GRUPO STUDIUM FORMACIÓN

Taller Tema 3: Búsqueda y Ordenación de tablas

Introducción

En la teoría del presente tema hemos visto los diferentes usos de las tablas y su gran utilidad. Hemos visto una serie de operaciones como son el rellenado inicial, llenado introduciendo los valores por teclado, recorrer las tablas para realizar ciertas operaciones a todos sus elementos, etc.

En el taller práctico del tema vamos a trabajar con dos nuevas operaciones: la búsqueda de un elemento en la tabla y la ordenación de los elementos de una tabla según un criterio.

Búsqueda de un elemento en una tabla no ordenada

Partimos de dos posibilidades de encontrar un elemento en una tabla: si la tabla está o no ordenada.

Primero veremos el caso de que no esté ordenada y tras la explicación de la ordenación, se afrontará la búsqueda en una tabla ordenada.

En el caso de que la tabla NO esté ordenada, para encontrar un elemento no tenemos más remedio que recorrer **toda** la tabla o como mínimo hasta que encontremos el elemento, que para el peor de los escenarios posibles, estará en la última posición o peor aún, no lo encontraremos. Esta posibilidad se debe tener en cuenta: el elemento no está en la tabla.

Partimos de una tabla de 10 elementos de tipo entero ya rellena (tabla[10]). Pediremos por teclado un elemento que será el buscado (elemento). El algoritmo sería de la forma:

```
ENTERO tabla[10]
ENTERO i
BOOLEANO encontrado
ENTERO elemento

PARA i = 0 HASTA 9 HACER
      ESCRIBIR "Dame un valor"
      LEER tabla[i]
FIN PARA

ESCRIBIR "Ahora dame el valor a buscar"
LEER elemento

encontrado = FALSO
i = 0
MIENTRAS (NO encontrado) Y (i<10) HACER
      SI tabla[i] = elemento ENTONCES
            encontrado = VERDADERO
      SINO
            i++
      FIN SI
FIN MIENTRAS
SI encontrado ENTONCES
      ESCRIBIR "El elemento se encontró en la posición " + i
SINO
      ESCRIBIR "El elemento NO se ha encontrado"
FIN SI
```

En primer lugar, declaramos una variable *encontrado* que nos servirá para saber si he encontrado o no el elemento buscado. También sirve para salir del bucle MIENTRAS cuando encuentre el elemento.

La variable *i* servirá para recorrer la tabla.

En la condición del MIENTRAS tenemos dos condiciones: Nos saldremos del bucle cuando no se cumpla alguna de las condiciones, o bien YA haya encontrado el elemento, o bien hayamos llegado al final de la tabla (i=10).

Dentro del bucle, simplemente debemos comprobar si el elemento buscado (*elemento*) coincide con el elemento de la tabla en la posición actual (*tabla[i]*).

La última instrucción del bucle sería avanzar al siguiente elemento de la tabla, aumentando el índice *i*.

Una vez fuera del bucle, preguntamos simplemente por la variable *encontrado*, si tiene el valor VERDADERO, se mostrará el mensaje indicando la posición en la que se ha encontrado, que se ha quedado registrada en la variable *i*.

Si es FALSO, se indica que se ha llegado al final de la tabla y NO se ha encontrado el elemento buscado.

Ordenación de una tabla

Pasemos ahora a ordenar una tabla. Existen varios métodos para la ordenación de tablas, desde los más sencillos y lentos a otros más elaborados y complejos pero mucho más rápidos.

Los ejemplos aquí mostrados se harán con una tabla de 10 números enteros, pero con mínimas modificaciones se pueden ordenar tablas de reales o de cadenas o de Objetos.

Empezaremos con el método de "**Intercambio**" por ser el más sencillo e intuitivo. Este método consiste en comparar cada elemento de cada posición con el resto de los elementos e ir intercambiando en caso de que no estén ordenados.

- El elemento en la posición 1 con el elemento de la posición 2, con la posición 3, con la posición 4,... con el último.
- El elemento en la posición 2 con el de la posición 3, con el de la posición 4, con el de la posición 5, ...
- ...
- El penúltimo con el último.

Si tenemos N elementos, se realizarán N-1 comparaciones, luego el tiempo de ejecución será de O(n).

