2 - Conceptos fundamentales de ficheros. Organizaciones y acceso. Organizaciones secuenciales.

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

2.2 - Organización de información y métodos de acceso. Organización secuencial.

2.3 - Operaciones de mantenimiento de ficheros y su implicación en la organización. Ordenación de ficheros secuenciales.

2.4 - Otras organizaciones secuenciales: Organización secuencial encadenada. Desarrollo de una implementación sencilla.

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

Punto de vista del usuario:

Un fichero es una "colección de datos (información) interrelacionados" almacenados, normalmente, sobre un soporte de almacenamiento masivo y permanente de información. Además:

- Se dispone de algún mecanismo de identificación (nombre)
- Se pueden realizar operaciones de mantenimiento y recuperación de información

Punto de vista representación física:

colección de "bloques" sobre un dispositivo físico concreto

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

Punto de vista del programador (además) :

Diferentes organizaciones de la información

- A nivel de los datos (registros)
- A nivel de organización (relaciones) entre los datos

Solución: **establecer niveles de abstracción** ⇒ ***Sistema Gestor de Ficheros*** ligado al ***Sistema Operativo***

- visión del sistema como una colección de **ficheros físicos**
- conjunto de operadores básico (a nivel de registro)

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

idea fundamental: **establecer independencia entre niveles**

mayor nivel de abstracción ⇒ **nuevo *Sistema Gestor de Ficheros*** ligado al ***lenguaje/aplicación***

- visión del sistema como una colección de **ficheros lógicos**
- conjunto de operadores ligados a la organización elegida
- diferentes organizaciones

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

Problemas ligados al diseño del Sistema Gestor de Ficheros :

P1) Los soportes físicos a utilizar pueden ser de distinto tipo y distintas características (**independencia del dispositivo**)

P2) El **tiempo de acceso** es, normalmente, excesivamente grande, debiéndose minimizar su efecto

P3) Tiene que servir para soportar un **número de ficheros muy diverso** (posiblemente grande), y **de tamaños muy diferentes**

P4) **Gestión eficiente** del espacio de almacenamiento

P5) Tiene que garantizarse a un nivel razonable la **fiabilidad** y **seguridad** de la información (frente a errores, fallos del sistema, y accesos)

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

P1) Los soportes físicos a utilizar pueden ser de distinto tipo y distintas características (**independencia del dispositivo**)

S1) Estructuración de la gestión en una jerarquía con niveles independientes

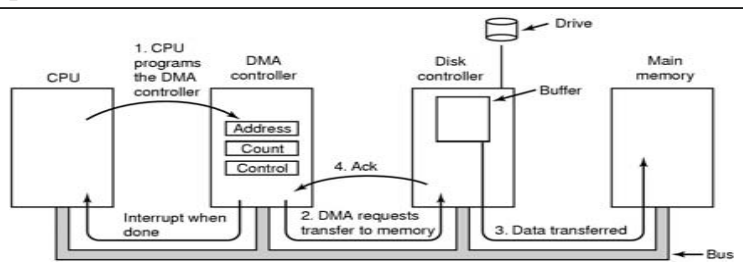
Los operadores de cada nivel se implementan a partir de los definidos en el nivel inferior:

* software a nivel de usuario: ***Gestor de ficheros***
* software a nivel de dispositivos: ***Gestor de periféricos***
* controladores de dispositivo

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

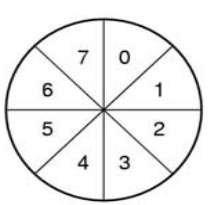
P2) El **tiempo de acceso** es, normalmente, excesivamente grande, debiéndose minimizar su efecto

S2)
- tratamiento de **bloques** de información en vez de datos simples, y zonas de memoria tampón auxiliares (**buffers**)
- técnicas especiales de comunicación entre la memoria del computador y los periféricos (DMA, canal, etc.)



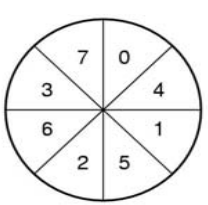
2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

S2)
- técnicas de **optimización del acceso físico** a los bloques (sectores):
  * *intercalado de sectores*



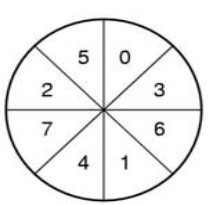
(a)

Sin intercalado
(factor intercalado = 1)



(b)

Intercalado sencillo
(factor intercalado = 2)

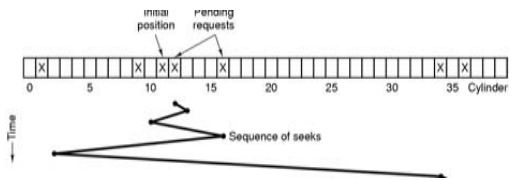


(c)

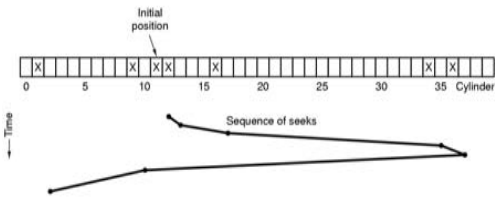
Intercalado doble
(factor intercalado = 3)

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

S2) • * *planificación de los movimientos* de las cabezas
=> gestión de cola de peticiones
- localización más próxima
- algoritmo del elevador
- FCFS (First Come First Served) • • •



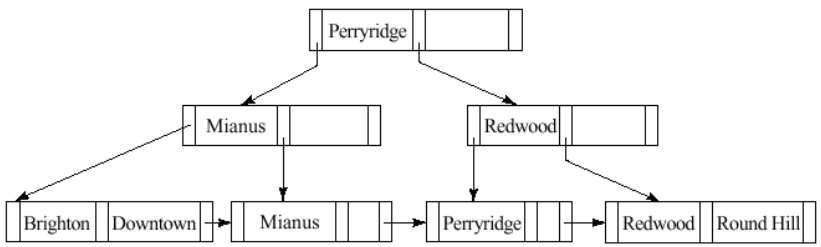
localización más próxima



algoritmo del elevador

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

S2) • técnicas de estructuración de la información
(implementaciones basadas en listas, pilas, árboles, etc.)



2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

P3) Tiene que servir para soportar un *número de ficheros muy diverso* (posiblemente grande), y *de tamaños muy diferentes*

S3) • Organización de los ficheros en base a **directorios** a varios niveles. Además se suelen establecer **particiones** de los dispositivos
• En el directorio se almacena información (o referencia) acerca de:
* la organización del fichero
* la ubicación física de los bloques
* otras propiedades ligadas al acceso, protección, tiempo, etc.

- Facilita la organización de la información al usuario y al sistema
- incrementa la eficiencia en el acceso

☛ Jerarquía de directorios → un directorio es un fichero más

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

P4) **Gestión eficiente** del espacio de almacenamiento

a nivel de bloques:

S4) • la elección del tamaño de bloque más adecuado (factor de bloque) para optimizar el compromiso de aprovechamiento *tiempo/memoria*. Debe servir para diferentes aplicaciones (factor común y posterior elección de múltiplos)
• Organización de los *bloques libres y usados* (información/defectuosos)

- Deben existir bloques de disco "especiales" *(posic. fijas ?)*
- * lista descriptora de los nºs de bloque libres.
- * mapa de bits.

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

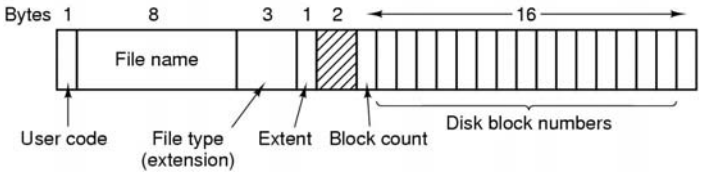
a nivel de almacenamiento del fichero:

S4)
- Contiguo ⇒ problemas gestión y baja eficiencia; fragment. externa
- Lista enlazada de bloques (o de secuencias de datos)
  - * Pb. de tamaño no pot. de 2 => fragment. interna
  - * Pb. de acceso directo complejo
- **FAT**
  - * tabla de ubicación (mapa o referencia) de bloques
  - * una entrada por bloque
    - → Problema de tamaño
- Estructuras más complejas: **nodo i** (UNIX)

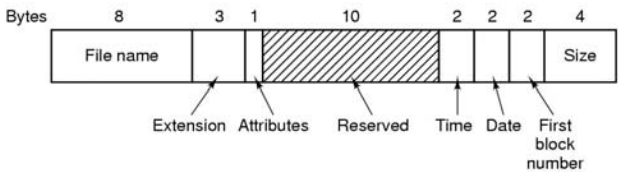
⇒ ***Compromiso simplicidad / potencia***

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

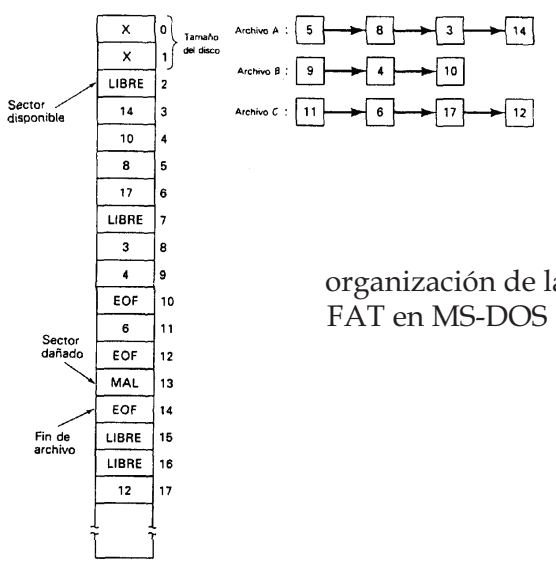
a) formato de directorio de CP/M



b) entrada de directorio de MS-DOS



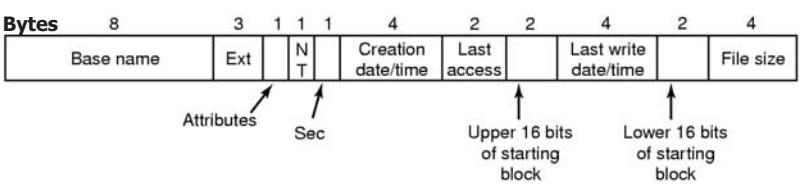
2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.



