teleriloves
www.wuolah.com/student/teleriloves

ADA1ejercicios (1).pdf

Ejercicios Resueltos de Análisis de Algoritmos

2° Análisis de Algoritmos y Estructuras de Datos

Grado en Ingeniería Informática

Escuela Superior de Ingeniería
UCA - Universidad de Cádiz

Tema 1: Órdenes asintóticos

Definición de orden:

$$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \quad f(n) \leq c \cdot g(n)$$

Conviene dar no solo el c, sino también el n_0. Entiendo que no lo has hecho porque en tu solución siempre basta con $n_0 = 0$, pero conviene decirlo explícitamente.

Ejercicio 1

Demuestre, a partir de la definición de O, que son ciertas las siguientes relaciones:

a) $\mathbf{1000 \in O(1)}$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \ 1000 \leq c \cdot 1$, para c=1000 se cumple, para $n_0 = 0$.

b) $\boldsymbol{n \cdot \sqrt{n} \in O(n^2)}$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \ n\sqrt{n} \leq c \cdot n^2$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \ n\sqrt{n} \leq c \cdot n \cdot n$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \ \sqrt{n} \leq c \cdot n$

para c=1 se cumple, ya que cualquier número natural es mayor o igual que su raíz, para $n_0 = 0$.

Solución 2:

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \ n\sqrt{n} \leq c \cdot n^2$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \ n \cdot n^{1/2} \leq c \cdot n \cdot n$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \ n^{1/2} \leq c \cdot n$

c) $\mathbf{2^n \in O(3^n)}$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \quad 2^n \leq c \cdot 3^n$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \quad \dfrac{2^n}{3^n} \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \quad \left(\dfrac{2}{3}\right)^n \leq c$

como $\left(\dfrac{2}{3}\right)^n$ es una función exponencial de base menor que 1, función que tiende a 0 luego con constante c = 1 ya se satisface, para $n_0 = 0$.

escuela de negocios
CÁMARA DE SEVILLA

diferénciate

Con la mejor formación práctica

www.mastersevilla.com

Titulación de prestigio
en el sector empresarial

MÁSTER EN DIRECCIÓN Y GESTIÓN DE RECURSOS HUMANOS

d) $\boldsymbol{n! \in O((n+1)!)}$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad n! \leq c \cdot (n+1)!$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \frac{n!}{n+1!} \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \frac{1}{n+1} \leq c$

como $\frac{1}{n+1}$ es una función decreciente, función que tiende a 0 luego con constante c = 1 ya se satisface, para $n_0 = 0$.

e) $\boldsymbol{n \notin O(\sqrt{n})}$

Supongamos lo contrario.

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad n \leq c \cdot \sqrt{n}$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \frac{n}{\sqrt{n}} \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \frac{n^{\frac{2}{2}}}{n^{\frac{1}{2}}} \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad n^{\frac{1}{2}} \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \sqrt{n} \leq c$

como $\sqrt{n}$ es creciente, función que tiende a ∞ luego con constante c habrá valores con los que ya no se cumpla, sea cuál sea el valor de n_0.

f) $\boldsymbol{n \cdot \log n \notin O(n)}$

Supongamos lo contrario.

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad n \cdot \log n \leq c \cdot n$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \frac{n}{n} \cdot \log n \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \log n \leq c$

como $\log n$ es una función creciente, función que tiende a ∞ luego con constante c habrá valores con los que ya no se cumpla, sea cuál sea el valor de n_0.

g) $\boldsymbol{3^n \notin O(2^n)}$

Supongamos lo contrario.

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad 3^n \leq c \cdot 2^n$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \frac{3^n}{2^n} \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \left(\frac{3}{2}\right)^n \leq c$

como $\left(\frac{3}{2}\right)^n$ es una función exponencial de base mayor que 1, función que tiende a ∞ luego con constante c habrá valores con los que ya no se cumpla, sea cuál sea el valor de n_0.

h) $\boldsymbol{O((n+1)!) \notin O(n!)}$

Supongamos lo contrario.

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad (n+1)! \leq c \cdot n!$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \frac{(n+1)!}{n!} \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad \frac{(n+1) \cdot n!}{n!} \leq c$

$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad n+1 \leq c$

como n+1 es una función creciente, función que tiende a ∞ luego con constante c habrá valores con los que ya no se cumpla, sea cuál sea el valor de n_0.

Ejercicio 2

Sea $k \geq 0$ una constante y f, g y h, funciones. Demuestre las siguientes propiedades a partir de la definición de O y de sus propiedades qye relacionan pertenencia y contención.

