Algoritmos de ordenación internos

Shell:

La ordenación Shell debe el nombre a su inventor, D. L. Shell. Se suele denominar también ordenación por inserción con incrementos decrecientes. Se considera que el método Shell es una mejora del método de inserción directa.

En el algoritmo de inserción, cada elemento se compara con los elementos contiguos de su izquierda, uno tras otro. Si el elemento a insertar es el más pequeño, hay que realizar muchas comparaciones antes de colocarlo en su lugar definitivo. El algoritmo de Shell modifica los saltos contiguos resultantes de las comparaciones por saltos de mayor tamaño, y con ello se consigue que la ordenación sea más rápida. Generalmente, se toma como salto inicial n/2 (siendo n el número de elementos), y luego se reduce el salto a la mitad en cada repetición hasta que sea de tamaño 1. El Ejemplo 6.1 ordena una lista de elementos siguiendo paso a paso el método de Shell.

Ejemplo 6.1

Obtener las secuencias parciales del vector al aplicar el método Shell para ordenar de modo creciente la lista:

6 5 2 3 4 0

El número de elementos que tiene la lista es 6, por lo que el salto inicial es *6/2 = 3*. La siguiente tabla muestra el número de recorridos realizados en la lista con los saltos correspondiente.

Recorrido	*Salto*	*Intercambios*	*Lista*
1	3	(6,2),(5,4),(6,0)	2 4 0 3 5 6
2	3	(2,0)	0 4 2 3 5 6
3	3	ninguno	0 4 2 3 5 6
salto 3/2=1			
4	3	(4,2),(4,3)	0 2 3 4 5 6
5	1	ninguno	0 2 3 4 5 6

Los pasos a seguir por el algoritmo para una lista de n elementos son:

1. Se divide la lista original en n/2 grupos de dos, considerando un incremento o salto entre los elementos de n/2.
2. Se clasifica cada grupo por separado, comparando las parejas de elementos, y si no están ordenados se intercambian.
3. Se divide ahora la lista en la mitad de grupos (n/4), con un incremento o salto entre los elementos también mitad (n/4), y nuevamente se clasifica cada grupo por separado.

4. Así sucesivamente, se sigue dividiendo la lista en la mitad de grupos que en el recorrido anterior con un incremento o salto decreciente en la mitad que el salto anterior, y luego clasificando cada grupo por separado.
5. El algoritmo termina cuando se llega a que el tamaño del salto es 1.

Por consiguiente, los recorridos por la lista están condicionados por el bucle,

intervalo ← n / 2 mientras (intervalo > 0) hacer

Para dividir la lista en grupos y clasificar cada grupo se anida este código,

```
desde i ← (intervalo + 1) hasta n hacer
        j ← i - intervalo
        mientras (j > 0) hacer
                k ← j + intervalo
                si (a[j] <= a[k]) entonces
                        j ← 0
                sino
                        Intercambio (a[j], a[k]);
                        j ← j - intervalo
                fin _ si
        fin _ mientras
fin _ desde
```

donde se observa que se comparan pares de elementos de índice j y k, a[j], a[k]), separados por un salto de intervalo. Así, si n = 8, el primer valor de intervalo = 4, y los índices i = 5, j = 1, k = 6. Los siguientes valores que toman i = 6, j = 2, k = 7 y así hasta recorrer la lista.

Para realizar un nuevo recorrido de la lista con la mitad de los grupos, el intervalo se reduce a la mitad.

Intervalo ← intervalo / 2

Y así se repiten los recorridos por la lista, mientras intervalo > 0.

6.6.2. Codificación del algoritmo de ordenación Shell

Al codificar el algoritmo es preciso considerar que Java toma como base en la indexación de *arrays* índice 0 y, por consiguiente, se han de desplazar una posición *a la izquierda* las variables índice respecto a lo expuesto en el algoritmo.

```
public static void ordenacionShell(double a[])
{
   int intervalo, i, j, k;
   int n= a.length;

   intervalo = n / 2;
   while (intervalo > 0)
   {
      for (i = intervalo; i < n; i++)
      {
         j = i - intervalo;
         while (j >= 0)
         {
            k = j + intervalo;
            if (a[j] <= a[k])
               j = -1; // par de elementos ordenado
            else
            {
               intercambiar(a, j, j+1);
               j -= intervalo;
            }
         }
      }
      intervalo = intervalo / 2;
   }
}
```

Quicksort:

