ESTRUCTURAS DE DATOS Y ALGORITMOS



GUÍA ILUSTRADA PARA PROGRAMADORES

MARIONA NADAL FARRÉ

ANAYA MULTIMEDIA

ESTRUCTURAS DE DATOS Y ALGORITMOS

ESTRUCTURAS DE DATOS Y ALGORITMOS

GUÍA ILUSTRADA PARA PROGRAMADORES

MARIONA NADAL FARRÉ

ANAYA MULTIMEDIA

Títulos Especiales

Diseño de cubierta: Celia Antón Santos

Ilustración de cubierta: Eduardo Rodríguez Meliá

Revisión: Gelsys M. García Lorenzo

Maquetación: Claudia Valdés-Miranda Cros

Responsable editorial: Eugenio Tuya Feijoó

1.ª edición digital: 2022

Todas las denominaciones de programas, sistemas operativos, equipos, etc., que aparecen en este libro son marcas registradas de sus respectivas compañías y organizaciones.

Reservados todos los derechos. El contenido de esta obra está protegido por la ley, que establece penas de prisión y/o multas, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujeren, plagiaren, distribuyeren o comunicaren públicamente, en todo o en parte, una obra literaria, artística o científica, o su transformación, interpretación o ejecución artística fijada en cualquier tipo de soporte o comunicada a través de cualquier medio, sin la preceptiva autorización.

Edición española:

© EDICIONES ANAYA MULTIMEDIA (GRUPO ANAYA, S.A.), 2022
Calle de Juan Ignacio Luca de Tena, 15
28027 Madrid
www.anayamultimedia.es

ISBN: 978-84-415-4639-4
Edición electrónica sobre la 1.ª edición impresa

A mi primera promoción[1] de GDAM en la escuela CES, juntos hemos probado en vivo el funcionamiento del libro de Java, mientras dabais vuestros primeros pasos en el mundo de la programación. También me habéis acompañado en el crecimiento de este libro, el de algoritmos, y me habéis inspirado para futuras obras.

[1] **Grupo de primero, turno de tarde, curso 2021-2022.**

Agradecimientos

A toda la gente mencionada en los agradecimientos del libro anterior y además a todos aquellos que me han ayudado a dar un paso más en mi carrera como profesora, dejando atrás mis tiempos como desarrolladora en la empresa privada:

- A Carlos, de LinkedIn Learning, por seguir confiando en mí para elaborar más y más videocursos.
- A Eugenio, de Anaya Multimedia, por ofrecerme hacer este segundo libro y los que vendrán.
- A Elena, Carles y Carlos, de la UOC, por abrirme la puerta a la docencia universitaria.
- A Sonia, de la escuela CES, por creer en mis posibilidades como profesora presencial y darme no solo una sino dos oportunidades (también en la escuela CEV) y así permitirme cambiar de vida.

Y específicamente sobre esta obra:

- A Pablo, por releer todos los capítulos, encontrar esos errores que a veces se escapan y escribir el prólogo.
- A Eduardo, por crear a Eday y por el resto de ilustraciones.
- A Gelsys, Claudia y Celia, por sus revisiones, maquetaciones y portada.
- A resto del equipo de Anaya Multimedia, y a todo el personal involucrado en la impresión, distribución, exposición, reposición, venta, recomendación, reparto… sin ellos no llegaríamos a las librerías ni a tus manos.

Sobre la autora



Mariona Nadal es ingeniera informática por la Universidad Politécnica de Madrid (2003).

Cuenta con más de 20 años de experiencia en la práctica del desarrollo de aplicaciones Java en entornos empresariales para grandes clientes internacionales, experiencia que le ha permitido ser formadora en J2EE de jóvenes ingenieros en su primer empleo como programadores.

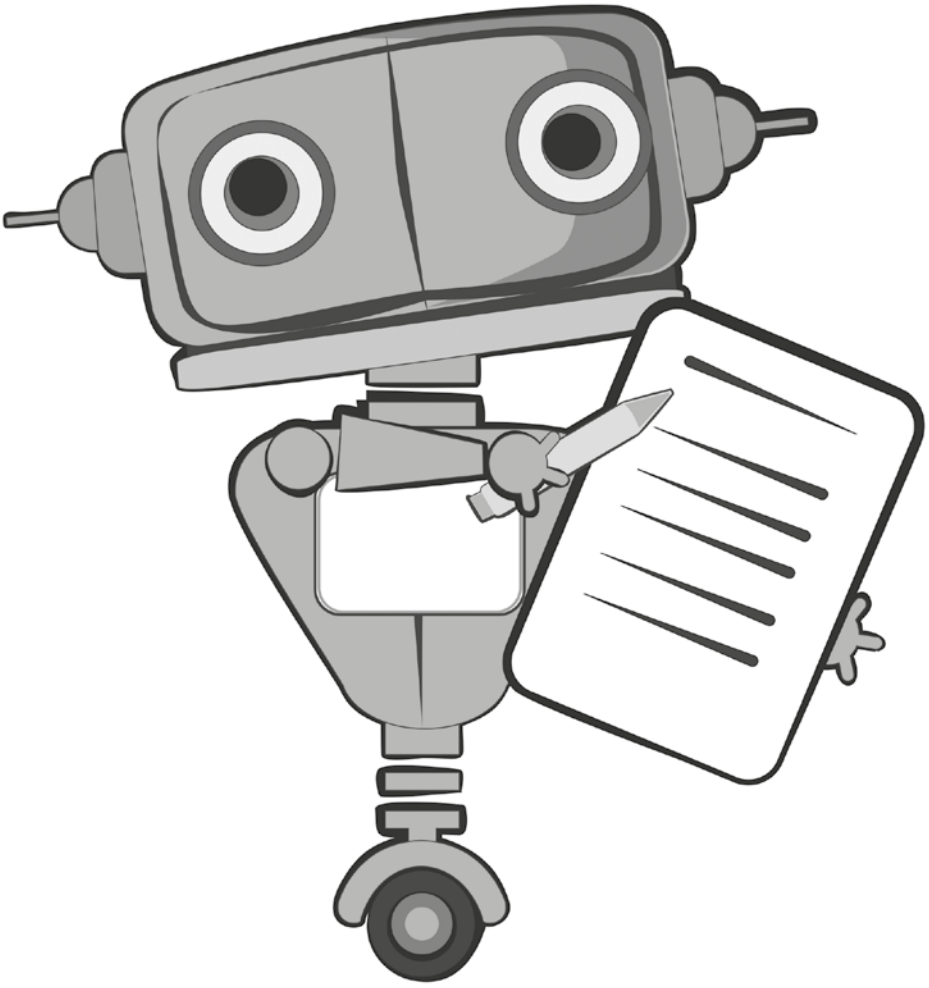
Actualmente es profesora de programación en grados superiores de Formación Profesional. También es instructora de LinkedIn Learning, donde cuenta con un número creciente de cursos sobre Fundamentos de la Programación y Java. Su primer libro fue el *Manual Imprescindible. Curso de Programación Java*, publicado en junio de 2021.

