ORDENAMIENTO POR OSCILACION

Por JORGE PODJARNY * y JULIO C. GIANIBELLI **

INTRODUCCION

La aparición de la microcomputadoras en el campo de la informática trajo aparejada la necesidad de definir nuevos parámetros de optimización para el desarrollo de la programación, ya que las soluciones óptimas para macrosistemas (gran capacidad de memoria y muy alta velocidad de comunicación periférica), dejaron de serlo en el caso de memoria principal reducida (64 KB) y memoria periférica comparativamente lenta (minidiscos flexibles).

Asimismo, las microcomputadoras no son normalmente programadas por especialistas en el área, sino por los mismos usuarios, lo que requiere que las rutinas de uso permanente sean de fácil conocimiento e implementación, en contraste con las bibliotecas de utilitarios normalmente existentes en los macrosistemas.

Uno de los problemas más comunes con los que se enfrenta el programador durante el desarrollo de los sistemas de aplicación, es la necesidad de ordenar un archivo de acuerdo a un campo preestablecido; esto, en los procesadores que soportan base de datos, está resuelto por el apoyo brindado por la base misma, no siendo así en

* Jefe de Trabajos Prácticos de la Cátedra de Optimización. Investigador del Instituto de Investigación y Práctica de la Facultad de Matemática Aplicada de la UCALP. Lic. en Análisis de Sistemas.

** Profesor Asociado de la Cátedra de Optimización. Director del Instituto de Investigación y Práctica de la Facultad de Matemática Aplicada de la UCALP. Lic. en Geofísica.

el caso de los microcomputadores, ya que por su capacidad intrínseca no pueden soportarla viéndose obligado el programador a realizar el ordenamiento de la información a través de un algoritmo propio.

Existe abundante bibliografía al respecto, orientada fundamentalmente a los macrosistemas, siendo por lo tanto compleja su utilización, por parte de usuarios de microcomputadoras, lo que lleva a la necesidad de desarrollar algoritmos específicos para dichos usuarios.

ESPECIFICACIONES

De acuerdo a las características de ordenamiento de los datos podemos clasificar a los archivos en:

1. Desordenados.
2. Parcialmente ordenados.
3. Ordenados.

Dentro de los archivos parcialmente ordenados debemos distinguir entre las siguientes categorías:

1. Ordenados por sectores.
2. Parcialmente ordenados propiamente dichos.

Ordenados por Sectores.

Son aquellos archivos en los que conocemos los límites de los sectores ordenados; en este caso lo conveniente es realizar un ordenamiento por lista a través de un "MERGE".

Parcialmente ordenados propiamente dichos.

Son aquellos en que sus características, ya sea a través del ordenamiento previo por un campo diferente al actualmente requerido, o por la generación misma de la información, trae implícita un parcial ordenamiento; por ejemplo en el primer caso, un archivo previamente ordenado por número de documento que deseamos ordenarlo por edad; en el segundo un archivo de un plan de cuentas de una contabilidad que en el momento de su generación se realizó siguiendo la lógica contable.

Este segundo caso es el que queremos atacar a través de un algoritmo que reúna las características previamente enunciadas (sim-

plicidad en su lógica y fácil implementación) para su uso en microcomputadoras.

ALGORITMO

Dado un vector con los datos a ordenar, la idea básica del algoritmo consiste en comparar los elementos del mismo de a pares consecutivos, en dos recorridas del vector.

En la primer recorrida comparamos los elementos de la siguiente manera: 1 con 2, 3 con 4, etc., invirtiendo aquellos que no están ordenados.

En la segunda recorrida comparamos los elementos: 2 con 3, 4 con 5, etc., invirtiendo aquellos que no están ordenados.

De esta manera colocando una bandera (flas) en el caso de haber realizado una inversión, (lo que implica que un elemento no estaba en su lugar correcto), podemos detectar el fin del ordenamiento por ausencia de la misma.

Para la realización de los ciclos (recorridas) debemos calcular los límites superiores, ya que difieren en el caso de que la cantidad de elementos sea par o impar.

Para el caso par el límite superior del primer ciclo es igual a la cantidad de elementos menos 1, ya que la comparación se realiza del elemento iesimo con el elemento iesimo + 1, siendo este último, para el caso de i = al límite superior, el último elemento del vector, y el límite superior del segundo ciclo es igual a la cantidad de elementos menos 2 por ser, para i = al límite superior, el último elemento impar del vector.

Para el caso impar los límites se invierten: el límite superior del primer ciclo es igual a la cantidad de elementos menos 2 y el límite superior para el segundo ciclo es igual a la cantidad de elementos menos 1.

Si el vector posee un solo elemento, éste está obviamente ordenado.

Si el vector posee 2 elementos, sólo es necesario compararlos entre sí, e invertirlos en caso de que estén desordenados.

Para 3 ó más elementos ya el algoritmo realiza el ordenamiento.

Si la cantidad de elementos supera la capacidad de memoria de la microcomputadora, puede realizarse el ordenamiento sobre el so-

porte de memoria periférica (minidisco flexible) siempre y cuando el archivo sea un archivo de acceso directo (Random).