Veamos el algoritmo al detalle:

```
ENTERO array[10]
ENTERO i,j,aux
PARA i = 0 HASTA 9 HACER
      ESCRIBIR "Dame un valor"
      LEER array[i]
FIN PARA

PARA i = 0 HASTA 9-1 HACER
      PARA j=i+1 HASTA 9 HACER
            SI array[i] > array[j] ENTONCES
                  aux = array[i]
                  array[i] = array[j]
                  array[j] = aux
           FINSI
      FIN PARA
FIN PARA
```

Veamos un sencillo ejemplo:

Ordenemos la siguiente tabla de enteros de menor a mayor: 4 2 5 7 3

Cogemos el elemento de la primera posición (4) y lo comparamos con el segundo (2). Al estar desordenados entre ellos, los intercambiamos y la tabla quedará: 2 4 5 7 3

Ahora comparamos el primer elemento de nuevo, que ahora vale diferente (2) con el tercer elemento (5), pero al estar ordenados no se hace nada.

Comparamos primero con cuarto (7) y con quinto (3) sin hacer nada pues está ordenado. Al acabar de comparar el primero con todos empezamos con el segundo:

Segundo (4) con tercero (5)→Nada

Segundo (4) con cuarto (7)→Nada

Segundo (4) con quinto (3)→Intercambiamos → 2 3 5 7 4

Tercero (5) con cuarto (7)→Nada

Tercero (5) con quinto (4)→Intercambiamos→ 2 3 4 7 5

Cuarto (7) con quinto (5)→Intercambiamos→ 2 3 4 5 7

Y ya tenemos la tabla ordenada.

El siguiente método en complejidad es el de "**Selección**". Este método consiste en buscar el elemento más pequeño de la tabla y ponerlo en primera posición; luego, entre los restantes, se busca el elemento más pequeño y se coloca en segundo lugar, y así sucesivamente hasta colocar el último elemento. Si la tabla tiene N elementos, el número de comprobaciones que hay que hacer es de N*(N-1)/2, luego el tiempo de ejecución está en $O(n^2)$.

Veamos un ejemplo:

Ordenemos la tabla anterior de enteros de menor a mayor: 4 2 5 7 3

Suponemos que el menor está en la primer posición, es decir, indice_menor = 0, y buscamos el índice del elemento menor recorriendo el resto de los elementos. Así llegamos a encontrar que indice_menor vale 1 con valor 2 y lo ponemos en la primera posición: 2 4 5 7 3

Ahora nos vamos a la segunda posición y repetimos el proceso con la búsqueda del índice cuyo elemento es el menor de los que queda, sin contar con el primero que ya está ordenado y en su sitio:

indice_menor = 4 → Intercambiamos → 2 3 5 7 4

Pasamos a la tercera posición

indice_menor = 4 → Intercambiamos → 2 3 4 7 5

Pasamos a la cuarta posición

indice_menor = 4 → Intercambiamos → 2 3 4 5 7

Con este método realizamos el mismo número de comparaciones pero menos intercambios, con lo cual será más rápido que el método anterior.

Veamos el algoritmo al detalle:

```
ENTERO tabla[10]
ENTERO i,j,indice_menor,aux

PARA i = 0 HASTA 9 HACER
      ESCRIBIR "Dame un valor"
      LEER tabla[i]
FIN PARA

PARA i = 0 HASTA 9-1 HACER
      indice_menor = i
      PARA j=i+1 HASTA 9 HACER
            SI tabla[j] < tabla[indice_menor] ENTONCES
                  indice_menor=j
            FINSI
      FIN PARA
      aux = tabla[i]
      tabla[i] = tabla[indice_menor]
      tabla[indice_menor] = aux
FIN PARA
```

Vamos con el método de "**Inserción**". Este método va metiendo los datos en la tabla de forma ordenada, es decir, cogemos el primer elemento y lo colocamos en su posición correcta, después el segundo y así con todos.

```
ENTERO tabla[10]
ENTERO i,j,aux

PARA i = 0 HASTA 9 HACER
      ESCRIBIR "Dame un valor"
      LEER tabla[i]
FIN PARA

PARA i = 0 HASTA 9 HACER
       j = i;
       aux = tabla[i]
       MIENTRAS j > 0 y aux < tabla[j-1] HACER
            tabla[j] = tabla[j-1]
            j = j -1
       FIN MIENTRAS
       tabla[j] = aux
FIN PARA
```

El método consiste en ir cogiendo sublistas empezando con los dos primeros y ordenarlos. Después coger una sublista con los dos ya ordenados y el tercer elemento colocando éste en su lugar correspondiente y así sucesivamente.

