

Tema 7: Organización de Ficheros:

Organizaciones Auxiliares

- **Introducción**
 - Concepto de Índice y Apuntamiento
 - Diseño básico de índices
 - Ventajas e inconvenientes de su aplicación
 - Operaciones y Coste de Procesos sobre Org. Indizadas
- **Taxonomías de Índices: simples e Índices Multinivel**
- **Indización por Árboles B**
- **Estructuras Aux. Especiales: *Cluster, Árbol R, Bitmap***
- **Otros procesos indizados: el *Acceso Invertido***



Las organizaciones base suelen establecerse **entorno a un proceso privilegiado** (o unos pocos procesos).

Serial → **privilegia inserciones**

Secuencial → **privilegia algún acceso ordenado** (y alguna localización)

Direccionada → **privilegia localizaciones a través de una clave**

- El resto de procesos selectivos (clave alternativa) son pesados (*full scan*)
- Si una **cl. alternativa es muy frecuente**, se puede almacenar en un **archivo aparte** la ubicación física de cada valor de esa clave.

Ejemplo: en un libro, la clave *título_capítulo* se almacena asociada a la ubicación del registro (núm. página) en un archivo aparte (**índice**).

Tipo de archivo auxiliar: **índice** (directorío)

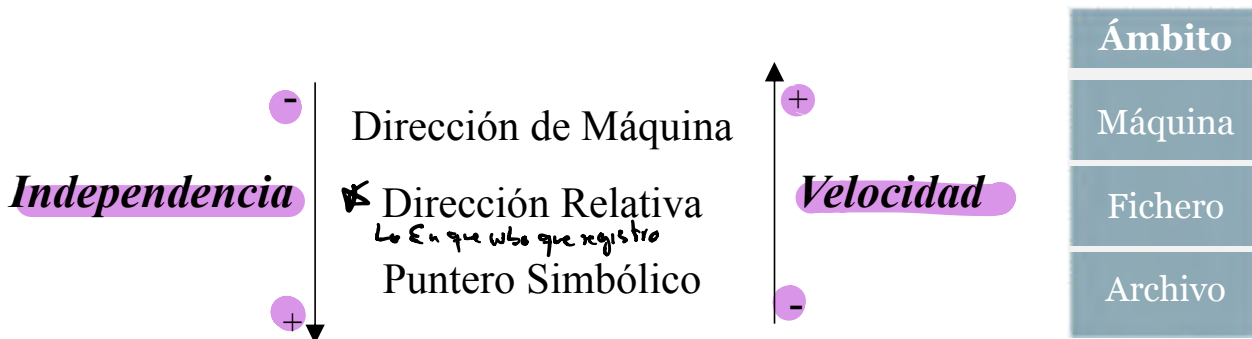
Clave privilegiada: **clave de indización**

Por ser auxiliar, se pueden establecer cuantos índices se estime oportuno



Tipos de Puntero (según su dominio):

- Dirección de Máquina: la dirección física del registro
- ✦ • Dirección Relativa: del registro en el espacio de dircmto. del fichero
- Puntero Simbólico (identificador): identificación lógica del registro
(Puntero Simbólico no identificador: caracterización lógica de un cjto. de registros)





- *Entrada*: registro formado por punteros
- *Directorio*: archivo formado por entradas (y por ende, por punteros)
Algunos usos:
 - virtualizar direccionamientos (ver lect. comp. 6)
 - asociar registros (p.e., personas con sus coches)
 - traducir punteros

ÍNDICE: *directorio cuya entrada se refiere a un solo registro*
- *Es como un listado para traducir punteros (lógicos a relativos)*

- “*Almacenamiento auxiliar utilizado para localizar los registros*”
 - Los índices se almacenan en un fichero (o varios) de índices.
 - Los registros, en el fichero de datos manteniendo su *organización base*
- *La clave siempre es un puntero lógico* (no necesariamente unívoco)
- *El otro puntero suele ser relativo* (dir. del cubo / posición en el cubo)
 - Si el uso del índice es el filtrado (del cjto. dir. relevantes) → sólo parte alta pto.
 - Si la org. base es virtual, se utilizará la dirección virtual



- **PRIMARIO**: la clave de indización es **identificativa**
1 registro \leftrightarrow 1 valor de clave \leftrightarrow 1 entrada (1 puntero)

Asi-Fin	3 ●		Dum-Mos	Los Tres Mosqueteros	Dumas	...
Dum-Con	4 ●		Per-Cap	El Capitán Alatriste	Pérez-Reverte	...
Dum-Mos	1 ●		Asi-Fin	El Fin de la Eternidad	Asimov	...
Per-Cap	2 ●		Dum-Con	El Conde de Montecristo	Dumas	...
...

Asimov	●		Dum-Mos	Los Tres Mosqueteros	Dumas	...
Dumas	● ●		Per-Cap	El Capitán Alatriste	Pérez-Reverte	...
Pérez-Reverte	●		Asi-Fin	El Fin de la Eternidad	Asimov	...
...	●		Dum-Con	El Conde de Montecristo	Dumas	...
...

- **SECUNDARIO**: la clave de indización es **no identificativa**
n registros \leftrightarrow 1 valor de clave \leftrightarrow 1 entrada (n punteros)



• Diseño Físico-Lógico de la entrada de índice Primario:

ENTRADA \equiv clave \cdot puntero externo \rightarrow Apunta a otro fichero.
(longitud)
[tamaño = tamaño clave + tamaño puntero ext]

• Diseño Físico-Lógico de la entrada de índice Secundario:

- Habrá varios registros con el mismo valor de clave...
- Se almacena sólo una vez el valor de cada clave, con todos sus punteros.

