

Ficheros

J. R. Paramá



- Introducción
- Conceptos básicos de ficheros
- Características del medio físico
- Organización de los registros en archivos
 - Ficheros Montículo
 - Ficheros Ordenados
 - Ficheros de acceso directo (Hash)
 - Hash Extensible

- Aunque los SGBD proporcionan una visión de alto nivel de los datos, en realidad los datos están almacenados como bits en uno o más dispositivos de almacenamiento.
- Lo normal es que los usuarios manejen un SGBD usando un modelo de datos lógico, como el relacional, que muestra los datos en relaciones (o tablas). El objetivo es ocultar al usuario la complejidad de los detalles físicos.
- Sin embargo las características físicas de los dispositivos de almacenamiento y las características lógicas de los datos (estructura y métodos de acceso) tienen un fuerte impacto en el rendimiento de los SGBD.
- Por ejemplo, un acceso a disco magnético requiere decenas de milisegundos, mientras que el acceso a memoria invierte unos pocos nanosegundos.

- Introducción
- Conceptos básicos de ficheros
- Características del medio físico
- Organización de los registros en archivos
 - Ficheros Montículo
 - Ficheros Ordenados
 - Ficheros de acceso directo (Hash)
 - Hash Extensible

Registros y tipos de registros

- Los archivos se organizan lógicamente como **secuencias de registros**.
- Cada registro consta de una colección de **valores** o *elementos* de datos relacionados, donde cada valor está formado de uno o más bytes y corresponde a un determinado **campo** del registro.
- El tipo de datos de cada campo, especifica el tipo de valores que el campo puede tomar.

```
struct alumno{  
    char nombre[30];  
    date fecha-nacimiento;  
    int curso;  
    char titulación[50];  
};
```

Claves

- Una **clave** (en el entorno de ficheros) es un campo o conjunto de campos usados para **identificar u ordenar los registros de un fichero**.
- Una clave **externa** se deriva de alguna característica física del registro almacenado, por ejemplo, su posición dentro del fichero.
- Una **clave primaria** es aquella clave tal que no existen dos registros que puedan tener el mismo valor.

Registros de longitud fija y registros de longitud variable

Un **fichero** es una **secuencia** de registros: de **registros de longitud fija** o **registros de longitud variable**.

- Registros del mismo tipo pero con **campos de longitud variable**.
- Registros del mismo tipo pero con **campos repetitivos**.
- Los **registros del fichero son de distintos tipos**.

Conceptos básicos de ficheros

Registros de longitud fija y registros de longitud variable

Nombre	Dirección	Dept	Puesto trabajo	Salario
30 bytes	50 bytes	10 bytes	20 bytes	18 bytes

Registro de empleado de tamaño fijo

Nombre	Dirección	Dept	Puesto trabajo	Salario Mes	
Nombre	Dirección	Dept	Puesto trabajo	Salario Hora	Horas trabaj

Registro de longitud variable de distinto formato.

Nombre	Dirección	Dept	Puesto trabajo	Salario
Nombre	Dirección	Dept	Puesto trabajo	Salario

Registro de longitud variable con campos de longitud variable.

Nombre	Dirección	Dept	Puesto trabajo	Nombre Hijo 1	
Nombre	Dirección	Dept	Puesto trabajo	Nombre Hijo 1	Nombre Hijo 2

Registro de longitud variable con grupos repetitivos.

Conceptos básicos de ficheros

Registros de longitud fija

- Si suponemos que nuestros registros ocupan 40 bytes, un enfoque sencillo usaría los 40 primeros bytes del archivo para el primer registro, los cuarenta siguientes para el segundo, y así sucesivamente.
- Pero esto tiene el problema de que resulta complicado borrar. O se rellena el hueco con otro registro o se marca como borrado.
- Cuando se borra un registro se puede sustituir por el siguiente, pero esto implica que habría que hacerlo con el resto de registros hasta el final del fichero.
- Esto implica desplazar gran número de registros. Puede que fuese más sencillo ocupar el borrado con el último registro del fichero.

Conceptos básicos de ficheros

Registros de longitud fija

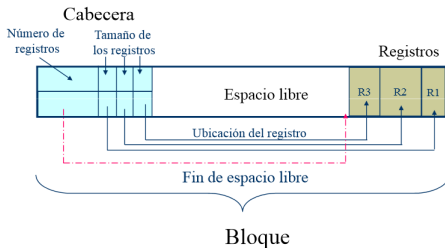
- En cualquier caso no resulta deseable reescribir el espacio ocupado por un registro borrado por representar un costo alto que haría enlentecer los borrados.
- Resulta más aceptable dejar el espacio del borrado libre y esperar una futura inserción que reutilizaría ese espacio.
- No basta con una marca de borrado, porque sería muy lento encontrar los borrados cuando se hace la inserción. Por tanto es necesaria una estructura adicional que permita localizar las posiciones borradas de forma eficiente.
- Lo más sencillo es almacenar en la cabecera del fichero la posición del primer registro borrado, y usar el primer registro borrado para almacenar la posición del segundo, y así sucesivamente.
- La inserción y borrado de registros de longitud fija es sencilla dado que el espacio que deja un borrado es justo el que necesita una inserción.

