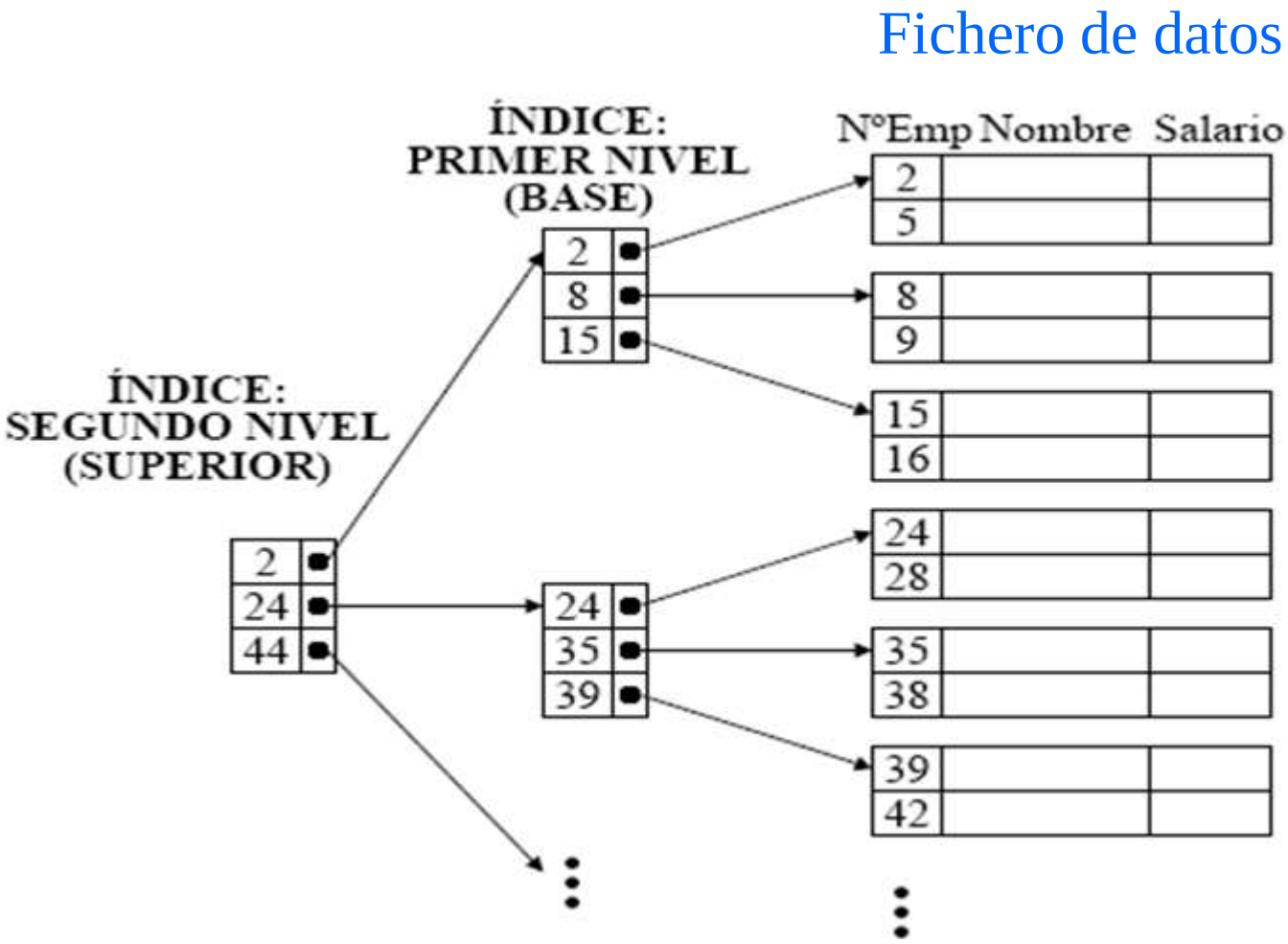
Índice

5.2 Árboles (índices multinivel)



Índice Primario de dos niveles

5.2 Árboles (índices multinivel)

- Hasta ahora:
  - Fichero de índices ordenados
  - Búsqueda binaria al índice para localizar punteros (a bloques o registros) que tienen un cierto valor en campo de indexación: $log_2\ b_i$ accesos a bloque para un índice con b_i bloques.
- *Índices Multinivel* reducir la búsqueda binaria en el índice (*fbl_i, factor de bloque del índice):*
  - *Si $fbl_i >> 2$ → reducción rápida del espacio de búsqueda*
  - *fbl_i* se denomina *fan-out*, ***fo***.
  - Búsqueda en un índice multinivel requiere aprox. $log_{fo}b_i$ accesos a bloques:

Comparación en términos de *bi* (un nivel). Cálculo de número de niveles (t)/accesos: $\log_{fo}r1$ (r1: núm. entradas nivel 1).

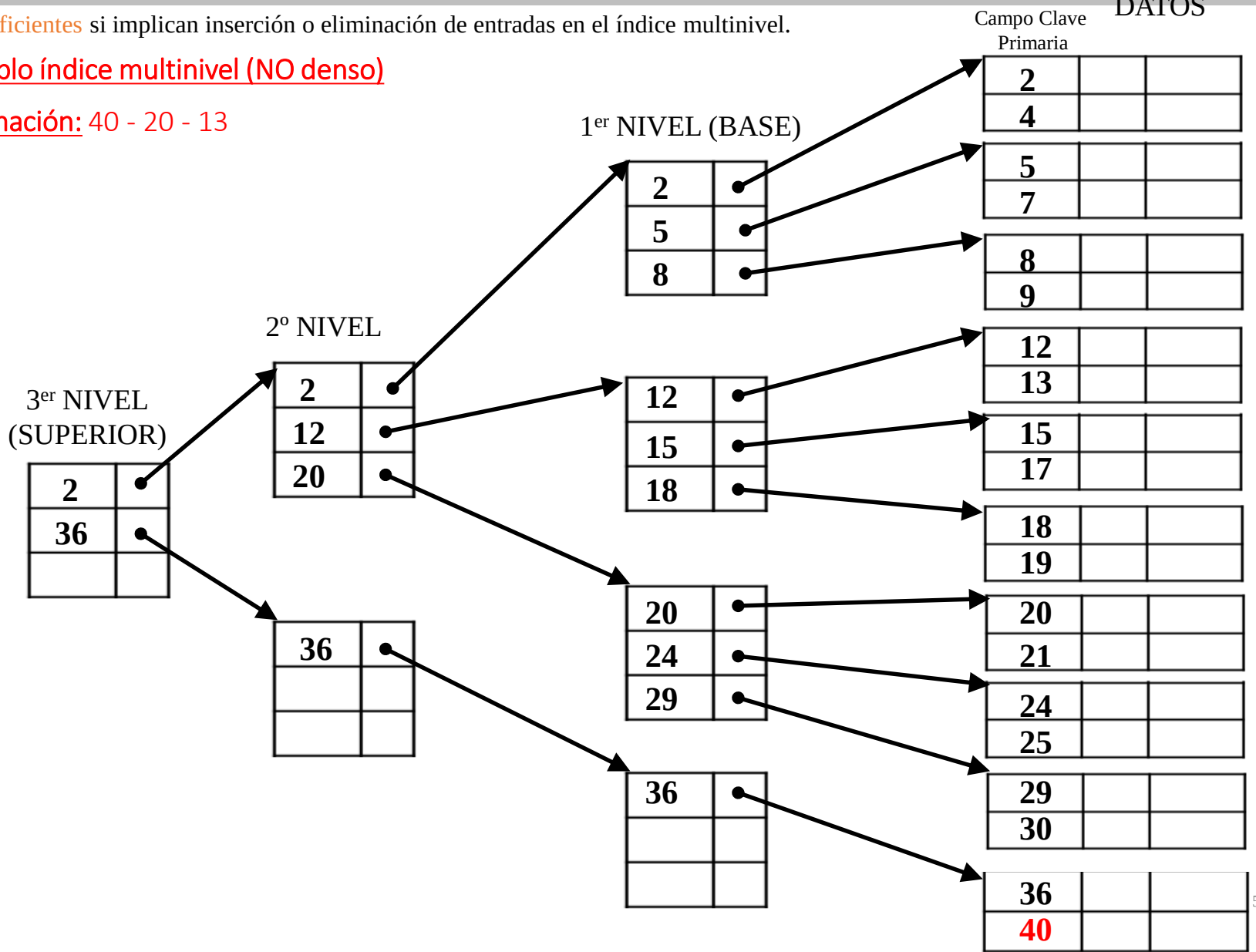
- *Índices Multinivel* considera
  - el fichero del índice (denotado como *primer nivel o nivel base*) como un *fichero ordenado con un valor <> para cada K(i).*
  - Podemos crear un *Índice Primario* para este primer nivel, llamado *segundo nivel*. Podemos repetir el proceso.
- Todos los niveles son ficheros ordenados luego esquemas válidos para IP, IA e IS siempre que en primer nivel tenga valores <> para cada K(i) y entradas de long. fija.
- Ineficientes si implican inserción o eliminación de entradas en el índice multinivel.
- Solución: índices **dinámicos** de múltiples niveles (árboles B^+, B, B^*)