ENTRADA \equiv clave \cdot long_lista \cdot (puntero externo)^{long_lista}
La marca de iteración ↳ tantos punteros como registros

• Corolarios:

- El número de entradas es igual a la card. del dominio: $e = \#valores(CI)$
si priviledio sin priv. todas las posibles valores.
- Al buscar, sólo hay que recorrer el índice hasta encontrar una entrada
- Para insertar en listas de punteros hay que buscar la entrada correspondiente y en caso de que no exista (tras recorrer todo el fichero) insertar al final.
 → Con listas, conviene tener el índice ordenado, y siempre no consecutivo
- La longitud media de la lista es la coincidencia de clave $k = r / \#valores(CI)$



- El conjunto de direcciones relevantes puede ser filtrado (reducido) gracias a la aplicación de índices (uno o más) en cl. de búsqueda.
- Un índice primario proporciona un puntero → filtrado máximo
 - si puedo aplicar un primario, aplico ese (y sólo ese)
- Un índice secundario suele filtrar menos (puedo aplicar varios)
 - *Selección multiclave mediante índices secundarios:*
 - se descompone la consulta en subconsultas (un índice por subconsulta); ese índice proporcionará un cjto. resultado (parcial) para esa subconsulta
 - se operan los conjuntos obtenidos (álgebra) para reconstruir la consulta original
 - se obtiene el conjunto de direcciones relevantes final
 - Ejemplos: sean a y b consultas simples; A y B sus respectivos cjtos. resultado
$$a \wedge b \equiv A \cap B$$
$$a \vee b \equiv A \cup B$$
$$a \wedge -b \equiv A - B$$
$$-a \equiv \aleph - A$$
(donde \aleph es el espacio de direcciones global)



Ventajas

1. Acceso por Claves Alternativas

Se gana eficiencia en la localización por claves (hasta ahora) no privilegiadas

2. Aumento de la Tasa de Acierto (en M_{int})

El índice tiene menos cubos y de acceso más recurrente. Con buena gestión de M_{int} , el acierto es muy elevado en los accesos al índice (que son la mayor parte del acceso indizado) haciendo que se dispare la tasa de acierto global.

3. Reorganización Menos Costosa

Ya que los índices tienen menos bloques que el f. de datos, este coste es menor

Inconvenientes

1. Procesos de Actualización Más Costosos

2. Necesidad de Almacenamiento Auxiliar

3. Necesidad de Operaciones de Mantenimiento

~ El + importante

Tema 7.1.4: Operaciones sobre Ficheros Indizados



- Operaciones sobre la totalidad del fichero:
 - **Creación**: hay que crear el índice (al crear el fichero o posteriormente)
 - **Borrado**: si se borra el fichero de datos, hay que borrar el índice
- se puede destruir el índice sin borrar el fichero de datos.
 - **Consulta**: generalmente, interesa hacerla sin contar con el índice
(excepción: proceso ordenado, con índice ordenado y f. de datos no sec.)
- Operaciones selectivas (sobre registros aislados):
 - **Localización** (por CI): se hará a través del índice
 - **Operaciones de Actualización**: pueden afectar al índice
(es necesario actualizarlo)

Tema 7.1.4: Coste de Procesos Ficheros Indizados



- **Localización a través del Índice:** acceso al índice (según su naturaleza)
- **Localización por varios índices:** suma del acceso a cada índice

- **Recuperación:**

$$C(O_i, P_j) = \text{acceso_índice} + \text{acceso_datos}$$

$$\text{acceso_datos} = \# \text{cubos} \cdot E_c$$

- **Actualización:**

- Inserciones: suelen requerir inserción de entradas

$$\Delta \text{coste} = \text{acc_índice} + 1$$

- Borrados: pueden localizarse con el índice

$$\Delta \text{coste} = \text{acc_índice} + 1$$

- índice primario: suelen requerir borrado de entradas

- índice secundario: pueden requerir modificación de entradas

- Modificaciones: pueden localizarse con el índice

- CI: suele implicar borrado + reinserción de entrada

$$\Delta \text{coste} = 2 \cdot \text{acc_índice} + 2$$

- CD/CO: cambia ubicación reg. → cambia puntero

$$\Delta \text{coste} = \text{acc_índice} + 1$$



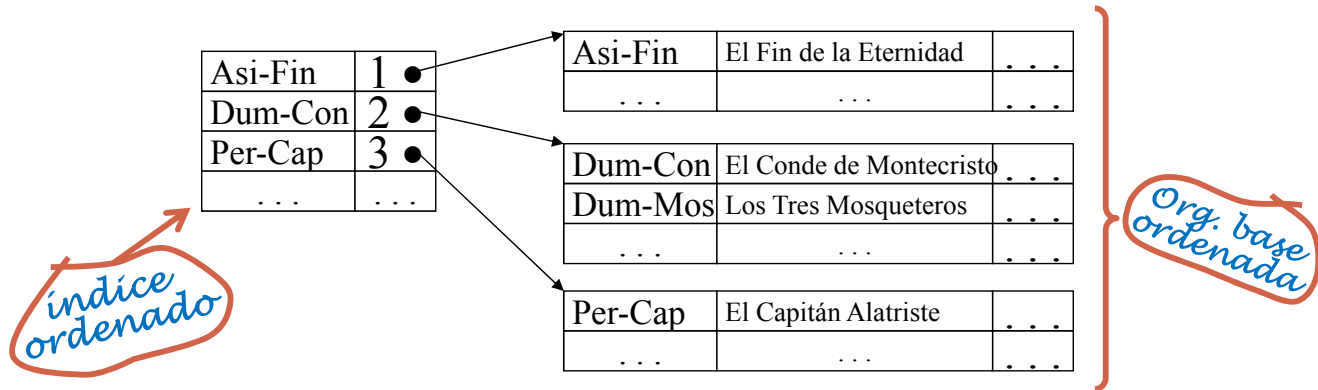
- **Naturaleza:** serial, secuencial, o direccionado
- **Coste:** dependiendo de su naturaleza (igual que un fichero de datos análogo)
- **Restricciones:** se debe aplicar sobre claves no privilegiadas

Ejemplos:

- si el índice se usa para un proceso ordenado por CI (y además $CI \neq CO$)
- si la CO se usa para un proceso ordenado y la CI para procesos selectivos
- si el índice se usa para un proceso especial (ver acceso invertido, tema 7.5)

Mantenimiento:

- si es ordenado o disperso, puede desbordar → requiere reorganización
- debe evitarse la degeneración de la estructura
 - índice ordenado: preferible inserción ordenada + reorganización local
 - índice disperso: pierde eficiencia si cambia (si es volátil)
es más útil como índice temporal (procesos puntuales)

