Fundamentos de Análise e Deseño de Algoritmos e Estruturas de Dados

---

TEMA 0: PRELIMINARES MATEMÁTICOS.

0.1.- Exponentes.

$x^a \cdot x^b = x^{a+b}$

$x^a / x^b = x^{a-b}$

$(x^a)^b = x^{a \cdot b}$

$x^n + x^n = 2x^n$

$2^n + 2^n = 2^{n+1}$

0.2.- Logaritmos.

$\log_x a = y \Leftrightarrow x^y = a$

$$\frac{\log_c b}{\log_c a} = \log_a b$$

$\log_x a + \log_x b = \log_x (a \cdot b)$

$\log_x a - \log_x b = \log_x (a/b)$

$a \cdot \log_x b = \log_x (b^a)$

Para todo $x > 0$, $\log x < x$

As demostracións logran-se aplicando a definición de algoritmo. Por exemplo:

* demostración $\log_c b / \log_c a = \log_a b$

$\log_c b = x \Leftrightarrow c^x = b$

$\log_c a = y \Leftrightarrow c^y = a$

$\log_a b = z \Leftrightarrow a^z = b$

$\Rightarrow$

$b = a^z = (c^y)^z$

$b = c^x$

$\Rightarrow$

$y \cdot z = x$

$\Rightarrow$

$x / y = z$ c.q.d.

* demostración $\log_x a + \log_x b = \log_x(a \cdot b)$

$\log_x a = w \Leftrightarrow x^w = a$

$\log_x b = y \Leftrightarrow x^y = b$

$\log_x(a \cdot b) = z \Leftrightarrow x^z = a \cdot b$

$\Rightarrow$

$x^z = a \cdot b = x^w \cdot x^y = x^{w+y}$

$\Rightarrow$

$z = w + y$ c.q.d.

0.3.- Séries xeométricas.

$1+2+2^2+2^3+2^4+2^5+...$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$1+a+a^2+a^3+a^4+a^5+...$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a-1}$$

-Nesta última expresión podemos obter a seguinte desigualdade:

$$\text{se } 0 < a < 1 \Rightarrow \sum_{i=0}^{n} a^i \leq \frac{1}{1-a}$$

-E se ademais n tende a infinito temos non xá unha desigualdade senón unha igualdade:

$$\text{se } 0 < a < 1 \Rightarrow \sum_{i=0}^{\infty} a^i = \frac{1}{1-a}$$

-A demostración de esta última expresión é:

$$\text{se } 0 < a < 1 \Rightarrow \sum_{i=0}^{\infty} a^i = 1+a+a^2+a^3+a^4+a^5+... = \text{"S"} \Leftrightarrow$$

$\Leftrightarrow a+a^2+a^3+a^4+a^5+... = aS \Rightarrow$

$$\begin{array}{l} 1+a+a^2+a^3+a^4+a^5+... = S \\ \quad a+a^2+a^3+a^4+a^5+... = aS \\ \hline 1 \qquad\qquad\qquad = S(1-a) \end{array} \quad (*)$$

$\Rightarrow S = 1/(1-a)$ c.q.d.

(*)(a resta só se pode facer para séries de converxência)

$$\sum_{i=1}^{\infty} \frac{i}{2^i} = 2$$

-A demostración de esta última expresión é:

S = 1/2^1 + 2/2^2 + 3/2^3 + 4/2^4 + 5/2^5 + ...
2S = 1 + 1 + 3/2^2 + 4/2^3 + 5/2^4 + ...

2S = 1 + 1 + 3/2^2 + 4/2^3 + 5/2^4 + ...
S = 1/2^1 + 2/2^2 + 3/2^3 + 4/2^4 + ...

S = 1 + 1/2 + 1/2^2 + 1/2^3 + 1/2^4 + ...

entón S = 2 c.q.d.

0.4.- Séries aritméticas.

-Calquer série aritmética se pode avaliar coa seguinte fórmula:

1+2+3+4+5+...

$$\sum_{i=1}^{N} i = \frac{N\cdot(N+1)}{2} \approx \frac{N^2}{2}$$

-Exemplo: calcular o valor da suma 2+5+8+...

2+5+8+...+(3·k-1) =
3·1-1+3·2-1+3·3-1+...+(3·k-1) =
3·(1+2+3+...+k)-(1+1+1+...+1) =

$$3\cdot\frac{k\cdot(k+1)}{2} - k = \frac{3k^2+3k-2k}{2} = \frac{3k^2+k}{2}$$

-Fórmulas de uso menos frecuente son:

$$\sum_{i=1}^{N} i^2 = \frac{N\cdot(N+1)(2N+1)}{6} \approx \frac{N^3}{2}$$

$$\sum_{i=1}^{N} i^{k} \approx \frac{N^{k+1}}{|k+1|} \qquad k \neq -1$$

-Cando **k é -1** usa-se a seguinte expresión:

$$H_N = \sum_{i=1}^{N} \frac{1}{i} \approx \log_e N$$ (H_N "números harmónicos")

-Estas duas fórmulas son só operacións alxebraicas xerais:

$$\sum_{i=1}^{N} f(N) = N \cdot f(N)$$

$$\sum_{i=n0}^{N} f(i) = \sum_{i=1}^{N} f(i) - \sum_{i=1}^{n0-1} f(i)$$

0.5.- Bases teóricas das demostracións.