5.2 Árboles (índices multinivel)

- Ineficientes si implican inserción o eliminación de entradas en el índice multinivel.

Ejemplo índice multinivel (NO denso)

Eliminación: 40 - 20 - 13

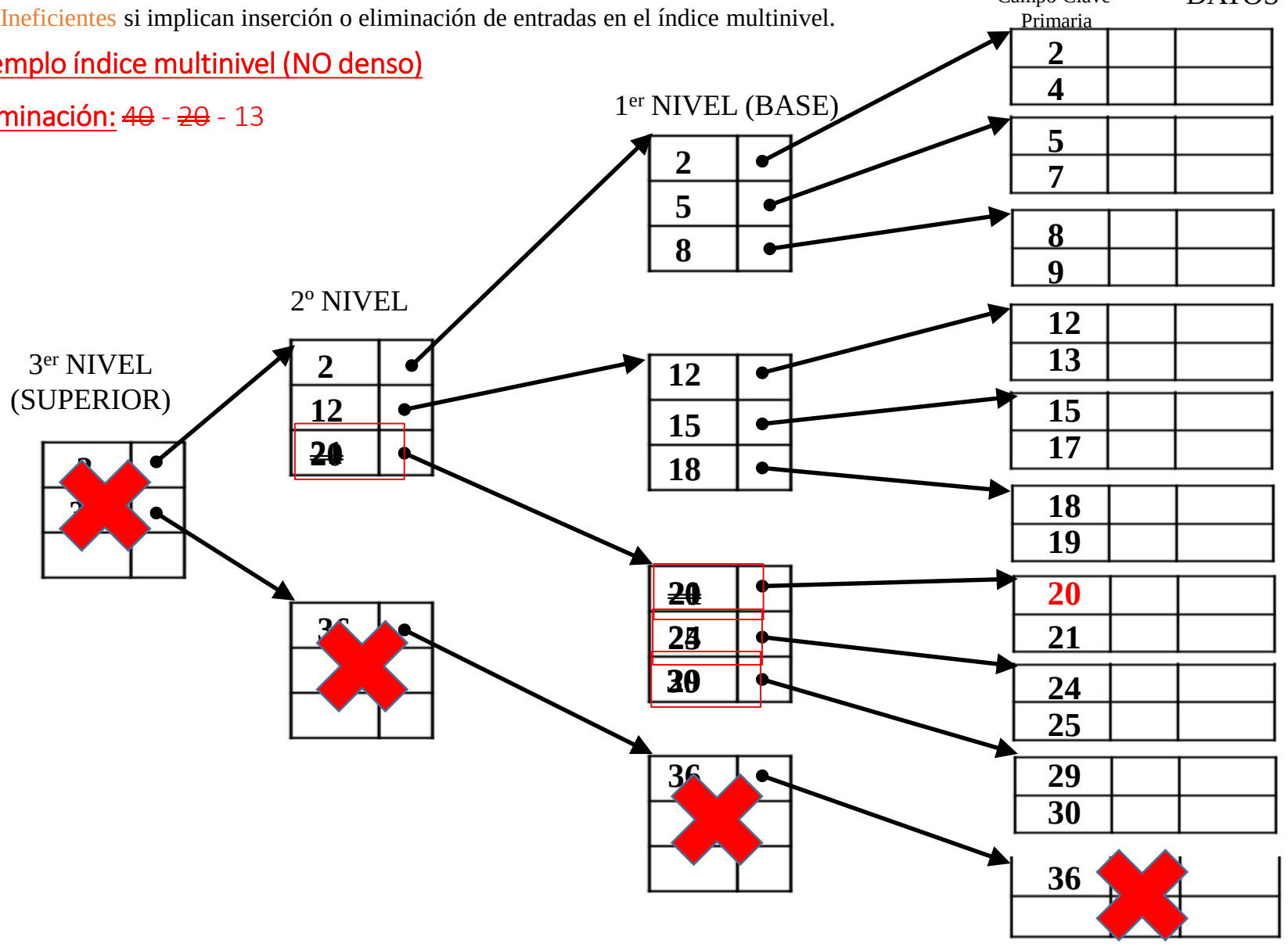


5.2 Árboles (índices multinivel)

- Ineficientes si implican inserción o eliminación de entradas en el índice multinivel.

Ejemplo índice multinivel (NO denso)