NOTA: No olvides que en un examen tendrías que justificar todos los pasos (con un comentario entre paréntesis a la derecha de cada paso).

a) $\boldsymbol{O(f) = O(g) \Longrightarrow O(f^k) = O(g^k)}$

De la definición de pertenencia ($\in$) y contención ($\subseteq$)

$O(f) = O(g) \Rightarrow f \in O(g) \wedge g \in O(f)$ {definición}

Primero demostramos

$$\begin{aligned}
O(f) = O(g) \quad & \Rightarrow f \in O(g) \\
& \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad f(n) \leq c \cdot g(n) && \{\text{definición}\} \\
& \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad (f(n))^k \leq (c \cdot g(n))^k && \{\text{ elevamos }\} \\
& \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad f(n)^k \leq c^k \cdot g(n)^k && \{\text{ Potencia de un producto }\} \\
& \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad f(n)^k \leq c' \cdot g(n)^k && \{\text{sustituimos } c' = c^k\ \} \\
& \Rightarrow f^k \in O(g^k)
\end{aligned}$$

Y análogamente demostramos que:

$O(f) = O(g) \quad \Rightarrow g \in O(f)$

b) $\boldsymbol{O(g) \subseteq O(f) \Longrightarrow O(f+g) = O(f)}$

De la definición de pertenencia ($\in$) y contención ($\subseteq$)

$O(g) \subseteq O(f) \Leftrightarrow g \in O(f) \wedge g \in O(f)$ {definición}

Primero demostramos

$$\begin{array}{lll} O(g) \subseteq O(f) & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) \leq c \cdot f(n) & \{\text{definición de orden}\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) + f(n) \leq c \cdot f(n) + f(n) & \{\text{ suma ambos lados igualdad }\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) + f(n) \leq (c+1) \cdot f(n) & \{\text{ factor común }\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) + f(n) \leq c' \cdot f(n) & \{\text{ Renombramos } c' = c+1\ \} \\ & \Rightarrow g + f \in O(f) & \{\text{ pertenencia }\} \\ & \Rightarrow O(g+f) \subseteq O(f) & \{\text{ contención }\} \end{array}$$

Luego tenemos que demostrar que:

$$\begin{array}{lll} O(f) \subseteq O(f+g) & \Rightarrow f \in O(f+g) & \{\text{ definición }\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad f(n) \leq c \cdot (f(n) + g(n)) & \{\text{ definición de pertenencia }\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad f(n) \leq c \cdot f(n) + c \cdot g(n) & *\ \{\text{ distributiva }\} \\ & \Rightarrow f \in O(f+g) & \{\text{ definición de pertencia }\} \\ & \Rightarrow O(f) \subseteq O(f+g) & \{\text{ definición de contención }\} \end{array}$$

**NOTA: Puesto que $f(n) \leq c \cdot f(n) + c \cdot g(n)$ es trivialmente cierto, ya que el tiempo de una función $g(n) \geq 0$*

Puesto que tenemos que:

$O(g) \subseteq O(f) \Rightarrow O(g+f) \subseteq O(f) \wedge O(f) \subseteq O(f+g)$

Podemos afirmar que:

$O(g) \subseteq O(f) \Rightarrow O(g+f) = O(f)$

c) $\boldsymbol{O(g) = O(h) \Longrightarrow O(f+g) = O(f+h)}$

De la definición de pertenencia ($\in$) y contención ($\subseteq$)

$O(g) = O(h) \Leftrightarrow O(f+g) \subseteq O(f+h) \wedge O(f+h) \subseteq O(f+g)$ {definición}

Primero demostramos que

$O(g) = O(h) \Rightarrow O(f+g) \subseteq O(f+h)$

Partiendo de

$$\begin{array}{lll} O(g) = O(h) & \Rightarrow O(g) \subseteq O(h) & \{\text{debilitamos las restricciones}\} \\ & \Rightarrow g \in O(h) & \{\text{def. pertenencia y contención}\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) \leq c \cdot h(n) & \{\text{definición de orden}\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) + f(n) \leq c \cdot h(n) + f(n) & \{\text{suma ambos lados igualdad}\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) + f(n) \leq c \cdot h(n) + c \cdot f(n) & \{\text{debilitamos la desigualdad}\} \\ & \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) + f(n) \leq c \cdot (h(n) + f(n)) & \{\text{factor común}\} \\ & \Rightarrow f + g \in O(h+f) & \{\text{pertenencia}\} \\ & \Rightarrow O(f+g) \subseteq O(h+f) & \{\text{contención}\} \end{array}$$

De manera análoga demostramos que

$O(g) = O(h) \Rightarrow O(f+h) \subseteq O(f+g)$

d) $\boldsymbol{O(g) = O(h) \Longrightarrow O(f \cdot g) = O(f \cdot h)}$

De la definición de pertenencia ($\in$) y contención ($\subseteq$)

$O(g) = O(h) \Leftrightarrow O(f \cdot g) \subseteq O(f \cdot h) \wedge O(f+h) \subseteq O(f+g)$ {definición}