As duas formas mais comúns de demostrar proposicións na análise de estruturas de dados son a indución e a reducción ao absurdo.

A mellor forma de demostrar que un teorema é falso é dar un contraexemplo.

A reducción ao absurdo consiste en supor que se cumpren as hipóteses do teorema e ao mesmo tempo a negación da conclusión (en outras palabras, que o teorema é falso) e de elas chegar a unha contradicción.

A demostración por indución consiste, primeiro, en comprobar un caso base ou trivial que normalmente involucra ao cero, un, ou algún número natural pequeno; o segundo paso da indución consiste en, suposta certa a afirmación para un número natural calquer n, demostrar que daquela tamén o é para o seguinte, n+1. Asi conclui-se que a propriedade se cumpre para todos -ou casi todos, dependendo de cal sexa o caso base- os números naturais.

Exemplo de indución: demostrar unha propriedade relativa à sucesión de Fibonacci. A función de Fibonacci, da que se obtên os termos da sucesión homónima, asigna un número inteiro a cada inteiro non negativo do seguinte xeito:

Fib : Z → Z
0 → 1
1 → 1
n → Fib (n-1) + Fib (n-2) sendo n>1

Demostrar que Fibonacci (i) < $(5/3)^i$ ∀ i >= 1

"caso base"

i = 1 : Fib (1) = 1 < (5/3)

"hipótese de indución"

supomos certa a afirmación para i = k : Fib (k) < $(5/3)^k$

"paso inductivo"

i = k+1 : Fib (k+1) = Fib (k) + Fib (k-1) pola hipótese de indución: Fib (k) < $(5/3)^k$; Fib (k-1) < $(5/3)^{k-1}$ entón: Fib (k+1) < $(5/3)^k$ + $(5/3)^{k-1}$ = $(5/3)^k$ ·5/3·3/5 + $(5/3)^{k-1}$·$5^2/3^2$·$3^2/5^2$ =
$(5/3)^{k+1}$·3/5 + $(5/3)^{k+1}$·$3^2/5^2$ = $(5/3)^{k+1}$ ·(15+9)/25 do que se ten que Fib (k+1) < 0.96·$(5/3)^{k+1}$ ⇒ Fib (k+1) < $(5/3)^{k+1}$ c.q.d.

Outra das técnicas empregadas no deseño de algoritmos para o manexo de estruturas de dados é a da recorrência (ou recursividade). Un algoritmo dise recursivo cando se define en termos de si mesmo. Dada unha función ou un procedimento recursivo, este debe ter ao menos un caso base, no que a solución é inmediata sen que o subprograma se chame a si mesmo de novo. Ademais é necesária a progresión do algoritmo (que non entre nunha secuência infinita senón que cada vez que se chama a si mesmo, o sistema evolucione cara o caso base, é dicer que teña un final seguro).

Vexamos o que acontece cando se construi unha función recursiva sen dotá-la da ideia de progreso:

```
función mala ( n : inteiro )
     se n = 0
     entón devolver 0
     senón devolver mala ( n div 3 + 1 ) + n-1
     finse
finfunción
```

Nesta función, se se calcula mala (1), como 1 div 3 + 1 é 1, éntra-se nun bucle no que para calcular mala (1) se necesita calcular mala (1) o cal é absurdo.

Asi, poden-se definir três regras necesárias para implementar unha función recursiva:
1º) rexeitamento de parámetros non válidos para a función (por exemplo, non aceptar un número negativo na función que calcula o factorial)
2º) especificación do caso base e da sua solución
3º) verificación de que se cumpre a ideia de progreso

Un exemplo de función recursiva é o seguinte -supomos que temos implementada a función visualizar_díxito-, trata-se de unha función para visualizar inteiros positivos:

```
procedimento visualizar ( n : inteiro )
     se n < 0
     entón error ( “n é negativo” )
     senón
          se n < 10
          entón visualizar_díxito ( n )
          senón visualizar ( n div 10 ) ;
                visualizar_díxito ( n mod 10 )
          finse
     finse
finprocedimento
```

A recorrência e a indución están relacionadas. Pode-se demostrar por indución sobre i o número de díxitos do inteiro positivo n que o procedimento visualizar funciona:

“caso base”

i = 1 : n ten un díxito, entón é menor que dez, daquela visualiza-se n directamente, co que funciona.

“hipótese de indución”

dado un inteiro positivo n de k díxitos, o procedimento visualizar amósao exitosamente.

“paso inductivo”

dado un número n inteiro positivo con k+1 díxitos: realiza-se visualizar (n div 10) que é un número de k díxitos. Pola hipótese de indución, visualiza-se exitosamente n div 10. A continuación visualiza-se o díxito n mod 10.

Nota: ainda que matemàticamente son equivalentes os cálculos n mod 10 e n - (n div 10) * 10, esta segunda expresión é mais eficiente porque a operación mod é mais lenta que a div.

Podemos revisar as regras necesárias para a validez da aplicación da recursividade:
1º) incluir o caso base.
2º) verificar a ideia de progreso.
3º) regra de deseño: supoña-se que todas as chamadas recursivas funcionan.
4º) regra do interese composto: nunca duplicar un traballo resolvendo un problema en chamadas recursivas separadas.
* Nunca usar a recursión para substituir un bucle.

<fin do tema 0>