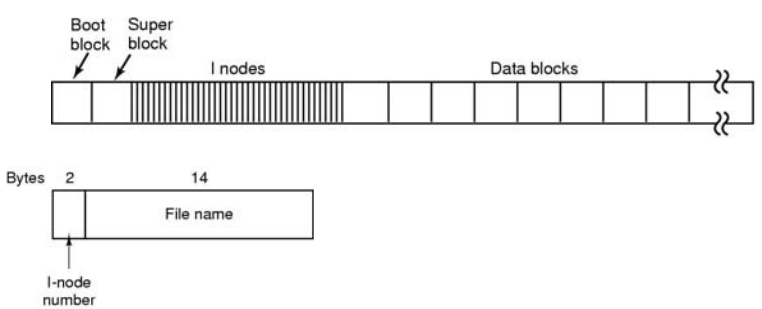
organización de la FAT en MS-DOS

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

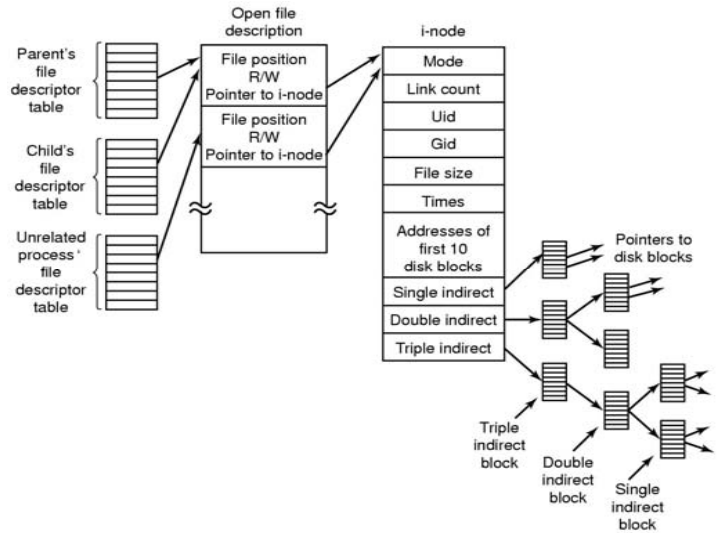
c) entrada de directorio en Windows 98



d) organización en UNIX



2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.



2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

Para mejorar el rendimiento:

S4)
- Utilización de buffer de bloques (páginas)
  - * Suficientemente grande (anticipación y reducc. accesos)
  - * Política de reemplazo adecuada (FIFO, LRU, etc.)
  - * Política de salvaguarda (hay bloques esenciales)
- Estrategias heurísticas para ubicar la información
  - * bloques lo más próximos (mismo cilindro) → simple con mapa bits
  - * utilizar pistas centrales para bloques con mayor acceso (nodos i)
  - * pistas libres reservadas en el mismo cilindro para extensiones, bloques malos, etc.

2.1 - Concepto de fichero. El Sistema Gestor de Ficheros.

P5) Tiene que garantizarse a un nivel razonable la **fiabilidad** y **seguridad** de la información (frente a errores, fallos del sistema, y accesos)

S5)
- la adición de <u>redundancias</u> en la información:
  - * códigos detectores/correctores (CRC, etc.) en los bloques de control (a nivel de dispositivo)
  - * en las estructuras de datos, …
    - ✓ FAT duplicada
    - ✓ información replicada en bloques descriptores (cabecera) del fichero
    - ✓ • • •
- técnicas de control de acceso, criptografía, etc.

2.1 - El Sistema Gestor de Ficheros: organización

Gestor del Periférico:

- dispositivo almacenamiento como "vector" de "**páginas**" (bloques)
  - ➪ tamaño fijo (1024, 2048, 4096, etc.)
  - ➪ identificador (número) único
- organizadas como colección de **conjuntos de páginas** disjuntos
  - ➪ identificador único
  - ➪ como mínimo conjunto páginas libres y/o usables
- operadores (básicos)
  - • **lectura** *de la página* p {*o página* p *del conjunto* c}
  - • **escritura** *de la página* p {*o página* p *del conjunto* c} ⇒ modificación
  - • **adición** *de la página* p *al conjunto* c
  - • **eliminación** *de la página* p *del conjunto* c

  → basados en operaciones del gestor del dispositivo (sectores, etc.)

2.1 - El Sistema Gestor de Ficheros: organización

Gestor de Ficheros:

- dispositivo almacenamiento como **colección de ficheros** (conj. registros)
  - * *nombre* (explícito o implícito) único (identif. interna)
  - * organizados como un *árbol*
- ficheros organizados como lista de secuencias de páginas (extensiones) + descriptor (bloque) de información
- operadores (básicos)
  - **creación** *del fichero* f *{+ especificación propiedades}*
  - **eliminación** *del fichero* f
  - **lectura** *de un dato (registro)* r *del fichero* f
  - **escritura** *de un dato (registro)* r *en fichero* f
  - **adición** *de un dato (registro)* r *al fichero* f
  - **eliminación** *de un dato (registro)* r *del fichero* f

  {+ especificaciones adicionales }

  → basados en operaciones del gestor del periférico

2.1 - El Sistema Gestor de Ficheros: organización

Gestor de Ficheros:

- ❑ cada fichero tiene asociada una E.D. interna (memoria) para su gestión
  - * buffer de páginas
  - * datos auxiliares (punteros, contadores, etc.)

  } ⇒ *"grande"*
- ❑ operadores (básicos)
  - **"abrir"** *el fichero* f *{+ especificación propiedades}*
  - **"cerrar"** *el fichero* f
- ❑ normalmente ligado al ***SISTEMA OPERATIVO*** del computador, con un ***conjunto operadores muy básico***
  - • ∄ estr. información *(no están "tipados")*
  - • sólo hay acceso directo (a un byte)

  ⇒ necesidad de *implementar gestores específicos*, adaptados al Pb.

2.1 - El Sistema Gestor de Ficheros: organización

Ejemplo de gestor de ficheros (C):

```
FILE *fp;   { declaración }
FILE *fopen(char *nombre, char modo)
int fclose(FILE *fp)

int getc(FILE *fp)
int putc(int c, FILE *fp)
```

"r" → *lectura*
"w" → *escritura*
"a" → *añadir*
+ b → *binario*

```
size_t fread (void *ptrB,       size_t sizObj, size_t numObj,
              FILE *stream)

size_t fwrite(const void *ptrB, size_t sizObj, size_t numObj,
              FILE *stream)

int fseek(FILE *stream, long offset, int origen)
```


2.1 - El Sistema Gestor de Ficheros: organización

E/S de bajo nivel en UNIX (C):

```
int  fd;  { "descriptor" de fichero ≡ referencia interna}
int read (int  fd, char *buf, int n)
int write(int  fd, char *buf, int n)

int open (char *nombre, int flags, int perms)
int creat(char *nombre, int perms)
```

devuelve el nº bytes leídos(-1 = error)

devuelve el nº bytes escritos

<fcntl.h>
<sys/file.h>

```
int lseek(int  fd, long offset, int origen)
```

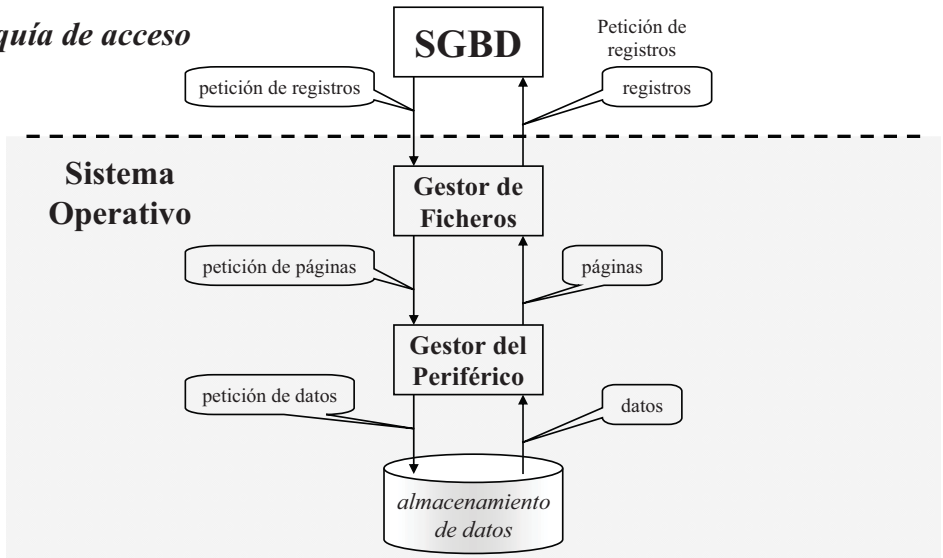
```
typedef struct _iobuf{
   int  cnt;     /* caracteres que quedan */
   char *ptr;    /* posición del siguiente carácter */
   char *base;   /* localización del buffer */
   int  flag;    /* modo de acceso al fichero */
   int  fd;      /* descriptor de fichero */
}FILE;
```