Eliminación: ~~40~~ - ~~20~~ - 13

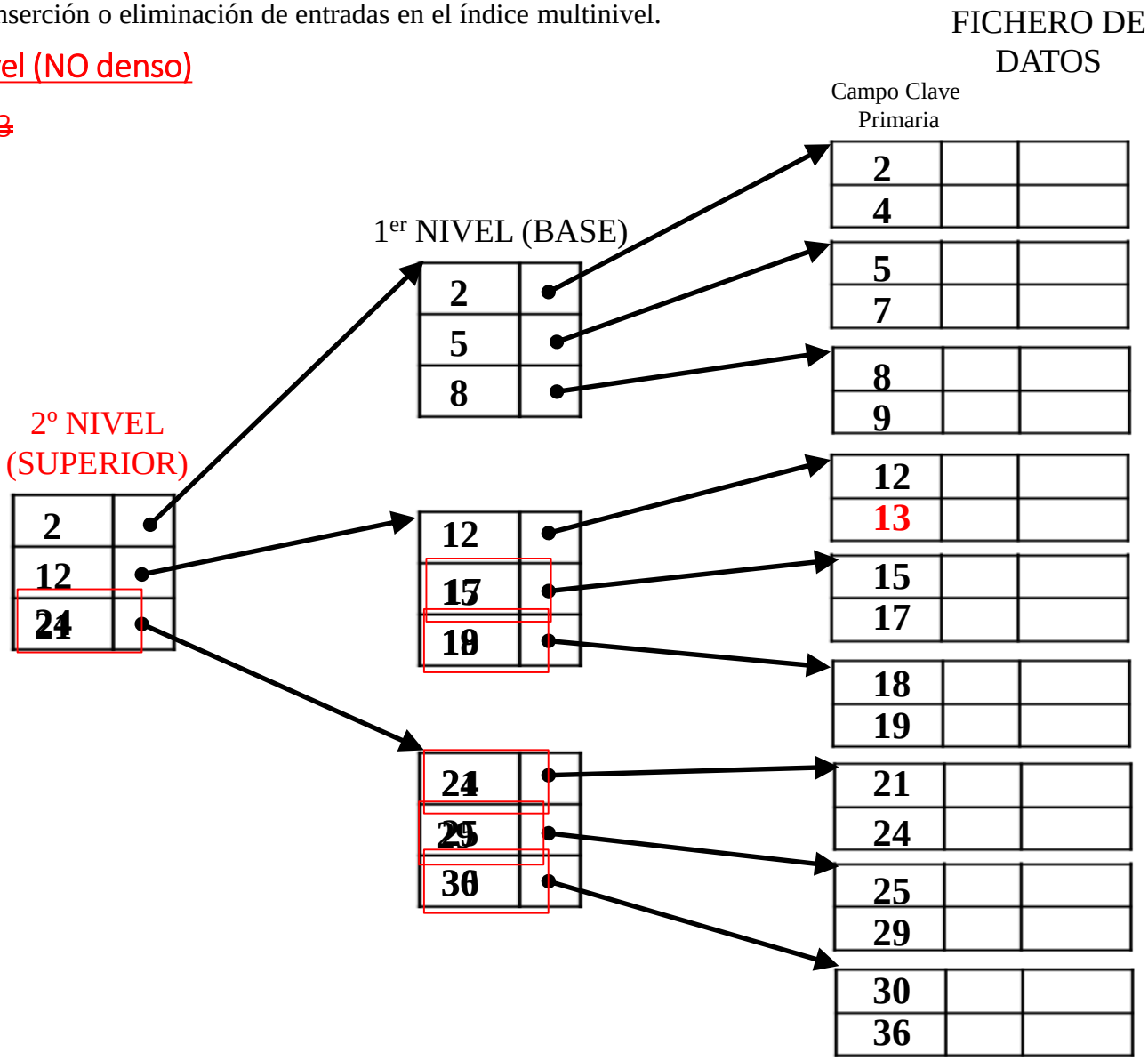


5.2 Árboles (índices multinivel)

- Ineficientes si implican inserción o eliminación de entradas en el índice multinivel.

Ejemplo índice multinivel (NO denso)

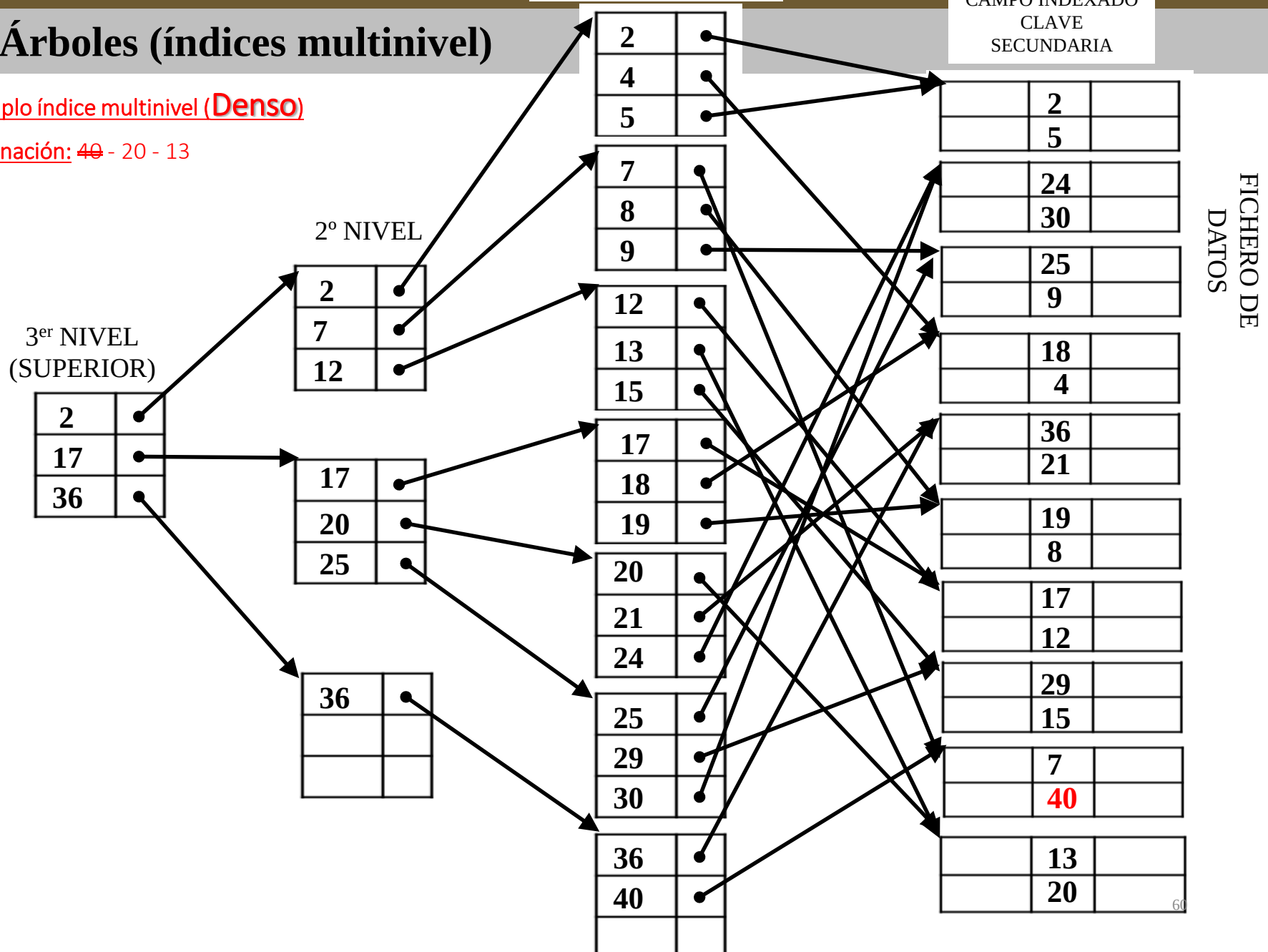
Eliminación: ~~40~~ - ~~20~~ - ~~13~~



5.2 Árboles (índices multinivel)

Ejemplo índice multinivel (Denso)