Su metodología fresca, directa y realista hace sus cursos amenos, claros y útiles, llevándote a través de la práctica a un estilo de programación de fácil mantenimiento y alta empleabilidad.

En http://school.dsrroma.es encontrarás todos sus cursos y más información.

Web DSR School
bit.ly/dsrschoool

Cursos en LinkedIn Learning
bit.ly/lilmnadal



Índice de contenidos

Cómo usar este libro 12
Prólogo 15

Parte 1. Estructuras de datos

1 **Introducción a las estructuras de datos** 18

Estructuras de datos y tipos abstractos de datos 19
Introducción a la complejidad 20
Análisis y elección de una estructura de datos 22
Test y ejercicios 24
Soluciones 25

2 **Tipos abstractos de datos** 26

Tipos básicos 27
Tuplas o registros 29
Listas 31
Conjuntos 33
Pilas y colas 35
Diccionarios 37
Árboles 39
Grafos 42
Iteradores 45
Test y ejercicios 46
Soluciones 48

3 **Estructuras de datos** 50

Introducción 51
Listas estáticas 54
Listas dinámicas 57
Operaciones sobre listas 61
Tablas hash 65
Árboles 69
Grafos 71
Test y ejercicios 74
Soluciones 76

4 **Colecciones en Java** **78**

Introducción 79
ArrayList 80
HashSet 90
HashMap 99
TreeSet 113
Test y ejercicios 126
Soluciones 126

Parte 2. Algoritmos

5 **Introducción a los algoritmos** **130**

Introducción 131
Características de un algoritmo 132
Metodología de creación de algoritmos 133
Lenguajes para la definición de algoritmos 135
Estructuras básicas en los algoritmos 136
Test y ejercicios 138
Soluciones 139

6 **Caja de herramientas** **141**

Introducción 142
Variables y constantes 142
Bifurcaciones 143
Iteraciones 144
Funciones y métodos 146
Recursividad 148
Complejidad 149
Test y ejercicios 150
Soluciones 151

7 **Algoritmos muy simples** **153**

Introducción 154
Algoritmo: Mínimo de tres números 154
Algoritmo: Sucesión de Fibonacci 162
Test y ejercicios 166
Soluciones 167

8 **Algoritmos de ordenación** **170**

Introducción 171
Algoritmo: Ordenación por inserción (insertion sort) 171
Algoritmo: Ordenación de burbuja (bubble sort) 175
Algoritmo: Ordenación por mezcla (merge sort) 178
Algoritmo: Ordenación rápida (quicksort) 189
Test y ejercicios 197
Soluciones 198

9 **Algoritmos de búsqueda** **199**

Introducción 200
Algoritmo: Búsqueda lineal o secuencial 201
Algoritmo: Búsqueda binaria o dicotómica 203
Test y ejercicios 210
Soluciones 210

10 **Algoritmos de texto** **212**

Introducción 213
Algoritmo de cifrado César 213
Algoritmo de búsqueda de cadenas Boyer-Moore 218
Algoritmo de compresión Huffman 227
Test y ejercicios 248
Soluciones 249

11 **Algoritmos matemáticos** **252**

Introducción 253
Algoritmo de Euclides (máximo común divisor) 253
Criba de Eratóstenes (números primos) 257
Algoritmo de Karatsuba (multiplicación de números grandes) 263
Test y ejercicios 267
Soluciones 268

Índice alfabético **269**

Cómo usar este libro

A quién va dirigido y qué es necesario para empezar

Este libro se dirige a estudiantes de programación y a curiosos de la informática que quieren afianzar sus conocimientos de programación y consolidar dos piezas básicas de la misma: las estructuras de datos y los algoritmos.

Poco importa qué lenguaje o lenguajes conozcan o quieran aprender, aunque los ejemplos de código incluidos en este libro son básicamente en Java, JavaScript o Python, o incluso alguno en Haskell, los conocimientos adquiridos son aplicables a cualquier otro lenguaje, siempre acompañados de pseudocódigo y diagrama de flujo independientes del lenguaje.

Para seguir los ejemplos y los ejercicios del libro necesitarás preferiblemente un ordenador con:

- Un IDE (entorno de desarrollo integrado) adecuado para el lenguaje elegido, que es un editor de texto adaptado para programadores que integra muchas funcionalidades útiles, como la de compilar y ejecutar el código. No es imprescindible, pero, según van creciendo los proyectos, se agradece tenerlo. En este libro, para Java, he utilizado Eclipse IDE, pero, si prefieres otro, también te sirve. Puedes descargar Eclipse en eclipse.org.
- Acceso a internet y navegador web para utilizar compiladores online de los distintos lenguajes. Yo he utilizado:
  - Para Python: https://www.programiz.com/python-programming/online-compiler/.
  - Para JS: https://js.do.
  - Para Haskell: https://paiza.io.
  - Para Blockly: https://developers.google.com/blockly.

Estructura del libro

Esta obra se divide en dos partes: la primera dedicada a las estructuras de datos y la segunda a los algoritmos. Aunque puedes realizar una lectura lineal del libro, probablemente su uso más habitual sea ir directamente a la estructura de datos o al algoritmo que te interese en cada momento.

En la parte 1, empezamos en el primer capítulo con una introducción a las estructuras de datos, para profundizar en el capítulo 2 en los tipos abstractos de datos y en el 3 en las estructuras de datos que los implementan. El cuarto capítulo se dedica a las colecciones de Java, algunos conceptos serán aplicables a otros lenguajes y otros no.

La segunda parte también empieza con una introducción, en este caso a los algoritmos. El capítulo 6, la caja de herramientas, te permitirá conocer las bases para trabajar con algoritmos. Los capítulos siguientes están dedicados cada uno a un tipo de algoritmos, empezando por unos muy simples, luego los de ordenación, después los de búsqueda, para terminar con los algoritmos sobre textos y los matemáticos.