Primero demostramos que

$O(g) = O(h) \Rightarrow O(f \cdot g) \subseteq O(f \cdot h)$

Partiendo de

$$\begin{aligned}
O(g) = O(h) \quad & \Rightarrow O(g) \subseteq O(h) && \{\text{debilitamos las restricciones}\} \\
& \Rightarrow g \in O(h) && \{\text{def. pertenencia y contención}\} \\
& \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) \leq c \cdot h(n) && \{\text{definición de orden}\} \\
& \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) \cdot f(n) \leq c \cdot h(n) \cdot f(n) && \{\text{multiplicamos ambos lados}\} \\
& \Rightarrow \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \quad g(n) \cdot f(n) \leq c \cdot (h(n) \cdot f(n)) && \{\text{factor común}\} \\
& \Rightarrow f \cdot g \in O(h \cdot f) && \{\text{pertenencia}\} \\
& \Rightarrow O(f \cdot g) \subseteq O(h \cdot f) && \{\text{contención}\}
\end{aligned}$$

De manera análoga demostramos que

$O(g) = O(h) \Rightarrow O(f \cdot h) \subseteq O(f \cdot g)$

Ejercicio 3

Generalice, por inducción, la regla del máximo a un número arbitrario de funciones. Es decir, sean k funciones $f_1, ..., f_k$, demuestre que:

$$\boldsymbol{O(f_1 + ... + f_k) = O(\text{máx}\{f_1, ..., f_k\})}$$

Para demostrar por inducción primero demostramos el caso base

$O(f_1 + f_2) = O(\text{máx}\{f_1, f_2\})$

Lo cuál es cierto ya que es una propiedad de los órdenes

Lo suponemos cierto para k

$O(f_1 + ... + f_k) = O(\text{máx}\{f_1, ..., f_k\})$

Y lo demostramos para k+1

$O(f_1 + ... + f_k + f_{k+1}) = O(\text{máx}\{f_1, ..., f_k, f_{k+1}\}) \Rightarrow O(f_1 + ... + f_k + f_{k+1}) = O(\text{máx}\{\text{máx}\{f_1, ..., f_k\}, f_{k+1}\})$

Cómo hemos supuesto válida la función máxima para k elementos y para 2 elementos, queda demostrado para $k + 1$.

Ejercicio 4

En primera instancia realizaremos solo las demostraciones de la parte de la implicación que contiene $\in$

a) $\lim \dfrac{f(n)}{g(n)} = 0 \Longrightarrow f \in O(g) \wedge g \notin O(f)$

Demostraremos que:

$\lim \dfrac{f(n)}{g(n)} = 0 \Longrightarrow f \in O(g)$

Para ello partimos de:

$$\lim \frac{f(n)}{g(n)} = 0 \Rightarrow \quad \forall \varepsilon > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 \quad \left| \frac{f(n)}{g(n)} - 0 \right| < \varepsilon \qquad \{\text{definición de límite}\}$$

$$\Rightarrow \quad \forall \varepsilon > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 \quad \frac{f(n)}{g(n)} < \varepsilon \qquad \{\text{operamos}\}$$

$$\Rightarrow \quad \forall \varepsilon > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 \quad \frac{f(n)}{g(n)} \leq \varepsilon \qquad \{\text{debilitamos restricciones}\}$$

$$\Rightarrow \quad \forall \varepsilon > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 \quad f(n) \leq \varepsilon \cdot g(n) \qquad \{\text{despejamos}\}$$

$$\Rightarrow \quad \forall \varepsilon \in \mathbb{R} \; \exists n_0 \in \; \forall n \geq n_0 \quad f(n) \leq \varepsilon \cdot g(n) \qquad \{...\}$$

$$\Rightarrow \quad f \in O(g) \qquad \{\text{definición de orden}\}$$

b) $\lim \frac{f(n)}{g(n)} = 0 \Longrightarrow g \in \Omega(f) \wedge f \notin \Omega(g)$

Ejercicio 5

a)

b)

c)

d)

e)

f)

g) $\boldsymbol{\Theta(\sum_{i=1}^{n} \frac{1}{i}) = \Theta(\log n)}$

Por el criterio de Stolz:

$$\lim \frac{\sum_{i=1}^{n} \frac{1}{i}}{\log n} = \lim \frac{\sum_{i=1}^{n} \frac{1}{i} - \sum_{i=1}^{n-1} \frac{1}{i}}{\log n - \log(n-1)} = \lim \frac{\frac{1}{n}}{\log n - \log(n-1)} = \lim \frac{\frac{1}{n}}{\log \frac{n}{n-1}} = \lim \frac{1}{n * \log \frac{n}{n-1}} =$$

$$\lim \frac{1}{\log(\frac{n}{n-1})^n} = \lim \left(\frac{n}{n-1} \right)^n = \frac{1}{loge} \in R^+ \Rightarrow \Theta(\sum_{i=1}^{n} \frac{1}{i}) = \Theta(\log n)$$

Muy bien. Hay que explicar que $\frac{1}{\log e} \in \mathbb{R}^+$ independientemente de la base del logaritmo.