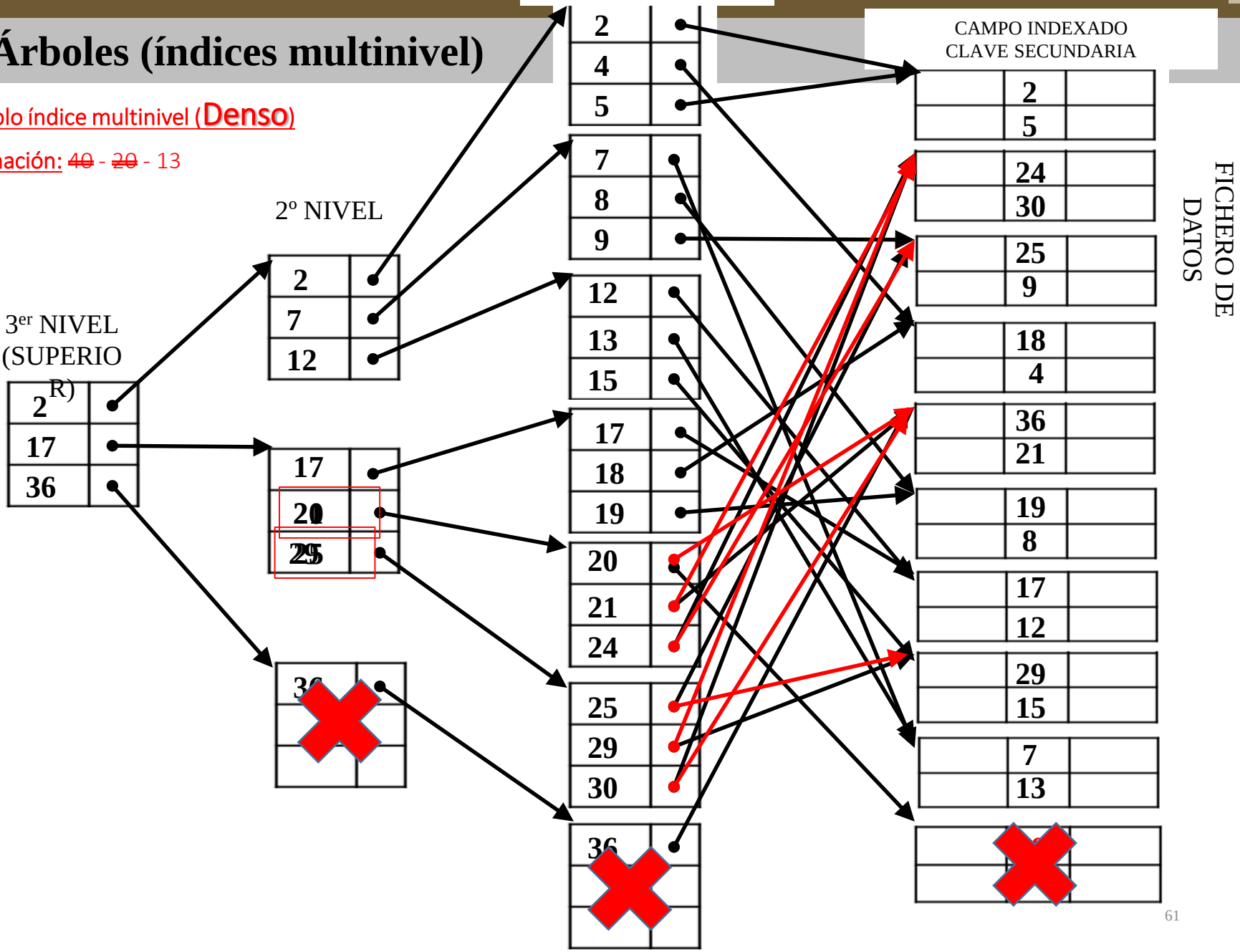
Eliminación: ~~40~~ - 20 - 13



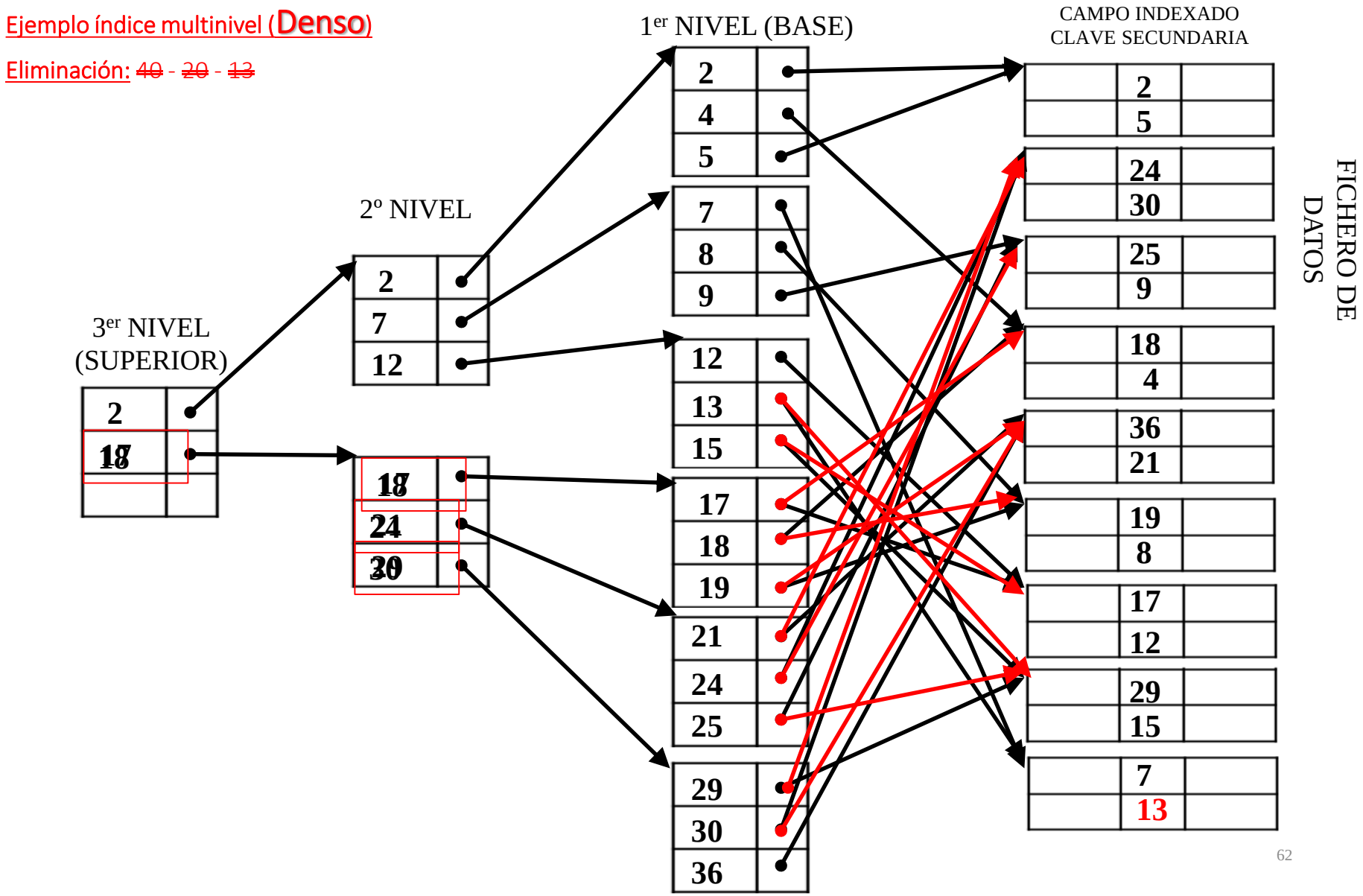
5.2 Árboles (índices multinivel)

Ejemplo índice multinivel (Denso)

Eliminación: ~~40~~ - ~~20~~ - 13



5.2 Árboles (índices multinivel)



5.2.1 Árboles (índices multinivel): Árbol B^+

- Solución a los problemas de inserción y eliminación.