Convenios utilizados en este libro

Para facilitar la comprensión de este manual se han utilizado algunos formatos especiales:

- Los nombres de comandos, menús, opciones, cuadros de diálogo y otros elementos aparecen en letra «de palo seco» para distinguirlos fácilmente del resto del texto, por ejemplo, la instrucción System.out.println.
- Las combinaciones de teclas aparecen separadas por un guion y también en un tipo de letra diferente, por ejemplo, Ctrl-Mayús-O.
- Los fragmentos de código los verás como textos de ancho fijo, al igual que los resultados de la ejecución de los programas. El color del texto te dará la pista para saber de qué tipo se trata:

```
fragmento de código
resultados de ejecución
```



- Los test y ejercicios se encuentran al final del capítulo, pero en el momento en el que ya sabes lo imprescindible para resolverlos, encontrarás estos iconos que referencian el número de test (blanco sobre negro) o de ejercicio (negro sobre blanco) que ya puedes intentar.
- A lo largo del libro aparecen notas informativas separadas del texto principal que proporcionan información, tales como aclaraciones, advertencias, curiosidades o consejos:

> Nota: Para facilitar o concretar información relacionada con el tema abordado, incluyen recomendaciones que conviene tener en cuenta.

Resumen

Breve resumen para afianzar el concepto recién explicado.

Información de soporte

Los ejemplos, ejercicios y proyectos desarrollados en este libro están disponibles en un repositorio git de acceso público: https://github.com/DSRschool/Anaya-Eday.



Repositorio en git

bit.ly/gitEday

Puedes descargarte todo en un fichero zip, pero te recomiendo aprender las operaciones básicas de git para sacarle el máximo provecho.

Algunas nociones para empezar a utilizar el repositorio:

- La interfaz de gitHub puede cambiar sin previo aviso, pero además de ver el código en la propia web, mediante el botón Code es posible clonarlo para tener tu propia versión del repositorio o descargártelo como zip.
- Viendo un fichero en la web, el botón History te permitirá conocer todos los *commits* hechos sobre ese fichero. Cuando cojas una versión en concreto, verás con el fondo blanco las líneas de código sin cambios respecto a las anteriores, con fondo rojo y un signo menos, las líneas que estaban antes y ya no están, y con fondo verde y un signo más, las nuevas. Si una línea ha sido modificada, aparecerá la versión antigua en rojo y la nueva en verde.

Recomendaciones y buenas prácticas

Durante toda la obra te llevo de la mano para ir aprendiendo nuevos conceptos, te propongo ejemplos y ejercicios para afianzar esos conocimientos, pero ¡hay que ser valiente! ¡Suéltate de mi mano y avanza más! Si te muestro un ejemplo que calcula el máximo, intenta por tu cuenta calcular el mínimo. Si durante el ejercicio se te ocurren mejoras o retos, ¡hazlos! El espacio en el libro es limitado, tu capacidad de aprendizaje es infinita.

Puede parecer extraño, pero conviene que tengas papel y lápiz a mano, poder tomar apuntes, hacer esquemas y diagramas ayuda, y mucho, a la hora de programar. Y es una herramienta casi imprescindible para probar los algoritmos.

Tienes todos los ficheros disponibles para descargar, pero si de verdad quieres aprender a programar, escríbelos tú. Como los cuadernos de caligrafía de la escuela, escribir letra a letra los programas te llevará a asimilar lo que estás escribiendo. Olvídate de copiar y pegar hasta que no sepas perfectamente qué estás copiando y pegando.

Para generar buenos programas, debemos ordenar nuestra mente, pero también nuestros ficheros. Clasifícalos bien en carpetas, da nombres significativos tanto a las carpetas como a los ficheros, de forma que te resulte fácil recuperar lo que estés buscando.

Contacta conmigo

Tus sugerencias, comentarios, correcciones, propuestas… son una fuente de crecimiento para mí y mis contenidos. Puedes escribirme a school@dsrroma.es. Procuraré contestar tan pronto como las circunstancias tengan a bien. También puedes seguirme en Twitter (@MarionaJava) o en LinkedIn (marionanadal) para estar al día de mis nuevas publicaciones.

Prólogo

Quien se adentra en el desarrollo de software necesita un sólido conocimiento de los bloques básicos que sirven de fundamento a esta profesión. Estos bloques son las estructuras de datos que son objeto de manipulación en los programas que creamos y los algoritmos mediante los cuales se produce dicha manipulación. Un conocimiento profundo de ambos es más importante que las particularidades de uno u otro lenguaje de programación, puesto que estos son solo la representación de aquellos en una forma concreta.

En esta obra, Mariona presenta de una forma a la vez amena y rigurosa los principales algoritmos y estructuras de datos que son imprescindibles para un profesional de la informática o un estudiante que aspire a serlo algún día y que deben manejar con soltura. Mariona, gracias a su experiencia en el desarrollo de software, tiene una profunda comprensión de estas materias y, gracias a sus años de dedicación a la enseñanza, es capaz de transmitirla de una manera muy eficaz.

