




Métodos Avanzados de Programación Científica y Computación

M^a Luisa Díez Platas

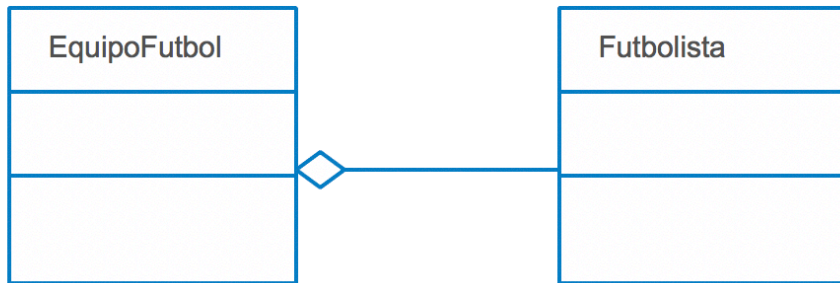
Tema 2. Relaciones entre clases y complejidad (II)

¿Cómo estudiar este tema?

IDEAS CLAVE	LO + RECOMENDADO	+ INFORMACIÓN	TEST
<p>¿Cómo estudiar este tema?</p> <p>Abstracción y herencia</p> <p>Conceptos avanzados de herencia</p> <p>Polimorfismo</p> <p>Composición y agregación</p> <p><code>This</code> y <code>super</code></p> <p>Complejidad de un algoritmo</p>	<p>No dejes de leer...</p> <p>Herencia</p> <p>Paquetes en Java</p> <p>Herencia en Java</p> <p>No dejes de ver...</p> <p> Proyecto en ArgoUML</p> <p> Herencia en Java</p> <p> Polimorfismo en Java</p>	<p>A fondo</p> <p>Ejemplo de herencia en Java</p> <p>Ejemplo de polimorfismo en Java</p> <p>UML Distilled</p> <p>Programación orientada a objetos usando Java</p>	

- Abstracción y herencia
- Polimorfismo
- Agregación
- Conceptos relacionados con los métodos

Agregación



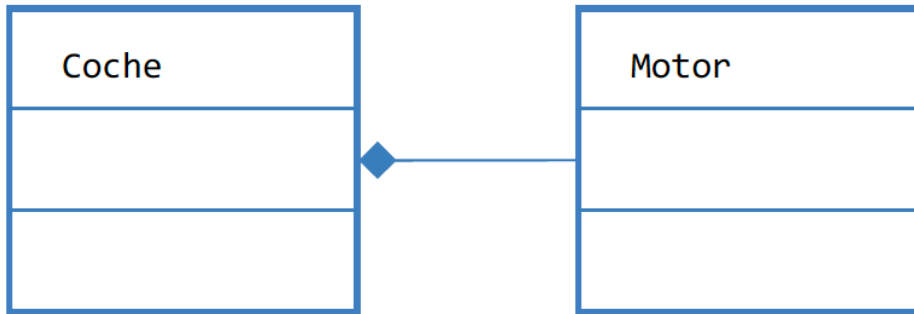
```
class Futbolista{
//....
public Futbolista(.....){//.....}
}
class EquipoFutbol {
    private Futbolista[] futbolistas;
public EquipoFutbol(.....){
    futbolistas=new Futbolista[20];
}
}
```

Se crea el array pero no cada una de los futbolistas

- Una clase esta formada por objetos de otra
- Los objetos contenidos no **se crean necesariamente al crear el objeto de la contenedora**
- La destrucción del objeto de la contenedora **no implica la destrucción de los objetos contenidos**

Composición débil

Composición



```
class Motor{
//...
public Motor(.....){//.....}
}

class Coche {
    private Motor elMotor;
public Coche(.....){
    elMotor=new Motor(...);
}
}
```

An arrow points from the text 'Se crea el objeto contenido al mismo tiempo que el contenedor' to the line `elMotor=new Motor(...);` in the Coche class definition.

