Algoritmos de ordenación (I)

Todos sabemos que los ordenadores, entre otras cosas, sirven para ordenar. Pero ¿Cómo lo hacen? ¿Existen distintos modos de hacerlo? ¿Depende de lo que tengamos que ordenar? ¿Es tan crítico elegir correctamente el método de ordenación?

En este artículo divulgativo se pretende dar respuesta a estas y a otras preguntas, mostrando como funcionan los algoritmos de ordenación más importantes y discutiendo las claves a tener en cuenta para seleccionar el algoritmo más apropiado para un problema concreto.

Eugenio Francisco Sánchez Úbeda

Doctor Ingeniero del ICAI
Instituto de Investigación Tecnológica y Departamento de Sistemas Informáticos de la Universidad Pontificia Comillas de Madrid, donde es coordinador del Área de Sistemas Inteligentes (ASI)
E-mail: Eugenio.Sanchez@iit.upco.es

Introducción

La ordenación es una operación básica en la vida diaria. El que uno sea más o menos hábil realizando pequeñas tareas cotidianas está directamente relacionado con el que las cosas implicadas estén más o menos ordenadas. La razón es obvia: es más sencillo buscar algo cuando los objetos están ordenados.

Quién no ha pensado "Tendría que ordenar los discos…" al buscar un disco de música en la estantería o, "Nunca recuerdo si está por su nombre, su apellido, o por su nombre de pila…" al tratar de encontrar el teléfono de un amigo en nuestra agenda. Encontrar una lata de pimientos en la despensa también puede ser todo un reto. Es muy fácil recordar otras fuentes de pequeños problemas organizativos en la vida cotidiana (mesilla de noche, despacho, caja de herramientas, botiquín, trastero, etc.)

Por tanto, parece evidente que las operaciones de ordenación y búsqueda son importantes y están estrechamente relacionadas.

Algo similar ocurre en computación: La ordenación es seguramente la operación elemental más importante y por tanto, mejor estudiada. No en vano utilizamos el término "ordenador" para referirnos a lo que otros llaman "computador" o "computer". Además, se puede afirmar con bastante seguridad que la gran mayoría de los datos generados por un

programa están ordenados de alguna manera y que la gran mayoría de los programas de un cierto tamaño necesitan realizar en algún momento una ordenación.

¿Por qué hay distintos algoritmos?

Casi todos nosotros hemos aprendido, de forma más o menos inconsciente y a partir de la experiencia, varias maneras distintas de ordenar.

Imaginemos, por ejemplo, que tenemos que ordenar una mano de cartas en una partida de mus o de poker. Seguramente la estrategia que seguiremos es muy diferente de la empleada cuando uno tiene que colocar por orden las 200 hojas de un informe desordenado que acaba de recoger del suelo. Además, posiblemente emplearemos una estrategia distinta si sabemos (o intuimos) que las hojas no están muy desordenadas.

Si la mayoría de nosotros empleamos distintas estrategias de ordenación, no es de extrañar que existan algoritmos de ordenación distintos.

¿Qué algoritmo emplear?

El decidir qué algoritmo de ordenación es el mejor para nuestro problema no es una labor sencilla. Dicha decisión se debe basar en varios criterios que normalmente son difíciles de evaluar con exactitud.

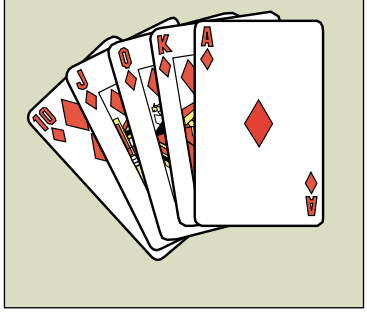
FIGURA 1. LA ORDENACIÓN ES UNA OPERACIÓN BÁSICA EN LA VIDA DIARIA

Volvamos al ejemplo del informe desordenado. Es después de recoger las hojas del suelo (y comprobar que están numeradas) cuando uno decide como ordenará el informe. Normalmente dicha decisión se realiza de forma inconsciente y tiene en cuenta ciertos criterios "borrosos" basados en el número de hojas a ordenar o en cómo de ordenadas estén las hojas. No es lo mismo ordenar 10 que 1000 hojas, como tampoco lo es ordenar algo que está totalmente desordenado que algo que está prácticamente ordenado.