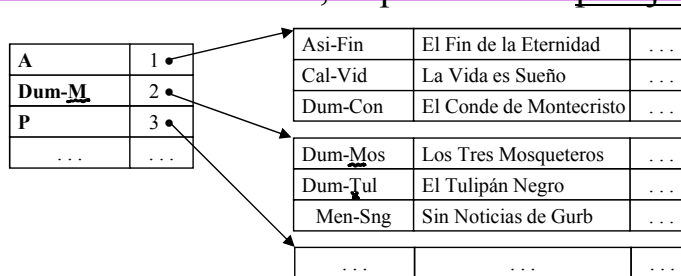


- **Concepto:** una entrada por cubo de datos (en lugar de una entrada por registro)
- **Restricción:** índice y organización base deben ser necesariamente secuenciales y con *clave_indización = clave_ordenación (CO=CI)*
- **Usos:** aporta varias posibilidades de acceso:
 - procesos ordenados (a la totalidad): acceso serial de la org. base (ordenada)
 - procesos selectivos (solución única): a través del índice
 - mixtos (selección de un rango): acc. indizado (1^{er} elemento) + serial



Ventajas:

- al ser de tamaño muy reducido, tiene menor coste (y mayor tasa de acierto)
- se ahorran muchas actualizaciones de índice (insertar/borrar/modificar registros a menudo no afectan al índice, salvo que sean el primer registro del cubo).
- en lugar de utilizar toda la *clave* en la entrada, se pueden usar prefijos (%tamaño)



Con el mínimo tamaño para reconocer.

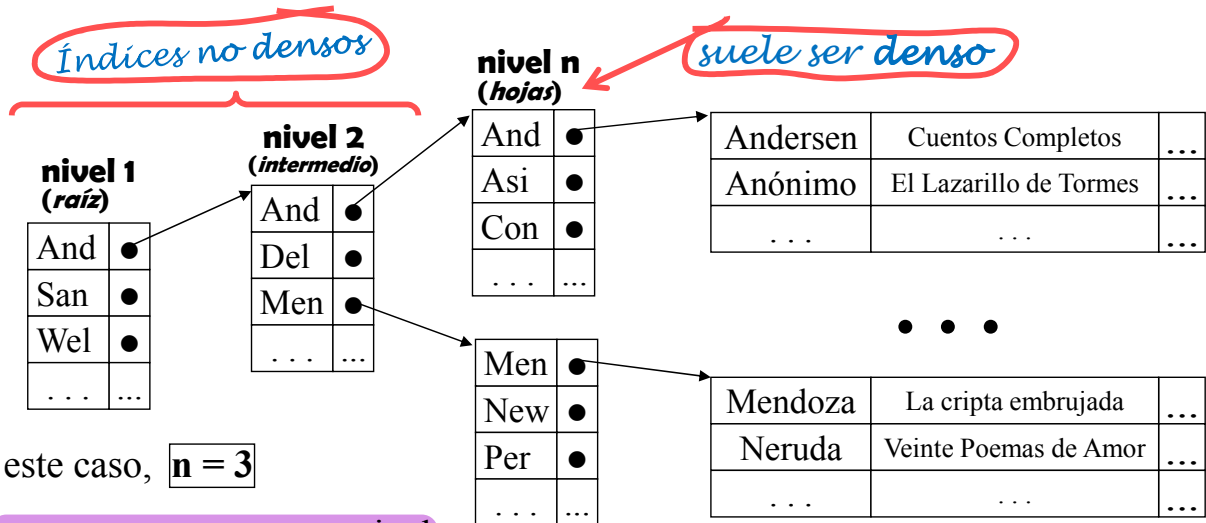
Inconvenientes:

- sólo puede existir un índice no denso para cada archivo
- la inserción del registro debe ser ordenada, pero se localiza con el índice.
- La inserción de la entrada es ordenada, y conlleva pesadas reorganizaciones
 - deben aplicarse mecanismos de ELD, rotación, partición celular, etc.



Es un árbol de índices, un índice de un índice.

Concepto: es un índice con n niveles (el nivel n es índice del nivel $n+1$)



- En este caso, $n = 3$
- El coste es un acceso por nivel
Interesa definir **nodos pequeños** (1 bloque) incluso a costa de tener más niveles
- Es ventajoso **bloquear la raíz** (nivel 1) en memoria intermedia (ahorra un acceso)
- El nivel n suele ser denso, pero como es secuencial puede ser un índice no denso



- El índice multinivel perfectamente construido es eficiente, pero degenera
- En ficheros constantes es buena solución
- En ff. volátiles se requiere reorganización local → evolución a otras estructuras

Árboles Binarios: cada nodo es una entrada del índice con dos punteros internos
Solución sencilla, pero presenta problemas de vecindad y desequilibrio.

- **Árbol AVL:** resuelve desequilibrio mediante procesos de reorganización local
- **Árboles Binarios Paginados:** resuelven el problema de vecindad *Almacena sus dos hijos, entran 2 niveles*
- **Árboles AVL-Paginados:** buen rendimiento, pero necesitan muchos punteros internos, presentan *bajísima densidad*, y reorganizaciones frecuentes.

útil ⇒

Solución:

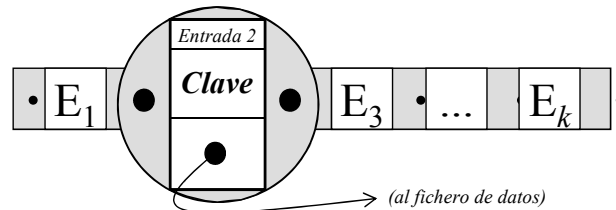
- Incluir varias entradas por nodo
(y, por tanto, varios descendientes)
- Construir el árbol en orden ascendente
(el separador pertenece al nodo que desborda)

} **árboles B** ← el mejor.