Ejemplo: 4 2 5 7 3

Nos quedamos con la sublista primera de dos elementos y los ordenamos:

2 4 5 7 3

Ahora, a la sublista resultante ya ordenada, le añadimos el siguiente elemento: 2 4 5 7 3

Y así sucesivamente: 2 4 5 7 3

Y por último: 2 3 4 5 7

No se realiza ninguna comparación pero sí se hacen muchos intercambios y asignaciones.

El método de la "**Burbuja**" consiste en comparar dos números adyacentes e intercambiar en caso necesario. Se comparan todos con el primero, todos con el segundo menos el último, y así sucesivamente.

```
ENTERO tabla[10]
ENTERO i,j,aux

PARA i = 0 HASTA 9 HACER
      ESCRIBIR "Dame un valor"
      LEER tabla[i]
FIN PARA

PARA i = 0 HASTA 9-1 HACER
      PARA j = 0 HASTA 9 - i - 1 HACER
            SI tabla[j] > tabla[j + 1] ENTONCES
                  aux = tabla[j]
                  tabla[j] = tabla[j + 1]
                  tabla[j + 1] = aux
            FINSI
      FIN PARA
FIN PARA
```

Ordenemos la tabla de nuestros ejemplos: 4 2 5 7 3

Comparamos el primero (4) con el segundo (2) e intercambiamos:

2 4 5 7 3

Ahora el segundo (4) con el tercero (5): 2 **4 5** 7 3

El tercero (5) con el cuarto (7): 2 4 **5 7** 3

El cuarto (7) con el quinto (3) e intercambiamos: 2 4 5 **3 7**

Y así obtenemos el mayor elemento en la última posición. Y continuamos ahora comparando todos con sus siguientes salvo el penúltimo:

2 4 5 3 7

2 **4 5** 3 7

2 4 **5 3** 7

Intercambiamos:

2 4 **3** **5** 7

Y seguimos:

2 **4** 3 5 7

2 **4** **3** 5 7

Intercambiamos:

2 **3** **4** 5 7

Y por último comparamos primero con segundo, aunque ya están ordenados:

2 3 4 5 7

El método de "**Hundimiento**" consiste en un algoritmo similar con la diferencia que realizamos las comparaciones empezando por el final y "bajando" elemento más pequeño al principio en vez de "subir" el más grande hacia arriba.

```
ENTERO tabla[10]
ENTERO i,j,aux

PARA i = 0 HASTA 9 HACER
      ESCRIBIR "Dame un valor"
      LEER tabla[i]
FIN PARA

PARA i = 1 HASTA 9-1 HACER
      PARA j = 9 - 1 HASTA j > = i INCREMENTO -1 HACER
            SI tabla[j - 1] > tabla[j] ENTONCES
                  aux = tabla[j]
                  tabla[j] = tabla[j - 1]
                  tabla[j - 1] = aux
            FINSI
      FIN PARA
FIN PARA
```

El método "**Shell**" es una mejora del método de inserción, utilizado cuando el array tiene un gran número de elementos. En este método no se compara a cada elemento con el de su izquierda, como en el de inserción, sino con el que está a un cierto número de lugares (llamado salto) a su izquierda. Este salto es constante, y su valor inicial es N/2 (siendo N el número de elementos, y siendo división entera). Se van dando pasadas hasta que en una pasada no se intercambie ningún elemento de sitio. Entonces el salto se reduce a la mitad, y se vuelven a dar pasadas hasta que no se intercambie ningún elemento, y así sucesivamente hasta que el salto vale 1.

Por ejemplo, lo pasos para ordenar el array {40,21,4,9,10,35} mediante el método de Shell serían:

Salto=3:

Primera pasada:

{9,21,4,40,10,35} ← se intercambian el 40 y el 9.