- Los árboles B, B^+ y B* son casos especiales de estructuras de datos en forma de árbol.
- Cuando un índice multinivel se implementa como una estructura de árbol, esta información incluye los valores del campo de indexación del fichero que se utiliza para guiar la búsqueda de un determinado registro.
- Nos vamos a centrar en los **Árboles B^+**

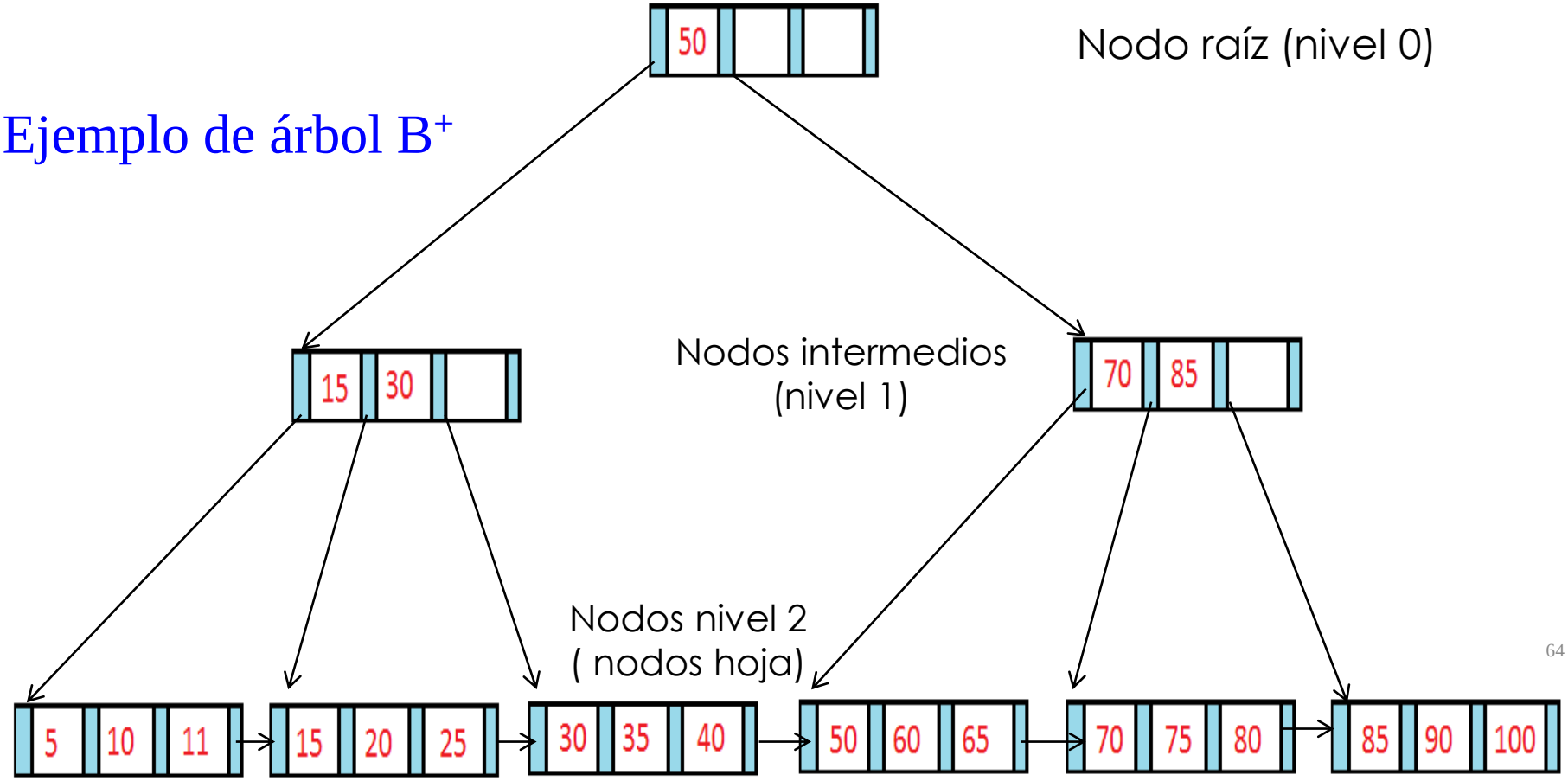
ESTRUCTURA de **Árbol B^+**:

- Es un árbol equilibrado (todos los nodos hoja están al mismo nivel)
- Longitud de los caminos de la raíz a las hojas es el mismo.
- Sea ***n*** el **orden** del árbol: número de punteros de cada nodo del árbol:
  - Cada nodo intermedio (no hoja), tiene entre $\lceil n/2 \rceil$ y ***n*** nodos hijo;
  - Cada nodo hoja puede tener entre $\lceil (n-1)/2 \rceil$ ***y n-1*** valores del campo de búsqueda.
  - El nodo raíz (si es interno) tendrá al menos **2** nodos hijo.

5.2.1 Árboles (índices multinivel): ESTRUCTURA Árbol B^+

- Los rangos de los valores del campo clave no se solapan en las hojas
- Sean L_i y L_j nodos hoja con i<j, entonces cada valor de la clave de búsqueda en L_i es menor que cada valor de clave en L_j.
- Si el índice del árbol B^+ es denso, cada valor de la clave de búsqueda debe aparecer en algún nodo hoja

