Capítulo 1

Introducción

Este capítulo introduce el concepto de la algoritmia desde un enfoque histórico y propone un contexto de estudio y los requisitos aconsejables para un buen aprovechamiento del texto.

1.1 El nacimiento de la programación

El desarrollo de programas como disciplina nace tras la aparición de los primeros lenguajes de programación en la década de los 50. En esta primera fase, la programación era dependiente de los procesadores y los conceptos de control de flujo y las estrategias algorítmicas eran bastante difusos. En esta fase temprana, la producción de programas estaba basada en la prueba y error, y las estrategias para la resolución algorítmica de ciertos problemas no tenían todavía un lenguaje de especificación de código con suficiente nivel de abstracción. Sin embargo, entre 1950 y 1960 aparecen varios lenguajes de programación como ALGOL y FORTRAN que permiten identificar elementos de diseño algorítmico, muchos de los cuales han pervivido hasta nuestros días.
Más tarde, a finales de la década de los sesenta, la forma tradicional de producción de los programas en base a un primer diseño seguido de una batería de pruebas con juegos de datos de diversos tipos, fue cuestionada fuertemente y se constató la falta de una metodología de desarrollo y verificación de los programas que hacía que éstos no fueran fiables, lo que constituyó una barrera importante para el arte de la programación.
Como consecuencia de estas reflexiones surgen en aquel momento dos líneas de investigación paralelas:

- Una línea de desarrollo de los métodos de programación que produjeran procesos manejables mentalmente y por tanto sencillos de entender y validar.

- Una línea de desarrollo de métodos de verificación de programas cuyo objeto era la formalización de procedimientos que permitieran decidir que un programa sa-

tisface las especificaciones por estudio de su estructura y no por las técnicas de prueba.

La primera de ellas introdujo la noción de programación estructurada, el concepto de módulo y, con el tiempo, la noción de objeto; la segunda desarrolló las bases axiomáticas para la verificación de programas y también en menor medida para la producción de programas en función de una especificación previa.

1.2 Algoritmos y programación estructurada

La programación estructurada constituyó una traslación a la programación de las técnicas generales de resolución de problemas (mecanizadas a partir de la idea de N. Simon y Newell con el General Problem Solver [NS63]) pero también, a partir de estas ideas, comienza a evolucionar la noción del diseño de programas como un proceso de ingeniería.

Las aportaciones de Edgser W. Dijkstra, C.A.R. Hoare, O.J. Dahl y N. Wirth, entre otros, introdujeron a finales de los años 60 y principios de los 70 la noción de programación *estructurada* [Dij68, Dij72, Wir71, DDH72], planteando abstracciones estructurales en el código, que hasta entonces se veía como una mera colección lineal de asertos, lo que supuso la primera evolución hacia el desarrollo disciplinar de programas. Fue con estas abstracciones como la la programación modular y las técnicas de descomposición de módulos desarrollada por D.L. Parnas [GP70, Par72] como se produjo la primera revolución conceptual de la programación, precursora de la segunda que fue la introducción de la programación orientada a objetos, seguida a su vez por la programación basada en patrones de diseño.

A comienzos de los setenta se gesta también otra revolución conceptual, esta vez acerca de la representación y especificación de tipos de datos. Alrededor del concepto de *tipo abstracto de datos* [Gut75] evolucionan las ideas de modularidad, genericidad, esquemas genéricos de resolución de problemas, formalización del concepto de implementación y especificación formal de tipos de datos; ideas, todas ellas, armónicas con el diseño descendente y la programación estructurada. Se trata de distinguir entre la especificación de un tipo de datos, o su comportamiento observable, y sus implementaciones. Los mecanismos de encapsulamiento para aislar el acceso y manipulación de tipos de datos de su implementación se incorporaron rápidamente a lenguajes de programación como Pascal y Modula-2 [Wir82] entre otros.