También hay que indicar en algún sitio que:

$\lim \left(\frac{n}{n-1} \right)^n = e$

Ejercicio 6

f) $\Theta(\sum_{i=1}^{n} i^k) = \Theta(n^{k+1})$

He resuelto uno de los apartados del ejercicio 6 de acotación integral, el segundo concretamente, que se corresponde con el apartado g) del ejercicio 5. Me gustaría que alguien que también lo haya resuelto me dijera si está bien o en caso contrario cuáles son los errores.

Lo he resuelto siguiendo los mismos pasos que el que está resuelto en las hojas de repaso.
Lo primero que he hecho es llamar Gn al sumatorio $\sum_{i=1}^{n} i^k$

Consideramos una extensión real del término general de la sucesión $f(x) = x^k$

Se observa que $f : \mathbb{R} \to \mathbb{R}$ es monótona creciente (Es decir cada término es menor o igual que el anterior) y es continua (ya que se trata de un polinomio y estos lo son siempre)

Luego al acotar de la siguiente forma:
$\int_0^n x^k\,dx \leq \sum_{i=1}^{n} i^k \leq \int_1^{n+1} x^k\,dx$

observamos que:
$\left[x^k + 1\right]_0^n \leq Gn \leq \left[x^k + 1\right]_0^{n+1}$

y en conclusión:
$n^{k+1} - 0^{k+1} \leq Gn \leq (n+1)^{k+1} - 1^{k+1}$
$n^{k+1} \leq Gn \leq (n+1)^{k+1} - 1$

Como Gn está acotada inferior y superiormente por funciones de órden polinomial, neceriamente $Gn \in O(n^{k+1})$

palomo

Repasemos las integrales que hacen falta para resolver el ejercicio 6:

6f)

$$\int x^k\ d\,x = \frac{x^{k+1}}{k+1}$$

6g)

$$\int \frac{1}{x}\,dx = \ln x$$

Como $\ln x$ no es continua en el 0, conviene eliminar el término problemático del sumatorio antes de acotar (como se hace en los materiales de repaso).

6h)

Hagámoslo con $\ln n!$ (para el orden que hay que calcular, da igual). Hay que tener en cuenta que:

$\ln n! = \ln \prod_{i=1}^{n} i = \sum_{i=1}^{n} \ln i$

Este es el ejemplo típico de integral que se resuelve por partes.

$$\int \ln x \, dx = x \ln x - \int \frac{1}{x} x \, dx = x \ln x - x$$

Al igual que antes, como $x \ln x - x$ no es continua en el 0, conviene eliminar el término problemático del sumatorio antes de acotar.

Tema 2: Análisis a la complejidad de los problemas

Ejercicio 6

Sean v y w vectores de dimensión $n \in \mathbb{N}^*$, el siguiente algoritmo determina si uno es permutación del otro-

```
permutación : v × w × n → w × r
si n = 1
   r ← v[1] = w[1]
si no
   p ← posición(v[n], w, n)
   si p > n
      r ← ⊥
   si no
      w[p] ↔ w[n]
      r ← permutación(v, w, n − 1)
```

```
posición : x × v × n → p
p ← 1
mientras p ≤ n ∧ x ≠ v[p]
        p ← p + 1
```

Analice, mediante ecuaciones de recurrencia, su eficiencia temporal en el peor caso. Seleccione cuidadosamente la operación crítica.

***Nota.* Obsérvese que el algortimo «destruye» w, ya que se trata de un parámetro de entrada y salida. En este caso, es posible evitar este efecto secundario definiendo w como un parámetro exclusivamente de entrada y pasándolo por copia. Sin embargo, en este y otros algoritmos, es importante suponer que los vectores se pasan siempre por referencia, aunque sean parámetros exclusivamente de entrada: el paso por copia incurriría, en la mayoría de las ocasiones, en un coste inacpetable que impediría considerarlo como operación elemental.**

Elegimos como operación crítica la comparación entre elementos del vector (Esto incluye las operaciones $x \neq v[p]$ y $x[1] = w[1]$). Ya que es la operación que mas se repite y además representa la esencia del algoritmo.

Caso base:
Cuando $n = 1$ se realiza una comparación entre elementos del vector, por lo que $t(1) = 1$.

Caso general:
Cuando $n > 1$ se realiza una llamada a posición, dentro de la cuál se produce la comparación entre elementos del vector, con un vector de n elementos y además vuelve a llamar a la propia función con $n - 1$ elementos. Por lo tanto cuando $n > 1$ realiza $t_pos(n) + t(n - 1)$ comparaciones.