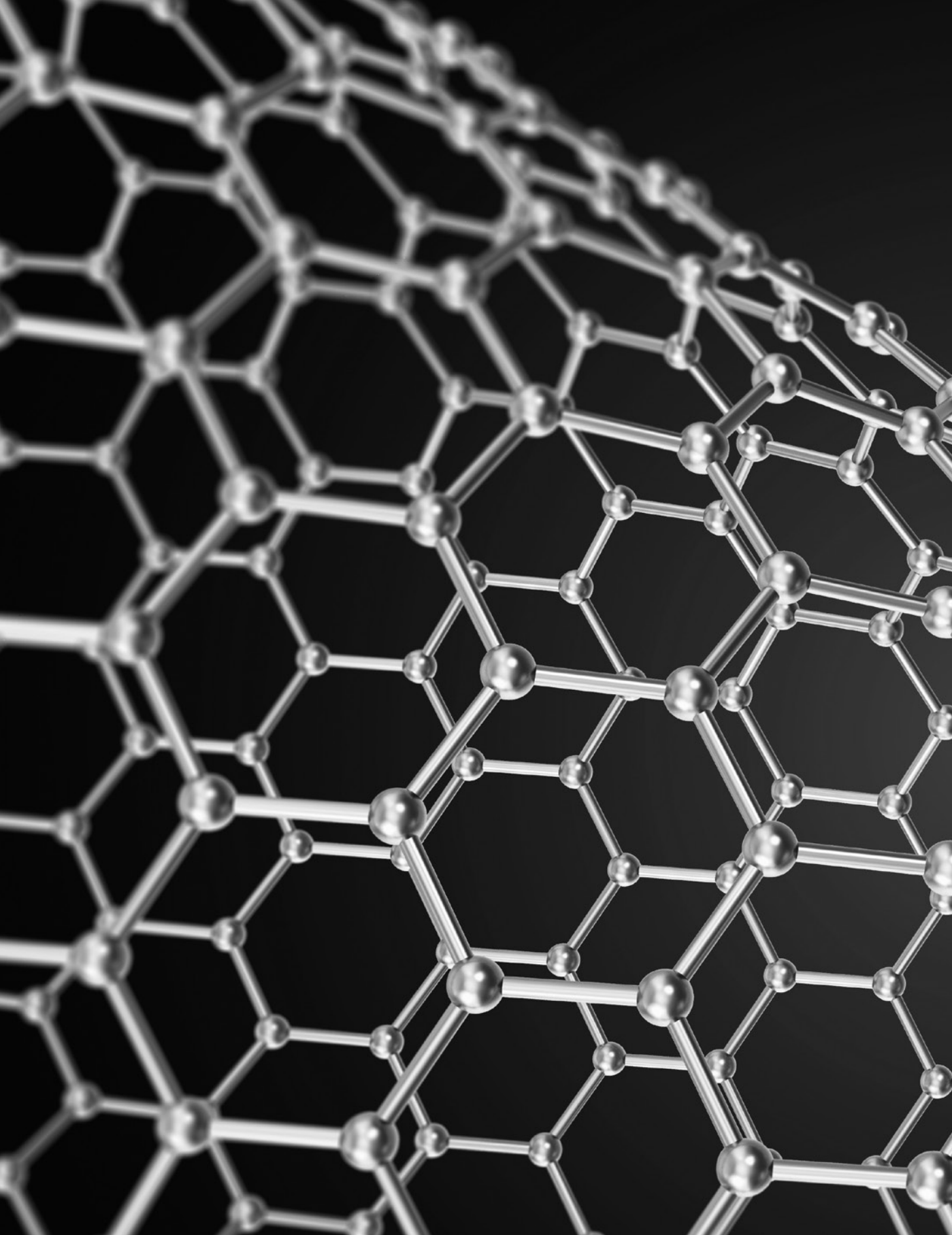
En el libro se asume que el lector tiene un conocimiento básico de los principios de la programación de ordenadores, pero no establece requisitos específicos de lenguajes o tecnología, lo cual lo hace apto para profesionales o estudiantes provenientes o enfocados en distintas áreas. Los ejemplos en el libro están dados en diferentes lenguajes de programación con lo que se pueden comparar distintas implementaciones de una misma solución y son fácilmente comprensibles incluso si no se es experto en dichos lenguajes.

La estructura de este libro comprende una parte inicial en la cual se tratan las estructuras de datos y una segunda dedicada a los algoritmos. En la primera parte, la autora explica en detalle las ideas que subyacen en las estructuras de datos que más utilizamos quienes nos dedicamos al desarrollo de software para hacer una representación de la realidad que nos permita atacar de una forma adecuada los problemas que debemos resolver en el desempeño de nuestra profesión. El conocimiento de las diferentes estructuras de datos y la comprensión de cuándo y por qué utilizar cada una de ellas es parte fundamental para la construcción de programas que sean a la vez eficientes, comprensibles y de fácil mantenimiento.

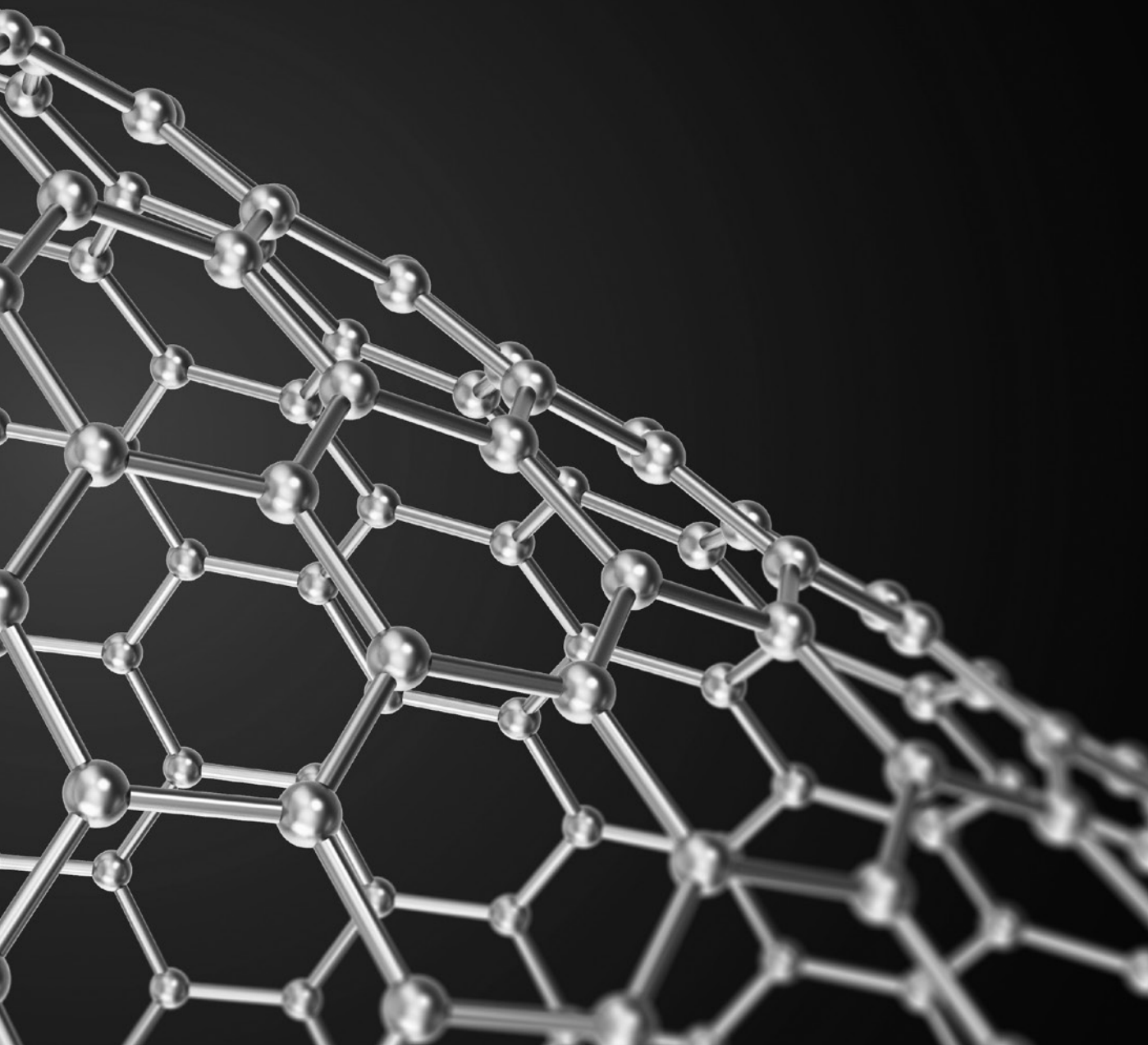
En la segunda parte del libro se tratan los algoritmos, partiendo de una presentación conceptual de los mismos, su definición y características generales, continuando con el estudio de diferentes algoritmos partiendo de los más simples y subiendo paulatinamente en complejidad. La función del aprendizaje de estos algoritmos no es necesariamente su utilización en la práctica, puesto que el desarrollador ya podrá encontrar bibliotecas que los implementan, sino ayudar a la comprensión de lo que sucede entre bambalinas cuando usamos estas bibliotecas y sobre todo estimular el hábito de atacar con método la resolución de problemas: lo importante no es el destino sino el camino.