Se crea el objeto contenido al mismo tiempo que el contenedor

- Una clase esta formada por objetos de otra
- Los objetos **contenidos se crean al crear el objeto de la contenedora**
- La destrucción del objeto de la contenedora **implica la destrucción de los objetos contenidos**

Complejidad

- ▶ Espacial- cantidad de memoria que necesita un algoritmo para su ejecución
- ▶ Temporal-cantidad de tiempo que un algoritmo consume para su ejecución
- ▶ Factores que influyen
 - Externos: la máquina, el modelo de gestión de la memoria
 - Internos: Número de instrucciones

Complejidad temporal. Te

- ▶ Complejidad temporal –función que expresa el número de pasos de programa que un algoritmo necesita ejecutar para cualquier entrada posible → **cálculo a posteriori**

- Paso de un programa: secuencia de operaciones
- Tamaño de la entrada

Número de instrucciones → depende de condiciones

caso mejor, caso medio y caso peor.

Etapas Complejidad temporal. Te

1. Determinación del tamaño del problema
 1. Determinación del caso mejor y peor: instancias para las que el algoritmo tarda más o menos
 - No siempre existe mejor y peor → el tamaño no influye
2. Obtención de los valores para cada caso.

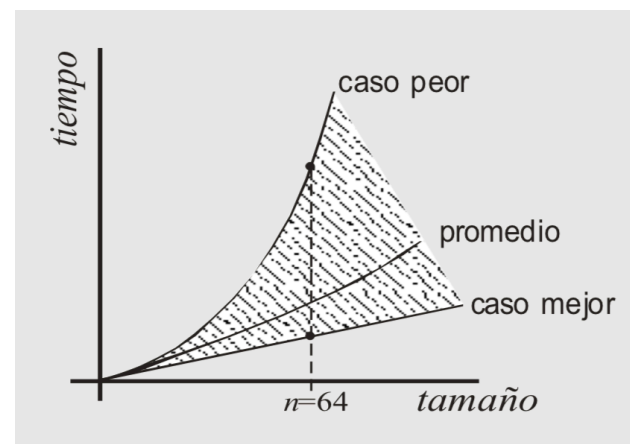
Métodos:

- cuenta de pasos
- relaciones de recurrencia (funciones recursivas)

Ejemplo

► Búsqueda en un vector:

1. Tamaño de la entrada \rightarrow longitud de un vector
2. Caso mejor: el elemento buscado está en el primer lugar
3. Caso peor: el elemento buscado no está en el vector



https://rua.ua.es/dspace/bitstream/10045/4411/17/ped-06_07-tema1.pdf

funcion buscar (**var** X:vector[N]; elem:Tipo) **devuelve** Entero

var i:Entero;

comienzo

i:=1;

mientras ($i \leq N \wedge X[i] \neq \text{elem}$) **hacer**

i:=i+1;

fin_mientras;

si ($i > N$) **entonces devuelve** 0;

si_no devuelve i;

fin_si

fin_funcion

Ejemplo

funcion buscar (**var** X:vector[N]; elem:Tipo) **devuelve** Entero

var i:Entero;

comienzo

i:=1;

mientras (i ≤ N ∧ X[i] ≠ elem) **hacer**

i:=i+1;

fin_mientras;

si (i > N) **entonces devuelve** 0;

si_no devuelve i;

fin_si

fin_funcion

Lo importante es
la relación con el
tamaño del array
en este caso
 $5n \rightarrow O(n)$
Complejidad lineal

- Caso mejor: el elemento está en el primero

$$T_{\text{mejor}} = 1 + 1 + 1 + 2 + 1 + 1 = 7$$

- Caso peor: el elemento no está

$$T_{\text{peor}} = 1 + ((n)(1 + 1 + 2 + 1) + 1 + 1) + 1 + 1 = 5n + 5$$

Ejemplo

funcion buscar (**var** X:vector[N]; elem:Tipo) **devuelve** Entero

var i:Entero;

comienzo

i:=1;

mientras (i < N \wedge X[i] \neq elem) **hacer**

i:=i+1;

fin_mientras;

si (X[i] \neq elem) **entonces devuelve** 0;

si_no devuelve i;

fin_si

fin_funcion

Lo importante es
la relación con el
tamaño del array
en este caso
 $5n \rightarrow O(n)$
Complejidad lineal