Conceptos básicos de ficheros

Registros de longitud variable

- Existen distintas estrategias para implementar ficheros con registros de longitud variable.
- Una es la **estructura de bloques con slots** (*slotted-page*).
- Los registros se colocan dentro de **bloques**.
- Cada bloque contiene:
 - El número de registros.
 - El espacio vacío.
 - Un array cuyas entradas contienen la ubicación y el tamaño de cada registro.

Registros de longitud variable



(<http://gpd.sip.ucm.es/yolanda/BDSI0708/S2FichLog.pdf>)

- Los registros reales se colocan en el bloque de una *manera contigua*, empezando por el final.
- El espacio libre es contiguo, entre la última entrada del array de la cabecera y el primer registro.
- Si se inserta un registro, se le asigna espacio al final del espacio libre y se añade a la cabecera una entrada que contiene su tamaño y ubicación.

Conceptos básicos de ficheros

Registros de longitud variable

- Si se borra un registro, se marca su entrada como *borrado*.
- Además se desplazan los registros del **bloque** situados antes del registro borrado, para que todo el espacio libre quede contiguo.
- Se puede cambiar el tamaño de los registros utilizando un método similar al del borrado.
- El coste de trasladar los registros dentro del bloque no es demasiado elevado porque los bloques son pequeños (normalmente 4 KB), y además se hace en memoria.
- Con esta técnica los punteros a registros no pueden ser a la posición exacta, sino al bloque donde está el registro, ya que como vimos la posición dentro del bloque de un registro puede cambiar.

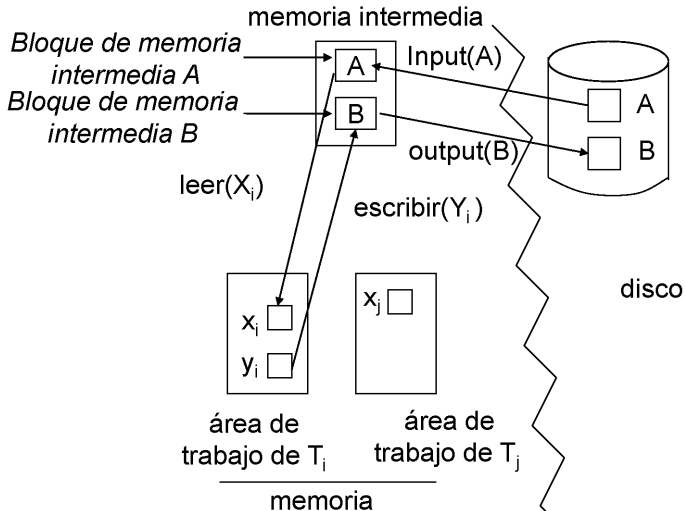
Conceptos básicos de ficheros

Las seis tareas básicas sobre ficheros

- **Añadir.** Añadir un nuevo registro al fichero.
- **Borrar.** Borrar un registro del fichero.
- **Leer todos los registros en cualquier orden.**
- **Leer todos los registros en el orden de clave.**
- **Leer un registro con un valor específico de clave.**
- **Actualizar el registro actual.**

Características del medio físico

Las dos operaciones básicas de E/S



- Introducción
- Conceptos básicos de ficheros
- Características del medio físico
- Organización de los registros en archivos
 - Ficheros Montículo
 - Ficheros Ordenados
 - Ficheros de acceso directo (Hash)
 - Hash Extensible

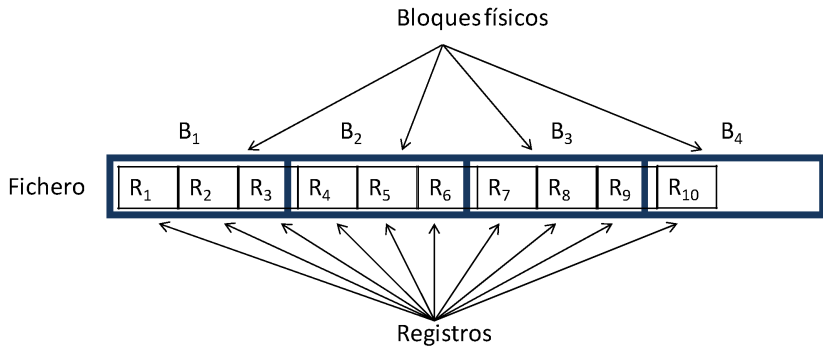
Características del medio físico

Bloques

- Los bloques físicos (o de disco) es la unidad de transferencia de disco a memoria.
- Los registros de un fichero se deben asignar a bloques del disco.
- El *factor de bloqueo* es el número de registros lógicos por bloque físico
- Si el tamaño de bloque es de *bloq* bytes. Para un fichero con registros de longitud fija con tamaño R , siendo $bloq \geq R$ podemos colocar $B = bloq/R$ registros por bloque.