Por tanto, nuestra ecuación de recurrencia será:

$$t(n) = \begin{cases} 1 & \text{si } n = 1 \\ t_{pos}(n) + t(n-1) & \text{si } n > 1 \end{cases}$$

Analizando el algoritmo de posición vemos que en el peor caso, cuando el elemento no pertenece al vector realiza n operaciones críticas, por lo que $t_{pos}(n) = n$. Luego nuestra ecuación de recurrencia en el caso general nos quedaría de la forma:

$t(n) = n + t(n - 1)$

Se trata de una ecuación de recurrencia lineal de coeficientes constantes no homogénea de grado 1. La resolvemos por el método del polinomio característico.

$t(n) - t(n-1) = n$ {Reorganizamos la ecuación}

La parte homogénea aporta el factor: $C_1 \leftarrow (x-1)$
La parte no homogénea aporta el factor: $C_2 \leftarrow (x-1)^2$

$c(x) = (x-1)(x-1)^2 = (x-1)^3$ {Polinomio característico}

$t(n) = \alpha + \beta n + \gamma n^2$ {ecuación característica}

Sustituyendo los parámetros ligados en la ecuación general obtenemos
$\beta n + \gamma n^2 - (\beta(n-1) + \gamma(n-1)^2) = n \Rightarrow ... \Rightarrow \beta - \gamma + 2\gamma n = n$

Por un lado tenemos que:

$$2\gamma n = n \Rightarrow \gamma = \frac{2n}{n} = \frac{1}{2}$$

Y por otro

$$\beta - \gamma = 0 \Rightarrow \beta - \frac{1}{2} = 0 \Rightarrow \beta = \frac{1}{2}$$

Sustituyendo los valores de los parámetros ligados en la ecuación general obtenemos la solución general que será

$t(n) = \alpha + \frac{1}{2}n + \frac{1}{2}n^2$ {solución general}

Aplicando las condiciones iniciales $t(1) = 1$ calculamos la solución particular

$t(1) = \alpha + \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1^2 = 1 \Rightarrow \alpha - 1 = 0 \Rightarrow \alpha = 1 - 1 \Rightarrow \alpha = 0$ {Aplicando condiciones iniciales}

$t(n) = \frac{1}{2}n + \frac{1}{2}n^2$ {Solución Particular}

$t \in \Theta(n^2)$ {Aplicando la regla del máximo}

Tema 3: Algunos algoritmos clásicos y su análisis

Ejercicio 5

Sea v un vector de dimension $n \in N^*$ tal que $v[1] <= ... <= v[n-1]$, el siguiente algoritmo termina de ordenaer el vector insertando en orden el elemento v[n] en la posición adecuada.

inserción: v x n $\longrightarrow$ v

si $n > 1$

si $v[n] < v[n-1]$

$v[n] \leftrightarrow v[n-1]$

inserción (n, n-1)

Analice, mediante ecuaciones de recurrencia, el número exacto de comparaciones entre los elementos del vector que realzia el algoritmo en el mejor caso, el peor caso y el caso promedio. Para el análisis en el caso promedio, suponga que el elemento v[n] tiene la misma probabilidad de acabar en cualquiera de las posiciones.

Pista: En el caso promedio le resultará útil considerar las relaciones existentes entre las probabilidades de que $v[n] < v[n-1], v[n-1] < v[n]$ y $v[n] = max_{1 \leq i \leq n} v[i]$.

La operación crítica del algortimo será $v[n] < v[n-1]$ puesto que al menos de ejecuta tantas veces como cualquier otra del algortimo y representa la esencia del mismo.

Mejor caso

En el mejor caso sería que v[n] fuera mayor que v[n-1] luego su posición sería la nueva posición última del vector y solo se ejecutaría una vez la operación crítica.

$t_{min}(n) = 1 \in \Theta(1)$

Peor caso

En el peor caso sería que v[n] fuera siempre menor que v[n-1] luego se efectuarían n-1 llamadas recursivas y se ejecutaría la operación crítica n-1 veces, n-1 veces porque si el elemento nuevo ha de viajar hasta la posición v[1] en la última llamada al programa sería la instrucción n >1 la que término las llamadas recursivas, no produciendose en este caso una ejecuciónd e la comparación, luego:

$t_{max}(1) = 0)$

$t_{max}(n) = 1 + t_{max}(n-1)$ si $n > 1$

De aqui deducimos fácilmente que $t_{max}(n) = n - 1 \in \Theta(n)$

Caso promedio

Concluyendo $t(n) \in \Omega(1) \cap \Theta(n) \Rightarrow O(1) \leq O(t(n)) \leq O(n)$, dada la diferencia asintótica entre el mejor y el peor caso, tiene sentido estudiar el caso promedio.

