Parte II: Estructuras de Datos y Algoritmos

UNIVERSIDAD DE CANTABRIA

1. Introducción al análisis y diseño de algoritmos.
2. Tipos abstractos de datos.
3. *Métodos de ordenación.*

4

Notas:

UNIVERSIDAD DE CANTABRIA

3. Métodos de ordenación.

- **El modelo de ordenación interna.**
- **Esquemas simples de ordenación.**
- **Ordenación Rápida.**
- **Ordenación por Cajas.**
- **Ordenación por Base.**

3.1. El modelo de ordenación interna.

UNIVERSIDAD DE CANTABRIA

Los procesos de ordenación pueden ser:

- ***internos*: para ordenar datos en la memoria del computador**
- ***externos*: para ordenar datos que residen en memoria secundaria (ficheros)**

Los objetos a ser ordenados son de cualquier tipo, aunque deben tener definida la operación de comparación estándar (<).

- **en Ada se puede redefinir esta operación**
- **en otros lenguajes, o también en Ada, se pueden ofrecer mediante funciones definidas al efecto**

Notas:

UNIVERSIDAD DE CANTABRIA

Los procesos de ordenación de listas de objetos, de acuerdo con una relación de orden lineal, son tan importantes y frecuentes que su análisis es de gran importancia.

Los procesos de ordenación pueden ser de dos tipos:

- **Ordenación interna**. Se realiza en la memoria del computador.
- **Ordenación externa.** Cuando el volumen de información es muy grande y no cabe en la memoria principal del computador, es preciso manejar la información en la memoria secundaria o masiva. En este caso el cuello de botella es el movimiento de datos entre la memoria principal y la secundaria, por lo que es preciso tener en cuenta técnicas especiales.

En este capítulo se analizarán algunos métodos de ordenación interna. Los objetos a ser ordenados pueden ser de cualquier tipo, siempre que exista una relación de orden lineal entre ellos. Supondremos que estos objetos tienen su relación de orden definida por medio de las operaciones de comparación del Ada (<). En caso de que los objetos no tengan esta relación predefinida, se escribirá por parte del usuario. En otros lenguajes, será necesario escribir una función para realizar esta operación de comparación.

El problema de la ordenación es organizar la secuencia de registros r_1, r_2, ..., r_n, obteniéndose la secuencia r_{i1}, r_{i2}, ..., r_{in}, tal que $r_{i1} \leq r_{i2} \leq \ ... \leq \ r_{in}$.

3.2. Esquemas simples de ordenación

UNIVERSIDAD DE CANTABRIA

Supondremos que hay que ordenar N elementos del array A

El método más sencillo para ordenar es el de ascenso de burbuja:

```
for i in 1..N-1 loop
   for j in reverse i+1..N loop
        if A(j) < A(j-1) then
             intercambia(A(j),A(j-1));
        end if;
   end loop;
end loop;
```

La dependencia temporal del ascenso de burbuja es $O(n^2)$

Notas:

UNIVERSIDAD DE CANTABRIA

Quizás el método de ordenación más sencillo es el de **ordenación por ascenso de burbuja**, que se puede imaginar como un array vertical en el que se almacenan los registros a ordenar, y en el que los registros de orden menor son más 'ligeros' que los de orden mayor y, por tanto, ascienden flotando a las posiciones superiores del array.

Para implementar este algoritmo en el computador se realizan múltiples pasadas por el array de abajo a arriba, analizando parejas de elementos contiguos, de modo que si su orden es incorrecto se intercambian. Suponiendo que los elementos del array A son los registros a ordenar (del tipo Elemento), en número N, el algoritmo se muestra en la transparencia anterior, donde el procedimiento intercambia será:

```
procedure intercambia(x,y : in out Elemento) is
        temp : Elemento:=x;
begin
        x:=y;
        y:=temp;
end intercambia;
```

Aunque este método es sencillo, su tiempo de ejecución es de tipo $O(n^2)$. Otros métodos sencillos, como el de inserción o el de selección también presentan tiempos de este orden. Por tanto estos métodos sencillos solamente son válidos cuando el número de elementos a ordenar es bajo.

