La enseñanza de la programación en los estudios de informática de la UPV

A. Casanova, M. J. Castro, V. Fuentes, I. Galiano, C. García, F. Marqués, N. Prieto

Departamento de Sistemas Informáticos y Computación

Universidad Politécnica de Valencia

(casanova,mcastro,vfuentes,mgaliano,cgarcia,pmarques,nprieto)@dsic.upv.es

Resumen

En esta comunicación se exponen las claves en que se basa la propuesta docente de las asignaturas básicas de programación en los estudios de informática de la UPV.

Las premisas adoptadas tienen su raíz en los autores clásicos que establecieron los fundamentos de la programación como una disciplina, y que en España tuvieron una primera acogida en las propuestas de la Universitat Politècnica de Catalunya.

A partir de estas premisas, los contenidos específicos de las asignaturas implicadas se basan en los siguientes puntos: la formalización de los problemas, el diseño sistemático de algoritmos, la relevancia de la abstracción y el papel fundamental de las prácticas.

1 Introducción

La evolución de la enseñanza de la programación a lo largo de las dos últimas décadas ha puesto de manifiesto la necesidad de combinar dos aspectos en la formación de los estudiantes universitarios de informática. Por una parte, citando a [Scholl,91] "un punto esencial es que el programador comprenda, ya durante su aprendizaje, que el producto de su trabajo consiste no sólo en el programa que obtiene finalmente sino, sobre todo, en una descripción de las etapas de su análisis que ponga en evidencia todas las elecciones que haya podido efectuar". Al mismo tiempo hay que garantizar la adquisición por parte del alumno del nivel práctico que se le va a exigir como ingeniero en informática.

La dificultad mayor en la conjugación de ambos aspectos parece residir en el papel que las técnicas formales desempeñan en el diseño de los programas.

En esta ponencia presentamos nuestra experiencia a este respecto y su concreción en unos ejes centrales alrededor de los cuales se estructuran los contenidos de las asignaturas de primer curso de algoritmos y estructuras de datos de los estudios de informática en la UPV: Introducción a la PRogramación (IPR), Algoritmos y estructuras de Datos I (AD1) y Algoritmos y estructuras de Datos II (AD2). Estos contenidos son, básicamente:

- IPR. Introducción a la definición de problemas y algoritmos mediante su especificación basada en la noción de estado. Instrucciones y tipos de datos elementales. Inicio a un lenguaje de programación imperativo.
- AD1. Constructores de tipo tupla y vector. Diseño iterativo guiado por la noción de invariante. Algoritmos de recorrido y búsqueda. Algoritmos de ordenación.
- AD2. Análisis de la eficiencia de los algoritmos. Inducción en el diseño de algoritmos, diseño recursivo. Diseño orientado a los datos. Tipos lineales y árboles binarios.

En el siguiente apartado se presentan la abstracción y el razonamiento inductivo como los ejes centrales antes mencionados. En el punto 3 presentamos las prácticas de laboratorio, que soportan la conjugación de los conocimientos teóricos con las habilidades prácticas, aspecto fundamental de nuestra propuesta docente. Adicionalmente, en el último punto se discuten algunas restricciones impuestas a las asignaturas por el plan de estudios vigente, junto con algunas propuestas de revisión.

2 Ejes centrales

2.1 Abstracción

Denominamos abstracción a la operación de eliminación de los detalles que se consideran irrelevantes durante la elaboración de un modelo.

En el ámbito de la computación el uso de la abstracción es tan significativo que una de las definiciones que se recogen en [Denning,89] es la siguiente: "la ciencia de la computación es el estudio de la abstracción, y el dominio de la complejidad". Esta definición, aunque excesivamente generalista, revela sin embargo la gran importancia concedida al concepto en el conjunto de la disciplina.

En el ámbito de la creación de software, creemos que el uso de la abstracción es imprescindible como una de las herramientas mentales para abordar, con garantía de éxito, la producción del mismo. La complejidad del producto a realizar y la del proceso de producción exigen al programador una segmentación del diseño en distintos niveles de abstracción, que le faciliten su comprensión y su solución efectiva.

Como ya hemos dicho, uno de los objetivos fundamentales de los estudios de programación es la capacitación del alumno en el diseño del software. El logro de este objetivo implica su formación en el proceso de segmentación mencionado y su capacitación para razonar de forma adecuada en el marco de cada nivel.

Es necesario, por lo tanto, que cualquier conjunto de asignaturas que, como las aquí presentadas, supongan la introducción de los alumnos al diseño y análisis de algoritmos y estructuras de datos, presenten desde el primer momento la problemática expuesta.