Ejemplo de árbol B^+

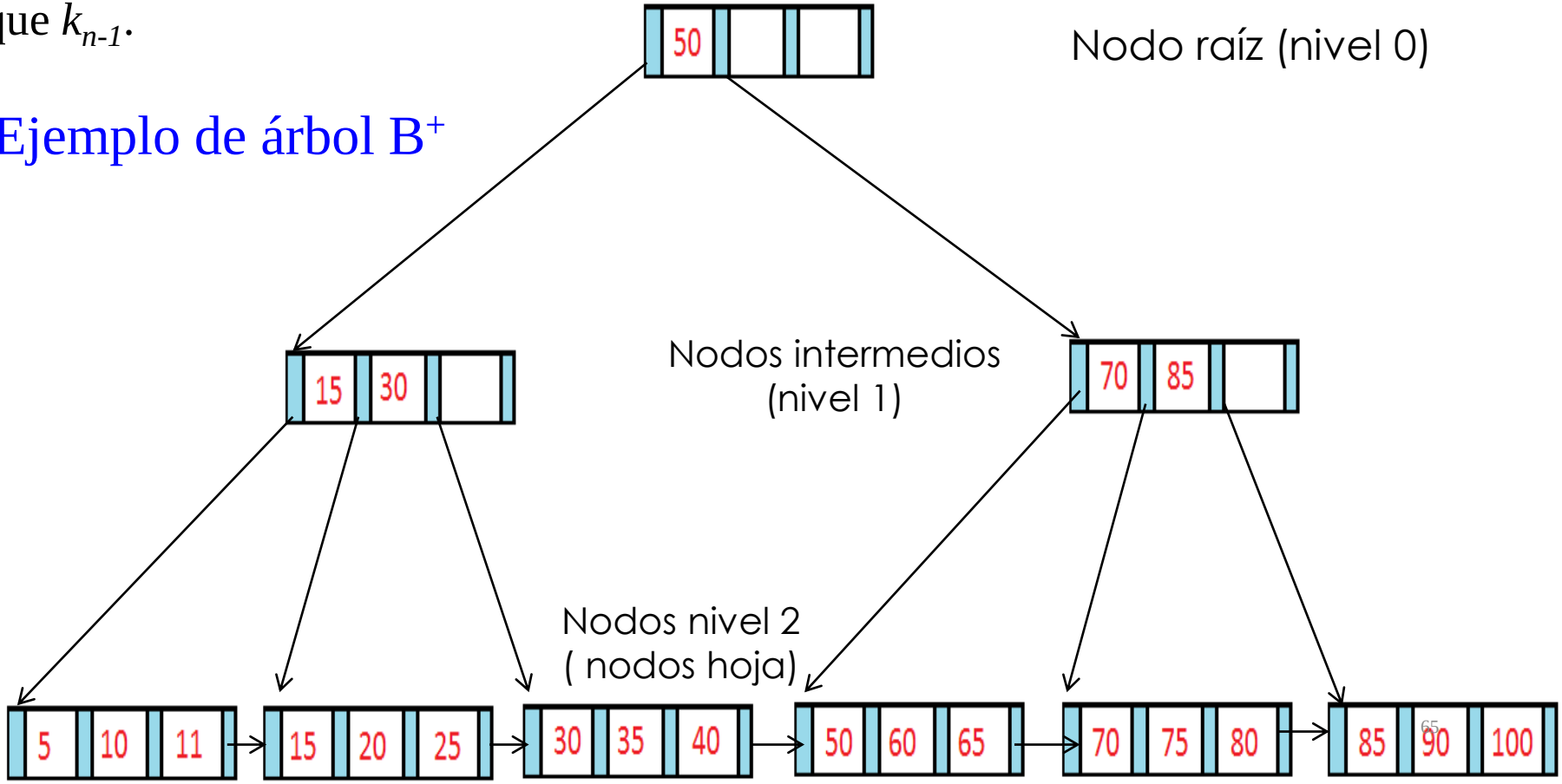


5.2.1 Árboles (índices multinivel): Árbol B^+

Consideremos un nodo con ***n*** punteros:

- P_1 apunta al subárbol que contiene los valores de clave de búsqueda menor que k_1.
- Para ***i=2,3,…,n-1*** → P_i apunta al subárbol que contiene los valores de clave de búsqueda mayor o igual que k_{i-1} y menores que k_i
- P_n apunta al subárbol que contiene los valores de clave de búsqueda mayor o igual que k_{n-1}.

Ejemplo de árbol B^+

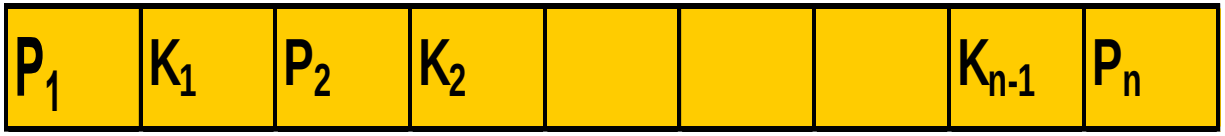


5.2.1 Árboles (índices multinivel): ESTRUCTURA Árbol B^+

- **¡Nodo hoja! de un árbol B^+ de nivel n**

  Hasta n-1 claves de búsqueda K_1, K_2..... K_{n-1}

  n punteros $P_1, P_2 \ldots\ldots\ldots\ldots P_n$ (El puntero P_n se utiliza para encadenar los nodos hoja en el orden de la clave de búsqueda)



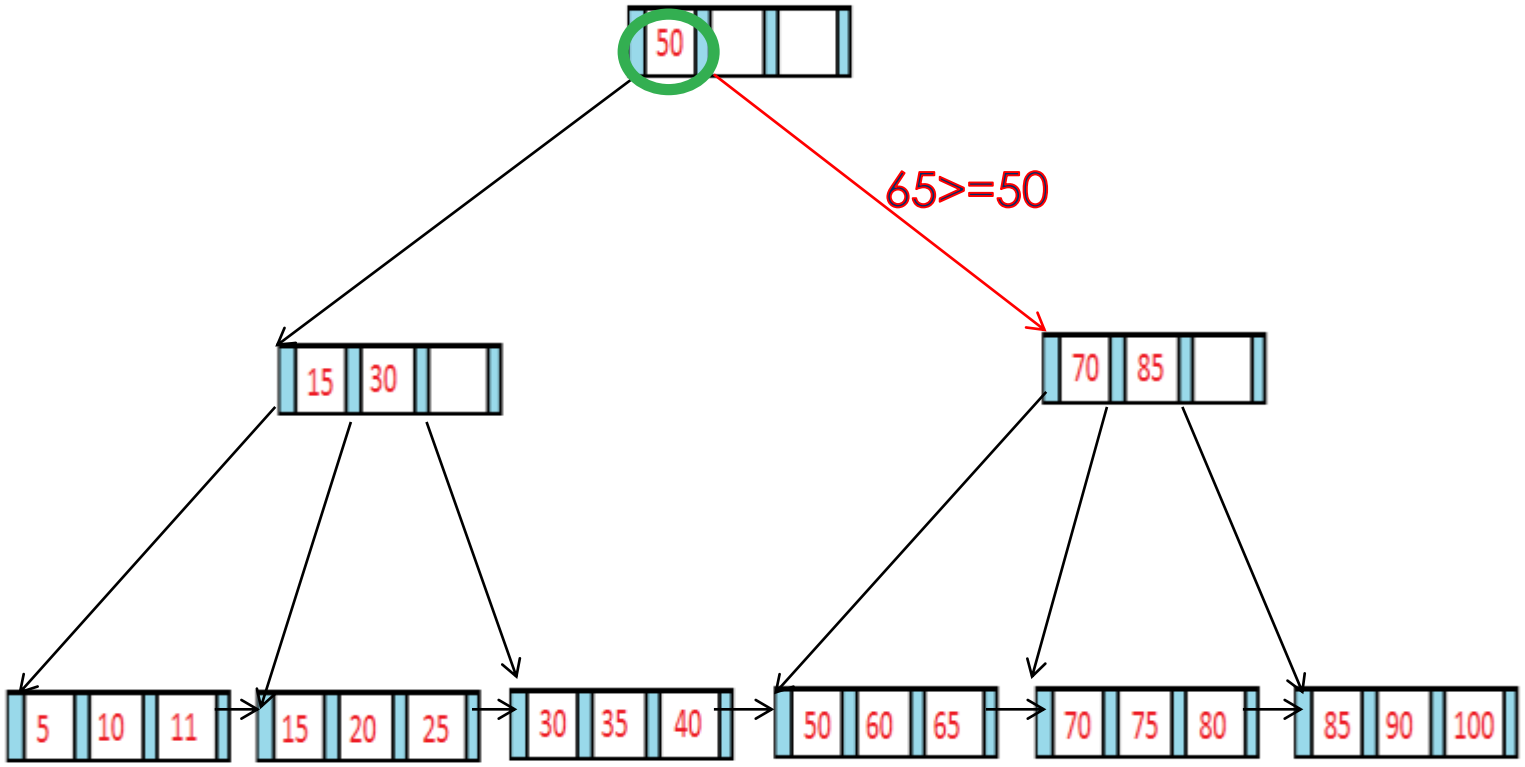
- Si el campo de búsqueda es un campo clave: el puntero P_i apunta a un registro (o al bloque que contiene el registro) del fichero de datos con valor de clave K_i
- Si el campo de búsqueda NO es un campo clave: P_i apunta a un bloque de punteros cada uno de los cuales apunta a un registro del archivo con valor de clave K_i (creándose un nivel extra de indirección).

Importante: En un árbol B^+ los punteros a datos solo se almacenan en los nodos hoja del árbol (por ello los nodos internos tienen estructura diferente de los nodos hoja).