Características del medio físico

Bloques



No existe una fórmula para calcular el factor de bloqueo adecuado. Existen dos límites uno superior y otro inferior.

- El **límite inferior** viene impuesto por la necesidad de evitar un número excesivo de transferencias de información cuando se está leyendo el fichero secuencialmente.
- El **límite superior**: tamaño del buffer de memoria principal o el tiempo para transmitir los datos, si sólo necesitamos un pequeño registro, se debe leer el bloque completo, con el consiguiente consumo de tiempo de transferencia.

El tamaño de bloque físico viene determinado por el formateo del disco.

Así que no es fácil cambiar el factor de bloqueo. Lo que podemos hacer es crear particiones de disco con un tamaño de bloque físico específico (si el sistema operativo lo permite.), y ese tamaño de bloque será el mismo para todos los ficheros en esa partición.

- Introducción
- Conceptos básicos de ficheros
- Características del medio físico
- Organización de los registros en archivos
 - Ficheros Montículo
 - Ficheros Ordenados
 - Ficheros de acceso directo (Hash)
 - Hash Extensible

Organización de los registros en archivos

- **Montículo**. Los registros se colocan en cualquier lugar en el que haya espacio suficiente. Los registros no se ordenan.
- **Ordenado**. Los registros se guardan según el valor de la *clave*.
- **Hash**

- Introducción
- Conceptos básicos de ficheros
- Características del medio físico
- Organización de los registros en archivos
 - [Ficheros Montículo](#)
 - Ficheros Ordenados
 - Ficheros de acceso directo (Hash)
 - Hash Extensible

Leer todos los registros en cualquier orden

- Es una lectura secuencial del fichero.
 - Se abre el fichero.
 - Se van leyendo los registros y se procesan los que son válidos.
- Un factor de bloqueo alto favorece esta operación, ya que muchas lecturas lógicas se resuelven en memoria intermedia con un coste del orden de nanosegundos, en lugar de tener que acceder a disco (que tiene un coste del orden de milisegundos).

Siendo N el número de registros en el fichero.

Leer un registro con un valor de clave específico

- Que exista un único registro que tenga el valor de clave buscado El fichero es leído secuencialmente hasta que se encuentre el registro con la clave buscada. $N/2B$ lecturas físicas
- Que existan varios registros con el valor de clave buscado Recorrer el fichero por completo el fichero. N/B lecturas físicas.
- Que no exista ningún registro con el valor de clave buscado. Recorrer el fichero por completo el fichero. N/B lecturas físicas.

Leer todos los registros por orden de clave

La mejor solución para este caso es la utilización del algoritmo **Merge Sort**.

- **1ª Fase.** Dividir el fichero en porciones que quepan en memoria principal. Cada porción se ordena en memoria principal utilizando cualquier algoritmo de ordenación en memoria. Se reescribe en disco la porción, pero ahora ordenada.
- **2ª Fase.** Merge.

FASE MERGE

Abre todos los archivos con porciones del archivo (ahora ordenadas):

- 1. Al abrirse se coloca un puntero al primer registro de cada fichero (porción).
- 2. Se compara el valor de la clave de los registros apuntados por el puntero de cada porción. El que tenga el valor más pequeño de clave se escribe en un nuevo fichero, avanzando el puntero de la porción procesada al siguiente registro.
- 3. Repetir 2 hasta que los punteros de todas las porciones lleguen al final

Ficheros Montículo

Tareas básicas

EJEMPLO

5	12	10	7	2	15	18	20	3	9	1	6	14	11	4
---	----	----	---	---	----	----	----	---	---	---	---	----	----	---

Supongamos que caben en memoria cuatro registros. Por lo tanto creamos cuatro porciones:

Porción 1	Porción 2	Porción 3	Porción 4
5 12 10 7	2 15 18 20	3 9 1 6	14 11 4

Ordenamos uno a uno, en memoria, las cuatro porciones:

Porción 1	Porción 2	Porción 3	Porción 4
5 7 10 12	2 15 18 20	1 3 6 9	4 11 14

A continuación se abren los cuatro fichero, leyendo el primer registro en cada una de ellas.

Porción 1	Porción 2	Porción 3	Porción 4
5 7 10 12 ↑	2 15 18 20 ↑	1 3 6 9 ↑	4 11 14 ↑

Ficheros Montículo

Tareas básicas

Se escoge el registro de clave más pequeño (en el orden escogido) de los cuatro registros apuntados actualmente, y se escribe en un nuevo fichero, haciendo avanzar el puntero de la porción procesada.

Porción 1	Porción 2	Porción 3	Porción 4																																
<table><tr><td>5</td><td>7</td><td>10</td><td>12</td></tr><tr><td>↑</td><td></td><td></td><td></td></tr></table>	5	7	10	12	↑				<table><tr><td>2</td><td>15</td><td>18</td><td>20</td></tr><tr><td>↑</td><td></td><td></td><td></td></tr></table>	2	15	18	20	↑				<table><tr><td>1</td><td>3</td><td>6</td><td>9</td></tr><tr><td></td><td>↑</td><td></td><td></td></tr></table>	1	3	6	9		↑			<table><tr><td>4</td><td>11</td><td>14</td><td></td></tr><tr><td>↑</td><td></td><td></td><td></td></tr></table>	4	11	14		↑			
5	7	10	12																																
↑																																			
2	15	18	20																																
↑																																			
1	3	6	9																																
	↑																																		
4	11	14																																	
↑																																			

Nuevo fichero

1														
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Ahora se escoge el siguiente registro de entre los apuntados por el puntero de cada porción, en este caso el 2.