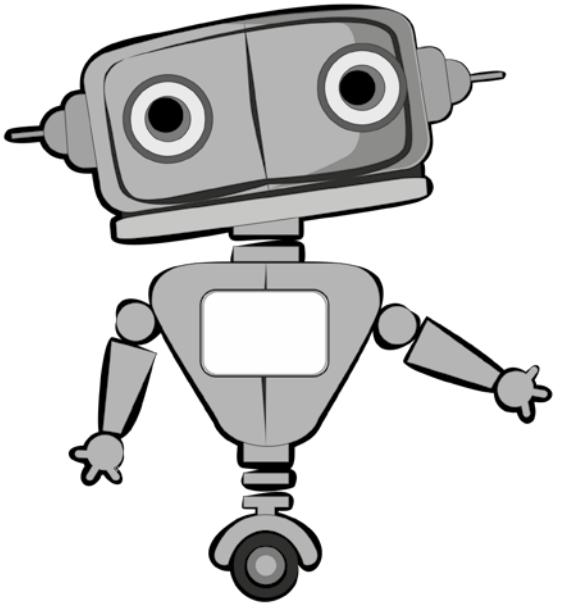
Cada capítulo tiene además una serie de ejercicios para ayudar al lector a fijar las ideas y evaluar su comprensión. En definitiva, un estudio cuidadoso de este libro ayudará en gran modo a los lectores a desarrollar las habilidades necesarias para resolver adecuadamente los problemas que se le presentarán en el desarrollo de esta profesión al tiempo que le ayudarán a comprender los retos que se plantean al resolver de forma eficiente un problema informático.

Pablo C. Lucien



Estructuras de datos





01

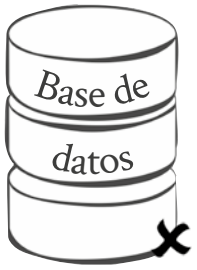
Introducción a las estructuras de datos

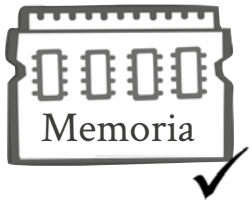
En este capítulo aprenderás a:

- Diferenciar entre estructuras de datos y tipos abstractos de datos.
- Identificar nociones de complejidad.
- Escoger tipos abstractos de datos y/o estructuras de datos.

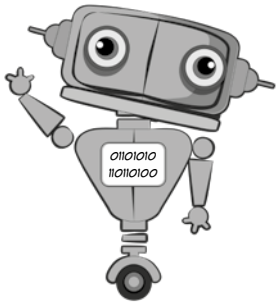
Estructuras de datos y tipos abstractos de datos

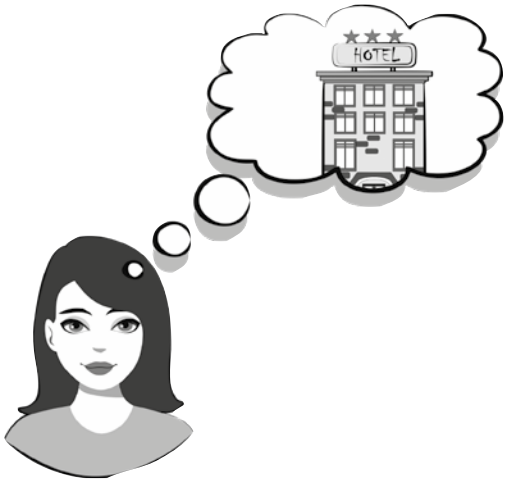
En este libro conoceremos varias formas de organizar los datos que maneja nuestro programa. No estamos hablando de los datos que persistirán en base de datos, es decir, que seguirán ahí almacenados esté ejecutándose o no nuestro programa, sino de los datos que un programa necesita tener en memoria durante su ejecución.



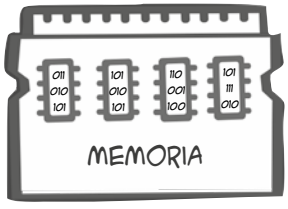


Aunque no quiero adentrarme en detalles demasiado técnicos, cada programa en memoria dispone de una zona de memoria para almacenar sus datos de trabajo. Una zona de memoria no es más que un espacio en el que se pueden almacenar ceros y unos porque, en esto de la informática y la programación, todo son ceros y unos. Los ordenadores funcionan con ceros y unos, pero los humanos necesitamos representar y organizar los datos de forma distinta.