Llamemos $t(n)$ al tiempo promedio. Si $n = 1$, se hacen 0 comparaciones. Si $n > 1$, se hace 1 comparación y, si esta es positiva, lo que ocurre con probabilidad $P(v[n] < v[n-1])$, se hacen $t(n-1)$ comparaciones más. Si la comparación resulta negativa no se hace ninguna más. Por tanto:

$$t(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ 1 + (P[v[n] < v[n-1]]) \cdot t(n-1) & \text{si } n > 1 \end{cases}$$

La probabilidad de que el elemento a insertar sea máximo en un vector de n elementos es $\frac{1}{n}$ ya que tiene la misma posibilidad de acabar en cualquier otra posición del vector, por tanto la probabilidad del caso promedio, de que el elemento no sea el máximo del vector será

$$P(v[n] < v[n-1]) = 1 - \frac{1}{n} = \ \ldots \ = \frac{n-1}{n}$$

Por tanto sustituyendo en el caso general

$t(n) = 1 + (P[v[n] < v[n-1]]) \cdot t(n-1) \Rightarrow$

$\Rightarrow t(n) = 1 + \frac{n-1}{n} \cdot t(n-1)$ {Sustituimos}

$n \cdot t(n) = n + (n-1) \cdot t(n-1)$ {Multiplicando por n}

$u(n) = u(n-1) + n$ {Cambio de variable $u(n) = n \cdot t(n)$, si $n > 1$}

Resolviendo por sumatorio nos quedaría

$u(n) = u(1) + \sum_{i=2}^{n} i$

Puesto que $u(1) = 1 \cdot t(1) = 1 \cdot 0 = 0$

$u(n) = \sum_{i=2}^{n} i$ {sustituyendo $u(1) = 0$}

$u(n) = \frac{(2+n)(n-1)}{2}$ {Resolviendo sumatorio}

$u(n) = \frac{n^2}{2} + \frac{n}{2} - 1$ {Operando}

$t(n) = \frac{n+1}{2} - \frac{1}{n}$ {Aplicando $t(n) = \frac{u(n)}{n}$}

$t \in \Theta(n)$ {Aplicando la regla del máximo}

Ejercicio 8

Considere el siguiente algoritmo, que calcula F_n, n-éismo número de Fibonacci.

```
f : n → r
⟨a, b⟩ ← ⟨0, 1⟩
mientras n > 1
        c ← a + b
        ⟨a, b⟩ ← ⟨b, c⟩
        n ← n − 1
si n = 0
  r ← a
si no
  r ← b
```

Conteste razonadamente a las siguientes cuestiones:

a) **¿Bajo que condiciones no es posible considerar a la suma como una operación elemental?**

Para que la suma se consideré una operación elemental n debe ser lo suficientemente pequeño como para que al programar el algoritmo, $f(n)$ quepa en una palabra de máquina. Esto se debe a que el tamaño de $f(n)$ crece linealmente respecto de n, esto es así por la forma en que crece los números de Fibonacci. Por tanto salvo que n sea lo suficientemente pequeño, no es realista considerar la suma de $f(n-1)$ y $f(n-2)$ como una operación de máquina elemental, se se desea calcular correctamente $f(n)$, sino como un cálculo independiente compuesto de $\Theta(n)$ operaciones de máquina (las sumas de los dígitos).

b) **¿Cuál sería entonces la operación aritmética elemental que podríamos tomar como operación crítica para analizar el algoritmo**

La operación elemental será suma de dos números se considera elemental cuando se puede realizar en tiempo $\Theta(1)$, es decir, cuando los números implicados son lo suficientemente pequeños. Esto quiere decir que n debe ser lo suficientemente pequeño como para que al programar el algoritmo, $f(n)$ quepa en una palabra de máquina.

c) **¿Cuál sería en tal caso el orden de la suma medido en dichas operaciones críticas en función del tamaño de sus operandos?**

Se observa que el algoritmo va sumando desde i igual a 2 hasta n los números de Fibonacci $a = F_{i-1}$ y $b = F_{i-2}$ para producir $c = F_i$.

Por tanto, su complejidad temporal es:

$$t(n) \in \sum_{i=2}^{n} \Theta(||F_{i-2} + F_{i-1}||) =$$

$$= \sum_{i=2}^{n} \Theta(||F_i||) = \quad \{\text{Definición de } F_i\}$$

$$= \sum_{i=2}^{n} \Theta(i) = \quad \{\text{Tamaño de } F_i\}$$

$$= \Theta(\sum_{i=2}^{n} i) = \quad \{\text{Suma de órdenes}\}$$

$$= \Theta\left(\frac{(n+2)(n-1)}{2}\right) = \quad \{\text{Suma aritmética}\}$$

$$= \Theta\left(\frac{1}{2}n^2 + \frac{1}{2}n - 1\right) = \quad \{\text{Simplifación algebraica}\}$$

$$= \Theta(n^2 \quad \{\text{Orden de un polinomio}\}$$

Nota: $||x||$ representa el tamaño de x, medido en número de dígitos.