La decisión del algoritmo a emplear debe de tener en cuenta al menos las siguientes consideraciones:

- El número de objetos a ordenar
- El tamaño de dichos objetos (cuanto cuesta moverlos)
- El nivel de desorden que aparece en el conjunto a ordenar

FIGURA 2. EN LOS JUEGOS DE CARTAS SE EMPLEAN DISTINTAS ESTRATEGIAS DE ORDENACIÓN



Cuanto mayores sean nuestros recursos disponibles para realizar la ordenación, menos críticas serán las consideraciones anteriores. Desde el punto de vista informático, el disponer de más memoria principal y de un procesador más rápido significa que podremos ordenar más datos, de mayor tamaño y

en menos tiempo. Sin embargo, una correcta selección del algoritmo puede significar un ahorro de recursos muy importante, fundamentalmente en el tiempo necesario para realizar la ordenación.

Además, existen otras aspectos que no deben olvidarse y que pueden ser determinantes a la hora de elegir el algoritmo. Por ejemplo, si tenemos ordenados nuestros datos de clientes según su primer apellido y a continuación queremos ordenarlos atendiendo a su antigüedad pero sin perder la anterior ordenación (es decir, que para una misma antigüedad los clientes sigan ordenados por su primer apellido), estamos obligados a utilizar un algoritmo "estable". Un algoritmo estable nos asegura que los objetos siempre conservan su posición relativa cuando tienen la misma clave.

¿Es tan crítica la selección del algoritmo?

La respuesta es sí, sin duda alguna. Una razón simple, pero de peso, la podemos encontrar en nosotros mismos: empleamos distintas estrategias de ordenación dependiendo de las características del problema. Una mala elección implica un gasto de recursos innecesario. ¿Qué ocurre si intentamos ordenar un informe de 200 hojas, que está muy desordenado, como si fuera una mano de cartas? ¿No es mejor hacer varios montones, ordenarlos por separado y luego juntarlo todo?

Desde el punto de vista algorítmico la razón es la misma, aunque se expresa de una forma un poco más técnica. Se dice que un algoritmo es "lineal" si su tiempo de ejecución es directamente proporcional al número de datos a ordenar. Es decir, tarda el doble en ordenar el doble de datos. Un algoritmo "cuadrático" es mucho peor que uno lineal ya que para ordenar el doble de datos necesita cuatro veces más tiempo.

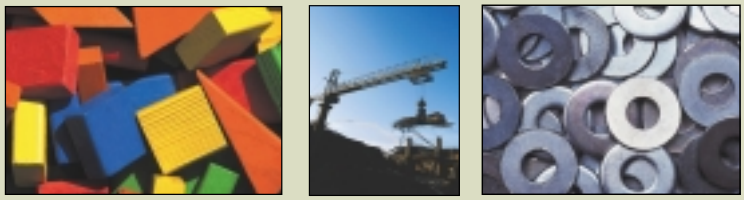
En cualquier caso, uno podría pensar que si dispone de grandes recursos para realizar la ordenación, (mucha memoria principal y un procesador rápido), ¿Por qué preocuparse del algoritmo si todos ordenan correctamente?

Supongamos que una empresa dispone de un supercomputador capaz de realizar 100 millones de operaciones por segundo. Entre los miembros de su personal existen muy buenos programadores que emplean varios días en poner a punto un código máquina altamente eficiente que implementa un algoritmo de ordenación ineficiente (por ejemplo, cuadrático).

Por otro lado, un estudiante tiene en casa un PC que realiza 1 millón de operaciones por segundo, es decir 100 veces más lento. Decide programar en un rato y utilizando un lenguaje de alto nivel un algoritmo muy eficiente (por ejemplo, lineal).

Si el estudiante y la empresa ordenaran el mismo millón de datos, el primero tendría que esperar tan sólo unos pocos segundos para obtener el resultado, frente a las varias horas empleadas por el supercomputador de la empresa.