- *Propuestos por Bayer y McCreight*

- **Idea:** ya que inicialmente no se conoce el elemento que es mejor separador, se comienza por las hojas. A medida que crezca, se construye hacia arriba.
- **Nodo:** cada nodo va a llenar la página; contiene entradas de índice (pares *clave indización-puntero* a los datos) y punteros (para apuntar nodos hijo)



- **Orden del árbol:** indica la capacidad de los nodos (y por ende, del árbol)

- según las entradas: el n° mínimo de claves de un nodo (Bayer)
- según los punteros (hijos): n° máximo de hijos de un nodo (Knuth)

Mejor la que os am



Árboles B : Observaciones

- Si el árbol es de orden m , cualquier nodo tendrá a lo sumo m descendientes
- Si un nodo tiene m descendientes (no hoja), tendrá $m-1$ entradas
- Corolario: un nodo de un árbol de orden m tiene a lo sumo $k=m-1$ entradas
- En un nodo (T_{nodo} bytes) caben m punteros internos y k entradas, luego:

$$m \cdot T_{\text{ptro_interno}} + k \cdot T_{\text{entrada}} \leq T_{\text{nodo}}$$

A mayor tamaño + niveles. → ya que k se compensa y m se reduce.

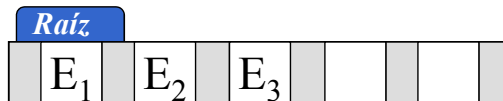
(con esta fórmula y $k=m-1$ se podrá hallar el orden del árbol)

- El nodo raíz tiene al menos un elemento y, por tanto, al menos 2 hijos.
- T_{nodo} es múltiplo de T_{bloque} y suele ser lo menor posible (típicamente 1 bq)
- T_{entrada} es la suma del tamaño real de la clave (fija/marcada/codificada...) más el/los puntero/s interno/s (pueden ser muchos punteros, si es secundario)



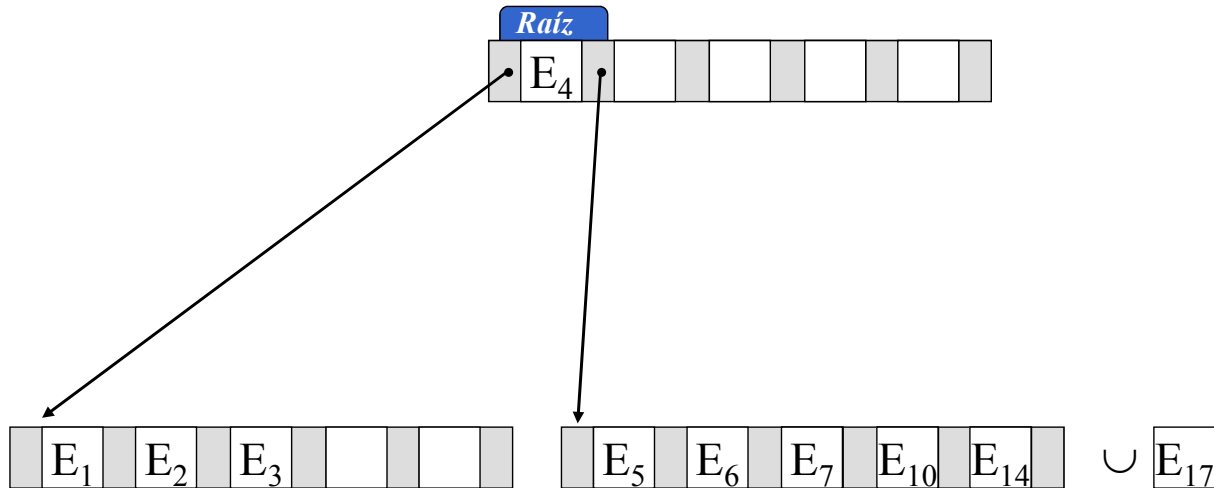
Árboles B : Partición y Promoción

- Las entradas dentro de un nodo van ordenadas
- Cuando un nodo *desborda*, se divide en dos y se *promociona* el elemento intermedio hacia el nivel superior (ese elemento se lleva dos punteros: uno hacia cada hijo, es decir, hacia cada uno de esos dos nuevos nodos)





Árboles B : Más Partición y Promoción (otro ejemplo)





Árboles B : Propiedades

- Todos los nodos menos el raíz garantizan una **ocupación mínima**:

$$k_{\min} = \left\lfloor \frac{k}{2} \right\rfloor$$

entradas

Corolario

primario: $r = e$
 — = entradas

secundario: $e = \# \text{ clave}$
 Dem de clave ind. raíz.

- ¿Cuántos descendientes como mínimo tienen los nodos intermedios?
 (suponiendo política de 'dividir cuando desborda')

$$m_{\min} = k_{\min} + 1 \rightarrow m_{\min} = \left\lfloor \frac{m+1}{2} \right\rfloor$$

primario

- Tamaño del fichero de índices

Se puede obtener una **cota superior** del fichero de índices

$$N_{\max}^{\circ} \text{ nodos fichero} = \frac{n^{\circ} \text{ entradas fichero}}{k_{\min}} = 1 + (n^{\circ} \text{ ent. fich}) / k_{\min}$$

$$T_{\max} \text{ fichero} = n_{\max}^{\circ} \text{ nodos fichero} \cdot T_{\text{nodo}}$$

Esta vale.



Árboles B : Propiedades (II)

El nº de niveles (n) para un árbol de orden m y e entradas tiene cota superior
Cuántas entradas tiene cada nivel del árbol. El peor caso

nivel	nodos	entradas	acumulado
1	1 ↳ hijos 2	1	1
2	2	$2 \cdot k_{\min}$	$1 + 2k_{\min}$
3	$m_{\min} \cdot 2$	$2 \cdot m_{\min} \cdot k_{\min}$	$1 + 2k_{\min} + 2m_{\min}k_{\min}$
n	$2 m_{\min}^{n-2}$	$2 m_{\min}^{n-2} \cdot k_{\min}$	$(2 m_{\min}^{n-1}) - 1$
$n+1$	$2 m_{\min}^{n-1}$	$2 m_{\min}^{n-1} \cdot k_{\min}$	$(2 m_{\min}^n) - 1$

 $\leq e$
 $\leq e$
 $\leq e$
 $> e$ Nivel imposible

Cada nodo m_{\min} hijos.