<stdio.h>

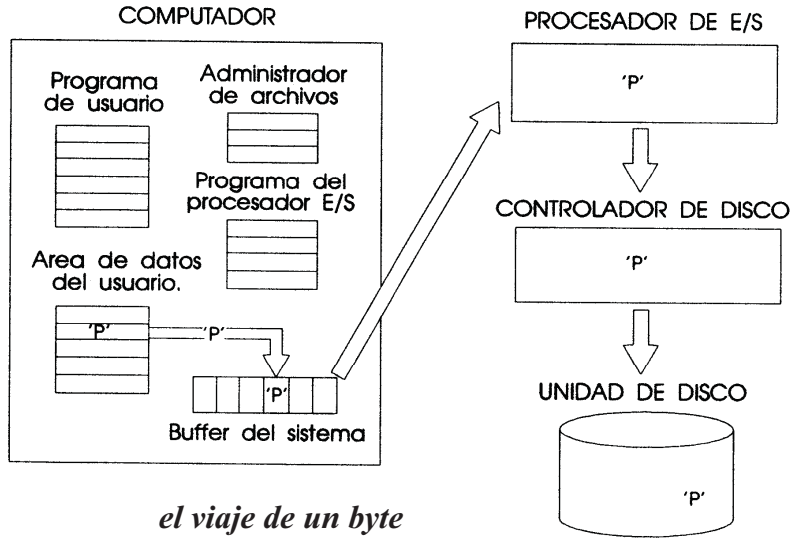
Kernighan & Ritchie

2.1 - El Sistema Gestor de Ficheros: organización

Jerarquía de acceso



2.1 - El Sistema Gestor de Ficheros: organización



el viaje de un byte

2.1 - El Sistema Gestor de Ficheros: organización

1	El programa pide al sistema operativo escribir el contenido de la variable `c` en la siguiente posición disponible en el fichero TEXTO.
2	El sistema operativo transfiere el trabajo al administrador de ficheros.
3	El administrador de ficheros busca a TEXTO en una tabla que contiene información acerca del fichero, para determinar si el fichero está abierto y disponible para usarse, qué tipos de acceso están permitidos, si los hay, y a qué nombre físico corresponde el nombre lógico TEXTO.
4	El administrador de ficheros busca en una tabla de asignación de ficheros la localidad física del sector que contendrá el byte.
5	El administrador de ficheros se asegura de que el último sector del fichero se ha almacenado en un buffer de E/S que el sistema mantiene en memoria RAM, y despues deposita la 'P' en la posición apropiada dentro del buffer.
6	El administrador de ficheros da instrucciones al procesador de E/S acerca de dónde está almacenado el byte en memoria RAM y en qué parte del disco debe quedar.
7	El procesador de E/S encuentra un momento en el cual la unidad de disco esté disponible para recibir los datos y los coloca en el formato adecuado para el disco. También puede almacenar temporalmente los datos para enviarlos en partidas del tamaño apropiado para el disco.
8	El procesador de E/S envía los datos al controlador del disco.
9	El controlador indica a la unidad de disco que mueva la cabeza de lectura y escritura a la pista adecuada, espera a que el sector deseado esté debajo de la cabeza de lectura y escritura, y entonces envía el byte a la unidad para depositarlo, bit por bit, en la superficie del disco.

lógico

capas de procedimientos que intervienen

físico

2.2 - Organización de información y métodos de acceso. Organización secuencial.

Fichero colección (secuencia ?) de *registros* { * longitud fija; * longitud variable

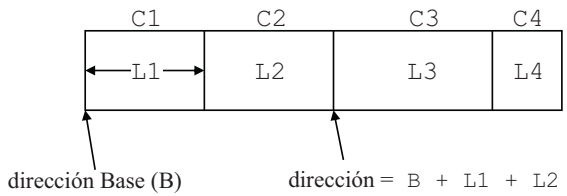
- campos de *longitud variable* (p.e. Cadena)
- campos con *valores múltiples* (repetitivos, p.e. Teléfono)
- campos *opcionales* (p.e. datos del cónyuge)
- compuesto de *registros de tipo diferente* (e.j. lista datos relacionados)

→ *gestión más compleja*

Representación ⇒ *varias soluciones*

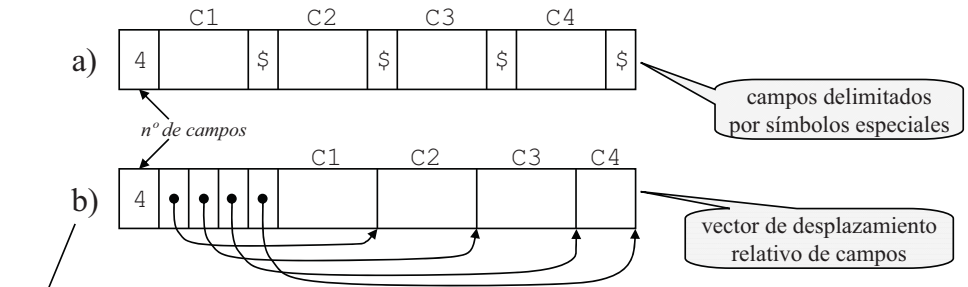
- *separadores* de campos y *finalizador* de registro
- *< id. Campo><valor>* (p.e. pocos campos opcionales de muchos posibles)
- *longitud secuencia + secuencia de bytes*

Formato de registro: longitud fija



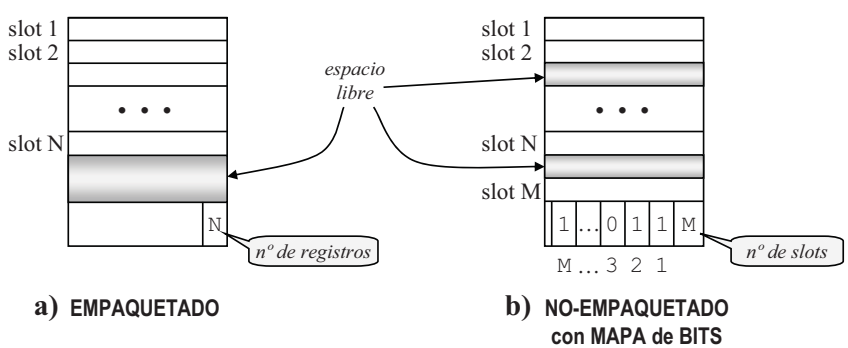
- La información sobre el tipo de los campos es la misma para todos los registros del fichero, y se almacena en el *catálogo del sistema*
- Para buscar el campo *i-ésimo* hay leer (cargar) el registro entero

Formato de registro: longitud variable



☛ *permite **acceso directo** al campo i-ésimo, representación eficiente de los valores nulos, mínimo coste del "directorio"*

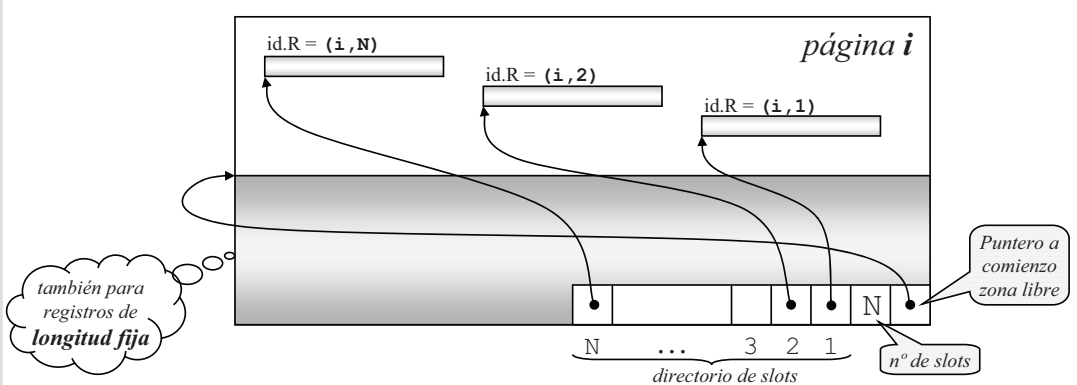
Formato de página: registros de longitud fija



☛ *id. registro* = **<id. página, n° slot>**

☛ *mover los registros para gestionar el espacio libre puede no interesar (cambia el id. Registro)*

Formato de página: registros longitud variable



☛ *se pueden mover los registros en la página sin cambiar su* **id.R** ⇒ ***interesante***

2.2 - Organización de información y métodos de acceso.