La abstracción en la enseñanza de la programación

El proceso de segmentación por niveles al que hemos aludido se refleja principalmente en las asignaturas presentadas en dos aspectos relativos al diseño de programas. Por una parte, en el diseño de operaciones que representarán, de forma abstracta, un algoritmo o cómputo; por otra, en la idea central de tipo de datos, que abstrae a un conjunto de operaciones la complejidad subyacente a la realización de un nuevo tipo.

Además, para el tratamiento adecuado de cada uno de los dos aspectos es necesario la introducción de formalismos que provean el rigor que necesita la discusión, pero que sean alcanzables por los alumnos en esa etapa de su formación. Este nivel medio, correspondiente al habitual de un alumno de nuevo ingreso en unos estudios universitarios, presenta carencias, particularizadas en los ámbitos de la lógica de primer orden, que hacen necesario un refuerzo formativo adicional.

Conceptos significativos relacionados con el uso de la abstracción, y que se presentan en las asignaturas mencionadas, son los conceptos de estado, de variable, de procesador virtual y de operación. Una operación, que se concibe como un algoritmo parametrizado, se define, mediante notación pre-post, en términos del conjunto de estados permisible inicial y del conjunto de estados final, solución del problema.

Abstracción y tipos de datos

La noción de tipo de datos se introduce inicialmente al presentarse los tipos de datos elementales. Un tipo de datos es visto como un conjunto de valores, junto con una familia de operaciones. Tras dicha introducción se presentan a los alumnos los conceptos de *tupla* y de *vector*, más como mecanismos constructores de nuevos tipos, que como entes abstractos.

Creemos que la noción de secuencia, junto con la de tratamiento secuencial, son clave en la formación de los alumnos. Sin embargo, nuestra experiencia muestra que la manipulación, por parte de los alumnos, del tipo abstracto secuencia, como elemento básico en la formación correspondiente al primer semestre académico, es muy problemática debido principalmente a la complejidad excesiva que introduce la especificación de algoritmos con dicho tipo. Pensamos, y nuestra experiencia docente lo confirma, que resulta mucho más conveniente la introducción del tipo vector como estructura fundamental en la que basar todo el desarrollo formativo que supone la introducción al diseño iterativo. Los vectores son conceptualmente más simples que las secuencias, están más cercanos a la problemática real de la programación actual y, sobre todo, facilitan considerablemente la especificación de problemas, objetivo básico durante el primer semestre.

En el segundo semestre académico (asignatura AD2) se introduce de forma natural el diseño recursivo y el análisis de la eficiencia de los algoritmos. Ambos son imprescindibles para estudiar adecuadamente la abstracción de datos.

La última mitad del segundo semestre está dedicada a la introducción de la abstracción de datos y al estudio de los denominados tipos lineales. Como ya hemos dicho, un tipo de datos es visto como un conjunto de valores junto con una familia de operaciones. Para un tipo de datos diferenciamos entre su definición, o especificación, que se efectúa por medio de notación pre-post, y su implementación, que presenta una factibilidad que debe ser analizada. Las prácticas de laboratorio relativas al diseño modular intentan realizar una breve introducción al diseño orientado a los datos.

El estudio de diferentes tipos de datos introduce principalmente los tipos lineales clásicos: pilas y colas, con su definición y representación habitual, junto con las listas con punto de interés [Franch,93], que es el modelo de secuencia subyacente al utilizado durante todo el curso académico, y al que otorgamos gran importancia.

Finalmente, siguiendo una metodología similar a la ya señalada, se hace una breve introducción a un modelo de datos no lineal: la arborescencia binaria, mediante la misma se inicia el estudio de diferentes abstracciones de datos que se realizará en la siguiente asignatura del ciclo: Algoritmos y estructuras de Datos III (AD3).

2.2 La inducción

Como hemos mencionado anteriormente, uno de los objetivos primordiales de estas asignaturas es enseñar al alumno a diseñar y analizar algoritmos. A grandes rasgos, podemos distinguir dos líneas fundamentales de trabajo:

- Presentar las grandes familias de algoritmos, en orden creciente de dificultad, agrupados según algún criterio que puede ser desde el problema que resuelven (búsqueda, ordenación, ...), hasta la pertenencia a algún esquema que generalice la estructura del algoritmo. Esta línea perseguiría dotar al alumno con el conocimiento de un repertorio suficiente de algoritmos fundamentales, y capacitarle para desarrollar por analogía sus propios algoritmos.
- Estudiar técnicas formales de diseño de algoritmos: se enseña al alumno alguna técnica de verificación formal que muestre qué propiedades debe tener un algoritmo correcto, y que deben guiarle en el desarrollo de sus propios programas.

Ambas líneas de trabajo sólo son incompatibles llevadas a sus últimos extremos. El proyecto docente que presentamos aquí, se basa en conciliar los objetivos básicos de ambas líneas.