Cota sup.

$$n \leq 1 + \log_{\lfloor \frac{m+1}{2} \rfloor} \left(\frac{e+1}{2} \right)$$



- Para **recuperar una entrada**: $\#accesos = \#niveles$
 - Dado que la raíz estará siempre en memoria, contamos un acceso menos
- Para **recuperar un registro aleatorio**, se recuperan la entrada y tantos cubos de datos como punteros tenga la entrada (es decir, k cubos)

$$C(O_i, P_j) = (n-1) \cdot T_{\text{nodo}} + c \cdot E_c$$

- El **coste de cualquier actualización** sobre el índice en árbol B es el coste de **localización más un acceso** de escritura: $\Delta C(\text{actualización}) = (n-1) + 1 = n$
- El **coste extra de una partición** es de **dos accesos** de escritura (actualizar el nodo antecesor y escribir el nodo nuevo). *No la computamos.*
- El tiempo de acceso así calculado es una *cota superior al tiempo de acceso*.
- Puede **calcularse la cota inferior** (en base al número de niveles del árbol perfectamente construido), para **conocer su coste óptimo y valorar el beneficio** de ejecutar la reorganización del índice.



Árboles B : Valoración

↳ Es bueno para índices primarios, para el secundario escaso.

Aspectos Positivos:

- Existe una cota superior razonable del número de accesos a soporte
- Generalmente, las operaciones (de inserción o borrado) requieren reestructurar una página. Y si son más, suelen ser pocas páginas.
- En el peor caso, las ^{local}páginas están ocupadas a la mitad (aproximadamente)

Aspectos a Mejorar

no fue necesario.

- Si las entradas son grandes, el orden puede ser demasiado pequeño
- La densidad (mínima) de los nodos es muy mejorable
- En las hojas se desperdicia mucho espacio (no necesitan punteros)



Idea: se pretende **aumentar la densidad** de los nodos

Para ello, en lugar de dividir un nodo en dos, se dividirán dos nodos en tres.

Así, en lugar de conseguir una ocupación mínima del 50% se obtendrá el 66%
 $\frac{1}{2}$ $\frac{2}{3}$

- Cuando un nodo desborda, en lugar de dividir, se procurará ceder uno de sus elementos a su vecino (**rotación**). *Cambiando el discriminante por el desbordado.*
- Si el nodo vecino también está lleno, se parte (dos nodos llenos en tres nodos)
- Por lo demás, **el resto del funcionamiento es como el de los árboles B.**

• **Ventajas:**

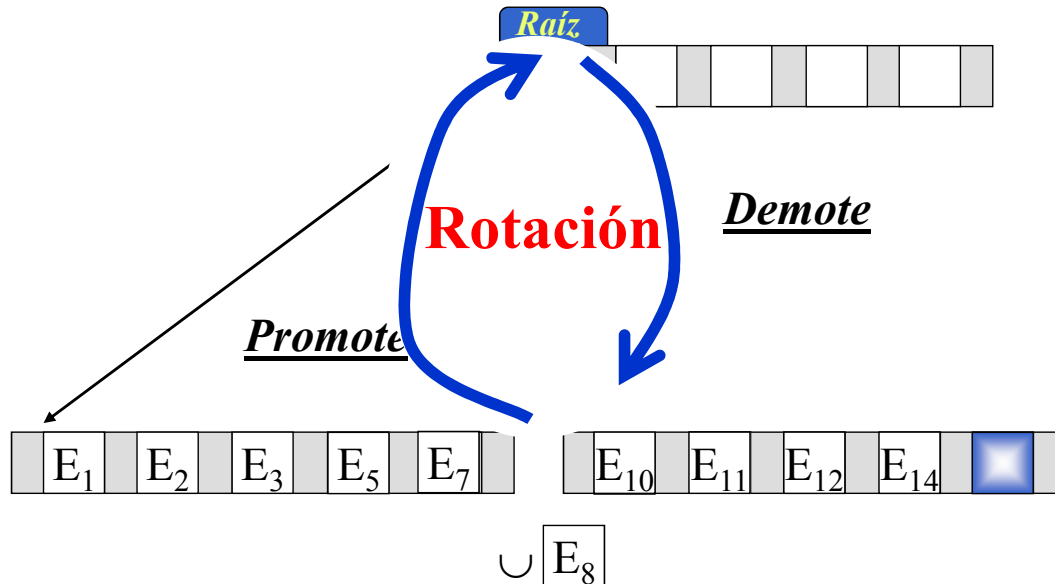
- *Aumento de la densidad (al 66%)*
- *Un desbordamiento no siempre supone partición/promoción*

• **Desventajas:**

- *Aumenta la probabilidad de desbordamiento (nodos más llenos)*



Rotación, Partición y Promoción (ejemplo)





Propiedades

- Todos los nodos menos el raíz garantizan una ocupación mínima:

$$k_{\min} = \left\lfloor \frac{2k}{3} \right\rfloor$$

- Los nodos intermedios cuentan con $\frac{2k}{3} + 1$ descendientes \rightarrow
(suponiendo política de 'dividir cuando desborda')

$$m_{\min} = \left\lfloor \frac{2m+1}{3} \right\rfloor$$

Cálculo de Costes

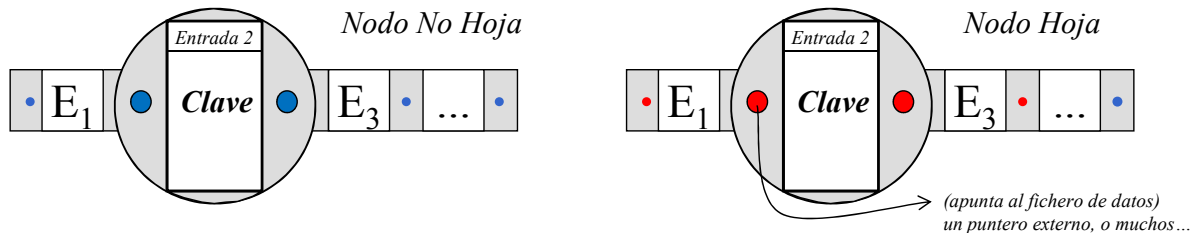
- Como la localización es idéntica al árbol B, también es igual el cálculo de costes
- El coste extra de una rotación es de tres accesos (lectura del nodo contiguo, más la escritura de ese nodo y del nodo antecesor). $\Delta C(\text{rotación}) = 3 \text{ acc}$
- La partición implica cuatro accesos extra (la lectura del nodo contiguo, más la escritura de los nodos contiguo, nuevo, y el antecesor). $\Delta C(\text{partición}) = 4 \text{ acc}$
- También se puede contemplar la rotación bidireccional. En este caso la densidad es 75% ($k_{\min} = \left\lfloor 3k/4 \right\rfloor$) pero también el coste de inserción (rotación 4, y partición 5).