5.2.1 Árboles (índices multinivel): Árbol B+

Ejemplo de búsqueda del valor 65 en el árbol B+

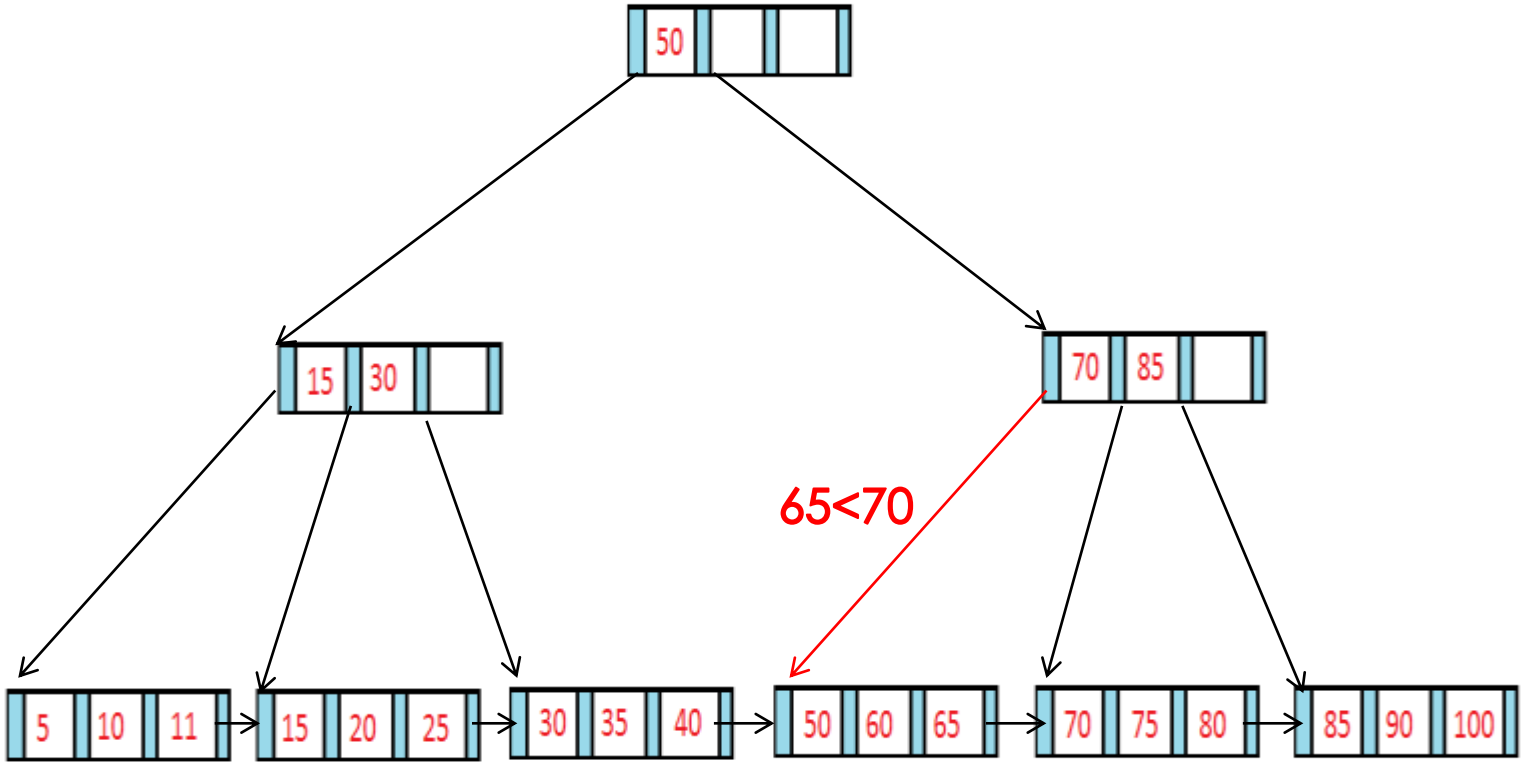
Nos fijamos en el nodo raíz, si el número es mayor o igual al de la raíz, vamos por el camino de la derecha, en caso contrario por la izquierda.



5.2.1 Árboles (índices multinivel): Árbol B^+

Ejemplo de búsqueda del valor 65 en el árbol B^+

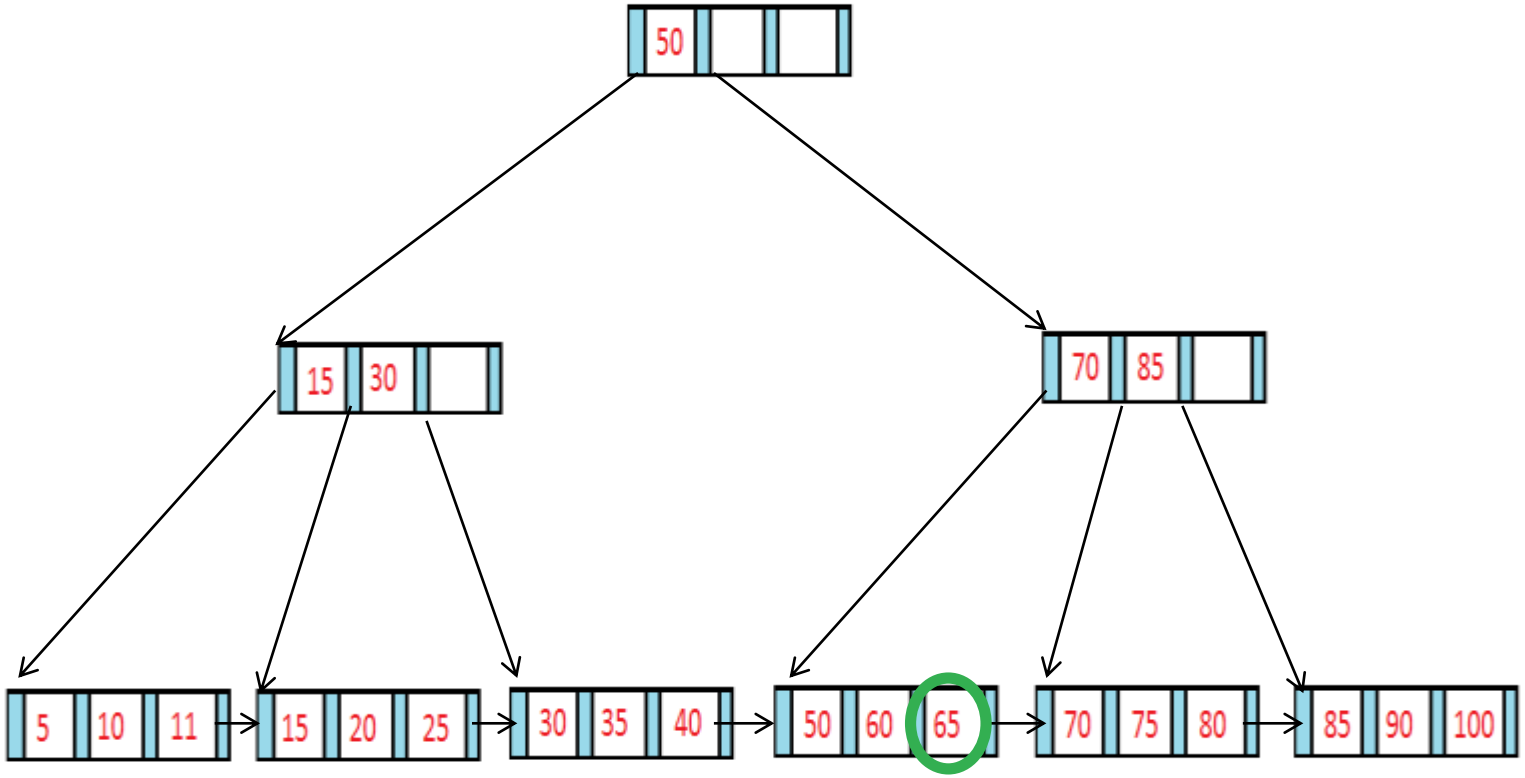
En los nodos intermedios observamos entre que números estaría el nuestro y elegimos un camino siguiendo el criterio anterior.