También a mediados de los 70 la algorítmica adquiere forma de disciplina dominante en el campo de la informática con el desarrollo de muchos lenguajes de alto nivel y de métodos formales de desarrollo, que trajeron también la necesidad de analizar formalmente la complejidad de los algoritmos con la introducción del concepto de tratabilidad y de la distinción entre el crecimiento exponencial y el crecimiento polinomial por los trabajos de A. Cobhan y J Edmonds [Cob65]. Finalmente es Knuth quien introduce la

notación O, Θ y Ω [Knu76], la cual permite facilitar enormemente el estudio formal de la eficiencia de los algoritmos, y que ha llegado hasta nuestros días.
Otro de los campos que han influido en la algorítmica aunque en menor medida es el de la verificación de programas, que viene a recoger la aserción de Edger W. Dijkstra de que los casos de prueba sirven para detectar la presencia de errores, pero no para asegurar su ausencia. En esta línea de razonamiento y buscando métodos de validación sistemática se posicionaron a mediados de los años 60 John McCarthy, P. Naur y sobre todo R. Floyd [McC63, Flo67]. Aunque el origen de la noción de verificación se puede remontar a Alan Turing [Tur49], fue Robert Floyd el primero en proponer el desarrollo de un método sistemático de verificación de algoritmos con la publicación en 1967 de su artículo "*Assigning Meanings to Programs*", en el que se expone la idea de etiquetar en forma de asertos lógicos las sentencias de los programas de forma que se reflejaran los efectos de las mismas a partir de una definición formal de la semántica del lenguaje de programación. Fue poco después C. Hoare quien en 1969 desarrolló a partir de los trabajos de Floyd el cálculo semántico de pre y postcondiciones en algoritmos y el concepto de *invariante* en un bucle [Hoa69]. De estos trabajos desarrollados para la verificación de algoritmos deterministas surgió lo que se conoce como *Lógica de Hoare* [Krz81], a cuyo posterior desarrollo y ampliación contribuyeron entre otros N. Wirth y D. Gries [Gri82]. Estas reglas axiomáticas de prueba propuestas tuvieron pronto una enorme acogida que se amplió hacia la programación concurrente y a la programación distribuida [OG76, Owi76, Krz86].

1.3 Esquemas algorítmicos

La idea de que una familia de problemas pueden compartir una solución algorítmica eficiente adquiere gran interés en esta época, donde abundan las publicaciones con soluciones a determinados problemas abordables computacionalmente como los algoritmos de ordenación [Hoa62, Wil64], problemas de caminos mínimos en grafos y un largo etcétera. Como consecuencia de lo anterior, se introduce poco a poco la idea de que es posible agrupar determinados problemas algorímicos bajo un esquema de resolución común.
Es en los años 60 donde se menciona por primera vez el concepto de esquema algorítmico que aparece desarrollado en la obra de Knuth "*The Art of Programming*", auténtica biblia de la algorítmica comenzada en 1970 con la ayuda de Floyd (también en el CS Department, en Stanford) quien ayudo en las primeras revisiones, y continuada y ampliada hasta la actualidad con un total de 7 volúmenes, algunos de ellos todavía en renovación.
Posteriormente se acuñan determinados nombres para las familias de algoritmos desarrolladas hasta el momento; así, la definición de algoritmo *voraz* se debe a Jack Edmonds [Edm71]. El estudio del problema de la ordenación, con los desarrollos de los algoritmos de ordenación rápida y la creación de las estructuras de datos de *montículos* para la solución de Williams [Hoa62, Wil64] se apoyan en la descomposición algorít-

mica que dio lugar al esquema de tipo Divide y Vencerás. Por otro lado, la descomposición en subejemplares solapados derivó en las técnicas de programación dinámica desarrolladas por Bellman, Floyd y Warshall, entre otros. El nombre del esquema conocido como *Divide y Vencerás* se atribuye a J. Mauchly, impulsor del proyecto de construcción de la ENIAC y uno de los fundadores de la ACM.