Ejercicio 11

El algoritmo recursivo trivial para el cálculo del determinante de una matriz cuadrada de orden n procede de la siguiente forma:

- **Si $n = 1$, el determinante se calcula sin ninguna operación escalar, ya que es el único elemento que posee la matriz.**

- **En caso contrario, se calculan los determinantes de los adjuntos de cada uno de los elementos de, por ejemplo, la primera fila (que son a su vez matrices cuadradas de orden $n-1$). El determinante de la matriz original se obtiene a partir de éstos tras $2n-1$ operaciones escalares.**

Dé un valor aproximado del número de operaciones escalares que realiza este algoritmo cuando n es lo suficientemente grande. Calcule su orden.

Pista. **Suponga n lo suficientemente grande como para poder considerar $\sum_{i=0}^{n} \frac{1}{i!} \approx e$.**

Planteamos la siguiente ecuación de recurrencia

$$t(n) = \begin{cases} 0 & \text{si, } n = 1 \\ n \cdot t(n-1) + 2n - 1 & \text{si, } n > 1 \end{cases}$$

Analizamos el caso general

$t(n) = n \cdot t(n-1) + 2n - 1 \Rightarrow$

$\Rightarrow n! \cdot u(n) = n! \cdot u(n-1) + 2n - 1 \Rightarrow$ {cambio de variable $t(n) = n! \cdot u(n)$}

$\Rightarrow u(n) = u(n-1) + \frac{2n-1}{n!} \Rightarrow$ {dividimos ambos lados por $n!$}

$\Rightarrow u(n) = u(n-1) + \frac{2n}{n!} - \frac{1}{n!} \Rightarrow$ {descomponemos la fracción}

Transformamos en sumatorio, usando el caso general $n > 1$

$$\Rightarrow u(n) = u(2-1) + 2\sum_{i=2}^{n-1} \frac{1}{i!} - \sum_{i=2}^{n} \frac{1}{i!} =$$

$$= u(n) = 0 + 2\sum_{i=2}^{n-1} \frac{1}{i!} - \frac{1}{n!} - \sum_{i=2}^{n-1} \frac{1}{i!} =$$ {Sacamos último término del sumatorio}

$$= (2-1)\sum_{i=2}^{n-1} \frac{1}{i!} - \frac{1}{n!} =$$ {factor común}

$$= \sum_{i=2}^{n-1} \frac{1}{i!} - \frac{1}{n!} =$$ {simplificamos}

Puesto que hemos considerado un n lo suficientemente grande y sabemos que $\sum_{i=0}^{n} \frac{1}{i!} \approx e$

$$= \sum_{i=0}^{n} \frac{1}{i!} - \frac{1}{0!} - \frac{1}{1!} - \frac{1}{n!} - \frac{1}{n!} \approx$$ {sumamos y restamos}

$$\approx e - 1 - 1 - \frac{1}{n!} - \frac{1}{n!} =$$

$$= e - 2 - \frac{2}{n!}$$

Deshaciendo el cambio $t(n) = n! \cdot u(n)$

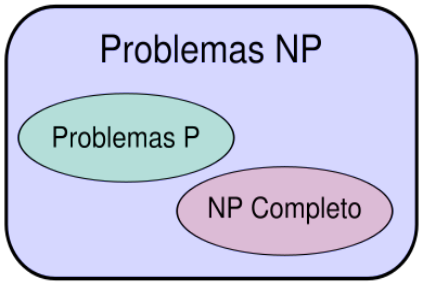
$\Rightarrow t(n) \approx n! \cdot (e - 2 - \frac{2}{n!})$ {Deshaciendo el cambio}

$\Rightarrow t(n) \approx e \cdot n! - 2 \cdot n! - n! \cdot \frac{2}{n!}$ {Multiplicamos}

$\Rightarrow t(n) \approx e \cdot n! - 2 \cdot n! - 2$ {Simplificando}

$\Rightarrow t \in \Theta(n!)$

Tema 4: Introducción a la complejidad de los problemas



Ejercicio 1

Conteste razonadamente a las siguiente cuestiones.

a) **¿Qué características fundamente posee la clase de problemas NP?**

La clase de problemas NP son un conjunto de problemas de decisión que disponen de un algoritmo de verificación (para comprobar si una solución es válida o no) de tiempo polinómico.

Un problema de decisión acepta una entrada x si y solo si hay un testigo y tal que el algoritmo de verificación acepta el par (x, y) en tiempo polinómico.

b) **¿Qué se deduce, en términos de dificultad relativa, del hecho que un problema de decisión sea reducible polinómicamente a otro?**

A $\leq_p$ B

A se reduce polinómicamente a B, quiere decir, que problema A es igual o más fácil que el problema B.