Porción 1	Porción 2	Porción 3	Porción 4																																
<table><tr><td>5</td><td>7</td><td>10</td><td>12</td></tr><tr><td>↑</td><td></td><td></td><td></td></tr></table>	5	7	10	12	↑				<table><tr><td>2</td><td>15</td><td>18</td><td>20</td></tr><tr><td></td><td>↑</td><td></td><td></td></tr></table>	2	15	18	20		↑			<table><tr><td>1</td><td>3</td><td>6</td><td>9</td></tr><tr><td></td><td>↑</td><td></td><td></td></tr></table>	1	3	6	9		↑			<table><tr><td>4</td><td>11</td><td>14</td><td></td></tr><tr><td>↑</td><td></td><td></td><td></td></tr></table>	4	11	14		↑			
5	7	10	12																																
↑																																			
2	15	18	20																																
	↑																																		
1	3	6	9																																
	↑																																		
4	11	14																																	
↑																																			

Nuevo fichero

1	2													
---	---	--	--	--	--	--	--	--	--	--	--	--	--	--

Ficheros Montículo

Tareas básicas

Ahora el proceso continúa del mismo siguiendo el mismo procedimiento, el siguiente elemento es el 3 (en la porción 3).

Porción 1				Porción 2				Porción 3				Porción 4			
5	7	10	12	2	15	18	20	1	3	6	9	4	11	14	
↑					↑					↑		↑			

Nuevo fichero

1	2	3													
---	---	---	--	--	--	--	--	--	--	--	--	--	--	--	--

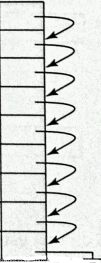
El proceso termina cuando todos los registros se han procesado.

- Introducción
- Conceptos básicos de ficheros
- Características del medio físico
- Organización de los registros en archivos
 - Ficheros Montículo
 - [Ficheros Ordenados](#)
 - Ficheros de acceso directo (Hash)
 - Hash Extensible

Ficheros ordenados

- Para permitir la recuperación rápida de los registros según el orden de clave, éstos están enlazados mediante punteros.
- Además, para minimizar el número de accesos a los bloques, los registros se guardan físicamente en el orden indicado por la clave, o lo más cercano posible.

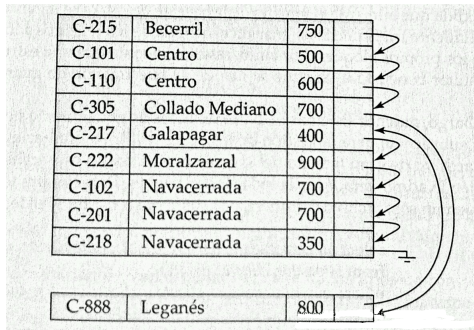
C-215	Becerril	750	
C-101	Centro	500	
C-110	Centro	600	
C-305	Collado Mediano	700	
C-217	Galapagar	400	
C-222	Moralzarzal	900	
C-102	Navacerrada	700	
C-201	Navacerrada	700	
C-218	Navacerrada	350	



The diagram illustrates a linked list structure where each record in the table is connected to the next record by a pointer. The pointers are represented by arrows on the right side of the table, indicating the sequence of records based on their key values. The sequence of records as indicated by the pointers is: C-215 (750) → C-101 (500) → C-110 (600) → C-305 (700) → C-217 (400) → C-222 (900) → C-102 (700) → C-201 (700) → C-218 (350).

- Resulta difícil mantener el orden físico secuencial debido a las inserciones y borrados, dado que como ya comentamos resulta costoso desplazar muchos registros como consecuencia de una sola operación de borrado o inserción.
- Los borrados e inserciones se pueden gestionar con punteros, pero esto puede obligar a que una lectura en orden de clave no coincida con el orden físico, perdiendo las ventajas del factor de bloqueo y aumentando los tiempos de acceso a disco por el posible movimiento de las cabezas.
- Para evitar este problema puede ser necesario *reorganizar* el fichero cada cierto tiempo, de modo que los registros estén en el orden de clave físicamente.
- Esta reorganización es muy costosa.

