Algoritmos de Ordenamiento

¿Qué es ordenamiento?

Es la operación de arreglar los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento.
El ordenamiento se efectúa con base en el valor de algún campo en un registro.
El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado.
Ej. de ordenamientos:
Dir. telefónico, tablas de contenido, bibliotecas y diccionarios, etc.

El ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia tal que represente un orden, el cual puede ser numérico, alfabético o incluso alfanumérico, ascendente o descendente.

¿Cuándo conviene usar un método de ordenamiento?
Cuando se requiere hacer una cantidad considerable de búsquedas y es importante el factor tiempo.

Tipos de ordenamientos:
Los 2 tipos de ordenamientos que se pueden realizar son: los internos y los externos.

Los internos:
Son aquellos en los que los valores a ordenar están en memoria principal, por lo que se asume que el tiempo que se requiere para acceder cualquier elemento sea el mismo (a[1], a[500], etc).

Los externos:
Son aquellos en los que los valores a ordenar están en memoria secundaria (disco, cinta, cilindro magnético, etc), por lo que se asume que el tiempo que se requiere para acceder a cualquier elemento depende de la última posición accesada (posición 1, posición 500, etc).

Eficiencia en tiempo de ejecución:

Una medida de eficiencia es:
Contar el # de comparaciones (C)
Contar el # de movimientos de items (M)
Estos están en función de el #(n) de items a ser ordenados.
Un "buen algoritmo" de ordenamiento requiere de un orden *nlogn* comparaciones.

La eficiencia de los algoritmos se mide por el número de comparaciones e intercambios que tienen que hacer, es decir, se toma n como el número de elementos que tiene el

arreglo o vector a ordenar y se dice que un algoritmo realiza O(n2) comparaciones cuando compara n veces los n elementos, n x n = n2

Algoritmos de ordenamiento:

Internos:

- Inserción directa.
    - Inserción directa.
    - Inserción binaria.
- Selección directa.
    - Selección directa.
- Intercambio directo.
    - Burbuja.
    - Shake.
- Inserción disminución incremental.
    - Shell.
- Ordenamiento de árbol.
    - Heap.
    - Tournament.
- Sort particionado.
    - Quick sort.
- Merge sort.
- Radix sort.
- Cálculo de dirección.

Externos:

- Straight merging.
- Natural merging.
- Balanced multiway merging.
- Polyphase sort.
- Distribution of initial runs.

Clasificación de los algoritmos de ordenamiento de información:

El hecho de que la información está ordenada, nos sirve para poder encontrarla y accesarla de manera más eficiente ya que de lo contrario se tendría que hacer de manera secuencial.

A continuación se describirán 4 grupos de algoritmos para ordenar información:

Algoritmos de inserción:

En este tipo de algoritmo los elementos que van a ser ordenados son considerados uno a la vez. Cada elemento es INSERTADO en la posición apropiada con respecto al resto de los elementos ya ordenados.

Entre estos algoritmos se encuentran el de INSERCION DIRECTA, SHELL SORT, INSERCION BINARIA y HASHING.

Algoritmos de intercambio:

En este tipo de algoritmos se toman los elementos de dos en dos, se comparan y se INTERCAMBIAN si no están en el orden adecuado. Este proceso se repite hasta que se ha analizado todo el conjunto de elementos y ya no hay intercambios.
Entre estos algoritmos se encuentran el BURBUJA y QUICK SORT.

Algoritmos de selección:

En este tipo de algoritmos se SELECCIONA o se busca el elemento más pequeño (o más grande) de todo el conjunto de elementos y se coloca en su posición adecuada. Este proceso se repite para el resto de los elementos hasta que todos son analizados. Entre estos algoritmos se encuentra el de SELECCION DIRECTA.

Algoritmos de enumeración:

En este tipo de algoritmos cada elemento es comparado contra los demás. En la comparación se cuenta cuántos elementos son más pequeños que el elemento que se está analizando, generando así una ENUMERACION. El número generado para cada elemento indicará su posición.