Se puede conseguir un algoritmo de A transformar cada ejemplar del problema A en un ejemplar de B y disponiendo de un algoritmo de B.

La transformación de ejemplares de A en B ha de ser polinómica,

A $\leq_p$ B

A se reduce polinómicamente a B, luego B es al menos tan difícil como A.

Luego si B es tratable (como mucho polínomico) entonces A será tratable.

Pero si A es no tratable entonces B no será tratable (porque B es siempre igual o más díficil que A).

c) **¿Qué ocurriría si el problema del árbol de Steiner pudiera reducirse polinómicamente a un problema NP dado?**

[Mirar el esquema]

A.S. es NPC

Y como $NPC \Rightarrow NP$

A.S en NP

$NP_{AS} \leq_p NP_B$

Luego $NP_B =_p NP_{AS}$, siendo B cualquier problema NP.

Recordar que,

NPC_A se reduce a NP_B quiere decir que el primero es igual o más fácil que el segundo, y NPC es la clase de los problemas más complejos luego estaremos en una igualdad, en la que $NP_B =_p NP_A$ donde B es cualquier problema NP.

d) ¿Qué ocurriría si el problema del árbol de Steiner pudiera reducirse polinómicamente al del árbol de expasión de coste mínimo?

[Mirar el esquema]

El problema de A.S. es NPC y el problema AECM es P, luego,

$$NPC_{AS} \leq_p P_{AECM}$$

Recordar que $NPC_{AS} \Rightarrow NP_{AS}$ luego

$$NP_{AS} \leq_p P_{AECM}$$

Y eso significa que NP_{AS} es igual o más fácil que P_{AECM}, como NP son más díficiles que P, por definición, entonces:

$$NP_{AS} =_p P_{AECM} \Rightarrow NP = P$$

Ejercicio 4

Demuestre que el problema 3-COL es reducible a SAT en tiempo polinóico. Acote inferior y superiormente el número de cláusulas y literales de la fórmula resultante en función del número de vértices del grafo.

Hay que justificar la existencia de un algoritmo capaz de construir a partir de cualquier grafo una fórmula lógica en forma normal conjuntiva (FNC) que exprese si el grafo es 3-coloreable. Para demostrar que la reducción se realiza en tiempo polinómico hay que comprobar que el tiempo que se emplea en la transformación está acotado por un polinomio en el tamañoo de la entrada.

Para cada vértice i del grafo se definen tres variables booleanas Ri , Vi y Ai , que indican si el vértice puede ser coloreado con un determinado color, por ejemplo, rojo, verde y azul, respectivamente (aunque los colores concretos son irrelevantes).

A continuación se expresan las restricciones respecto a los colores mediante cláusulas de una fórmula lógica FNC. Cada cláusula es una disyunción lógica de literales. La fórmula es la conjunción lógica de todas las cláusulas.

- Cada vértice ha de poseer un único color.

$$\begin{array}{ll} C_1 = R_i \vee V_i \vee A_i & \text{(es rojo, verde, azul o una mezcla)} \\ C_2 = \neg R_i \vee \neg V_i & \text{(no es rojo o verde a la vez)} \\ C_3 = \neg R_i \vee \neg A_i & \text{(no es rojo o azul a la vez)} \\ C_4 = \neg V_i \vee \neg A_i & \text{(no es verde o azul a la vez)} \end{array}$$

- Cada arista ha de poseer extremos de distintos colores.

$$
\begin{array}{ll}
C_5 = \neg R_i \vee \neg R_j & \text{(no es rojo en ambos extremos)} \\
C_6 = \neg V i \vee \neg V j & \text{(no es verde en ambos extremos)} \\
C_7 = \neg A i \vee \neg A j & \text{(no es azul en ambos extremos)}
\end{array}
$$

Para cada uno de los n vértices se necesitan cuatro cláusulas y nueve literales. Para cada arista se necesitan otras tres cláusulas y seis literales. En un grafo de n vértices, el número a de aristas está acotado:

$$
0 \leq a \leq \binom{n}{2} = \frac{n(n-1)}{2}
$$

y, por lo tanto, el número de cláusulas, c, y literales, l, también:

$$
\begin{array}{lll}
c = 4n + 3a & 4n \leq c \leq 4n + \dfrac{3}{2} n(n-1) & c \in \Omega(n) \cap O(n^2) \\
l = 9n + 6a & 9n \leq l \leq 9n + 3n(n-1) & l \in \Omega(n) \cap O(n^2)
\end{array}
$$

La fórmula ocupa espacio polinómico y puede construirse en tiempo polinómico generando secuencialmente las cláusulas requeridas por cada vértice y arista.