Ficheros ordenados



Ficheros secuenciales

Tareas básicas

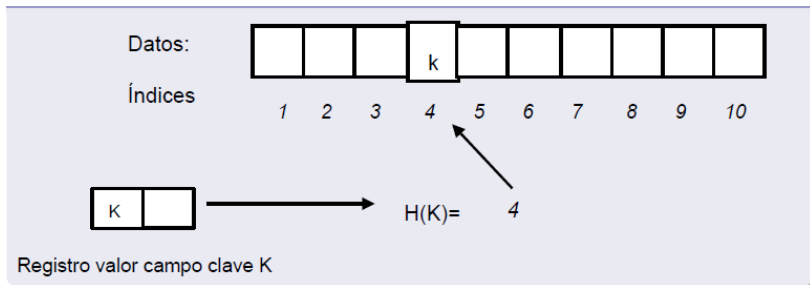
- **Leer un registro con un valor específico de clave** Bajo ciertas condiciones se podría realizar búsqueda binaria (si sólo hay un registro con el valor buscado se necesitan de media $\log_2(N)$ lecturas físicas). Pero rara vez se usa en disco, así que normalmente son N/B lecturas físicas.
- **Leer todos los registros en orden de clave.** Leer los registros siguiendo los punteros (N/B lecturas físicas).

- Introducción
- Conceptos básicos de ficheros
- Características del medio físico
- Organización de los registros en archivos
 - Ficheros Montículo
 - Ficheros Ordenados
 - Ficheros de acceso directo (Hash)
 - Hash Extensible

Ficheros de acceso directo (HASH)

- En los ficheros de acceso directo los registros pueden ser accedidos utilizando en **número de registro** como clave externa.
- De este modo el fichero es similar a un array unidimensional de tamaño fijo donde cada elemento del array es un registro y el índice del array el número de registro. A cada posición de ese imaginario array le denominamos **slot**.
- Pero los registros no se asignan arbitrariamente a los slots. La posición de un registro dado se deriva de su clave, aplicando sobre dicha clave una **función de hash $H(\text{clave})$** .

Ficheros de acceso directo (HASH)



Funciones de hash

Una buena función de hash debe tener tres características:

- Los números de registro que genere deben estar distribuidos **uniformemente y aleatoriamente** sobre el fichero.
- **Pequeñas variaciones en el valor de la clave** deben causar **grandes variaciones en el valor de $H(\text{clave})$** . **Todas las partes de la clave** deben ser utilizadas por la función de hash.
- La función de hash debe minimizar la creación de sinónimos.
 - Un **sinónimo** es una clave que al serle aplicada la función de hash el resultado es el mismo que el de una clave diferente. Es decir, si $H(X) = H(Y)$, entonces X e Y son sinónimos, también denominadas **colisiones**.

Funciones de hash

- El problema de encontrar una buena función de hash está relacionado con el de encontrar un buen generador de números pseudoaleatorios.
- La diferencia de la función de hash está en que debe ser **repetible**.
- Un fichero de N slots tendrá un espacio de direcciones de N direcciones, mientras que normalmente la clave tendrá un espacio de direcciones mucho más grande.
- Por lo tanto no existe modo alguno de asignar a todo valor del espacio de direcciones de la clave una dirección de fichero única.

Funciones de hash

- Si la clave no es numérica se debe **FOLD=plegar** para transformarla en un número. Una práctica común y sencilla es tomar los valores numéricos de los códigos ASCII de los caracteres que forman la clave, y aplicar sobre ellos alguna operación matemática sobre ellos.
- Una vez que se dispone de un valor numérico, la fórmula más utilizada por su sencillez y efectividad es:

$$H(K) = K \bmod N$$

- donde K es la forma numérica de la clave y N es el número de slots del fichero. Observe que $0 \leq H(K) < N$. Para obtener buenos resultados K debe ser mucho mayor (varios órdenes de magnitud) que N .

Ficheros de acceso directo (HASH): Funciones de hash

- $H(K) = K \bmod N$ puede dar malos resultados si el valor de N no se escoge con cuidado.
- Si $N = 1000$, $H(K)$ será siempre en los tres dígitos de menor orden. Los otros seis dígitos no tendrán ninguna influencia en el resultado de la función de hash.
- Para solucionar esto, simplemente escogemos un número primo como N .

Ficheros de acceso directo (HASH): Estructuras para manejar sinónimos

Sea M el número de registros almacenados en un fichero de N slots, el **factor de carga** es $(M \div N)$.

Núm regs k	Factor de carga									
	20 %	50 %	70 %	80 %	85 %	90 %	95 %	98 %	99 %	100 %
0	.81873	.60653	.49659	.44933	.42741	.40657	.38674	.37531	.37158	.36788
1	.16375	.30327	.34761	.35946	.36330	.36591	.36740	.36780	.36786	.36788
2	.01637	.07582	.12166	.14379	.15440	.16466	.17452	.18022	.18209	.18394
3	.00109	.01264	.02839	.03834	.04375	.04940	.05526	.05887	.06009	.06131
4	.00005	.00158	.00497	.00767	.00930	.01111	.01313	.01442	.01487	.01533
5	.00000	.00016	.00070	.00123	.00158	.00200	.00249	.00283	.00294	.00307
6	.00000	.00001	.00008	.00016	.00022	.00030	.00039	.00046	.00049	.00051
7	.00000	.00000	.00001	.00002	.00003	.00004	.00005	.00006	.00007	.00007
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00001	.00001	.00001