3.3 Ordenación rápida

UNIVERSIDAD DE CANTABRIA

El algoritmo de ordenación rápida ("*quicksort*") es $O(n \log n)$ en promedio ($O(n^2)$ en el peor caso):

El algoritmo es recursivo, y su pseudocódigo es:

```
procedimiento Quicksort(i,j,A) is
begin
     if A(i)...A(j) posee, al menos, dos valores diferentes then
        v:= pivote de A(i)...A(j);
        -- el pivote de un conjunto de elementos es el mayor
        -- de la primera pareja de elementos distintos
        Particion de A(i..j);
        -- Reordenar el array de forma que para
        -- algún k entre i+1 y j A[i]...A[k-1] sean menores
        -- que v, y A[k]...A[j] sean mayores o iguales que v
        Quicksort(i,k-1);
        Quicksort(k,j);
    end if;
end Quicksort;
```


Notas:

UNIVERSIDAD DE CANTABRIA

Cuando es preciso ordenar listas de gran número de objetos es preciso recurrir a algoritmos más rápidos, aunque también más complejos. Generalmente estos algoritmos presentan tiempos de ejecución de tipo O(n·log n). El algoritmo "Quicksort" presenta tiempos de ejecución promedios de tipo O(n·log n), pero en el peor caso pueden llegar a ser $O(n^2)$.

Este algoritmo permite ordenar un array A(1) ... A(n) tomando un valor del array, denominado **pivote**, y organizando todos los elementos del array de forma que para un k dado todos los elementos A(1) ... A(k-1) sean menores que el pivote y A(k) ... A(n) sean mayores o iguales.

Posteriormente se aplica el mismo procedimiento de forma recursiva a estos dos conjuntos de elementos sucesivamente hasta que todo el array aparezca ordenado.

Puede observarse que este procedimiento será tanto más eficiente cuanto más se acerque el elemento pivote al valor central de los elementos del array, para el que k=n/2.

Este algoritmo de ordenación rápida se puede expresar en pseudocódigo en la forma indicada en la transparencia anterior. Su implementación se muestra en las transparencias siguientes.

Implementación de "Quicksort"

UNIVERSIDAD DE CANTABRIA

```
subtype Indice is Integer range 0..Max_Elem;
type Tipo_Elementos is array (Indice range 1..Max_Elem)
     of Elemento;

procedure Ordenacion_Rapida (A : in out Tipo_Elementos;
                             N : in Indice) is
    function Encuentra_Pivote
         (i,j : in Indice; A : in Tipo_elementos) return Indice is
        --   Devuelve 0 si todos los elementos son iguales.
    begin
        for k in i+1..j loop
            if A(k)>A(i) then
                return(k);
            elsif A(k)<A(i) then
                return(i);
            end if;
        end loop;
        return(0); --no hay elementos diferentes
    end Encuentra_Pivote;
```


Notas:

UNIVERSIDAD DE CANTABRIA

```
  procedure Particion (i,j     : in Indice;
                       Pivote : in Elemento;
                        A       : in out Tipo_Elementos;
                        k       : out Indice)
  is
        l : Indice:=i;
        r : Indice:=j;
  begin
        loop
              Intercambia(A(l),A(r));
              while A(l) < Pivote loop
                  l:=l+1;
              end loop;
              while A(r) >= Pivote loop
                  r:=r-1;
              end loop;
              exit when l>r;
        end loop;
        k:=l;
  end Particion;
```

Implementación de "Quicksort" (cont.)

