n_0 \\ f(n-k) + h(n) & \text{caso contrário} \end{cases}$$

Onde:
$f(n-k)$: Chamada recursiva
n_0: Caso base
c_1: Resultante do caso base
$h(n)$: Função secundária em n (geralmente é constante).

(5.1) Equações de recorrência em divisão e conquista:
Recorrências da forma:

$$T(n) = \begin{cases} c_1 & \text{se } n = n_0 \\ a.T\left(\frac{n}{b}\right) + f(n) & \text{caso contrário} \end{cases}$$

Onde:
n_0: Caso base
c_1: Resultante do caso base
$\frac{n}{b}$: Tamanho dos subproblemas
a: Número de subproblemas
$f(n)$: Custo de etapas anteriores.

(5.2) Teorema mestre:
Maneira eficiente de determinar diretamente a complexidade de algoritmos de divisão e conquista:
Sejam $a \geq 1$, $b \geq 2$ e $T(n) : \mathbb{N} \to \Re$, definida como segue:
$T(n) = a.T\left(\frac{n}{b}\right) + f(n)$, então:

(1) Se $f(n) \in O(n^{(log_b a)-\epsilon})$ para certo $\epsilon > 0$, então $T(n) \in \Theta(n^{log_b a})$.
(2) Se $f(n) \in \Theta(n^{log_b a})$ então $T(n) \in \Theta(n^{log_b a}.log\, n)$.
(3) Se $f(n) \in \Omega(n^{(log_b a)+\epsilon})$ para certo $\epsilon > 0$ e se $af\left(\frac{n}{b}\right) \leq cf(n)$ para $c < 1$ e n suficientemente grande, então $T(n) \in \Theta(f(n))$.

---

(6) Tentativa e Erro:
Adequado quando é necessário usar recursividade para problemas onde é preciso tentar todas as alternativas possíveis. A idéia é decompor o processo em um número finito de subtarefas parciais.

$$\begin{cases} \text{(i) Passos em direção à solução final são tentados e registrados} \\ \text{(ii) Se esses passos não levam à solução final, então eles podem ser retirados do registro.} \end{cases}$$