Cuando el factor de carga se acerca al 100 %, la fracción de slots a los que la función de hash no asigna ningún registro se acerca a $1/e$ (0.368). La fracción de slots a los que la función de hash asigna un único registro es otra vez $1/e$

Los restantes $(1 - 2/e)$ slots (o 26,4 % de los slots), la función de hash les asigna los restantes $(1 - 1/e)$ (o 63,2 %) registros

Estructuras para manejar sinónimos: Fichero de overflow

- Si el factor de carga del fichero principal se mantiene bajo, la proporción de sinónimos será baja, y el fichero de overflow no necesitará mucho espacio.
- Este fichero puede organizarse como se desee. La forma más sencilla es organizarlo como un fichero montículo. Mientras su tamaño sea pequeño, los retardos serían aceptables.
- Se podría organizar como otro fichero de acceso directo, pero más pequeño. De este modo necesitaríamos a su vez otro fichero de overflow, una vez más, todavía más pequeño.
- Existen dos desventajas para el fichero de overflow.
 - El fichero principal no debe tener un factor de carga demasiado alto.
 - Dado que el fichero principal debe tener un factor de carga bajo, supondrá un desperdicio de espacio libre considerable debido a los slots vacíos.

Estructuras para manejar sinónimos: Direcccionamiento abierto

- Para aprovechar el 36,7 % de slots que siempre están vacíos (a un 100 % de factor de carga) necesitamos algún método para almacenar los sinónimos en los slots vacíos.
 - El **direcccionamiento abierto** se basa en la existencia de una secuencia predecible de slots a examinar cuando se está buscando un slot vacío para introducir un registro.
 - Cuando se busca ese registro, dado que la secuencia es predecible, es **repetible**. Por lo tanto se recorre del mismo modo que se hizo cuando se buscaba el slot vacío.
- Hay dos métodos de direcccionamiento abierto: El **prueba lineal** y **rehashing**.

Direccionamiento abierto: Prueba lineal

- **Prueba lineal** simplemente establece que cuando estamos ante un sinónimo, el hueco se buscará en los slots siguientes a la posición natural ($H(k)$) del registro.
- Si el registro debería estar en el slot i y está ocupado, se buscará en los slots $i + 1, i + 2, \dots$ (en módulo N) hasta que se encuentre un slot vacío.
- En el caso de que se llegue al final del fichero se pasa al principio del mismo.
- Con esto acabamos de describir la operación de **Añadir** un registro al fichero.

Leer un registro con un valor de clave específico con prueba lineal

```
boolean busca-reg(clave-buscada,n)
k:=PLEGAR(clave-buscada); /*Plegado de la clave*/
r:= k mod n; /*Cálculo de la dirección natural*/
LEER(r); /*Leer dirección natural*/
WHILE r no tenga marca de vacío AND CLAVE(r)  $\neq$  clave-buscada
    DO BEGIN
        /* Mientras no se encuentra el registro seguimos buscando*/
        r:=(r+1) mod n; /*Sigue la prueba lineal*/
        LEER(r);
    END;
IF CLAVE(r) = clave-buscada
    THEN RETURN TRUE
    ELSE RETURN FALSE;
```

Prueba lineal: Problemas

- Con un factor de carga cercano al 100 %, puede que algunas áreas estén demasiado congestionadas.
- Estudios empíricos muestran que con un 99 % de factor de carga, la búsqueda de un registro representa **inspeccionar de media 25 slots**, en los **peores casos 1000 slots**.
- Si el factor de carga se mantiene por debajo del 85 %, el rendimiento es aceptable.
- AL realizar una búsqueda **secuencial** desde la posición natural del registro \rightarrow beneficia un factor de bloqueo alto.
- En el caso anterior si en lugar de tener un **factor de bloqueo 1**, pasamos a tener **50**, el **número medio de lecturas físicas pasa 1.5**.

Prueba lineal: Problemas

- No es fácil borrar registros. El algoritmo de búsqueda **busca-reg**, interpreta un registro vacío como la prueba de que un registro no está en el fichero.
- Una solución simple pero costosa es continuar la búsqueda hasta que se encuentre el registro o se haya recorrido todo el fichero.
- Más razonable sería marcar los slots como “abandonados”, pero no borrarlos. El algoritmo **busca-reg** no terminaría su búsqueda al encontrar un slot “abandonado”, sin embargo, ese slot podría ser utilizado a la hora de insertar un nuevo registro.

- Leer de todos los registros en cualquier orden Lectura secuencial.
- Leer todos los registros en orden de clave. Dos posibilidades
 - Utilizar el algoritmo Merge-Sort.
 - Realizar la secuencia:
 - Hacer una lectura secuencial del fichero para obtener todos los valores de clave en el fichero.
 - Ordenarlas en un array en memoria con cualquier método de ordenación en memoria.
 - Leer el array por orden, y aplicar para cada clave el algoritmo busca-reg.

Rehashing

- Necesita **dos funciones de hash**. La primera función es tal y como la hemos presentado anteriormente: $H(K) = K \bmod N$.
- Si el slot apuntado por $H(K)$ está ocupado, se aplica la segunda función de hash $D = (K \bmod J) + 1$.
- J es un número primo diferente y más pequeño que N .
- D se utiliza como un desplazamiento sobre $H(K)$. Dado que $H(K) + D$ puede ser mayor que N , esta suma se realiza módulo N .