Saber
como
teoria, pero
no para
ejercicios.



Idea: coste proporcional a la profundidad → crecer en amplitud
→ aumentar el #hijos por nodo → **aumentar el orden**

- En los **nodos con hijos** (nodos no hoja) se **suprimen los punteros externos** (así caben más discriminantes, y por ende se tienen **más punteros internos**).
- En los **nodos hoja** no hay punteros a nodo hijo, pero sí habrá punteros externos. Para **apuntar a los datos**, la entrada debe estar en una hoja → todas las entradas están en nodos hojas, y en los no hoja sólo hay copias discriminantes

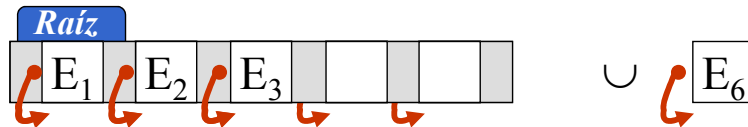


- Especialmente eficiente con punteros externos grandes → **índice secundario**

Tema 7.3.3: Indización en Árbol B+ Partición y Promoción



Ejemplo de Partición y Promoción: nodo hoja/raíz (*primera partición*)

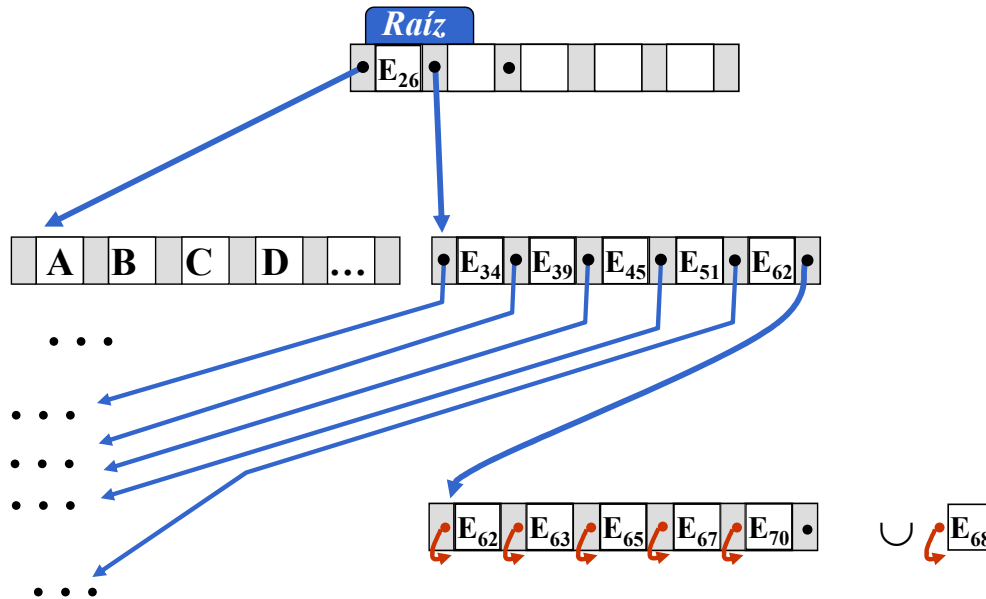


- Observar que en las hojas se usa un puntero interno adicional (puntero encadenamiento) para apuntar al siguiente nodo hoja (hermano). Esto se realiza durante la partición:
 - el `ptro_encadenamiento(nodo_nuevo) := ptro_encadenamiento(nodo_viejo)`
 - el `ptro_encadenamiento(nodo_viejo) := dirección(nodo_nuevo)`
- El *encadenamiento de hojas* proporciona un mecanismo de acceso alternativo.

Tema 7.3.3: Indización en Árbol B+ Partición y Promoción



- En nodos no hoja, la promoción es igual que en nodos de árbol B
- Ejemplo:**



Tema 7.3.3: Indización en Árbol B+

Propiedades (k_{mín} y m_{mín})



- Orden del árbol (m): se calcula para nodos no hoja, como en árboles B; teniendo en cuenta que las entradas esos nodos carecen de puntero externo

$$m \cdot T_{\text{puntero_interno}} + (m-1) \cdot T_{\text{clave}} \leq T_{\text{nodo}}$$

- Ocupación máxima (k) de los nodos hoja: si los tamaños de los punteros interno y externo son distintos, convendría calcularla por separado

$$k \cdot (T_{\text{clave}} + T_{\text{puntero(s)_externo}}) + T_{\text{puntero_interno}} \leq T_{\text{nodo}}$$

debe contener
al menos una
entrada comp.

encadmt. bidireccional req. dos punteros internos

- La ocupación mínima de las hojas será:
(suponiendo política de 'dividir cuando desborda')

$$k_{\text{mín}} = \left\lfloor \frac{k+1}{2} \right\rfloor$$

- La ocupación mínima de los nodos intermedios será $\lfloor k/2 \rfloor \rightarrow$
- (la promoción en estos se opera como en los nodos de un árbol B)

$$m_{\text{mín}} = \left\lfloor \frac{m+1}{2} \right\rfloor$$

Tema 7.3.3: Indización en Árbol B+ Profundidad y Tamaño



Cálculo del número de niveles:

- El nivel de las hojas es el **nivel n** . ¿Cuántas hojas?

$$n^{\circ} \text{ hojas} = \lfloor e / k_{\min} \rfloor$$

$e = n^{\circ} \text{ total de entradas}$

- El n° de nodos en el **nivel $n-1$** depende del número de nodos del nivel n

$$n^{\circ} \text{ nodos } (n-1) = \lfloor n^{\circ} \text{ nodos } (n) / m_{\min} \rfloor$$

- Cuando se llega a un nivel con un solo nodo (la raíz), este será el **nivel 1**.
(se tiene que el nivel $n-x=1$, y se puede despejar $n = \text{profundidad del árbol}$)

- Tamaño máximo del fichero índice:**

se calcula como la suma de los nodos necesarios

para cada nivel $(\sum_{i=1}^n \text{nodos}(i))$ multiplicado por el tamaño de un nodo.