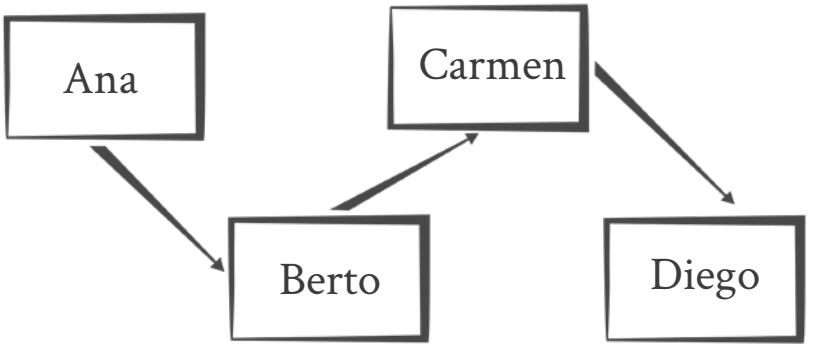
Cuando tenemos muy pocos datos que manejar, podemos usar los tipos básicos, o quizá registros o tuplas, pero para grandes cantidades de datos, hay que manejar estructuras de datos más complejas, como listas, conjuntos, diccionarios…



Hablamos de estructuras de datos cuando nos referimos a la organización, digamos, física, de los datos en memoria.

Cuando abordemos cada **estructura de datos**, estudiaremos cómo sería su representación en memoria, así como las operaciones que podemos hacer sobre ella y qué costes tienen.

Pero a un nivel más alto, antes debemos aprender algunos **tipos abstractos de datos**. Los tipos abstractos de datos son un conjunto de datos a los cuales se les asocian unas operaciones. Nos ofrecen una interfaz que define cómo acceder a esos datos o cómo manipularlos.





Si las estructuras de datos hemos dicho que eran la organización física de los datos, los tipos abstractos de datos nos mostrarán la organización lógica de estos. Un mismo tipo abstracto puede ser implementado usando distintas estructuras de datos. Del mismo modo, algunas estructuras podrían usarse para varios tipos abstractos. No te preocupes, vamos a verlos por separado, pero, a la vez, estudiaremos sus vínculos.

Introducción a la complejidad

El cálculo de la complejidad de los algoritmos y operaciones de un programa es toda una ciencia. Así que mejor no nos meternos en exceso en detalle, porque no quiero que cierres ya el libro. Pero es preciso que introduzcamos un poco la idea.

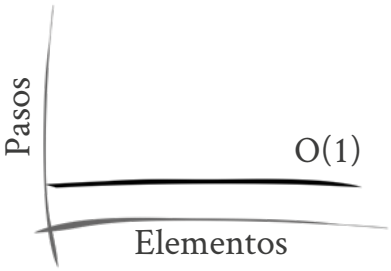
Tenemos varios tipos abstractos de datos con ciertas operaciones para acceder y manejar a los datos, y distintas estructuras de datos para implementarlos. Y en función de cómo implementemos, por ejemplo, una lista, el coste de cada operación que podemos hacer sobre una lista varía.

Siguiendo con el ejemplo de la lista, operaciones típicas sobre ella son añadir o encontrar un elemento. Contaremos con implementaciones en las que quizá añadir un elemento tenga una solución muy directa, pero encontrarlo requiera una búsqueda elemento a elemento. También podemos tener una implementación en la que el coste de añadir un elemento sea elevado, pero encontrarlo sea muy fácil.

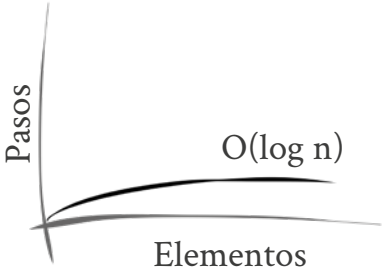
Pero la programación es una ciencia, y eso de decir que es «fácil» o «difícil» de encontrar es poco científico, poco preciso. El cálculo de la complejidad nos permite ponerle una cifra al coste de cada operación, y así comparar costes entre distintas operaciones o entre las distintas implementaciones de una misma operación. Esta cifra no se da como una unidad de tiempo (3 s), sino como un número de operaciones, pues el tiempo dependerá de las características de la computadora en la que se ejecute. Además, no se busca una cifra exacta, sino un «orden» de complejidad, es decir, si es de decenas, centenas, miles, millones… de operaciones.

Principalmente, podemos hablar de cinco niveles de complejidad. Iremos de menor a mayor, de más rápido a más lento:

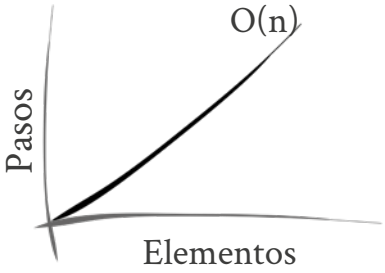
- **Complejidad constante, cuya notación es O(1):** el coste de una operación no depende del tamaño de nuestra estructura de datos, se trata de un acceso directo. Por ejemplo, cuando para acceder a los datos de la posición siete, podemos ir directamente a la posición siete y consultarlos. O cuando para añadir un elemento, nos vamos al final de nuestra estructura directamente, y lo añadimos. Poco importa el tamaño de la estructura, el número de datos que tengamos.



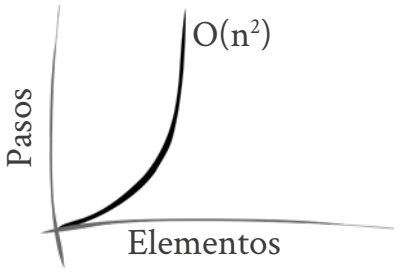
- **Complejidad logarítmica, O(log n):** la complejidad crece con el tamaño de los datos de entrada, pero de forma, digamos, moderada. Si hacemos una búsqueda binaria de un elemento en una lista ordenada, como en cada iteración o recursión usamos la mitad de la lista: si la lista inicial tiene ocho elementos, partiremos y tendremos cuatro, partiremos y tendremos dos, partiremos y tendremos uno, así que en cuatro pasos resolvemos. Si la lista inicial tiene 30 elementos, partiremos, tendremos 15 elementos, luego ocho, y así seguimos, así que aunque la lista de 30 es mucho más larga que la de ocho, solo necesitaremos dos pasos más.

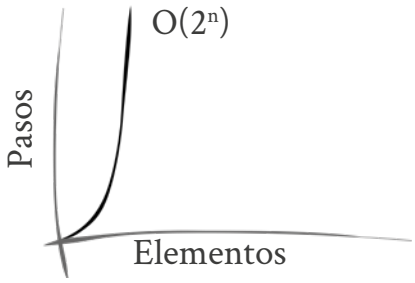


- **Complejidad lineal, O(n):** los pasos crecen de forma lineal en función del tamaño de los datos. Por ejemplo, en una lista enlazada, en la que cada elemento apunta al siguiente, para contar cuántos elementos tenemos, necesitamos recorrer la lista, elemento a elemento.