Rehashing

- Con esto se evita el problema de la prueba lineal que tendía a congestionar demasiado zonas del fichero, y de este modo, las búsquedas de un registro requerirán menos lecturas físicas debido a las pruebas de distintos slots.
- Sin embargo, dado que D casi siempre va a ser mayor que el factor de bloqueo, las ventajas asociadas se pierden.
- Se intenta insertar en el slot $H(K)$, si está ocupado, se intenta en $(H(K) + D) \bmod N$, si éste también lo está, se trata de insertar en $(H(K) + 2D) \bmod N$ y así sucesivamente.

Problemas

- Cualquier par de claves que son sinónimos con la función de hash principal, es altamente deseable que no lo sean en el segunda, sino estaría siempre compitiendo por los mismos slots.
- D siempre debe ser mayor que cero o el procedimiento iterará por siempre. Esto se asegura al añadir un $+1$ en la expresión que calcula D .
- Podríamos pensar que el algoritmo que inserta un nuevo registro puede entrar en un **bucle sin fin** a pesar de haber slots vacíos. Sin embargo, al **ser N primo**, y ser el **desplazamiento** (dado que se suma a $H(K)$ en módulo N) **menor que N** , esto nunca puede ocurrir.

- Introducción
- Conceptos básicos de ficheros
- Características del medio físico
- Las dos operaciones básicas de E/S
- Organización de los registros en archivos
 - Ficheros Montículo
 - Ficheros Ordenados
 - Ficheros de acceso directo (Hash)
 - [Hash Extensible](#)

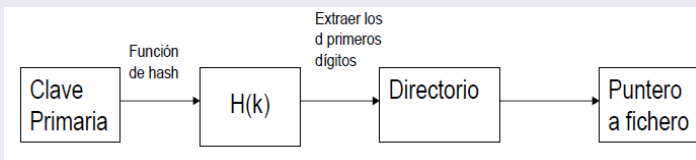
- La principal limitación de los ficheros de acceso directo es **su tamaño es fijo**.
- Si se **cambia el tamaño del fichero**, se debe cambiar la función de hash, y todos los registros antes ubicados según la función de hash antigua deben ser **recolocados** según la nueva función de hash.
- Necesitamos un fichero que **cambie de tamaño dinámicamente**.

Tres propiedades

- El fichero se **expandirá automáticamente** según las necesidades de alojamiento de nuevos registros. La expansión **no requerirá la reorganización del fichero de más de una pequeña parte de los registros.**
- El fichero se **contraerá cuando sea necesario**, de modo que la probabilidad de que el factor de carga del fichero baje del 50 % sea muy baja. Como en el caso de la expansión, la contracción **no requerirá una reorganización del fichero de una porción significativa de registros.**
- La estructura del fichero permitirá la recuperación de un registro por su clave primaria **con un acceso a disco.**

Hash extensible

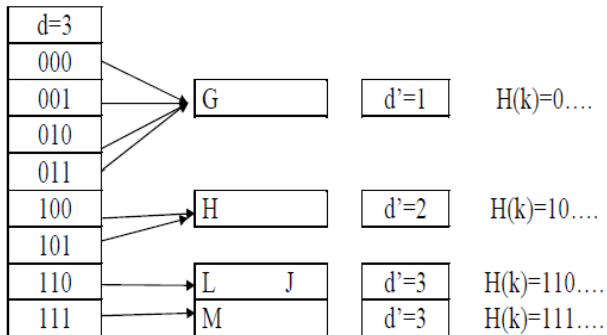
El hash extensible en lugar de hacer una única transformación de la clave, con esta técnica es necesario realizar varias transformaciones.



El puntero a un cubo se lee en una lectura física, es decir, es un bloque físico, y normalmente aloja a varios registros (sinónimos) disminuyendo de este modo las colisiones.

- La primera transformación es similar a la que vimos en los ficheros de acceso directo.
- El espacio de direcciones producido por la función de hash debe ser próximo a una potencia de 2, el rango de la función es ahora algo arbitrario.
- La segunda transformación extrae una porción relativamente pequeña de dígitos de $H(k)$
- Normalmente se utilizan números binarios y se extraen los dígitos de mayor orden, que se usan como un índice dentro de un array unidimensional de punteros.
- Este array se llama **directorio** y contiene 2^d entradas.

Hash extensible

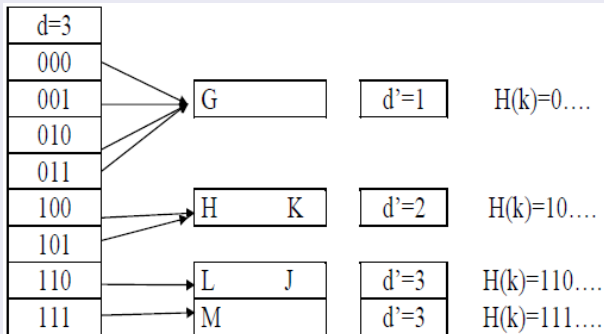


La secuencia completa

- Se le aplica la función de hash sobre la clave, $H(k)$.
- Se extraen los primeros d dígitos de $H(k)$.
- Se utilizan los d dígitos para buscar en el directorio el puntero a cubo apropiado.
- El puntero se utiliza para leer el cubo y traerlo a memoria principal.
- El registro deseado se localiza dentro del cubo.