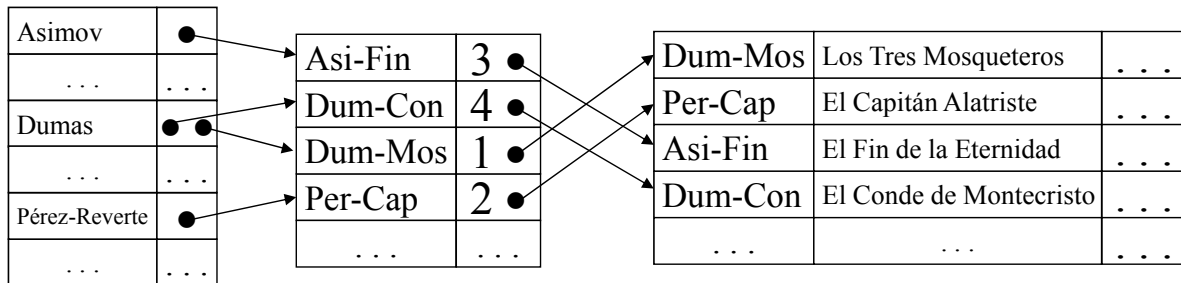
Consideraciones finales

- El encadenamiento de hojas proporciona mecanismos de acceso alternativo.
- Ejemplos: (índice en árbol B⁺ con clave indización '*fecha* (dd-MM-AAAA)')
 - procesos a la totalidad ordenados
Ejemplo: sacar un listado de todos los registros ordenados cronológicamente
 - procesos selectivos con tasa de actividad elevada
Ejemplo: recuperar todos los registros con fecha en mes de 'Mayo'
 - procesos ordenados con varios resultados (*rangos*) → acceso mixto
Ejemplo: recuperar todos los registros entre el 01-05-2005 y el 30-06-2005
 - Los accesos mixtos consisten en recuperar la primera entrada (01-05-2005) a través del árbol, y el resto de entradas se recuperarán con el encadenamiento
- Se puede **organizar un fichero de datos en árbol**
→ es como tener un f. secuencial con part. celular y un índice B⁺ no denso.
- Las mejoras logradas con árboles B⁺ y B* son combinables



Índice Intermedio

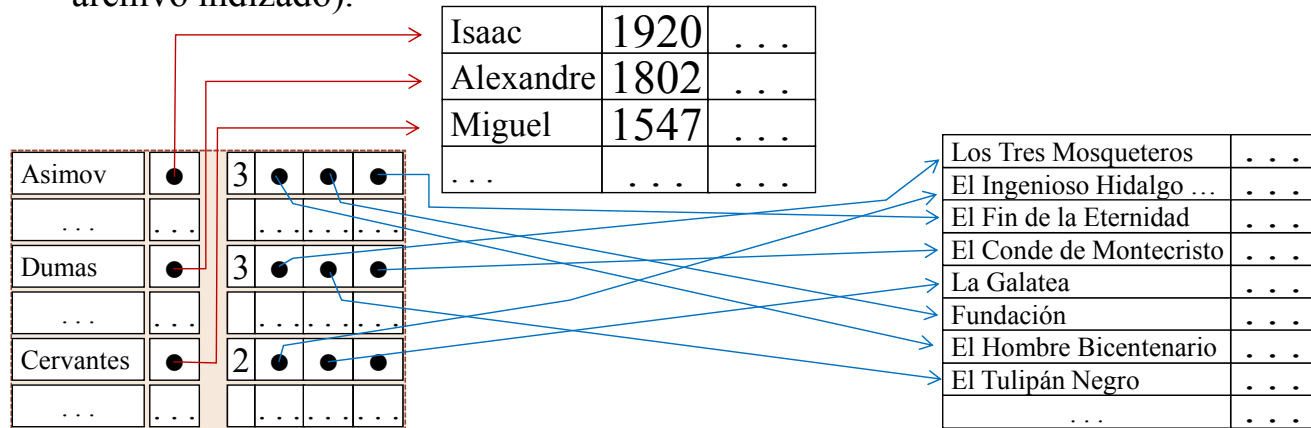
- Si los registros (en el f. de datos) cambian de ubicación, es necesario actualizar todos los índices de ese archivo.
- ^{→ Denso, primario y exhaustivo}
Índice Intermedio: índice primario cuyos punteros apuntan a los datos, y el resto de los índices apuntan a este. Al cambiar los registros de ubicación, sólo es necesario actualizar punteros en este índice.
- Este índice debe ser muy eficiente: bloqueado en memoria privilegiada de tamaño reducido, y (casi) constante (poco o nada volátil)





Índice Agrupado o Cluster

- Dos (o más) índices sobre distintos archivos con la misma CI y valores validados (por integridad referencial) pueden combinarse (*index join*).
- También puede crearse una estructura única de indexación (índ. *cluster*).
- La entrada tendrá una clave de indización y uno o más esquemas de punteros (un puntero o una lista de punteros, según sea la naturaleza de la clave en cada archivo indizado).





Esquemas de bits (BITMAP)

Para dominios con cardinalidad pequeña,

Un *esquema de bits* para un campo es un vector de valores booleanos. A cada **valor** del dominio se le hace corresponder una **posición**.