- **Complejidad cuadrática, $O(n^2)$, o polinómica $O(n^a)$:** los pasos crecen de forma cuadrática, cúbica… Hallamos complejidad cuadrática en algunos algoritmos de ordenación simples y la cúbica en algunas de las operaciones sobre matrices, como la multiplicación. En la complejidad cuadrática, dependerá de la cantidad de filas y columnas de nuestra estructura. En una matriz de tres por dos, el coste sería de seis (unidades sin definir), pero si la matriz es de cien por cincuenta, ¡el coste se nos dispararía hasta cinco mil!
- **Complejidad exponencial, $O(a^n)$:** si alguna estructura de datos presenta una complejidad exponencial en alguna de sus operaciones… quizá deberíamos plantearnos buscar otra implementación. Un ejemplo sería la partición de conjuntos.

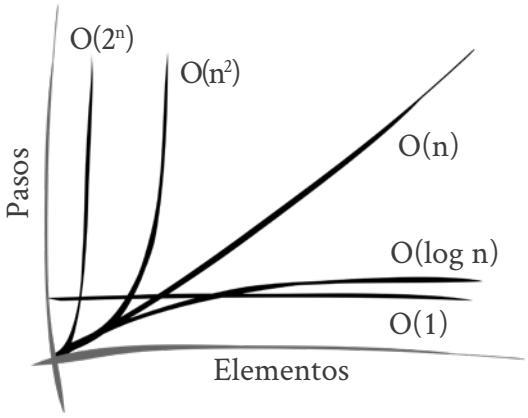




T1.2

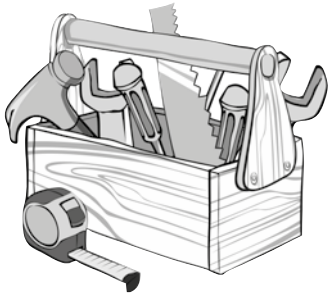
E1.1

Lo importante sobre la complejidad es que te quedes con la idea que mejor la constante que la lineal, mejor la lineal que la cuadrática, y saber más o menos de qué tipo es una operación.



Análisis y elección de una estructura de datos

A lo largo del libro veremos varios tipos abstractos de datos y estructuras de datos para implementarlos. El objetivo es que conozcas su existencia, sus características y así los añadas en tu caja de herramientas de programador. Es necesario que domines estos tipos abstractos de datos y estas estructuras de datos para que, llegado el momento, sepas escoger cuál es la herramienta adecuada en cada problema que debas resolver.

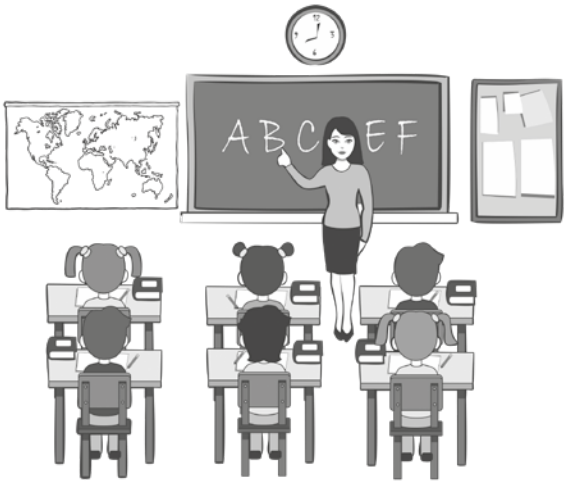


Por cada **tipo abstracto de datos**, veremos cuál es su propuesta para organizar los datos y qué operaciones ofrece sobre los mismos. Por tanto, has de tener en cuenta si esa organización y esas operaciones se adaptan a tu situación. Por ejemplo, habrá casos que admiten duplicados y otros que no, otros que tienen orden o que no, que se pueden recorrer o que solo nos dan un dato a la vez, que cuentan con clave y valor o solo con valor…

En cuanto a las **estructuras de datos**, por cada una conoceremos, a parte de su organización física interna, el coste computacional de cada una de sus operaciones. Porque en función de cómo se organicen los datos en memoria será más directo (complejidad constante) o menos (complejidad lineal) acceder a un dato en concreto, o contar cuántos elementos hay, o insertar un dato nuevo al principio, al final o en medio.

Cuando nos enfrentemos a un problema por resolver, estudiaremos qué operaciones más frecuentes vamos a ejecutar sobre esa estructura y qué operaciones usaremos solo de forma puntual. De esa manera, ponderamos el peso de cada operación, y decidimos la más adecuada. Buscaremos que las más usadas sean lo más rápidas posible, pero toleraremos que aquellas operaciones más ocasionales puedan ser un poco más pesadas.

Veámoslo con un par de ejemplos.



Los más vendidos de Robotalia

CURSO DE PROGRAMACIÓN JAVA
Curso de Programación Python
Curso de programación C/C++
CURSO DE PROGRAMACIÓN ANDROID CON KOTLIN

Piensa en la lista de alumnos de una clase de primaria y también en la lista de productos más vendidos de una web.

La lista de alumnos de primaria se construye a principios de curso y, ocasionalmente, durante el año escolar, puede sufrir algún alta o baja. Puede que el nuevo alumno se añada en su posición según el orden alfabético de los apellidos, pero también puede ser que se opte por añadirlo al final de la lista, para no cambiar su número de lista al resto de las criaturas. Sin embargo, los valores asociados a cada estudiante serán modificados casi a diario por los maestros, para controlar su asistencia o registrar los resultados de sus trabajos y exámenes. Por otro lado, en el caso de la lista de productos más vendidos de una web, estaremos añadiendo y quitando productos de forma casi constante, y también reordenando la lista, no sé si a cada hora, pero sí seguramente al menos cada día.

Para implementar la lista del alumnado de primaria, buscaremos una estructura que nos ofrezca un acceso rápido a los datos de cada alumno, pero no nos importará que la construcción inicial de la lista o sus modificaciones con las altas o bajas puntuales que se puedan dar durante el curso tengan un coste un poco mayor. Por el contrario, en el caso de la lista de más vendidos, preferiremos una estructura de datos que admita algoritmos de ordenación muy rápidos y, a su vez, también querremos que el añadir o quitar productos de la lista tenga un coste bajo. Quizá poco nos importe el coste de calcular la longitud de la lista, pues siempre habrá la misma cantidad de productos en la lista. Así pues, en situaciones distintas, escogeremos soluciones diferentes, fijándonos en especial en el coste de las operaciones que serán más frecuentes sobre los datos que vamos a tratar. ¡Evitaremos usar siempre la misma solución porque sea la única que dominamos!