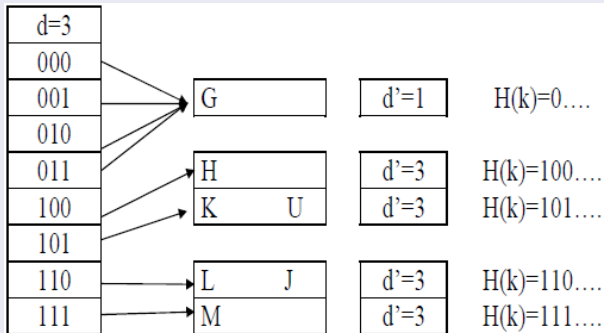
Expansión y contracción directorio

Insertando K donde, $H(K) = 10100111 \dots$



Expansión y contracción directorio

Insertando U donde, $H(U) = 1011001\dots$



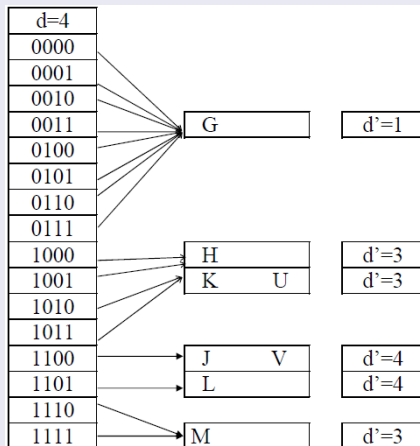
El parámetro d'

- El parámetro d' indica el número de dígitos de $H(K)$ cuyo valor es común a todos los registros del cubo. Este valor siempre debe ser menor o igual a d . El número de punteros que apuntan a un cubo determinado es $2^{(d-d')}$.
- Esta división de cubos, puede continuar según el fichero crezca y mientras existan al menos 2 punteros que apuntan al cubo que se debe dividir.
- Llegados al punto de necesitar insertar un registro en un cubo lleno y con un sólo puntero apuntándole (es decir $d = d'$) el directorio se debe duplicar.

Hash extensible

Expansión y contracción directorio

Insertando V donde, $H(V) = 1100\dots$



Duplicación directorio

- Cada puntero en el directorio original es duplicado y ocupa dos posiciones consecutivas en el nuevo directorio.
- A continuación es posible dividir el cubo origen de la duplicación de modo que un puntero apuntará al cubo que contiene los registros cuya función de hash empieza por ...0 y otro puntero hará lo propio con los que empiezan por ...1.

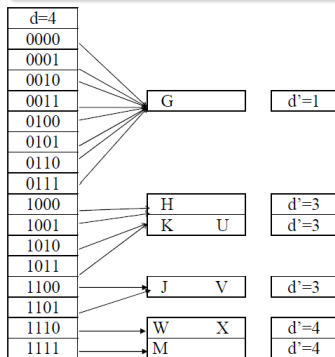
Refundir 2 cubos

Condiciones:

- El factor de carga medio de los dos cubos no puede exceder el 50 %.
- Los cubos a combinar deben tener el mismo valor de d' .
- Los valores de la función de hash aplicada a las claves de los registros en los dos cubos deben ser iguales en los $d' - 1$ primeros bits.

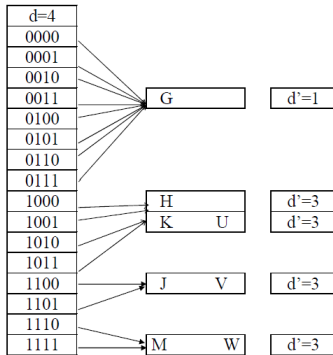
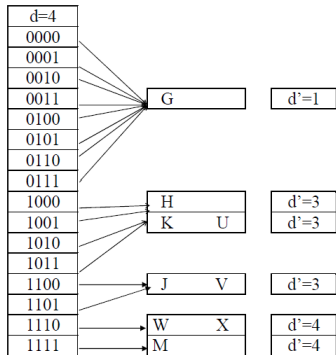
Borrando X

- Ambos tienen un factor de carga del 50 %.
- Ambos tienen el mismo valor de d' .
- Al aplicar la función de hash sobre las claves de los registro en su interior, los $d' - 1$ primeros bits son iguales (111).



Hash extensible: Expansión y contracción directorio

Borrando X



Hash extensible: Expansión y contracción directorio

El directorio se debe contraer siempre que todos los pares de punteros (empezando desde la posición cero del directorio) tengan el mismo valor.

Dicho de otro modo, si todos los cubos tienen un valor de d' menor que d .