De acuerdo con la tesis de Church, la naturaleza de los algoritmos es recursiva. Desde este punto de vista, resolver un problema consiste en un análisis por casos que descompone el problema en subproblemas "más pequeños" de la misma naturaleza. Este es el enfoque adoptado, aunque no se le plantea al alumno abiertamente.

La presentación de los esquemas iterativos de recorrido y búsqueda sobre vectores en AD1, proporcionan un primer ejemplo sencillo de recurrencia con el que se introducen el concepto de invariante y función limitadora de una iteración. El invariante es el equivalente a la hipótesis de inducción de la recurrencia, y la función limitadora traduce el razonamiento sobre los datos a una inducción sobre los naturales. Los algoritmos "lentos" de ordenación se describen adecuadamente en este marco, y la técnica se ha demostrado especialmente útil para desarrollar aspectos problemáticos de ciertos algoritmos, como por ejemplo el manejo de los índices en la búsqueda binaria, o el algoritmo de partición en el que se basa la ordenación rápida de Hoare.

Este enfoque permite que, cuando el alumno accede a la siguiente asignatura (AD2), el concepto de talla se introduzca de forma natural, y el análisis de la eficiencia se pueda enfocar al desarrollo de los algoritmos mediante la combinación de dos ideas fundamentales: descomposición del problema buscando una reducción "rápida" de la talla, reduciendo en lo posible el coste de la descomposición. Este método de trabajo encontrará su continuación en una asignatura posterior (AD3), que presentará estrategias avanzadas como "divide y vencerás", y "métodos voraces" para algunos problemas de "búsqueda de soluciones".

La presentación del paradigma recursivo completa este planteamiento: se presentan inicialmente recursiones lineales que faciliten el tránsito desde el razonamiento iterativo al nuevo paradigma, para continuar con ejemplos de recursión múltiple. Para reforzar la integración del paradigma iterativo en el recursivo, se intenta unificar la terminología usada en ambos: en ambos se hacen pruebas por inducción, y en ambos se reducen las pruebas a inducción sobre los naturales a través de una función limitadora. Se ha desestimado el uso de inducción noetheriana, que sería más conveniente e incluso necesaria en

algoritmos que van más allá del alcance de asignaturas básicas.

No es este tampoco el lugar adecuado para discutir la equivalencia entre recursión e iteración a pila, pero el hincapié que se hace en algunas prácticas en realizar trazas de algoritmos recursivos observando directamente la pila de llamadas del compilador, proporciona al alumno una experiencia que le facilitará esta discusión en asignaturas más avanzadas.

En resumen, se pretende dotar al alumno con un método de diseño bien fundamentado que le permita razonar sobre las propiedades de corrección y eficiencia de los algoritmos.

Aunque la notación utilizada en la escritura de los algoritmos es un pseudocódigo cercano al lenguaje de programación imperativo usado en el laboratorio, con esta visión se espera además transmitir al alumno una capacidad de razonar sobre los algoritmos que le facilite en el futuro su expresión en un lenguaje funcional o declarativo.

Al tratarse de asignaturas de primer curso no es conveniente utilizar como ejemplos de la metodología propuesta algoritmos muy avanzados. No obstante sí que se consigue revisar un repertorio que llega a incluir algunos algoritmos recursivos de recorrido y tratamiento de árboles binarios, además de la implementación de las estructuras secuenciales clásicas (listas, pilas y colas).

Diversos autores nos proporcionan ejemplos de que este método de razonamiento inductivo facilita la discusión de algoritmos que van más allá del alcance de estas primeras asignaturas. Por ejemplo, en [Bentley,89] se presentan las operaciones de cola de prioridad implementada como un "heap" y su aplicación al algoritmo de ordenación "heapsort; el algoritmo de Dijkstra de caminos mínimos en un grafo y los algoritmos de Kruskal y Prim de cálculo de árboles de recubrimiento mínimos sobre un grafo conexo se demuestran por inducción en [Brassard,97]; en [Manber,88] se muestran ejemplos muy diversos de aplicación de esta metodología como el problema de la ordenación topológica en un grafo, el cálculo de factores de equilibrio en un árbol binario, un algoritmo eficiente para el cálculo de la distancia mínima de puntos en un plano, etc.

3 Prácticas

Las prácticas tienen un papel fundamental en nuestra propuesta docente.

Los objetivos que nos planteamos con las prácticas son de tres tipos. En primer lugar se pretende que los alumnos adquieran un conocimiento avanzado de un lenguaje de programación imperativo que dé el máximo soporte posible para la implementación de los conceptos introducidos en la teoría. El lenguaje elegido en este caso es el Pascal, más concretamente su dialecto `HP-Pascal (STANDARD ANSI/IEEE 1983)`. El conocimiento del lenguaje de programación debe abarcar desde las instrucciones del lenguaje, tipos de datos elementales, gestión de entrada/salida, ficheros, vectores, etc., hasta el uso de variable dinámica.