Los métodos simples son: Inserción (o por inserción directa), selección, burbuja y shell, en dónde el último es una extensión al método de inserción, siendo más rápido. Los métodos más complejos son el quick-sort (ordenación rápida) y el heap sort.
A continuación se mostrarán los métodos de ordenamiento más simples.

METODO DE INSERCIÓN.

Este método toma cada elemento del arreglo para ser ordenado y lo compara con los que se encuentran en posiciones anteriores a la de él dentro del arreglo. Si resulta que el elemento con el que se está comparando es mayor que el elemento a ordenar, se recorre hacia la siguiente posición superior. Si por el contrario, resulta que el elemento con el que se está comparando es menor que el elemento a ordenar, se detiene el proceso de comparación pues se encontró que el elemento ya está ordenado y se coloca en su posición (que es la siguiente a la del último número con el que se comparó).

Procedimiento ***Insertion Sort***

Este procedimiento recibe el arreglo de datos a ordenar **a[]** y altera las posiciones de sus elementos hasta dejarlos ordenados de menor a mayor. **N** representa el número de elementos que contiene a[].

```
paso 1: [Para cada pos. del arreglo]  For i <- 2 to N do
paso 2: [Inicializa v y j]                          v <- a[i]
                                                    j <- i.
paso 3: [Compara v con los anteriores]        While a[j-1] > v AND j>1 do
paso 4: [Recorre los datos mayores]                 Set a[j] <- a[j-1],
paso 5: [Decrementa j]                              set j <- j-1.
paso 5: [Inserta v en su posición]            Set a[j] <- v.
paso 6: [Fin]                                 End.
```

Ejemplo:

Si el arreglo a ordenar es a = ['a','s','o','r','t','i','n','g','e','x','a','m','p','l','e'], el algoritmo va a recorrer el arreglo de izquierda a derecha. Primero toma el segundo dato 's' y lo asigna a *v* y *i* toma el valor de la posición actual de *v*.

Luego compara esta 's' con lo que hay en la posición j-1, es decir, con 'a'. Debido a que 's' no es menor que 'a' no sucede nada y avanza i.

Ahora v toma el valor 'o' y lo compara con 's', como es menor recorre a la 's' a la posición de la 'o'; decrementa j, la cual ahora tiene la posición en dónde estaba la 's'; compara a 'o' con a[j-1] , es decir, con 'a'. Como no es menor que la 'a' sale del for y pone la 'o' en la posición a[j]. El resultado hasta este punto es el arreglo siguiente: a = ['a','o','s','r',....]

Así se continúa y el resultado final es el arreglo ordenado :
a = ['a','a','e','e','g','i','l','m','n','o','p','r','s','t','x']

MÉTODO DE SELECCIÓN.

El método de ordenamiento por selección consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo mas pequeño, y así sucesivamente hasta ordenar todo el arreglo.

Procedimiento ***Selection Sort***

```
paso 1: [Para cada pos. del arreglo]       For i <- 1 to N do
paso 2: [Inicializa la pos. del menor]              menor <- i
paso 3: [Recorre todo el arreglo]                   For j <- i+1 to N do
paso 4: [Si a[j] es menor]
        If a[j] < a[menor] then
paso 5: [Reasigna el apuntador al menor]                          min =
j
paso 6: [Intercambia los datos de la pos.
           min y posición i]
Swap(a, min, j).
paso 7: [Fin]
End.
```

Ejemplo:

El arreglo a ordenar es a = ['a','s','o','r','t','i','n','g','e','x','a','m','p','l','e'].
Se empieza por recorrer el arreglo hasta encontrar el menor elemento. En este caso el menor elemento es la primera 'a'. De manera que no ocurre ningún cambio. Luego se procede a buscar el siguiente elemento y se encuentra la segunda 'a'.
Esta se intercambia con el dato que está en la segunda posición, la 's', quedando el arreglo así después de dos recorridos: a = ['a','a','o','r','t','i','n','g','e','x','s','m','p','l','e'].

El siguiente elemento, el tercero en orden de menor mayor es la primera 'e', la cual se intercambia con lo que está en la tercera posición, o sea, la 'o'. Le sigue la segunda 's', la cual es intercambiada con la 'r'.