{9,10,4,40,21,35} ← se intercambian el 21 y el 10.

Salto=1:

Primera pasada:

{9,4,10,40,21,35} ← se intercambian el 10 y el 4.

{9,4,10,21,40,35} ← se intercambian el 40 y el 21.

{9,4,10,21,35,40} ← se intercambian el 35 y el 40.

Segunda pasada:

{4,9,10,21,35,40} ← se intercambian el 4 y el 9.

Con sólo 6 intercambios se ha ordenado el array, cuando por inserción se necesitaban muchos más.

El algoritmo quedaría en Pseudocódigo de la siguiente manera:

```
ENTERO array[TAM]
ENTERO salto,aux,i
BOOLEANO cambios

PARA i = 0 HASTA TAM-1 HACER
      ESCRIBIR "Dame un valor"
      LEER array[i]
FIN PARA

PARA salto = TAM/2 HASTA salto <> 0 INCREMENTO salto/2 HACER
      cambios = VERDADERO
      MIENTRAS cambios = VERDADERO HACER
            cambios = FALSO
            PARA i = salto HASTA i < TAM HACER
                  SI array[i-salto] > array[i] ENTONCES
                        aux = array[i]
                        array[i] = array[i-salto]
                        array[i-salto] = aux
                        cambios = VERDADERO
                  FINSI
            FIN PARA
      FIN PARA
FIN PARA
```

Por último veremos el método "**Quicksort**". Este método se basa en la táctica "divide y vencerás", que consiste en ir subdividiendo el array en arrays más pequeños, y ordenar éstos. Para hacer esta división, se toma un valor del array como pivote, y se mueven todos los elementos menores que este pivote a su izquierda, y los mayores a su derecha. A continuación se aplica el mismo método a cada una de las dos partes en las que queda dividido el array.

Normalmente se toma como pivote el primer elemento de array, y se realizan dos búsquedas: una de izquierda a derecha, buscando un elemento mayor que el pivote, y otra de derecha a izquierda, buscando un elemento menor que el pivote. Cuando se han encontrado los dos, se intercambian, y se sigue realizando la búsqueda hasta que las dos búsquedas se encuentran. Por ejemplo, para dividir el array {21,40,4,9,10,35}, los pasos serían:

{21,40,4,9,10,35} → se toma como pivote el 21. La búsqueda de izquierda a derecha encuentra el valor 40, mayor que pivote, y la búsqueda de derecha a izquierda encuentra el valor 10, menor que el pivote. Se intercambian:

{21,10,4,9,40,35} → Si seguimos la búsqueda, la primera encuentra el valor 40, y la segunda el valor 9, pero ya se han cruzado, así que paramos. Para terminar la división, se coloca el pivote en su lugar (en el número encontrado por la segunda búsqueda, el 9, quedando:

{9,10,4,21,40,35} → Ahora tenemos dividido el array en dos arrays más pequeños: el {9,10,4} y el {40,35}, y se repetiría el mismo proceso.

La implementación es claramente recursiva, y suponiendo el pivote el primer elemento del array, el algoritmo sería:

```
PROGRAMA Quicksort
VARIABLES
      ENTERO tabla[100]
      ENTERO i
INICIO
      PARA i = 0 HASTA 99 HACER
            ESCRIBIR "Dame un valor"
            LEER tabla[i]
      FIN PARA
      ordenar(tabla, 0, 99)
      PARA i = 0 HASTA 99 HACER
            ESCRIBIR tabla[i]
      FIN PARA
FIN
```

```
PROCEDIMIENTO ordenar(ENTERO array[100], desde, hasta)
VARIABLES
      ENTERO pivote
INICIO
      SI desde < hasta ENTONCES
            pivote=colocar(array, desde, hasta);
            ordenar(array, desde, pivote-1);
            ordenar(array, pivote+1, hasta);
     FIN SI
FIN
FUNCION colocar(ENTERO array[100], desde, hasta) DEVOLVER ENTERO
VARIABLES
      ENTERO i, pivote, valor_pivote, temp
INICIO
      pivote = desde
      valor_pivote = array[pivote]
      PARA i = desde + 1 HASTA i <= hasta HACER
            SI array[i] < valor_pivote ENTONCES
                  pivote = pivote + 1
                  temp = array[i]
                  array[i] = array[pivote]
                  array[pivote] = temp
            FIN SI
      FIN PARA
      temp = array[desde]
      array[desde] = array[pivote]
      array[pivote] = temp
      DEVOLVER pivote
FIN
```

Búsqueda de un elemento en una tabla ordenada

Hemos visto en el apartado anterior cómo ordenar una tabla con varios métodos. Ahora retomemos el tema de la búsqueda de un elemento en una tabla ordenada. Se trata de la **Búsqueda Binaria** o **Dicotómica**.