UNIVERSIDAD DE CANTABRIA

```
  procedure Quicksort(i,j : in Indice;
                      A   : in out Tipo_elementos) is
        Pos_Pivote,k : Indice;
  begin
        Pos_Pivote:=Encuentra_Pivote(i,j,A);
        if Pos_Pivote /= 0 then
              Particion(i,j,A(Pos_Pivote),A,k);
              Quicksort(i,k-1,A);
              Quicksort(k,j,A);
        end if;
   end Quicksort;

begin  -- Ordenacion_Rapida
    Quicksort(1,N,A);
end Ordenacion_Rapida;
```


Notas:

UNIVERSIDAD DE CANTABRIA

Los tiempos de ejecución de este algoritmo son de O(n·log n) en el caso medio y de $O(n^2)$ en el peor caso. Efectivamente, los tiempos de ejecución de peor caso para **encuentra_pivote** y **partición** son de tipo O(j-i+1). En el peor caso el número de niveles de recursión es n (caso en el que el pivote divide la lista en dos grupos uno de ellos de longitud 1) y, por tanto, el tiempo de peor caso es $O(n^2)$.

En cambio en el caso medio, el número de niveles de recursión es proporcional a log n, por lo que el tiempo de ejecución medio es O(n·log n).

Existen otros algoritmos que presentan tiempos de ejecución medios y de peor caso del orden de O(n·log n). Sin embargo el tiempo de ejecución medio del algoritmo 'quicksort' es inferior, por un factor constante, al de estos métodos.

Existen técnicas para mejorar el tiempo de ejecución de este algoritmo, basadas en realizar una mejor elección del pivote. Por ejemplo, se pueden tomar tres elementos del array al azar y tomar como pivote el de valor intermedio.

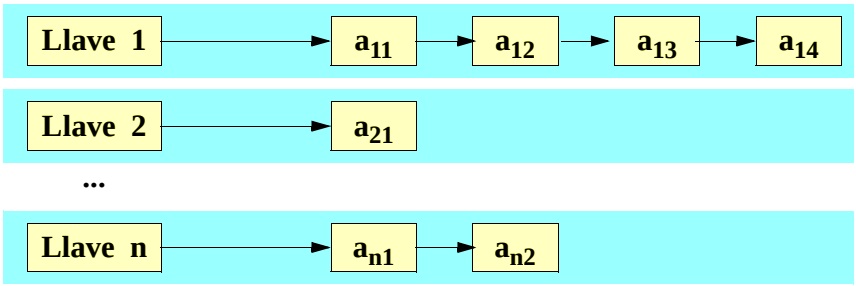
3.4. Ordenación por cajas

UNIVERSIDAD DE CANTABRIA

La ordenación por cajas o "*bin sorting*" sirve para ordenar conjuntos de elementos en base a una función llave discreta:

- **llave(elemento) = valor discreto**
- **los elementos se ordenan según su llave**

Para cada posible valor de la llave se crea una "*caja*" donde depositar elementos:



Notas:

UNIVERSIDAD DE CANTABRIA

Para algunos tipos particulares de datos a ordenar es posible utilizar algoritmos de ordenación con tiempos de ejecución de tipo O(n). Tal es el caso del método de ordenación por 'cajas', que se aplica a elementos para los que existe definida una función llave que sirve para ordenar los elementos, y cuyos valores sólo pueden presentar un conjunto discreto de valores, como por ejemplo números enteros comprendidos en un rango limitado.

Si suponemos que el tipo_llave = 1..n, y queremos ordenar un conjunto de n registros cuyos valores llave no están repetidos, puede utilizarse un array B(1..n) de 'cajas', con el siguiente algoritmo:

```
for i in 1..n loop
    B(Llave(A(i))):=A(i);
end loop;
```

Obviamente el tiempo de ejecución de este algoritmo es O(n). Sin embargo, si los elementos llave pueden tener valores repetidos es preciso poder almacenar varios registros en la misma caja. Para ello, el array B puede tener elementos que sean de tipo lista encadenada, como en la figura anterior.

En este caso el procedimiento de ordenación por cajas será el que se muestra en la próxima transparencia.