La ampliación de los esquemas iniciales, restringida a la resolución de los problemas de caminos mínimos y árboles de expansión mínimos en grafos explícitos y de ordenación, se complementa con la acuñación del concepto de Inteligencia Artificial por John McCarthy, que añadió a mediados de los 60 la investigación en algoritmos de búsqueda en grafos, búsquedas en anchura y en profundidad, y técnicas como el método de *minimax* y el algoritmo *A**, entre otros. Los algoritmos de ramificación y poda también son fruto de estas investigaciones y motivados principalmente con los problemas que plantea la Investigación Operativa. Entre la década de los 60 y comienzos de los 70 años, por ejemplo el *backtraking* o vuelta atrás se describe en 1965 por [GB65] con una solución algorítmica al problema de las ocho reinas.

Los *esquemas algorítmicos*, van cristalizando poco a poco y se asientan sus nombres y propiedades como elementos básicos de la computación. La importancia de los esquemas no se limita a la creación de soluciones genéricas a familias de problemas, y la evolución de éstos ha permitido, no solo la solución algorítmica de determinados problemas conocidos, sino la aparición de soluciones computables (tratables) de otros. Un ejemplo conocido de esto último es la FFT (transformada rápida de Fourier) que ha posibilitado el desarrollo del procesamiento digital de señales en tiempo real [BM67].

A partir de la década de los 70 la mayoría de los desarrollos realizados en esta línea se introducen de manera indiscutible como parte esencial de los currícula de la disciplina informática, que adquiere personalidad propia con la creación de las primeras facultades. De esta manera, el estudio de los esquemas algorítmicos y el conocimiento de sus múltiples aplicaciones ha llegado hasta nuestros días como un componente básico de la formación del ingeniero en informática.

1.4 Planificación del texto

Este libro está pensado como texto de apoyo a la enseñanza de la algorítmica en escuelas técnicas superiores de ingeniería informática.

Los contenidos combinan el estudio de algunas estructuras de datos y de los principales esquemas algorítmicos, junto con ejercicios y soluciones de los mismos. Cada capítulo, a su vez, se estructura con un desarrollo del tema que aborda seguido de ejercicios resueltos, propuestos y de notas bibliográficas. Hemos querido también realizar una presentación del tema al comienzo de cada capítulo para que el lector pueda conocer las dificultades que implica su estudio y los conocimientos previos requeridos.

1.4.1 Prerrequisitos

En primer lugar se supone una base matemática que implica entre otras cosas capacidad para la resolución de ecuaciones básicas y la realización de demostraciones por inducción. La base matemática se entiende fundamentada en los conocimientos habituales adquiridos en un primer curso de ingeniería e incluiría el cálculo de expresiones combinatorias básicas, nociones de probabilidad, cálculo de límites, polinomios, resolución de sistemas de ecuaciones lineales, y manejo de la notación matemática asociada.
En el aspecto de análisis algorítmico, se hará uso en mayor medida del cálculo y demostración de costes algoritmicos, por lo que puede necesitarse realizar demostraciones por reducción al absurdo o por inducción para la comprobación de optimalidad de determinados algoritmos y el manejo de sumatorios, combinatoria y cálculo de límites.
El texto da por conocidas las técnicas de análisis de coste de algoritmos y las notaciones O y Ω, aunque se usarán en muchas ocasiones las expresiones resueltas de las ecuaciones re recurrencia aplicadas a los algoritmos recursivos. Se suponen también conocidas las estructuras de datos básicas como listas, colas, árboles y sus operaciones básicas.