Tipos registros

- *físicos* → relacionados con los bloques dispositivo
- *lógicos* → relacionados con la estructura de la información

- *unblocked* ("sueltos") → un registro por bloque
- *blocked* ("en bloque") → varios registros por bloque

- *spanned* ("solapados") → entre dos (o más) bloques
- *unspanned* ("no solapados") → en un sólo bloque

gestión más compleja

☛ *La estructuración y organización de los registros tiene una gran incidencia en las prestaciones del sistema*

2.2 - Organización de información y métodos de acceso.

siguiente paso:

- establecer la ubicación física de los registros, y
- el mecanismo de acceso

- *Organización* del fichero
- *modo de acceso*

a diferentes niveles de abstracción

puede estar ligado a la información del registro (clave)

fuertemente ligado a los requerimientos del problema

Información (propia o añadida) que identifica un registro

2.2 - Organización de información y métodos de acceso.

organización de un fichero

Combinación de estructuras conceptuales y físicas empleadas para distinguir un registro de otro.

→ hace referencia a la organización de los datos del fichero en registros, bloques y estructuras de acceso (relaciones entre registros)

modo de acceso un fichero

especifica la manera en que se puede acceder a un registro

no son conceptos independientes: **Organización** ⇄ **Modo acceso**

2.2 - Organización de información y métodos de acceso.

como tipo (abstracto) de datos

conjunto de operadores y semántica asociada + *representación*

→ modo acceso

→ organización

notas de implementación:

- *la información de la organización en bloque descriptor (cabecera)*
- *los operadores pueden devolver el registro o su ubicación (locate mode)*

2.2 - Organización de información y métodos de acceso.

Operaciones básicas sobre un fichero

- *Creación*
- *Actualización*
  - *Inserción*
  - *Modificación*
  - *Eliminación*
- *Recuperación*
- *Mantenimiento*
  - *Estructuración*
  - *Reorganización*

2.2 - Organización de información y métodos de acceso.

modos de acceso a fichero

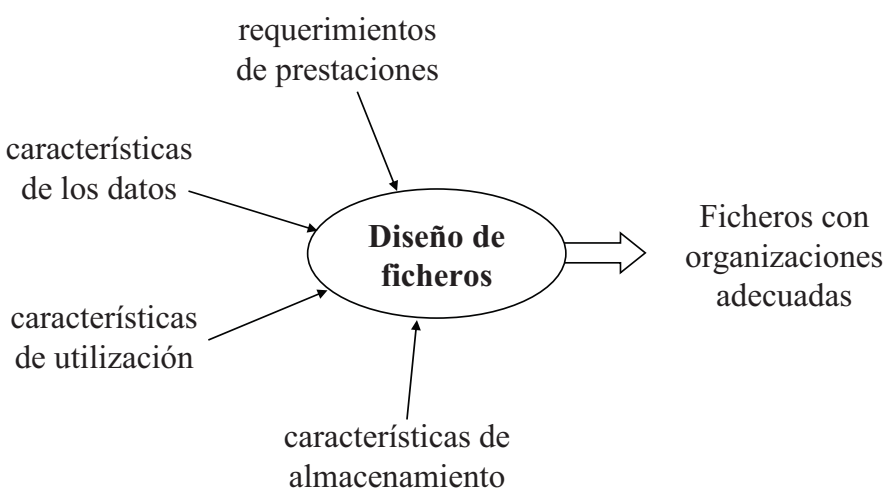
"ruta" de acceso
- ***Secuencial***
- ***directo***

"flujo" de información
- *sólo* ***lectura***
- *sólo* ***escritura***
- ***lectura/ escritura***

organizaciones de ficheros (básicas)

- ***secuencial***
- ***relativa*** → ***directa***, ***indexada***, ***dispersa***
- ***secuencial indexada***
- ***multiclave*** → ***ficheros invertidos***, ***multilista***

2.2 - Organización de información y métodos de acceso.



2.2 - Organización de información y métodos de acceso.

para cuantificar el impacto de las operaciones, se definen algunos parámetros:

- *tasa (ratio) de actividad (*hit rate*)* $\dfrac{\texttt{n°reg. accedidos}}{\texttt{n° reg. fichero}}$
- *volatilidad (*volatility*)* $\dfrac{\texttt{n°inserciones + n° eliminaciones}}{\texttt{n° reg. fichero}}$
- *tasa de crecimiento (*grow rate*)*
  - *real*
  - *aparente*

  ≠ si marcas de borrado
- *fan in/out ratio* $\dfrac{\texttt{n° accesos}}{\texttt{n° reg. accedidos}}$

. . .

2.2 - Organización secuencial: concepto

- ligada al concepto de secuencia
- la ubicación de un dato está determinada por la del anterior
  - *implícitamente (contigüidad)*
  - *explícitamente ("puntero")*
- acceso a un único dato

especificación básica ⇒ lectura/escritura incompatibles

tipos
- ficheros "serie" (heap)
- ficheros ordenados

Cambios en la semántica y la representación ⇒ Otras organizaciones secuenciales

2.2 - Organización secuencial: operadores básicos

operadores básicos:

- *IniciarLectura (f)*
- *LeerDato (f, d)*
- *FinFichero (f)*
- *SiguienteDato (f)*
- *Avanzar (f)*
- *IniciarEscritura (f)*
- *Escribir (f, d)*
- *Asociar (f, nombre)*
- *Disociar (f)*

representación: "**contigua**" ⇒
- implementación *simple* y *eficiente*
- *no adecuada a nivel físico*

☛ problema de implementación de operaciones de mantenimiento

2.3 Operaciones de mantenimiento de ficheros secuenciales y su implicación en la organización.

Se pueden definir algunos operadores adicionales *(razonablemente genéricos)* :

- operadores búsqueda *(representan acceso directo):*

definir la semántica como ejercicio

- *SigDatoValor (**ref** f: tpFSecDato;*
  *__función__ CondValor (d: tpDato) **devuelve** booleano;*
  __ref__ dato: tpDato;
  __ref__ encontrado: boolean)
- *SigDatoPos (**ref** f: tpFSecDato;*
  __valor__ pos: tpPosicion;
  __ref__ dato: tpDato;
  __ref__ encontrado: boolean)

coste lineal → mejora : **fichero ordenado**

2.3 - Operaciones de mantenimiento de ficheros y su implicación en la organización.

- operadores inserción*:*
  - *InsertarDatoValor (**ref** f: tpFSecDato;*
    *__función__ enOrden (d1, d2: tpDato) **devuelve** booleano;*
    __valor__ dato: tpDato)
  - *InsertarPos (**ref** f: tpFSecDato;*
    __ref__ pos: tpPosicion; {devuelve la posición real}
    __valor__ dato: tpDato)
- operadores eliminación y modificacion similares

ejercicio: defina la semántica

☛ coste (lineal) enorme, pues **hay que reescribir el fichero**

→ mejora algo con operadores de renombrado y eliminación de fichero

2.3 - Operaciones de mantenimiento de ficheros y su implicación en la organización.

mejor solución: *retrasar lo más posible la actualización en el fichero físico*.

⇩

mantener una lista de datos a actualizar { • *en memoria* • *en otro fichero* (más pequeño)

➪ *Fichero Maestro*
\+
➪ *Fichero de actualizaciones* → fichero de transacciones ordenado

≡ compromiso razonable de coste para tasas de actividad no elevadas

☛ implementación de operaciones más compleja (algorítmicamente)

→ en esencia es una operación de mezcla con tratamiento "especial"

2.3 - Operaciones de mantenimiento de ficheros y su implicación en la organización.

➪ otros operadores de mantenimiento:

• *mezclar (**ref** f1, f2, fRes: tpFSecDato;*
__función__ enOrden (d1, d2: tpDato) __devuelve__ booleano)

• *ordenar (**ref** f: tpFSecDato;*
__función__ enOrden (d1, d2: tpDato) __devuelve__ booleano)

→ *interesantes ejercicios de programación*

2.3 - Operaciones de mantenimiento de ficheros y su implicación en la organización.

mejora de prestaciones:

• *modificación de la semántica: lectura/escritura compatibles* → Pb. con dispositivo físico

⇩

operadores básicos:

- *Iniciar (f)*
- *LeerDato (f, d)*
- *FinFichero (f)*
- *Escribir (f, d)*

- *Asociar (f, nombre)*
- *Disociar (f)*

- *SiguienteDato (f)*
- *Avanzar (f)*

permite reducir algo el coste de las op. de mantenimiento: *ejercicio: definir una semántica*

2.3 - Operaciones de mantenimiento de ficheros y su implicación en la organización.