Ejemplo: *idioma* (castellano, inglés, francés, alemán, italiano)

- Si la cardinalidad (#valores) es alta, puede indicarse un subconjunto (índice parcial).
- Admite la multivaluación (varios valores para la misma entrada)
- Puede ser simple o multiclave, concatenando esquemas de varios campos

18,11
puntero

0001000000	0010	10100	10
departamento	categoría	idioma	sexo

- Puede utilizarse como *directorio de ocurrencia*
- **Proceso serial** (excepción: acceso invertido sobre bitmap)



Máscaras sobre BITMAP

- **Máscaras para condiciones de igualdad:**

- Se realizan con un bit para cada posible valor, en el conjunto $\{0, 1\}$

- La selección comprueba la condición

$$S \text{ AND } Q = Q$$

- Ejemplo: empleadas del dpto. informática que sólo sepan castellano

$$Q = 0001000000 \ 0001 \ 10000 \ 10$$

- **Máscaras con bits que admitan cualquier valor:**

- Se realizan con un bit para cada posible valor, en el conjunto $\{0, 1, q\}$

- La selección comprueba en lógica trivaluada la condición

$$S \text{ XOR } Q = 1$$

- Ejemplo: miembros del departamento de informática que sólo sepan inglés

$$Q = qqqlqqqqqq \ qqqq \ 01000 \ qq$$



Esq. de bits Simples vs. Multiclave

- Es conveniente que el diseño de los esquemas de bits se realice **atendiendo a las necesidades de procesamiento**.
- Ejemplo** para claves A (5 bits) y B (10 bits) en un fichero de 1000 registros, con puntero de 3 bytes, y tamaño de bloque 2KB:

Tamaño índices:	$T(A)=1000 \cdot (1+3) \rightarrow 2 \text{ bloques}$	Solución 1
	$T(B)=1000 \cdot (2+3) \rightarrow 3 \text{ bloques}$	
	$T(A+B)=1000 \cdot (2+3) \rightarrow 3 \text{ bloques}$	Solución 2

- ➔ caso i) Frecuencias relativas de uso: $f(A)=0.7$; $f(B)=0.2$; $f(A+B)=0.1$
- coste medio (solución 1) = $0.7 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot (2+3) = \mathbf{2.5 \text{ accesos}}$
 - coste medio (solución 2) = $0.7 \cdot 3 + 0.2 \cdot 3 + 0.1 \cdot 3 = \mathbf{3 \text{ accesos}}$
- ➔ caso ii) Frecuencias relativas de uso: $f(A)=0.1$; $f(B)=0.4$; $f(A+B)=0.5$
- coste medio (solución 1) = $0.1 \cdot 2 + 0.4 \cdot 3 + 0.5 \cdot (2+3) = \mathbf{3.9 \text{ accesos}}$
 - coste medio (solución 2) = $0.1 \cdot 3 + 0.4 \cdot 3 + 0.5 \cdot 3 = \mathbf{3 \text{ accesos}}$



- El acceso invertido es un tipo de acceso indizado multiclave orientado a optimizar el coste de acceso en procesos muy concretos
- Se trata de procesos en los que se pretende averiguar información delimitada de ciertos archivos con condiciones muy concretas.
- Del tipo: *¿Cuál es el valor del campo X en el archivo Y, para los registros cuyo campo Z vale 'valor'?*
dos o más claves (pero no muchas)
- El acceso invertido procurará [↙] averiguar toda esta información accediendo sólo a los índices (sin acceder al archivo de datos)
- Sus punteros relativos deben localizar unívocamente cada registro:
→ punteros con parte alta (*cubo*) y parte baja (*posición en el bloque*)



- Se ejecutarán primero las condiciones (para obtener conjunto resultado)
- Después se busca en los 'índices objetivo' (incógnitas), *pero al revés!* *→ Mirando los ptros. de la entrada.*
a partir de cada dirección (ptro. relativo) buscamos el valor (ptro. lógico)
- Ejemplo: *¿Cuántos tengo muchos punteros, si no no merece la pena el full scan del índice.*
¿Cuáles son los títulos de los libros de cualquier autor cuyo apellido sea 'Dumas'?

Índice Apellidos

...	...
Asimov	(1,3)(15,2)
Dumas	(2,1)(5,7)
Neruda	(10,5)
Pérez-Reverte	(4,1)
...	...
...	...

Índice Títulos

...	...
(2,1)	El Conde de Montecristo
...	...
(5,7)	Los Tres Mosqueteros
...	Los Tres Ositos
...	...

```
SELECT titulo
FROM autor
WHERE apellido='Dumas';
```

```
titulo
-----
El conde de Montecristo
Los Tres Mosqueteros

2 rows selected.
```



- El acceso invertido es eficiente si requiere acceder a pocos índices (si accediera a varios no contenidos en M_{int} , podría costar más que acceder a los datos).
- Los índices que soportan este acceso (puntero ext. doble precisión) se denominan **índices invertidos**. Los secundarios también se denominan **listas invertidas**.
- En este tipo de acceso, los índices bitmap son eficientes en dominios reducidos, mientras que las listas invertidas lo son en dominios de cardinalidad elevada.
- Un **fichero invertido** es el que soporta este tipo de acceso (dos o más índices invertidos). Un **fichero totalmente invertido** tiene todos los campos invertidos.
- En un fichero totalmente invertido el área de datos es redundante (prescindible).
- Sin embargo, se suele mantener para (a) evitar el alto coste de la recomposición de registros; y (b) mantener doble almacenamiento base (**sistema dual**) con **redundancia controlada en el tiempo**: uno de los almacenes (en este caso, el invertido) se actualiza eventualmente (cuando el sistema está ocioso), aumentando eficiencia de actualizaciones con poco perjuicio en las consultas.



- El coste del acceso invertido es la suma de los costes de sus dos partes:
 - Selección: Obtención de los punteros B o B^+ tanto como no doi. B^+ tantos como hojas
 - Listas invertidas no ordenadas: coste máx. n ; medio $(n+1)/2$
 - Listas invertidas ordenadas: $\log_2(n+1)$
 - Esquemas de bits: n
 - Otro tipo de índice: el coste correspondiente a esa estructura
 - Proyección: obtención de Claves correspondientes a los punteros
 - Esquemas de bits con puntero implícito: $\min(n, r)$
 - Cualquier otro caso: n
- Simbología: n es el número de bloques del índice; r es el número de resultados
- Si hubiera varios índices implicados en la condición o en la proyección, el coste en cada parte sería la suma de los costes individuales de cada índice implicado.