El método siguiente es mucho más rápido que la búsqueda vista en un apartado anterior pero solamente tiene efectividad si la tabla sobre la que se aplica está ordenada.

Se utiliza una variable *encontrado* con la misma función que en el método de la Búsqueda Secuencial.

El método consiste en comparar el elemento buscado con el elemento que ocupa la parte central de la tabla. Si es el buscado, se acaba la búsqueda. Si no es, comprobaremos en qué parte de la tabla puede estar el elemento buscado para desechar la otra parte y así vamos buscando solamente con la mitad de la tabla, olvidándonos de la otra mitad. Y así vamos dividiendo y quedándonos solamente con la parte de las sublistas que nos interesan.

```
      ENTERO tabla[10], elemento_buscado
      ENTERO inferior, superior, central, encontrado, i
      ESCRIBIR "Tabla"
      PARA i = 0 HASTA 9 HACER
            ESCRIBIR "Dame un valor"
            LEER tabla[i]
      FIN PARA
      ESCRIBIR "Dame el valor a buscar:"
      LEER elemento_buscado
      // ORDENAR TABLA
      encontrado = -1
      inferior = 0
      superior = 9-1
      MIENTRAS inferior <= superior HACER
            central = (inferior + superior) / 2
            SI tabla[central] = elemento_buscado ENTONCES
                  encontrado = central
                  inferior = superior + 1
            SINO
                  SI elemento_buscado < tabla[central] ENTONCES
                        superior = central - 1
                  SINO
                        inferior = central + 1
                  FIN SI
            FIN SI
      FIN MIENTRAS
      SI encontrado DISTINTO -1 ENTONCES
            ESCRIBIR "El elemento se encontró en la posición " +
encontrado
      SINO
            ESCRIBIR "El elemento NO se ha encontrado"
      FIN SI
```

Practicar

En los apartados anteriores se han presentado los métodos para buscar un elemento en una tabla y los métodos de ordenación. Todo en **Pseudocódigo**. Para practicar se pide realizar un proyecto en JAVA con los siguientes elementos:

1º La **creación** de una tabla de 1000 elementos de tipo **entero**. Para ello utilizaremos la generación **aleatoria** de números entre **-5000** y **5000**. No importa que se repitan los números.

2º Pediremos también un **número entero** entre -5000 y 5000 y lo buscaremos en la tabla anteriormente creada. Si lo encuentra, debemos indicar simplemente la primera posición donde aparece. Si no aparece, debemos indicarlo de alguna forma también.

3º Ahora hacemos 4 copias de la tabla inicial. Y utilizamos cada copia para aplicarle un método de ordenación: a una copia le aplicamos el método de **Intercambio**, a otra copia el de **Selección**, a otra el de **Burbuja** y por último a la cuarta copia el **Shell**.

4º Por último, volveremos a buscar el elemento introducido en el paso 2, pero ahora aplicando el método de búsqueda binaria. Debe dar el mismo resultado.

5º Para sacar un 10 en la práctica, debemos hacer las siguientes modificaciones en los pasos indicados:

Modificación Paso 2: Calcular el tiempo que tarda en encontrar el número buscado, en caso de que lo encuentre. Si no lo encuentra, también se muestra el tiempo transcurrido en dar el mensaje de "No encontrado".

Modificación Paso 3: Indicar el tiempo que tarda en ordenar la tabla por cada método aplicado. ¿Qué método tarda menos? ¿Y el más lento?

Modificación Paso 4: Anotar el tiempo que tarda en buscar el elemento introducido en el paso 2. ¿Ha tardado alrededor de la mitad que con la tabla desordenada?

08/08/2017