FIGURA 3. LA SELECCIÓN DEL ALGORITMO DE ORDENACIÓN IMPLICA UNA DECISIÓN MULTI-ATRIBUTO



La decisión de algoritmo a emplear debe de tener en cuenta al menos las siguientes consideraciones:

- El número de objetos a ordenar
- El tamaño de dichos objetos (cuánto cuesta moverlos)
- El nivel de desorden que aparece en el conjunto a ordenar

Algoritmos

Los distintos algoritmos de ordenación existentes describen de forma precisa como ordenar de forma creciente un conjunto de datos.

Normalmente los elementos a ordenar están almacenados en un vector. Dado que un elemento puede estar caracterizado por diversos atributos o campos, es necesario indicar el que se utilizará para ordenar. Dicho campo se suele denominar "clave".

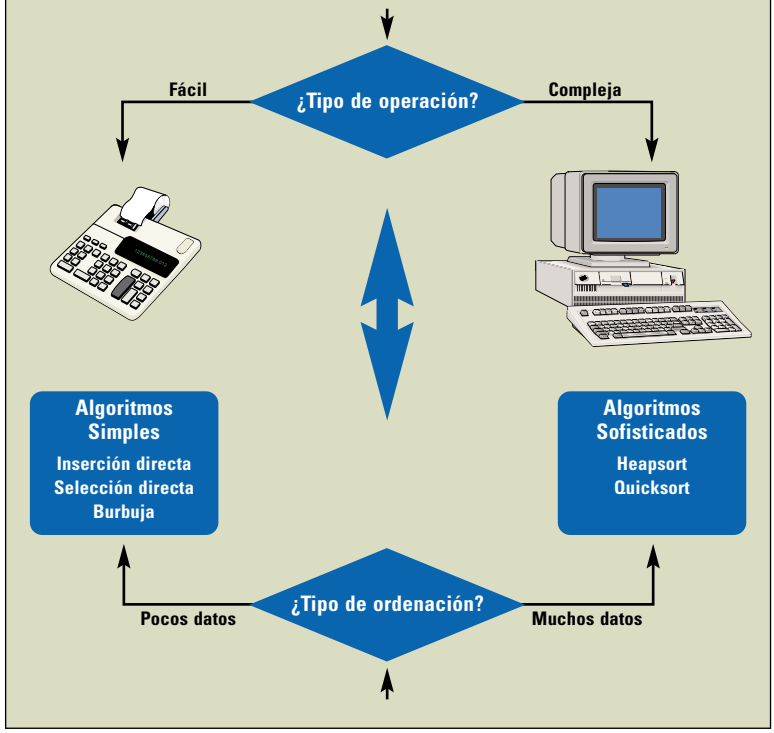
Los numerosos algoritmos de ordenación se pueden clasificar atendiendo a distintos criterios. Una clasificación comúnmente aceptada distingue entre algoritmos de ordenación externa e interna. Los primeros se deben emplear cuando no es posible almacenar todos los datos a ordenar en un vector dentro de la memoria principal del ordenador, siendo necesario emplear durante la ordenación un dispositivo de almacenamiento externo como una cinta o un disco (de ahí su nombre). Los algoritmos de ordenación interna están pensados para cuando no existe dificultad en el acceso a los datos ya que se tienen almacenados en memoria RAM como un vector de elementos. En este artículo nos vamos a centrar en este tipo de algoritmos por ser los más empleados en la práctica.

Además, existe un criterio adicional de economía de memoria RAM (es lo que se denomina ordenación in situ). En este artículo nos limitaremos a describir algoritmos de este tipo.

Simples y sofisticados

Atendiendo al número de objetos a ordenar se puede distinguir entre algoritmos "simples" y algoritmos "sofisticados". Los

FIGURA 4. ALGORITMOS DE ORDENACIÓN SIMPLES Y SOFISTICADOS



algoritmos simples (también denominados directos) son una buena solución cuando son pocos los elementos a ordenar. Cuando tenemos "bastantes" datos es muy conveniente optar por un algoritmo sofisticado.