→ actualización simple si datos de longitud fija (o tamaño menor)

→ eliminación simple con marcas de borrado ⇒ *crecimiento aparente*

→ inserción al final (si no ordenado) , o aprovechando "huecos" → usar bloques "no llenos"

☛ se complican las operaciones

☛ las prestaciones se degradan con el tiempo

→ *operaciones de reestructuración (copiado) periódicas*

2.3 - Mezcla de ficheros secuenciales

```
Algoritmo MezclaFSec(E/S f1,f2,f: tpFS; funcion enOrden(d1,d2:tpDato)devuelve booleano);
{ crea el fichero ordenado f a partir de la mezcla de todos los datos de los ficheros secuenciales ordenados f1 y f2.
  El criterio de ordenación está especificado por la función EnOrden}
Variables  d : tpDato;
Principio
  IniciarLectura(f1);      IniciarLectura(f2);   IniciarEscritura(f);
  Mientras Que not FinFichero(f1) and not FinFichero(f2) Hacer
    Si enOrden(SiguienteDato(f1), SiguienteDato(f2))
      Entonces { copiar dato de f1 } LeerDato(f1, d);
      Si no    { copiar dato de f2 } LeerDato(f2, d);
    Fsi;
    EscribirDato(f, d)
  FMQ
  Mientras Que not FinFichero(f1) Hacer { copiar resto f1 }
    LeerDato(f1, d); EscribirDato(f,d)
  FMQ
  Mientras Que not FinFichero(f2) Hacer { copiar resto f2 }
    LeerDato(f2, d); EscribirDato(f,d)
  FMQ
Fin.
```

→ ejercicio: mezcla múltiple

2.3 - Ordenación de ficheros secuenciales.

- ☛ *Es uno de los procesos más frecuentes y de más interés*
- ☛ *No se pueden aplicar los métodos de ordenación de E.D. acceso directo*

  → Todos los métodos se basan en operaciones de mezcla

esquema general:

- iteración de un proceso elemental (**pasada**), basado en operaciones de **mezcla** y **distribución**.
- al tratamiento de todos los datos de un fichero se llama **fase**

2.3 - Ordenación de ficheros secuenciales.

métodos:

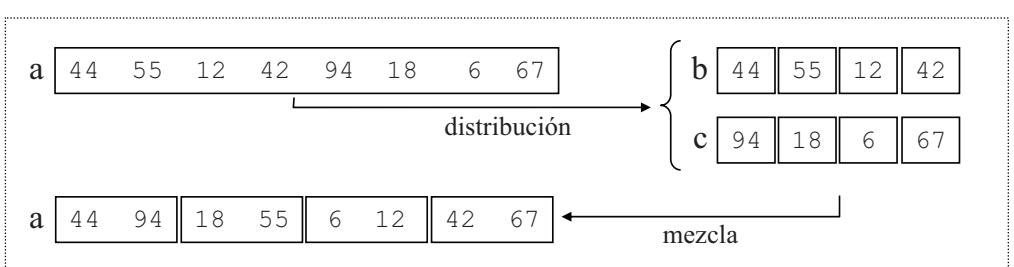
- ordenación por mezcla directa
- ordenación por mezcla natural
- ordenación por mezcla equilibrada múltiple
- ordenación por mezcla polifásica
- ordenación por mezcla en cascada

☛ *para compararlos: evaluación aproximada del coste para cada método*

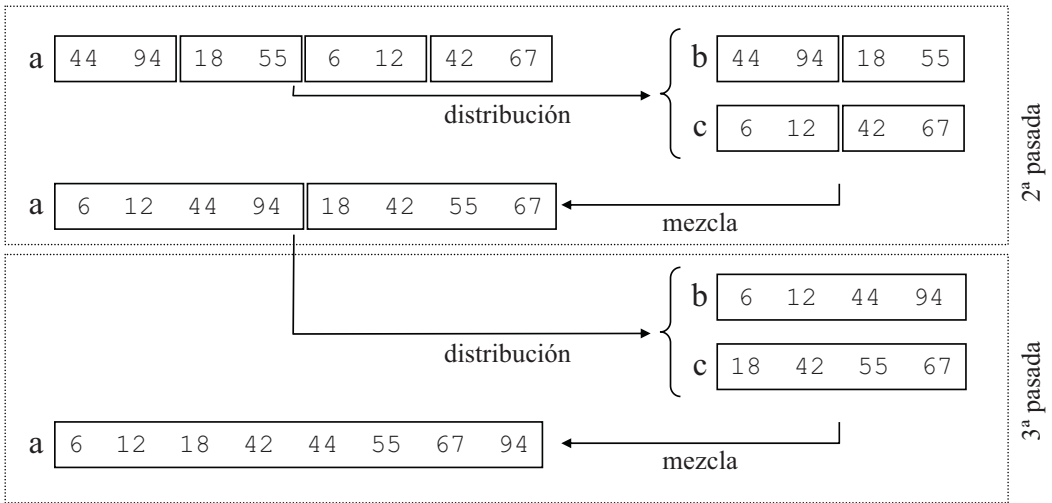
2.3 - Ordenación por mezcla directa

Ideas básicas:

- considerar el fichero como secuencias de longitud 1 (desordenado)
- la mezcla de dos secuencias (ordenadas) produce una secuencia (ordenada)



2.3 - Ordenación por mezcla directa



2.3 - Ordenación por mezcla directa

método: cada *pasada* consiste en

- división del fichero en otros 2, (todos de secuencias de longitud *l*)
- mezcla de las secuencias de los ficheros sobre el original (sec. de longitud *2*l*)

pb. técnicos:

- ⇨ el número de datos debe ser potencia de 2 ⇒ { - considerar datos "ficticios" - modificaciones algorítmicas
- ⇨ distribución de un fichero en otros 2 ≈ iguales → *trivial*

implementación: ejercicio interesante de programación

2.3 - Ordenación por mezcla directa

evaluación:

- son necesarios 3 ficheros (ocupación total = **2 * n**)
- nº de pasadas = $\lceil \log_2 n \rceil$

¿nº accesos físicos ?

nº movimientos = $Mv_d = 2 * n * \lceil \log_2 n \rceil$

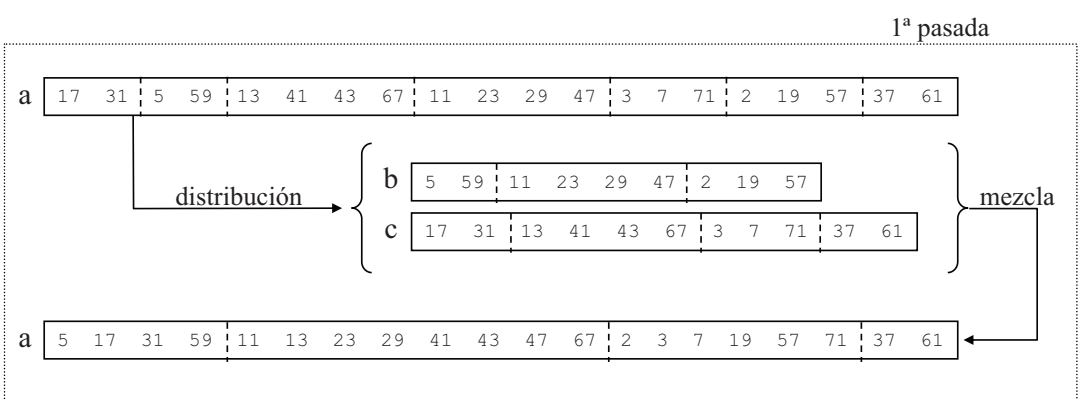
mejoras:

- eliminar la fase de distribución ⇒ usar 4 ficheros
- aprovechar al máximo la ordenación inicial de los datos → *mezcla natural*

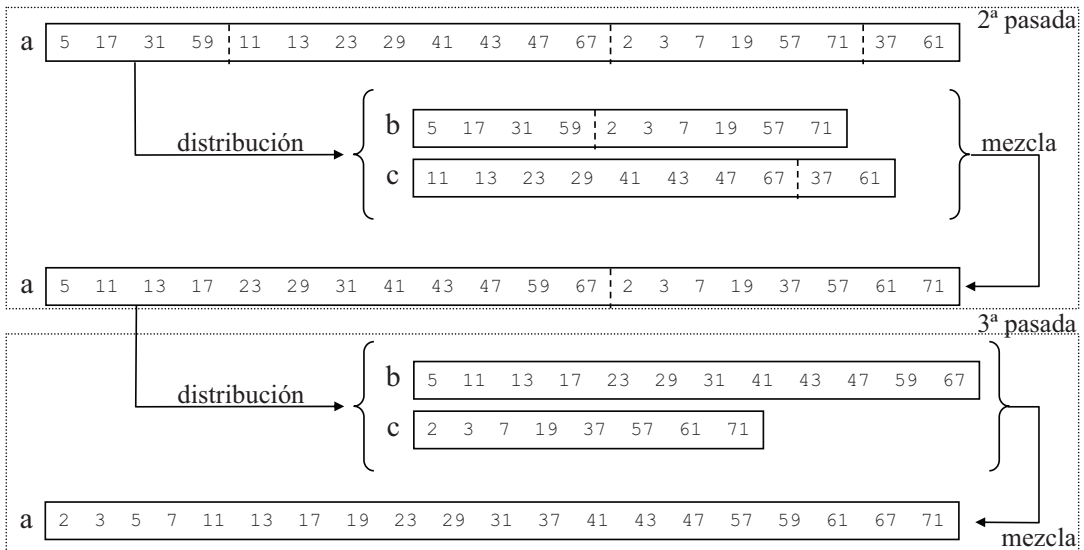
2.3 - Ordenación por mezcla natural

Ideas básicas:

- considerar el fichero como secuencias ordenadas (***tramos***)
- la mezcla de dos tramos produce un nuevo tramo



2.3 - Ordenación por mezcla natural



2.3 - Ordenación por mezcla natural

método: cada *pasada* consiste en *dos fases*

- *distribución* de los *tramos* del fichero en 2 ficheros
- *mezcla* de los *tramos* de los 2 ficheros sobre el original

pb. técnicos:

⇨ gestión de los tramos (≡ fin tramo) ⇒ *soluc.*: - FinFichero (f) OR - **NOT** enOrden (actual,SiguienteDato(f))

algoritmo

⇨ el nº de secuencias en la distribución - puede ser inferior al esperado - diferente en ambos ficheros (↓eficiencia)

soluc. → - guardar el último dato escrito en cada fichero - contar las secuencias en el proceso de mezcla

2.3 - Ordenación por mezcla natural

```
Algoritmo MezclaNatural (ref f: tpFS;  funcion enOrden(d1, d2: tpDato) devuelve booleano);
Variables numTramos: entero;   a, b: tpFS;
Principio
  Repetir
    IniciarLectura(f);  IniciarEscritura(a);  IniciarEscritura(b);
    Distribuir (f, a, b, numTramos);
    IniciarEscritura (f);  IniciarLectura(a);  IniciarLectura(b);
    Mezclar(a, b, f);
  Hasta Que numTramos = 1
Fin.
```

¡se cuentan mal los tramos!

```
Algoritmo Distribuir (ref f, a, b: tpFS;
Principio                ref numTramos: entero);
  numTramos := 0;
  MQ not FinFichero(f) Hacer
    copiarTramo(f, a);
    Si not FinFichero(f) Entonces copiarTramo(f, b) Fsi
    numTramos := numTramos + 1
  FMQ
Fin.
```

```
Algoritmo Mezclar (ref f, a, b: tpFS);
Principio
  MQ  not FinFichero(a) and not FinFichero(b) Hacer
    mezclarTramo(a, b, f)
  FMQ;
  Si not FinFichero(a) Entonces copiarTramo(a, f)
                       Si No    copiarTramo(b, f)
  FSi;
Fin.
```


2.3 - Ordenación por mezcla natural

```
Algoritmo MezclaNatural (ref f: tpFS; funcion enOrden(d1,d2: tpDato)devuelve booleano);
Variables numTramos: entero;     a, b: tpFS;
Principio
   Repetir
      IniciarLectura(f);      IniciarEscritura(a);   IniciarEscritura(b);
      Distribuir (f, a, b);
      IniciarEscritura (f);   IniciarLectura(a);     IniciarLectura(b);
      Mezclar (a, b, f , numTramos);
   Hasta Que numTramos = 1
Fin.
```

```
Procedimiento Distribuir (ref f, a, b: tpFS);
Principio;
  Mientras Que not FinFichero(f) Hacer
    CopiarTramo (f, a);
    Si not FinFichero(f) Entonces copiarTramo (f, b) Fsi
  FMQ
Fin.
```

** Se supone que el fichero no está vacío*

2.3 - Ordenación por mezcla natural

```
Procedimiento Mezclar (ref a, b, f: tpFS;  ref numTramos: entero);
Principio
   numTramos := 0;
   Mientras Que not FinFichero(a) and not FinFichero(b) Hacer
      MezclarTramo (a, b, f);      numTramos := numTramos + 1
   FMQ;
   Mientras Que not FinFichero(a) Hacer
      CopiarTramo (a, f);          numTramos := numTramos + 1
   FMQ;
   Mientras Que not FinFichero(b) Hacer
      CopiarTramo (b, f);          numTramos := numTramos + 1
   FMQ;
Fin.
```

```
Procedimiento CopiarTramo (ref x, y: tpFS);
Variables  fin_Tramo : boolean;
Principio
   Repetir CopiarDato (x, y, fin_Tramo) Hasta Que fin_Tramo
Fin.
```


2.3 - Ordenación por mezcla natural

```
Procedimiento MezclarTramo (ref a, b, f: tpFS);
Variables  fin_Tramo : boolean;
Principio
   Repetir
     Si enOrden (SiguienteDato(a), SiguienteDato(b))
        Entonces CopiarDato (a, f, fin_Tramo)
                 Si fin_Tramo Entonces CopiarTramo (b, f) FSi;

        Si no    CopiarDato (b, f, fin_Tramo)
                 Si fin_Tramo Entonces CopiarTramo (a, f) FSi;
     FSi;
   Hasta Que fin_Tramo;
Fin.
```

```
Procedimiento CopiarDato (ref x, y: tpFS; ref fin_Tramo : booleano);
Variables  actual : tpDato;
Principio
   LeerDato(x, actual);   EscribirDato(y, actual);
   Si FinFichero(x)
      Entonces fin_Tramo := verdad
      Si no    fin_Tramo := not enOrden(actual, SiguienteDato(x))
   Fsi
Fin.
```


2.3 - Ordenación por mezcla natural

evaluación:

- son necesarios 3 ficheros (ocupación total = **2 * n**)
- nº de pasadas $\cong \lceil \log_2 n_t \rceil$ siendo n_t el nº de tramos (<< n) **¿nº accesos físicos ?**

→ nº movimientos = $Mv_{nat} = 2 * n * \lceil \log_2 n_t \rceil$

mejoras:

- eliminar la fase de distribución ⇒ usar 4 ficheros ⇒ $Mv_{nat} = n * \lceil \log_2 n_t \rceil$
- Ordenación en memoria para la distribución inicial de los datos

→ *método del montículo* (orden m → long. Tramo $2*m$)

- equilibrado de tramos (mejora marginal)
- distribuir sobre más ficheros → *mezcla equilibrada múltiple*

2.3 - Ordenación por mezcla equilibrada múltiple

método: (similar a mezcla natural) cada *pasada* consiste en *dos fases*

- *distribución* de los *tramos* del fichero en ***m*** ficheros
- *mezcla* de los *tramos* de los ***m*** ficheros sobre el original

pb. técnicos: los de mezcla natural + gestión de la mezcla múltiple

evaluación:

- son necesarios ***m+1*** ficheros
- nº de pasadas $\cong \lceil \log_m n_t \rceil$ siendo n_t el nº de tramos (<< n)

→ nº movimientos = $Mv_{eqm} = 2 * n * \lceil \log_m n_t \rceil$

mejoras:

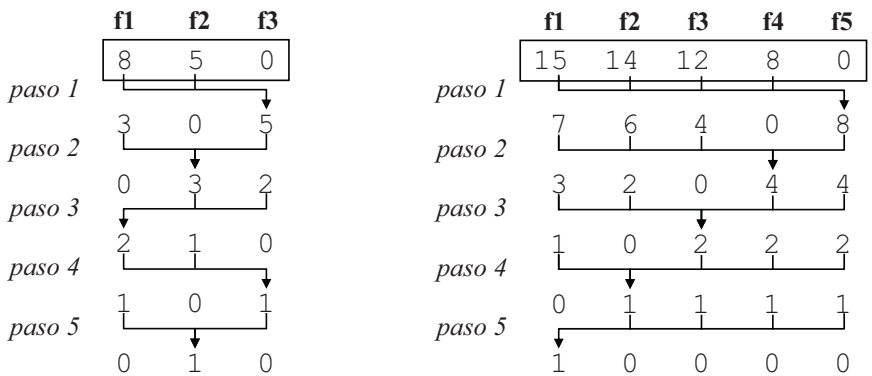
- eliminar la fase de distribución ⇒ usar 2m ficheros ⇒ $Mv_{eqm} = n * \lceil \log_m n_t \rceil$
- aprovechar más los ficheros (no haya ninguno vacío)

→ *mezcla polifásica*
→ *mezcla en cascada*

2.3 - Ordenación por mezcla polifásica

Idea básica: utilizar todos los ficheros en cada operación de mezcla

⇒ *cuando uno se queda vacío, pasa a ser destino de la mezcla*



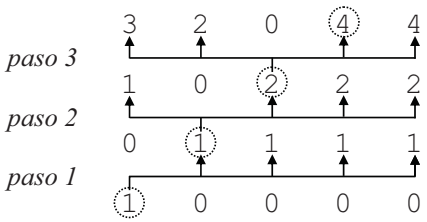
2.3 - Ordenación por mezcla polifásica

método:
- distribución inicial de los tramos sobre los ***m-1*** ficheros
- realizar *pasadas* de mezcla hasta que sólo quede un tramo
  - mezclar tramos de los ***m-1*** ficheros hasta que uno quede vacío
  - convertir éste en destino, y el anterior de destino, en origen

pb. técnicos:

⇨ los números de tramos debe ser valores concretos (n^{os} Fibonacci generalizados)

→ algoritmo constructivo simple



2.3 - Ordenación por mezcla polifásica

Algoritmo constructivo

$a_n[i] = a_{n-1}[i] + a_{n-1}[j]$

posición con 0 en fila n-ésima

nivel	f1	f2	f3	f4	f5	f6	*tramos totales*
0	0	0	0	0	0	1	1
1	1	1	1	1	1	0	5
2	0	2	2	2	2	1	9
3	2	0	4	4	4	3	17
4	6	4	0	8	8	7	33
5	14	12	8	0	16	15	65
6	30	28	24	16	0	31	129
7	61	59	55	47	31	0	253

pb. técnicos:

⇨ si los números de tramos reales no coinciden con estos valores (n^{os} Fibonacci)

tratamiento
- considerar tramos "ficticios" (simples contadores)
- ~~soluciones algorítmicas (complejas)~~

2.3 - Ordenación por mezcla polifásica

nivel	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	*tramos totales*
0	1	0	0	0	0	0	1
1	1	1	1	1	1	0	5
2	2	2	2	2	1	0	9
3	4	4	4	3	2	0	17
4	8	8	7	6	4	0	33
5	16	15	14	12	8	0	65
6	31	30	28	24	16	0	129
7	61	59	55	47	31	0	253

$a_n[i] = a_{n-1}[1] + a_{n-1}[i+1]$

d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	*tramos adicionales*
1	0	0	0	0	0	1
0	1	1	1	1	0	4
1	1	1	1	0	0	4
2	2	2	1	1	0	8
4	4	3	3	2	0	16
8	7	7	6	4	0	32
15	15	14	12	8	0	64
30	29	27	23	15	0	124

$d_n[i] = a_n[i] - a_{n-1}[i]$

tramos "ficticios" ≡ diferencia de tramos entre el nivel actual y el anterior. Al final, los que faltan para completar el número ideal

2.3 - Ordenación por mezcla polifásica

pb. técnicos:

- concatenación de tramos que no constituyen un nuevo tramo → añadir otro más

  ⇒ guardar último dato del tramo → caso particular en el primer nivel

- selección de fichero (distribución de tramos ficticios) → *solución de Knuth*

  ≈ "equilibrar" el nº de tramos ficticios

- mezcla de tramos "ficticios" y reales
  - ✓ si queda algún "ficticio", se elige (simplemente se descuenta)
  - ✓ se construye una lista con los ficheros reales a mezclar
  - ✓ si lista vacía → se produce un tramo ficticio

2.3 - Ordenación por mezcla polifásica

nivel	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	*tramos totales*
0	1	0	0	0	0	0	1
1	1	1	1	1	1	0	5
2	2	2	2	2	1	0	9
3	4	4	4	3	2	0	17
4	8	8	7	6	4	0	33
5	16	15	14	12	8	0	65
6	31	30	28	24	16	0	129
7	61	59	55	47	31	0	253

$a_n[i] = a_{n-1}[1] + a_{n-1}[i+1]$

d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	*tramos adicionales*
1	0	0	0	0	0	1
0	1	1	1	1	0	4
1	1	1	1	0	0	4
2	2	2	1	1	0	8
4	4	3	3	2	0	16
8	7	7	6	4	0	32
15	15	14	12	8	0	64
30	29	27	23	15	0	124

$d_n[i] = a_n[i] - a_{n-1}[i]$

nivel	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]
0	referencia (nº) de los ficheros que intervienen					
1	1	2	3	4	5	6
2	2	3	4	5	6	1
3	3	4	5	6	1	2
4	4	5	6	1	2	3
5	5	6	1	2	3	4
6	6	1	2	3	4	5
7	1	2	3	4	5	6

Antes de inicial la mezcla se inicia el vector con nºs consecutivos

2.3 - Ordenación por mezcla polifásica

fich. orig.: 170 183 80 188 86 150 240 167 127 182 202

197 117 73 11 313 256 303 113 96 53 222

distribución inicial :

fich. 1:	170	183			197	117			96	
fich. 2:	80	188			73				53	222
fich. 3:	86	150	240		11	313				
fich. 4:	167				256	303	113			
fich. 5:	127	182	202							

2.3 - Ordenación por mezcla polifásica

fich. 1:	F	170	183	197	117	96						
fich. 2:	F	80	188	73	53	222						
fich. 3:	F	F	86	150	240	11	313					
fich. 4:	F	167	256	303	113							
fich. 5:	F	127	182	202								
fich. 6:	----											

no existen, sólo están puestos para ilustrar el proceso de mezcla

fich. 1:	117	96										
fich. 2:	73	53	222									
fich. 3:	86	150	240	11	313							
fich. 4:	113											
fich. 5:	----											
fich. 6:	F	80	127	167	170	182	183	188	197	202	256	303

2.3 - Ordenación por mezcla polifásica

fich. 1:	96									
fich. 2:	53	222								
fich. 3:	11	313								
fich. 4:	----									
fich. 5:	73	86	113	117	150	240				
fich. 6:	80	127	167	170	182	183	188	197	202	256	303

fich. 1:	----
fich. 2:	----
fich. 3:	----
fich. 4:	todos los datos en orden
fich. 5:	----
fich. 6:	----

2.3 - Ordenación por mezcla polifásica

evaluación:

- son necesarios ***m*** ficheros
- nº de pasadas $\cong \lceil \log_{m-1} n_t \rceil$ siendo n_t el nº de tramos (<< n)

→ nº movimientos = $Mv_{pol} = n * \lceil \log_{m-1} n_t \rceil$

mejoras:

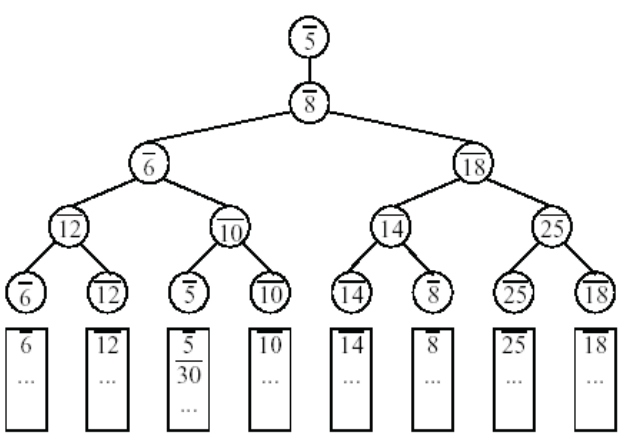
- ordenación en memoria en la fase de distribución inicial

→ *método del montículo* (orden $m \rightarrow$ long. tramo $\approx 2*m$)
algún pb. técnico en la integración

- algunas mejoras del algoritmo propuesto para casos particulares • • •
- árbol para mejorar la selección en la mezcla múltiple ⇨
- usar varios dispositivos físicos ⇒ paralelizar operaciones I/O (p.e. ***pipelining***) ⇨
- tamaño bloque (o buffer en memoria) grande → compromiso nº ficheros
- • • •

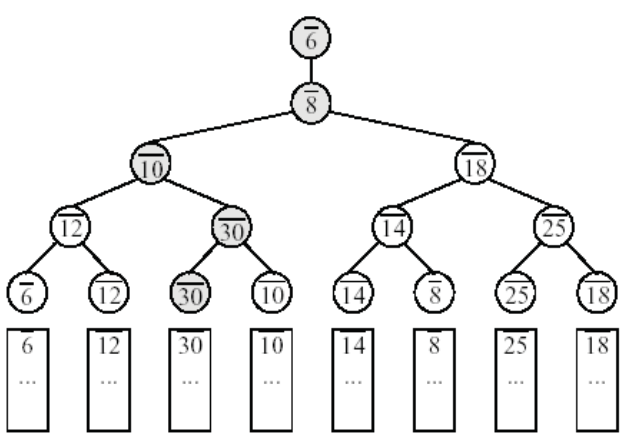
2.3 - Ordenación por mezcla polifásica

Mejorar la selección: árbol de perdedores



2.3 - Ordenación por mezcla polifásica

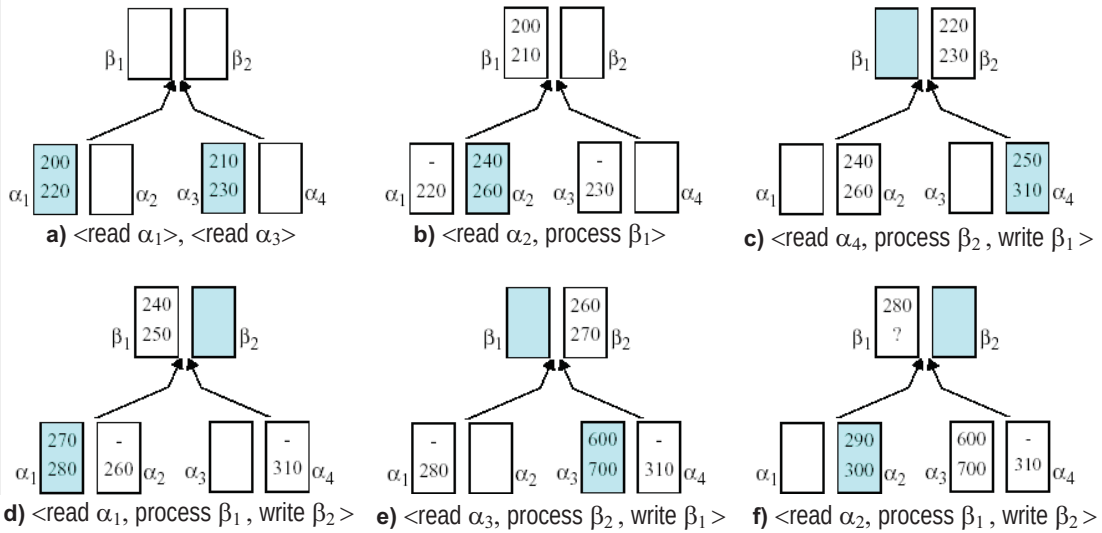
Mejorar la selección: árbol de perdedores