El arreglo ahora se ve de la siguiente manera: a = ['a','a','e','e','t','i','n','g','o','x','s','m','p','l','r'].

De esta manera se va buscando el elemento que debe ir en la siguiente posición hasta ordenar todo el arreglo.

El número de comparaciones que realiza este algoritmo es :
Para el primer elemento se comparan n-1 datos, en general para el elemento i-ésimo se hacen n-i comparaciones, por lo tanto, el total de comparaciones es:
la sumatoria para i de 1 a n-1 (n-i) = 1/2 n (n-1).

MÉTODO BURBUJA.

El bubble sort, también conocido como ordenamiento burbuja, funciona de la siguiente manera: Se recorre el arreglo intercambiando los elementos adyacentes que estén desordenados. Se recorre el arreglo tantas veces hasta que ya no haya cambios. Prácticamente lo que hace es tomar el elemento mayor y lo va recorriendo de posición en posición hasta ponerlo en su lugar.

Procedimiento ***Bubble Sort***

```
paso 1: [Inicializa i al final de arreglo]          For i <- N down to 1
do
paso 2: [Inicia desde la segunda pos.]                 For j <- 2 to i
do
paso 4: [Si a[j-1] es mayor que el que le sigue]           If a[j-1] <
a[j] then
paso 5: [Los intercambia]
Swap(a, j-1, j).
paso 7: [Fin]
End.
```

Tiempo de ejecución del algoritmo burbuja:

- Para el mejor caso (un paso) O(n)
- Peor caso n(n-1)/2
- Promedio $O(n^2)$

MÉTODO DE SHELL.

Ordenamiento de disminución incremental.

Nombrado así debido a su inventor Donald Shell.

Ordena subgrupos de elementos separados K unidades (respecto de su posición en el arreglo) del arreglo original. El valor K es llamado incremento.

Después de que los primeros K subgrupos han sido ordenados (generalmente utilizando INSERCION DIRECTA), se escoge un nuevo valor de K más pequeño, y el arreglo es de nuevo partido entre el nuevo conjunto de subgrupos. Cada uno de los subgrupos mayores es ordenado y el proceso se repite de nuevo con un valor más pequeño de K.

Eventualmente el valor de K llega a ser 1, de tal manera que el subgrupo consiste de todo el arreglo ya casi ordenado.

Al principio del proceso se escoge la secuencia de decrecimiento de incrementos; el último valor debe ser 1.

"Es como hacer un ordenamiento de burbuja pero comparando e intercambiando elementos."

Cuando el incremento toma un valor de 1, todos los elementos pasan a formar parte del subgrupo y se aplica inserción directa.

El método se basa en tomar como salto N/2 (siendo N el número de elementos) y luego se va reduciendo a la mitad en cada repetición hasta que el salto o distancia vale 1.

```
Procedimiento Shell Sort;

const

        MAXINC = _____;

incrementos = array[1..MAXINC] of integer;

var

        j,p,num,incre,k:integer;

begin

        for incre := 1 to MAXINC do begin /* para cada uno de los
incrementos */

                k := inc[incre];       /* k recibe un tipo de incremento
*/

                for p := k+1 to MAXREG do begin /* inserción directa para
el grupo que se encuentra cada K posiciones */

                        num := reg[p];

                        j := p-k;

                        while (j>0) AND (num < reg[j]) begin

                                reg[j+k] := reg[j];

                                j := j - k;

                        end;

                        reg[j+k] := num;

                end

        end
end;
```

Ejemplo:

Para el arreglo a = [6, 1, 5, 2, 3, 4, 0]

Tenemos el siguiente recorrido:

Recorrido	Salto	Lista Ordenada	Intercambio
1	3	2,1,4,0,3,5,6	(6,2), (5,4), (6,0)
2	3	0,1,4,2,3,5,6	(2,0)
3	3	0,1,4,2,3,5,6	Ninguno
4	1	0,1,2,3,4,5,6	(4,2), (4,3)
5	1	0,1,2,3,4,5,6	Ninguno

Autor: Raiger Murillo Moreno
Email: rmurillo95@yahoo.com
Ingeniero de Sistemas