- Caso mejor: el elemento está en el primero

$$T_{\text{mejor}} = 1 + 1 + 1 + 2 + 2 + 1 = 8$$

- Caso peor: el elemento no está

$$T_{\text{peor}} = 1 + ((n-1)(1 + 1 + 2 + 1) + 1 + 1) + 2 + 1 = 5n + 1$$

Ejemplo

funcion buscar (**var** X:vector[N]; elem:Tipo) **devuelve** Entero

var i:Entero;

comienzo

i:=1;

mientras (X[i]≠elem ∧ i < N) **hacer**

i:=i+1;

fin_mientras;

si (X[i]≠elem) **entonces devuelve** 0;

si_no devuelve i;

fin_si

fin_funcion

Lo importante es
la relación con el
tamaño del array
en este caso
 $5n \rightarrow O(n)$
Complejidad lineal

- Caso mejor: el elemento está en el primero

$$T_{\text{mejor}} = 1 + 2 + 1 + 2 + 1 = 7$$

- Caso peor: el elemento no está

$$T_{\text{peor}} = 1 + ((n-1)(2 + 1 + 1 + 1)) + 2 + 1 + 1 = 5n + 3$$

Ejemplo

funcion buscar (**var** X:vector[N]; elem:Tipo) **devuelve** Entero

var i:Entero;

comienzo

i:=1;

mientras (X[i]≠elem ∧ i < N) **hacer**

i:=i+1;

fin_mientras;

si (X[i]≠elem) **entonces devuelve** 0;

si_no devuelve i;

fin_si

fin_funcion

Lo importante es
la relación con el
tamaño del array
en este caso

$$\frac{5}{2}n \rightarrow O(n)$$

Complejidad lineal

- **Caso medio:** el bucle se ejecuta la mitad de veces

$$T_{\text{medio}} = 1 + ((m)(2 + 1 + 1 + 1) + 2 + 1) + 2 + 1 = 5m + 7 = 5(n-1)/2 + 7 = \frac{5}{2}n + \frac{9}{2}$$

$$m = (n-1)/2$$

Ejemplo, Caso mejor y peor

funcion buscar (**var** X:vector[N]; elem:Tipo) **devuelve** Entero

var i:Entero;

comienzo

i:=1;

mientras (i ≤ N) **hacer**

si (X[i]==elem) **entonces devuelve** i;

fin_si

 i:=i+1;

fin_mientras

devuelve 0;

fin_funcion

Lo importante es
la relación con el
tamaño del array
en este caso
 $4n \rightarrow O(n)$
Complejidad lineal

- Caso mejor: el elemento está en el primero

$$T_{\text{mejor}} = 1 + 1 + 2 + 1 = 5$$

- Caso peor: el elemento no está

$$T_{\text{peor}} = 1 + ((n)(1 + 2 + 1) + 1) + 1 = 4n + 3$$

Ejemplo. Caso medio

funcion buscar (**var** X:vector[N]; elem:Tipo) **devuelve** Entero

var i:Entero;

comienzo

i:=1;

mientras (i ≤ N) **hacer**

si (X[i]==elem) **entonces devuelve** i;

fin_si

 i:=i+1;

fin_mientras

devuelve 0;

fin_funcion

Lo importante es
la relación con el
tamaño del array
en este caso
 $2n \rightarrow O(n)$
Complejidad lineal

- Caso medio: el bucle se ejecuta la mitad de veces →

$$T_{\text{medio}} = 1 + ((m)(1 + 2 + 1)) + 1 + 2 + 1 = 4m + 5 = 4n/2 + 5 = 2n + 5$$

$$m = n/2$$

Escala de complejidad

$$O(1) \subset O(\lg \lg n) \subset O(\lg n) \subset O(\lg^{a>1} n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \lg n) \subset O(n^2) \subset \dots \subset O(n^{a>2}) \subset O(2^n) \subset O(n!) \subset O(n^n)$$

El tamaño de la complejidad puede variar según el valor de n

Entrada n	$10^6 n^2$	$5n^3$	Eficiencia de tiempo
100.000	10×10^{15}	5×10^{15}	$5n^3$
200.000	40×10^{15}	40×10^{15}	Igual
300.000	60×10^{15}	135×10^{15}	$10^6 n^2$

https://rua.ua.es/dspace/bitstream/10045/4411/17/ped-06_07-tema1.pdf

Conclusión

- ▶ La mayoría de las operaciones → complejidad constante
- ▶ Lo importante es la complejidad relacionada con el tamaño → determina el orden

UNIVERSIDAD
INTERNACIONAL
DE LA RIOJA

unir

www.unir.net