El algoritmo conocido como Quicksort (ordenación rápida) recibe su nombre de su autor, Tony Hoare. La idea del algoritmo es simple, se basa en la división en particiones de la lista a ordenar, por ello se puede considerar que aplica la técnica "divide y vencerás". El método es, posiblemente, el más pequeño de código, más rápido, más elegante y más interesante y eficiente de los algoritmos conocidos de ordenación. Este método se basa en dividir los n elementos de la lista a ordenar en dos partes o particiones separadas por un elemento: una partición izquierda, un elemento central denominado pivote o elemento de partición y una partición derecha. La partición o división se hace de tal forma que todos los elementos de la primera sublista (partición izquierda) sean menores que todos los elementos de la segunda sublista (partición derecha). Las dos sublistas se ordenan entonces independientemente. Para dividir la lista en particiones (sublistas) se elige uno de los elementos de la lista y se utiliza como pivote o elemento de partición. Si se elige una lista cualquiera con los elementos en orden aleatorio, se puede elegir cualquier elemento de la lista como pivote, por ejemplo, el primer elemento de la lista. Si la lista tiene algún orden parcial que se conoce, se puede tomar otra decisión para escogerlo. Idealmente, el pivote se debe elegir de modo que se divida la lista exactamente por la mitad de acuerdo al tamaño relativo de las claves. Por ejemplo, si se tiene una lista de enteros de 1 a 10, 5 o 6 serían pivotes ideales, mientras que

1 o 10 serían elecciones “pobres” de pivotes. Una vez que el pivote ha sido elegido, se utiliza para ordenar el resto de la lista en dos sublistas: una tiene todas las claves menores que el pivote y la otra, todos los elementos (claves) mayores o iguales que el pivote (o al revés). Estas dos listas parciales se ordenan recursivamente utilizando el mismo algoritmo; es decir, se llama sucesivamente al propio algoritmo quicksort. La lista final ordenada se consigue concatenando la primera sublista, el pivote y la segunda lista, en ese orden, en una única lista. La primera etapa de Quicksort es la división o “particionado” recursivo de la lista hasta que todas las sublistas constan de sólo un elemento.

Ejemplo 6.2

Se ordena una lista de números enteros aplicando el algoritmo quicksort, *como pivote se elige el primer elemento de la lista.*

1. *Lista original* 5 2 1 9 3 8 7

pivote elegido 5

sublista izquierda1 (elementos menores que 5) 2 1 3

sublista derecha1 (elementos mayores o iguales a 5) 9 8 7

2. El algoritmo se aplica a la sublista izquierda

Sublista Izda1 2 1 3

pivote

sublista Izda 1
sublista Dcha 3

Sublista Izda1

Izda	pivote	Dcha
1	2	3

3. El algoritmo se aplica a la sublista derecha

9 8 7

pivote

sublista Izda 7
sublista Dcha 8

Sublista Dcha1

Izda	pivote	Dcha
7	8	8

4. *Lista ordenada final*

Sublista izquierda	*pivote*	*Sublista derecha*
1 2 3	5	7 8 9

Burbuja:

El método de la burbuja es una comparación lineal con cada uno de los elementos, el elemento que sea menor contra el que se está comparado intercambiaran posiciones. Este método no es recomendado para grandes comparaciones, ya que es un proceso muy lento y requiere de una gran cantidad de Memoria RAM.

```
procedimiento DeLaBurbuja (a0, a1, a2, ..., a(n-1))
para i ← 1 hasta n hacer
para j ← 0 hasta n − i hacer
si a(j) > a(j+1) entonces
aux ← a(j)
a(j) ← a(j+1)
a(j+1) ← aux
fin si
fin para
fin para
fin procedimiento
```

Radix:

En informática, el ordenamiento Radix (radix sort en inglés) es un algoritmo de ordenamiento que ordena enteros procesando sus dígitos de forma individual. Como los enteros pueden representar cadenas de caracteres (por ejemplo, nombres o fechas) y, especialmente, números en punto flotante especialmente formateados, radix sort no está limitado sólo a los enteros.

Ejemplo

Vector original:

25 57 48 37 12 92 86 33

Asignamos los elementos en colas basadas en el dígito menos significativo de cada uno de ellos.

0:

1:

2:12 92

3:33

4:

5:25
6:86
7:57 37
8:48
9:
Después de la primera pasada, la ordenación queda:
12 92 33 25 86 57 37 48

Colas basadas en el dígito más significativo.
0:
1:12
2:25
3:33 37
4:48
5:57
6:
7:
8:86
9:92

Lista ordenada:
12 25 33 37 48 57 86 92

Referencias bibliográficas:

Libro: Estructuras de datos en Java

Autores: Luis Joyanes Aguilar Ignacio Zahonero Martínez

Edificio Valrealty, 1. ª Planta

Basauri, 17 28023 Aravaca (Madrid)

Editor: José Luis García Técnico

Editorial: Blanca Pecharromán

Compuesto en: Gesbiblo, S. L.

Diseño de cubierta: Gesbiblo, S. L.