Ahora bien, el problema es saber qué significa "pocos" o "bastantes" datos. Lamentablemente no existe un umbral claro que permita saber cuándo no compensa emplear un algoritmo sofisticado ya que el valor óptimo dependería del ordenador y del código máquina que implemente el algoritmo. Un umbral razonable puede ser 10 elementos. Este dilema es similar al que aparece cuando tenemos que realizar una operación numérica. Si es sencilla lo lógico es emplear una calculadora. Encender el ordenador seguramente es demasiado costoso para luego simplemente multiplicar 75 por 342. Sin embargo, puede compensar arrancar el ordenador si tenemos que realizar muchas operaciones.

Algoritmos simples

Inserción directa

Este algoritmo es el que empleamos típicamente y de forma natural para ordenar una mano de cartas. Los elementos (cartas) se dividen conceptualmente en dos conjuntos o secuencias, una ordenada (las ca-

ras que tenemos en la mano) y otra desordenada (las cartas que están boca abajo en la mesa).

Inicialmente todos los datos están en la secuencia desordenada. En cada paso se toma un elemento del conjunto desordenado (el siguiente) y se inserta en el sitio apropiado de la secuencia ordenada (de ahí el nombre del algoritmo). Como resultado, la zona desordenada va disminuyendo y la zona ordenada aumentando. Al final todos los datos estarán en el conjunto ordenado.

Inserción Directa es muy útil cuando tenemos que ordenar pocos datos y sabemos que están bastante ordenados. Si no es así, aunque sean pocos datos, es mejor emplear Selección Directa.

(Continuará en el próximo número)

Bibliografía

[1] T.H. Cormen, C.E. Leiseron y R.L. Rivest, Introduction to Algorithms, The MIT Press, 1994.

[2] C.A.R. Hoare, "Quicksort", The Computer Journal, Vol. 5, No. 1, pp. 10-15, Abril 1962.

[3] J. Jaja, "A perspective on Quicksort", Computing in Science and Engineering, pp. 43-49, Enero-Febrero 2000.

[4] D.E. Knuth, El arte de programar ordenadores. Volumen 3: Ordenación y búsqueda, Editorial Reverté, 1986.

[5] R. Sedgewick, Algoritmos en C++, Editorial Díaz de Santos, 1995.

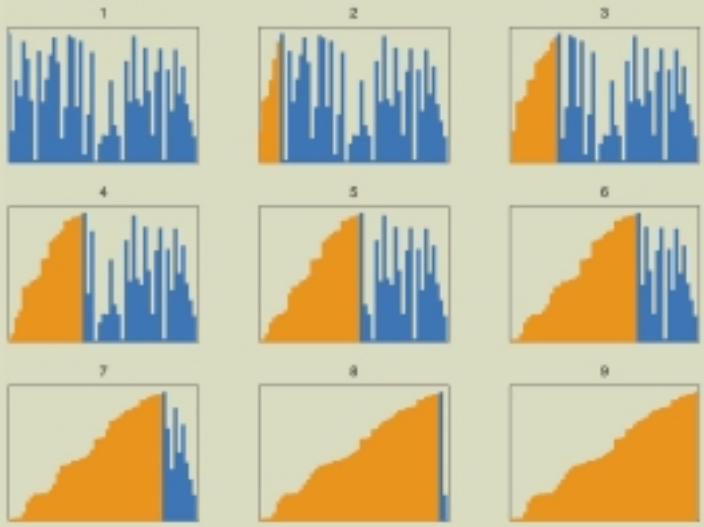
[6] M.A. Weiss, Estructuras de datos en Java, Addison Wesley, 2000.

[7] N. Wirth, Algoritmos + estructuras de datos = programas, Ediciones del Castillo, 1985.

FIGURA 5. INSERCIÓN DIRECTA

Hay dos zonas: una zona desordenada (en azul) y una zona ordenada (en naranja).

En cada iteración el algoritmo toma un elemento de la zona desordenada (el siguiente) y lo inserta en el sitio apropiado de la zona ordenada.




```
Inserción Directa(A)
1 For i=2 to Tamaño(A)
        carta = A[i];  % selecciona última carta
        j=i-1;
        2 While (j>0 & carta < A[j])
                A[j+1]=A[j];
                j=j-1;
        end 2
        A[j+1]=carta;   % inserta la carta
end 1
```

Elementos de la zona ordenada

Elementos de la zona desordenada