Implementación de "Binsort"

UNIVERSIDAD DE CANTABRIA

```
procedure Binsort (N : in Indice; A : in out Tipo_Elementos) is
     B   : array (Tipo_Llave) of Listas.Lista;
     Pos : Listas.Posicion;
     I   : Indice range A'Range:=A'First;
begin
     for V in Tipo_Llave loop
        Listas.Haz_Nula(B(V));
     end loop;
     for j in 1..N loop
        Inserta_Al_Principio(A(j),B(Llave(A(j))));
     end loop;
     for V in Tipo_Llave loop
        Pos := Primera_Pos(B(V));
        while Pos/=Posicion_Nula loop
           A(I):=Elemento_De(Pos,B(V));
           Pos:=Siguiente(Pos,B(V)); I:=Indice'Succ(I);
        end loop;
     end loop;
end Binsort;
```


Notas:

UNIVERSIDAD DE CANTABRIA

Donde Inserta_Al_Principio es una de las operaciones básicas de inserción de un elemento en una lista. Puede observarse que primero se insertan todos los elementos en las cajas apropiadas, y luego se recorren las cajas para extraer los elementos y volver a guardarlos en el array A.

Este algoritmo presenta tiempos de ejecución de tipo O(m+n), donde n es el número de elementos a ordenar y m el número de posibles valores del campo llave. Si $m \leq n$ entonces el tiempo es O(n). Sin embargo si, por ejemplo, $m=n^2$, entonces el tiempo será $O(n^2)$.

Una desventaja importante de este método es la necesidad de crear el array B de listas, que obviamente consume una cantidad importante de memoria.

3.5. Ordenación por base

UNIVERSIDAD DE CANTABRIA

La ordenación por Base siempre presenta tiempos O(n)

Es aplicable a elementos para los que el orden se define mediante una función llave que da valores discretos.

Sirve aunque el número de valores llave >> N

El proceso se puede asimilar a la ordenación de números (o llaves) de k cifras expresadas en base N (siendo k fijo):

- **primero se ordenan los números por su cifra más significativa**
- **luego por la siguiente cifra, etc.**

El proceso es O(k·n)=O(n)

Notas:

UNIVERSIDAD DE CANTABRIA

Para aquellos casos en los que m >> n, se puede aplicar una generalización del método de ordenación por cajas, denominada ordenación por base, que presenta siempre tiempos de ejecución de tipo O(n).

Supongamos que deseamos ordenar m enteros en base n comprendidos en el intervalo 0..n^2-1. Utilizaremos n cajas de listas, una para cada entero entre 0..n-1, y realizaremos el proceso de ordenación en dos fases:

- En la primera fase se inserta cada elemento i en la caja i MOD n, de modo que la inserción se realice siempre al final de la lista de cada caja.
- En la segunda fase se recorren los elementos de cada caja por orden, y se almacena cada elemento i en un nuevo conjunto de cajas en la posición i/n redondeado por abajo, que es el mayor entero igual o menor que i/n. De nuevo se insertan los elementos al final de cada lista.

Cuando este proceso, que es de tipo O(n) ha finalizado, la lista se halla ordenada. Este proceso se puede imaginar como un proceso de ordenación de números de dos cifras expresados en base n. En la primera fase se ordenan los números por su cifra menos significativa, mientras que en una segunda fase se hace por su cifra más significativa.

Este proceso se puede generalizar a números de k cifras en base n, para k fijo, siendo los tiempos de ejecución de tipo O(k·n) = O(n).

Ordenación por base (cont.)

UNIVERSIDAD DE CANTABRIA

La ordenación por base se puede también aplicar a datos compuestos por varios números en bases diferentes,

Por ejemplo se puede aplicar a fechas del tipo siguiente:

```
type Elemento is record
      Dia   : 1..31;
      Mes   : (Enero,....,Diciembre);
      Anyo  : (1900..2000);
end record;
```

En este caso primero se ordena por año, luego por mes, y luego por día.