En segundo lugar se pretende que el proceso completo de la programación se complemente con el uso de un entorno y unas herramientas que faciliten dicha tarea y proporcionen una primera visión de lo que es la programación. Nos referimos por una parte al sistema operativo (`UNIX`), con sus comandos elementales y su estructura de ficheros, y por otra a utilidades específicas como un editor de textos (`emacs`), un depurador de programas (`xdb`), un programa para el ajuste de mínimos cuadrados (de elaboración propia) y un programa para la representación gráfica de series de puntos y curvas (`gnuplot`); y por supuesto nos referimos también al compilador del `HP-Pascal`.

Y en tercer lugar está el objetivo de aprender a implementar (mediante programas concretos para resolver problemas concretos) los conceptos esenciales de programación estudiados en la teoría, de manera que el aprendizaje del lenguaje de programación antes mencionado quede al servicio de este objetivo. Estos conceptos esenciales los podemos agrupar en los siguientes bloques para su estudio práctico:

- Diseño orientado a tareas.
- Algoritmos de recorrido y búsqueda.
- Estudio práctico de la complejidad temporal de los algoritmos.
- Recursión y complejidad espacial.
- Diseño modular orientado a los datos. Implementación de tipos de datos lineales.

Queremos destacar que las prácticas de laboratorio son el lugar elegido para la introducción de aspectos técnicos que resultan difíciles de explicar en las clases teóricas. Por ejemplo, el estudio detallado del paso de parámetros, la medición de los tiempos de

ejecución de partes aisladas de un programa y la visualización del contenido de los registros de activación generados en una secuencia de llamadas recursivas (tanto para la depuración de posibles errores como para mejorar la comprensión del mecanismo de la recursión).

4 Observaciones al plan de estudios

La primera observación que queremos realizar es que todos los objetivos señalados en los apartados anteriores se deben conseguir a través de contenidos distribuidos en 15 créditos, los asignados a las asignaturas IPR, AD1 y AD2 (ver tabla 1). Consideramos que esta dedicación es claramente insuficiente. Por otra parte, el plan de estudios vigente obliga a una excesiva fragmentación de la materia en distintas asignaturas, lo que dificulta enormemente la organización de la docencia. Finalmente queremos destacar la ausencia de asignaturas optativas relacionas con los aspectos que aquí hemos tratado.

Algunas propuestas que pensamos que deberían de tenerse en cuenta en la revisión de planes de estudio que actualmente se plantea son las siguientes:

- Las asignaturas de primer curso deberían de formar una única asignatura anual de 15 créditos de estudio obligatorio.
- Contemplar adicionalmente en los cursos siguientes, dos asignaturas también de estudio obligatorio, especializadas en el análisis y diseño de algoritmos y en los tipos de datos.
- Ofertar asignaturas optativas que cubran las materias *Lenguajes de Programación* y *Tipos Abstractos de Datos.*

Agradecimientos

Deseamos expresar nuestro reconocimiento al resto de profesores que imparten estas asignaturas en la Escuela y Facultad de Informática de nuestra universidad y especialmente quisiéramos agradecer el soporte económico de la Escuela Universitaria de Informática.

Referencias

[Bentley,89] Jon Bentley. *Programming Pearls.* Addison-Wesley, 1989.

[Brassard,97] G. Brassard, P. Bratley. *Fundamentos de Algoritmia.* Ed. Prentice Hall , 1997.

[Castro,92] J.Castro, y otros. *Curs de Programació.* McGraw Hill, 1992.

[Denning,89] P.J.Denning, y otros. *Computing as a Discipline.* Communications of ACM, Vol. 32, No.1, pp. 9-23, 1989.

[Franch,93] X.Franch. *Estructures de Dades. Especificació, Disseny i Implementació.* U.P.C., 1993.

[Manber,88] U.Manber. *Using Induction to Design Algorithms.* Communications of the ACM, Vol 31, pp. 1300-1313, 1988.

[Scholl,91] P.C.Scholl., J.P.Peyrin. *Esquemas algorítmicos fundamentales. Secuencias e iteración.* Mason, 1991.

Tabla 1 Plan de ordenación académica para el estudio de los algoritmos y las estructuras de datos. Los créditos (C) se dividen en teóricos (CT) y prácticos (CP). La asignatura "Algorítmica" se imparte únicamente en la titulación de Ingeniero en Informática.

Sem.	Abrev.	Asignatura	C	(CT	+	CP)
1	IPR	Introducción a la programación	3	(1,5	+	1,5)
1	AD1	Algoritmos y estructuras de datos I	6	(3	+	3)
2	AD2	Algoritmos y estructuras de datos II	6	(3	+	3)
3	AD3	Algoritmos y estructuras de datos III	6	(3	+	3)
6	ALT	Algorítmica	6	(3	+	3)