1.4.2 Dependencias entre capítulos

El libro se estructura en dos partes: estructuras de datos y esquemas de algorítmica. La primera de ellas expone estructuras de datos con un objetivo práctico. Se estudiará la utilidad de los mismos en el ámbito de la implementación en algoritmos de diverso tipo. Por esta razón este capítulo hace hincapié en algunas implementaciones eficientes de los mismos y se realizan referencias a los ámbitos en los que se usan.
En segundo lugar, se abarca el estudio de los esquemas algorítmicos con ejemplos de aplicaciones en diversos ámbitos, como la planificación, la ordenación y la búsqueda, entre otros. Todos los capítulos de algorítmica usan en mayor o menor medida conceptos de estructuras de datos, por lo que se recomienda que éste capítulo se abarque en su totalidad antes de proceder al estudio de alguno de los esquemas.
En cuanto al orden de estudio de los esquemas algorítmicos, se recomienda abordar por un lado los algoritmos voraces antes que la programación dinámica, por otro el estudio de los algoritmos de tipo divide y vencerás, y por último el de vuelta atrás antes que el de ramificación y poda.

1.4.3 Organización del contenido

Desde el punto de vista del profesor, el contenido y la planificación que proponemos es la siguiente:

- El Capítulo 2 abarca el estudio de las estructuras de datos avanzadas necesarias para el desarrollo de los algoritmos y esquemas expuestos.

- El Capítulo 3 incluye el estudio de los algoritmos voraces. El estudio de este esquema se basa en el conocimiento de aplicaciones conocidas del mismo, sobre todo involucrando problemas con grafos (como el del árbol de recubrimiento mínimo o el de los caminos mínimos) y de planificación de tareas. Se recomienda empezar con la introducción teórica al esquema y algún ejercicio y posteriormente se abordarían las diferentes familias de problemas que se resuelven mediante este esquema: grafos, planificación y optimización en varias variantes. Es recomendable que el estudio se realice cuando se tienen recientes los conocimientos sobre grafos y montículos.

- El Capítulo 4 abarca el estudio de los algoritmos de tipo Divide y Vencerás. En este tipo de esquema, la variedad de subfamilias es menor. Debe hacerse más hincapié en la inducción realizada en las llamadas recursivas, y por tanto que el aspecto teórico del esquema se trabaje bien antes de abordar la instanciación. Se comenzaría con el estudio teórico y después se abordarían dos o tres problemas en profundidad que ilustren la instanciación del esquema.

- El Capítulo 5 estudia la resolución de problemas mediante programación dinámica. Es conveniente conocer previamente los esquemas de divide y vencerás y de algoritmos voraces, por utilizar la programación dinámica conceptos algorítmicos de ambos.

- El Capítulo 6 contempla el estudio de los algoritmos de *backtraking* o de retroceso. Habría que repasar el concepto de grafo y abordar el esquema exponiendo claramente el concepto de exploración en un grafo implícito frente al concepto de grafo explícito de la estructura de datos. De esta manera, es importante que se adquiera conciencia de que por un lado la exhaustividad, y por otro la generación implícita del recorrido son los aspectos fundamentales, además claro está, de la capacidad de retroceso en el espacio de búsqueda.

- Por último, en el Capítulo 7 y como variante de lo anterior, el esquema de ramificación y poda propone una búsqueda con objetivo de optimización. Se debe repasar la noción de montículo para el almacenamiento de los nodos pendientes de desarrollo y detallar porqué esta estructura de datos proporciona un mecanismo de ramificación basada en expandir en cada momento el nodo más prometedor. En los aspectos teóricos este esquema es similar al anterior, por lo que las explicaciones deben abordarlo como una variante e identificar los elementos nuevos, como son el montículo, ya mencionado, y la actualización de las cotas superiores e inferiores del objetivo que se precisa optimizar.

El texto se complementa con bibliografía en donde se profundiza en algunos aspectos de la algoritmia relacionados con la eficiencia o con variantes de determinados problemas, así como con la realización de pruebas de evaluación personal al concluir el estudio

de cada esquema. Por último es recomendable complementar el estudio teórico con la realización de prácticas en las que se aborde la resolución y codificación completa de los esquemas mencionados.