Ejemplo de ordenación por base (1/6)

UNIVERSIDAD DE CANTABRIA

```
with Var_Strings;
use Var_Strings;

package Elementos is

   subtype Tipo_Anyo is Integer range 1900..2000;
   subtype Tipo_Dia is Integer range 1..31;

   type Tipo_Mes is (Enero, Febrero, Marzo, Abril, Mayo,
      Junio, Julio, Agosto, Septiembre, Octubre,
      Noviembre, Diciembre);

   type Fecha is record
      Dia   : Tipo_Dia;
      Mes   : Tipo_Mes;
      Anyo  : Tipo_Anyo;
   end record;

   procedure Presenta_Fecha (La_Fecha : in Fecha);
```

Ejemplo de ordenación por base (2/6)

UNIVERSIDAD DE CANTABRIA

```
   type Persona is record
      Nombre            : Var_String;
      Fecha_Nacimiento  : Fecha;
      Direccion         : Var_String;
      Telefono          : Var_String;
   end record;

   type Ref_Persona is access Persona;

   subtype elemento is Ref_Persona;

   procedure Presenta_Persona (La_Persona : in Persona);

   function Fecha_De (La_Persona : in Persona) return Fecha;


end Elementos;
```


Ejemplo de ordenación por base (3/6)

UNIVERSIDAD DE CANTABRIA

```
with Elementos;
use Elementos;
package Grupos_Personas is

   Max_Personas : constant Integer :=10_000;

   type Personas is array (Integer range 1..Max_Personas)
           of Ref_Persona;

   type Grupo_Personas is record
      Las_Personas : Personas;
      Num_Personas : Integer range 0..Max_Personas;
   end record;

   procedure Ordena_Por_Fecha (
         El_Grupo       : in Grupo_Personas;
         Grupo_Ordenado : out Grupo_Personas);

end Grupos_Personas;
```

Ejemplo de ordenación por base (4/6)

UNIVERSIDAD DE CANTABRIA

```
with Colas;
package body Grupos_Personas is

   procedure Ordena_Por_Fecha (
               El_Grupo       : in Grupo_Personas;
               Grupo_Ordenado : out Grupo_Personas) is

      Caja_Anyo  : array(Tipo_Anyo) of Colas.Cola;
      Caja_Mes   : array(Tipo_Mes)  of Colas.Cola;
      Caja_Dia   : array(Tipo_Dia)  of Colas.Cola;
      La_Persona : Ref_Persona;

   begin

      --ordena por dia
      for I in 1..El_Grupo.Num_Personas loop
         Colas.Inserta(El_Grupo.Las_Personas(I),
           Caja_Dia(Fecha_De(El_Grupo.Las_Personas(I).all).Dia));
      end loop;
```


Ejemplo de ordenación por base (5/6)

UNIVERSIDAD DE CANTABRIA

```
      --ordena por mes
      for Dia in Tipo_Dia loop
        while not Colas.Esta_Vacia(Caja_Dia(Dia)) loop
            Colas.Extrae(La_Persona,Caja_Dia(Dia));
            Colas.Inserta(La_Persona,Caja_Mes(
                          Fecha_De(La_Persona.all).Mes));
         end loop;
      end loop;

      --ordena por anyo
      for Mes in Tipo_Mes loop
        while not Colas.Esta_Vacia(Caja_Mes(Mes)) loop
            Colas.Extrae(La_Persona,Caja_Mes(Mes));
            Colas.Inserta(La_Persona,Caja_Anyo(
                          Fecha_De(La_Persona.all).Anyo));
         end loop;
      end loop;
```

Ejemplo de ordenación por base (6/6)

UNIVERSIDAD DE CANTABRIA

```
      -- recoge resultados
      Grupo_Ordenado.Num_Personas:=0;
      for Anyo in Tipo_Anyo loop
        while not Colas.Esta_Vacia(Caja_Anyo(Anyo)) loop
            Grupo_Ordenado.Num_Personas:=
                  Grupo_Ordenado.Num_Personas+1;
            Colas.Extrae(La_Persona,Caja_Anyo(Anyo));
            Grupo_Ordenado.Las_Personas(
                  Grupo_Ordenado.Num_Personas):=La_Persona;
        end loop;
      end loop;
   end Ordena_Por_Fecha;

end Grupos_Personas;
```


UNIVERSIDAD DE CANTABRIA