2.3 - Ordenación por mezcla polifásica

Paralelizar operaciones ⇦

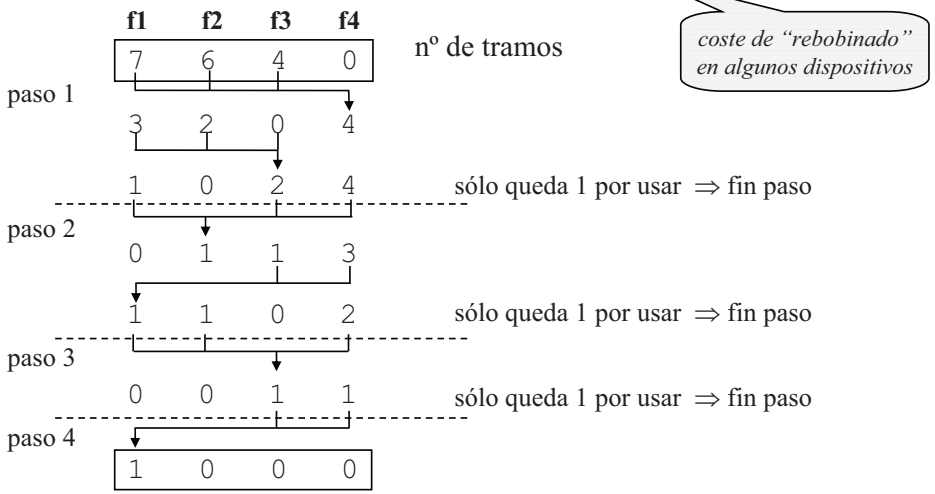
se usan 2 discos: 1 para destino y el otro para origen



2.3 - Ordenación por mezcla en cascada

Idea básica: no utilizar inmediatamente para lectura el fichero destino

coste de "rebobinado" en algunos dispositivos



2.3 - Ordenación por mezcla en cascada

método:
- distribución inicial de los tramos sobre los ***m-1*** ficheros
- realizar *pasadas* de mezcla hasta que sólo quede un tramo

```
k := m;
Repetir
  { mezclar tramos de los k-1 ficheros hasta que uno quede vacío }
  { convertir éste en destino; k := k - 1 (⇒ no se usará el de destino)}
Hasta Que {sólo quede un fichero con tramos: k = 1}
```

pb. técnicos:

➪ elección "optima" de la distribución de tramos inicial

evaluación:

➪ algo "peor" que la mezcla polifásica

2.4 - Otras organizaciones secuenciales: Organización secuencial encadenada.

objetivo: especificar una organización secuencial con mejores prestaciones de las operaciones de mantenimiento *(sobre todo el coste de la "reescritura" del fichero)*

idea básica: **implementar una lista**

- modificar la semántica de los operadores básicos *(lect./escr. compatibles)*
- añadir algún operador *(insertar y eliminar dato)*
- representación explícita del siguiente dato

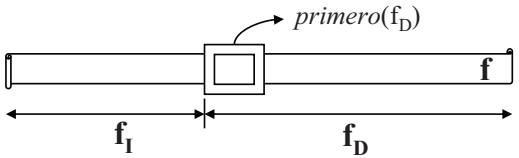
hipótesis (razonable): existe la capacidad de referenciar un registro (o bloque) físico
→ cierto a nivel del gestor del periférico y en la mayoría de gestores de ficheros

operadores básicos:

- *Asociar (***ref** f*: tpFSEnc;* nombre*: tpNombreFich)*
- *Disociar (***ref** f*: tpFSEnc)*

2.4 - Organización secuencial encadenada.

operadores básicos (cont.):



- *Iniciar (**ref** f: tpFSEnc; nombre: tpNombreFich)* → $f_I \leftarrow <>$; $f_D \leftarrow f$;
- *LeerDato (**ref** f: tpFSEnc; **ref** d: tpDato)* →
  d ← *primero*(f_D)
  $f_I \leftarrow f_I$ ° <*primero*(f_D)>
  $f_D \leftarrow$ *resto*(f_D)
- *FinFichero (**ref** f: tpFSEnc) **devuelve** booleano* → $f_D = <>$
- *EscribirDato (**ref** f: tpFSEnc; d: tpDato)* → $f_I \leftarrow f_I$ ° < d >; $f_D \leftarrow$ *resto*(f_D)

2.4 - Organización secuencial encadenada.

operadores básicos (cont.):

- *InsertarDato (**ref** f: tpFSEnc; d: tpDato)* → $f_I \leftarrow f_I$ ° < d >
- *EliminarDato (**ref** f: tpFSEnc)* → $f_D \leftarrow$ *resto*(f_D)

otros operadores :

- *SiguienteDato (**ref** f: tpFSEnc) **devuelve** tpDato*
  → **Si** $f_D \neq <>$ **entonces** *devuelve* (*primero*(f_D))
- *Avanzar (**ref** f: tpFSEnc)* →
  $f_I \leftarrow f_I$ ° <*primero*(f_D)>
  $f_D \leftarrow$ *resto*(f_D)

• • • •

2.4 - Organización secuencial encadenada.

representación : ⇒ debe existir acceso (físico) directo al registro (o bloque) → NRR

soluciones:

1) lista encadenada de registros (o bloques)
2) lista encadenada de registros (o bloques)
   + lista encadenada de registros (o bloques) libres
   ↳ *bloque descriptor del fichero*
3) listas de bloques de referencias de registros (o bloques)

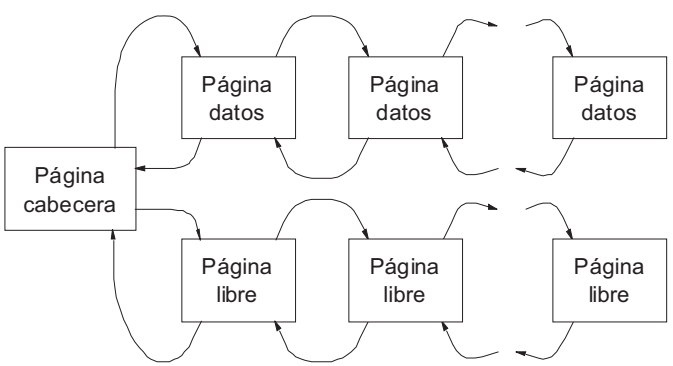
• • •

implementación:

tpRefBloque ≡ tipo para describir ref. física de bloque (entero)
valor especial *refBloqueNula*
función *NuevaRefBloque* ***devuelve*** *tpRefBloque*;
proced. *LeerBloque* (rfBloque: *tpRefBloque;* **ref** bloque: *tpBloque*);
proced. *EscribirBloque* (rfBloque: *tpRefBloque;* bloque: *tpBloque*);
función *DisponerBloque* (**ref** rfBloque: *tpRefBloque*);

2.4 - Organización secuencial encadenada. Desarrollo de una implementación sencilla.

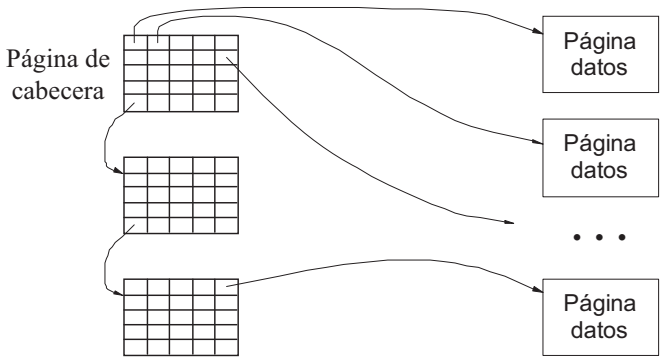
Implementación como lista bidireccional



- ✓ *bloque descriptor con comienzo de listas*
- ✓ *cada página contiene los datos (registros) y 2 referencias de bloque*

2.4 - Organización secuencial encadenada. Desarrollo de una implementación sencilla.

Implementación usando un directorio de páginas



☛ *cada entrada de página puede incluir el nº de bytes libres de la página*