Test y ejercicios

Test 1.1: ¿Cuál es la diferencia entre estructuras de datos (ED) y tipos abstractos de datos (TAD)?

a) Las ED son representaciones lógicas y los TAD son representaciones físicas.
b) Las ED son de almacenamiento permanente y los TAD son de almacenamiento durante la ejecución del programa.
c) Los TAD son representaciones lógicas y las ED son representaciones físicas.
d) Los TAD son de almacenamiento permanente y las ED son de almacenamiento durante la ejecución del programa.

Test 1.2: Ordena de más lenta a más rápida las siguientes complejidades: lineal, constante, logarítmica, cuadrática.

a) lineal > constante > logarítmica > cuadrática
b) cuadrática > lineal > logarítmica > constante
c) cuadrática > logarítmica > lineal > constante
d) constante > logarítmica > lineal > cuadrática

Test 1.3: Tengo un listado en el que añado y suprimo elementos con frecuencia, pero solo necesito ordenarlos al hacer el cierre del mes. ¿Qué combinación me interesa más?

a) Complejidad constante para la ordenación, lineal para la inserción y logarítmica para la supresión.
b) Complejidad cuadrática para la ordenación, lineal para la inserción y logarítmica para la supresión.
c) Complejidad cuadrática para la ordenación, constante para la inserción y logarítmica para la supresión.
d) Complejidad logarítmica para la ordenación, lineal para la inserción y lineal para la supresión.

Ejercicio 1.1: Número de operaciones según la complejidad.
Calcula el número de operaciones sobre 10, 100 y 1000 elementos para operaciones con las siguientes complejidades:

a) O(n)
b) O(1)
c) $O(n^2)$
d) O(log n)

Soluciones

Test 1.1: ¿Cuál es la diferencia entre estructuras de datos (ED) y tipos abstractos de datos (TAD)?

c) Las TAD son representaciones lógicas y los ED son representaciones físicas.

Test 1.2: Ordena de más lenta a más rápida las siguientes complejidades: lineal, constante, logarítmica, cuadrática.

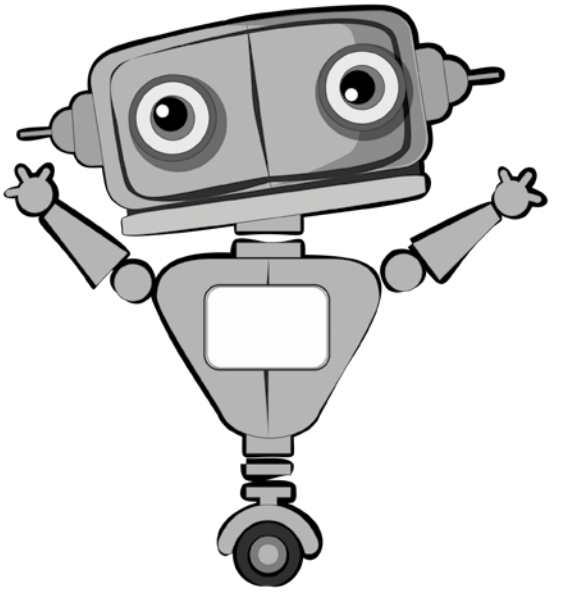
b) cuadrática > lineal > logarítmica > constante

Test 1.3: Tengo un listado en el que añado y suprimo elementos con frecuencia, pero solo necesito ordenarlos al hacer el cierre del mes. ¿Qué combinación me interesa más?

c) Complejidad cuadrática para la ordenación, constante para la inserción y logarítmica para la supresión.

Ejercicio 1.1: Número de operaciones según la complejidad.

a) O(n) = 10 / 100 / 1000
b) O(1) = 1 / 1 / 1
c) $O(n^2)$ = 100 / 10 000 / 1 000 000
d) O(log n) = 1 / 2 / 3



02

Tipos abstractos de datos

En este capítulo aprenderás a:

- Identificar los tipos abstractos de datos más comunes y sus operaciones más típicas.
- Entender mejor cada TAD gracias a ejemplos «de la vida real».

Tipos básicos

Podríamos discutir largo y tendido sobre si los tipos básicos son tipos abstractos de datos o no, pero la verdad es que me importa poco. Lo importante es conocer los tipos básicos, pues serán la base del resto de estructuras. De un lenguaje a otro puede que varíen. De hecho, varían.

Lo veremos de forma genérica, pero cuando trabajes con un lenguaje en concreto deberás fijarte en qué tipos de datos básicos te ofrece y en qué matices tiene cada uno para, de nuevo, escoger el más adecuado. Aunque, por cierto, hay lenguajes de programación que esto de los tipos de datos… bueno, no los manejan mucho. O al menos lo hacen de forma transparente para el programador. En esos lenguajes, cuando declaras una variable no hay que decir de qué tipo es y, además, ahora puedes inicializarla como un número y luego cambiarle el valor y darle un valor booleano.

Tabla 2.1. Lenguajes sin tipos vs. lenguajes fuertemente tipados.

En lenguajes sin tipos	En lenguajes fuertemente tipados

`var a`
`a = 6`
`a = true`
	`int n = 2`
	`boolean b = true`
	`n = 7`
	`b = false`

Mi preferencia personal son los lenguajes fuertemente tipados, en los que una variable numérica es una variable numérica y una variable booleana es una variable booleana. Lo defines en el momento de la declaración y lo mantienes durante todo su ámbito de uso. Quizá mi devoción por los lenguajes fuertemente tipados no es racional y viene solamente del hecho de que aprendí a programar en ese tipo de lenguajes, y casi siempre he trabajado con ellos. Pero en serio, pensándolo, evaluándolo a consciencia, creo que es cierto que el código es más legible y mantenible cuando los tipos están bien definidos y además se usan coherentemente.

- **Enteros:** números sin decimales, es decir, como -3, 8, 0, 1…

  Algunos lenguajes pueden ser aún más concretos y definir también los tipos:

  - **Naturales:** el 0 y los enteros positivos.
  - **Positivos:** a partir del uno, es decir, los naturales sin incluir el cero.

Por supuesto, si el lenguaje que estás usando te permite ser concreto, sé concreto. Si tienes un campo «edad» y puedes limitar sus valores a los números naturales, te ahorrarás los problemas que pueda generar tener una edad negativa. Es más, si el campo «edad» es para una escuela primaria, no creo que admita bebés de menos de un año, así que quizá convendría escoger un tipo de número positivo.