RENDIMIENTO

Por la cantidad de comparaciones realizadas, en el caso de un archivo totalmente desordenado, es equivalente a un ordenamiento por peso (Sort de Burbuja o sort Pesado); para archivos parcialmente ordenados, al detectar el fin del proceso, su rendimiento se acrecienta notablemente, siendo en algunos casos, equivalente al de un SORT dicotómico.

[+ 1] EJEMPLOS

Para facilitar su implementación, se adjuntan dos programas, el primero de ellos totalmente desarrollado y el segundo como subrutina empleando todas las facilidades brindadas por el BASIC específico.

Se eligió el lenguaje BASIC para los ejemplos por ser el lenguaje común a las microcomputadoras.

```
100 REM SORT POR OSCILACION.
110 REM ESTE PROGRAMA FUE REALIZADO UTILIZANDO UNICAMENTE LAS SENTENCIAS BASIC.
120 REM COMUNES A TODAS LAS VERSIONES EXISTENTES.
130 REM CI = CANTIDAD DE ELEMENTOS DEL VECTOR A ORDENAR.
140 REM A ( ) = VECTOR A ORDENAR.
150 REM L1 = LIMITE SUPERIOR DEL PRIMER CICLO.
160 REM L2 = LIMITE SUPERIOR DEL SEGUNDO CICLO.
170 REM MA = BANDERA 0 = > VECTOR ORDENADO.
180 REM                1 = > VECTOR NO ORDENADO.
190 REM RE = VARIABLE DE RESERVA DE VALORES.
200 REM I = INDICE DE LOS CICLOS.
210 PRINT "CANTIDAD DE ELEMENTOS A ORDENAR".
220 INPUT CI.
230 IF CI<1 THEN END.
240 DIM A(CI).
250 REM CARGA DEL VECTOR A ( ) CON VALORES ALEATORIOS.
260 FOR I = 1 TO CI.
270 A(I) = RND(I).
280 NEXT I.
290 REM VERIFICACION DE LA CANTIDAD DE ELEMENTOS DEL VECTOR.
300 IF CI > 2 THEN 360.
310 IF CI = 1 THEN 560.
320 REM RUTINA NARA EL CASO DE CI = 2.
```

```
330 I = 1.
340 GOSUB 620.
350 GOTO 560.
360 REM CALCULO DE LOS LIMITES DE LOS CICLOS.
370 REM VERIFICACION DE CI PAR O IMPAR.
380 IF CI/2 — INT (CI/2) >. 1 THEN 430.
390 REM CALCULO DE LOS LIMITES PARA CI PAR.
400 L1 = CI — 1.
410 L2 = CI — 2.
420 GOTO 460.
430 REM CALCULO DE LOS LIMITES PARA CI IMPAR.
440 L1 = CI — 2.
450 L2 = CI — 1.
460 REM              ORDENAMIENTO.
470 MA = 0
480 FOR I = 1 TO L1 STEP 2.
490 GOSUB 610.
500 NEXT I.
510 FOR I = 2 TO L2 STEP 2.
520 GOSUB 610.
530 NEXT I.
540 REM CONTROL DE FIN DEL ORDENAMIENTO.
550 IF MA = 1 THEN 470.
560 REM        FIN DEL ORDENAMIENTO.
570 FOR I = 1 TO CI.
580 PRINT A (I).
590 NEXT I.
600 END.
610 REM SUBRUTINA DE INVERSION DE VALORES EN EL VECTOR.
620 IF A (I) <A (I + 1) THEN RETURN.
630 RE = A (I).
640 A (I) = A (I + 1).
650 A (I + 1) = RE.
660 MA = 1.
670 RETURN.
    READY.
 10 REM PROGRAMA DE ORDENAMIENTO.
 20 REM CT % = CANTIDAD DE ELEMENTOS DEL VECTOR A OR-
    DENAR.
 30 REM L1 % = LIMITE SUPERIOR DEL PRIMER CICLO.
 40 REM L2 % = LIMITE SUPERIOR DEL SEGUNDO CICLO.
 50 REM I    = INDICE DE LOS CICLOS.
 60 INPUT "CANTIDAD DE ELEMENTOS": CT %.
```

```
 70 DIM A (CT%).
 80 FOR I = 1 TO CT%: A (I) = RND (1): PRINT A (I): NEXT.
 90 GOSUB 110.
100 FOR I = 1 TO CT%: PRINT A (I): NEXT I: END.
110 REM RUTINA DE ORDENAMIENTO.
120 L1% = CT% — 2: L2% = CT% — 1: IF CT% / 2
— CT% / 2 <. 1 THEN SWAP L1%, L2%.
130 MA% = 1.
140 WHILE MA% <> 0: MA% = 0.
150 FOR I% = 1 TO L1%: GOSUB 180: NEXT I%.
160 FOR I% = 1 TO L2%: GOSUB 180: NEXT I%.
170 WEND. RETURN.
180 IF A (I%) > A (I% + 1) THEN SWAP A (I%), A (I% + 1):
    MA% = 1.
190 RETURN.
```