5.2.1 Árboles (índices multinivel): Árbol B^+

Ejemplo de búsqueda del valor 65 en el árbol B^+

Por último, una vez hemos llegado a la hoja, buscamos nuestro número deseado.



5.2.1 Árboles (índices multinivel): Árbol B^+

Inserción árbol B^+

- Cuando queremos insertar un nuevo elemento en un bloque de un árbol B+, debe de hacerse de manera ordenada.
- El problema se produce cuando el elemento no entra; en este caso se produce un desbordamiento y hay que seguir ciertos pasos para dejar el árbol bien estructurado.

5.2.1 Árboles (índices multinivel): Árbol B^+

Algorithm 17.3. Inserting a Record with Search Key Field Value K in a B^+-Tree of Order p

```
n ← block containing root node of B+-tree;
read block n; set stack S to empty;
while (n is not a leaf node of the B+-tree) do
    begin
    push address of n on stack S;
        (*stack S holds parent nodes that are needed in case of split*)
    q ← number of tree pointers in node n;
    if K ≤ n.K1 (*n.Ki refers to the ith search field value in node n*)
        then n ← n.P1 (*n.Pi refers to the ith tree pointer in node n*)
        else if K ← n.Kq–1
            then n ← n.Pq
            else begin
                search node n for an entry i such that n.Ki–1 < K ≤ n.Ki;
                n ← n.Pi
                end;
        read block n
    end;
search block n for entry (Ki,Pri) with K = Ki; (*search leaf node n*)
if found
    then record already in file; cannot insert
    else (*insert entry in B+-tree to point to record*)
        begin
        create entry (K, Pr) where Pr points to the new record;
        if leaf node n is not full
            then insert entry (K, Pr) in correct position in leaf node n
            else begin (*leaf node n is full with pleaf record pointers; is split*)
                copy n to temp (*temp is an oversize leaf node to hold extra entries*);
                insert entry (K, Pr) in temp in correct position;
                (*temp now holds pleaf + 1 entries of the form (Ki, Pri)*)
                new ← a new empty leaf node for the tree; new.Pnext ← n.Pnext ;
                j ← ⌈(pleaf + 1)/2⌉;
                n ← first j entries in temp (up to entry (Kj, Prj)); n.Pnext ← new;
                new ← remaining entries in temp; K ← Kj ;
                (*now we must move (K, new) and insert in parent internal node;
                  however, if parent is full, split may propagate*)
                finished ← false;
                repeat
                if stack S is empty
                    then (←no parent node; new root node is created for the tree*)
                        begin
                        root ← a new empty internal node for the tree;
                        root ← <n, K, new>; finished ← true;
                        end
                    else begin
                        n ← pop stack S;
                        if internal node n is not full
                            then
                                begin (*parent node not full; no split*)
                                insert (K, new) in correct position in internal node n;
                                finished ← true
                                end
                            else begin (*internal node n is full with p tree pointers;
                                        overflow condition; node is split*)
                            copy n to temp (*temp is an oversize internal node*);
                            insert (K, new) in temp in correct position;
                            (*temp now has p + 1 tree pointers*)
                            new ← a new empty internal node for the tree;
                            j ← ⌊((p + 1)/2⌋;
                            n ← entries up to tree pointer Pj in temp;
                            (*n contains <P1, K1, P2, K2, ... , Pj–1, Kj–1, Pj >*)
                            new ← entries from tree pointer Pj+1 in temp;
                            (*new contains < Pj+1, Kj+1, ... , Kp–1, Pp, Kp, Pp+1 >*)
                            K ← Kj
                            (*now we must move (K, new) and insert in
                              parentinternal node*)
                        end
                    end
                until finished
                end;
end;
```

5.2.1 Árboles (índices multinivel): Árbol B^+

Inserción árbol B^+

- Veamos un ejemplo…..
- **Queremos construir un árbol B^+ de orden 4, por tanto:**
  - Cada nodo interno (NO hoja) tiene entre 2 y 4 nodos hijo:
    - $\lceil n/2 \rceil = \lceil 4/2 \rceil = 2$ y $n=4$.
  - Cada nodo hoja tiene entre 2 y 3 valores del campo clave:
    - $\lceil (n-1)/2 \rceil = \lceil 3/2 \rceil = 2$ y $n-1=3$.
  - Como n=4, tenemos 4 punteros por nodo.
- **Supongamos que los datos a insertar son:**

2, 3 , 5 , 7, 11, 17, 19, 23, 29, 31

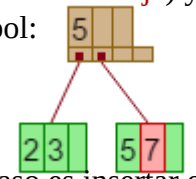
5.2.1 Árboles (índices multinivel): Árbol B^+

Vamos a construir un árbol B^+ de orden 4 (nodos internos: 2 y 4 hijos; nodos hoja: 2 y 3 valores, 4 punteros)

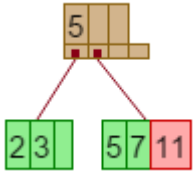
Los datos a insertar serán: (2, 3 ,5 , 7, 11, 17, 19, 23, 29, 31)

1. Introducimos los tres valores que nos permite el nodo:

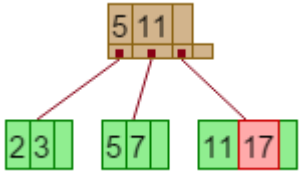
2. Al introducir el 7, el nodo debe dividirse en dos (con 2 valores mín. en nueva hoja) y subiríamos el 5 arriba como índice del árbol:



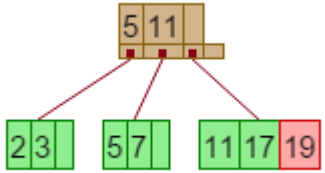
3. El siguiente paso es insertar el 11, que se añadirá sin problema alguno a la derecha del 7:



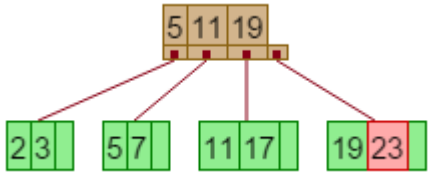
4. Insertamos el 17 y dividimos el nodo en dos, haciendo que suba el 11:



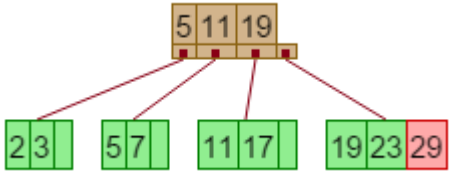
5. Insertamos el 19:



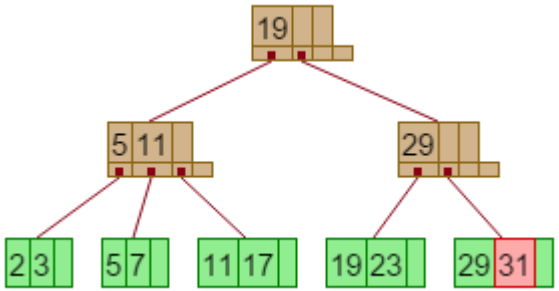
6. Insertamos el 23 y dividimos de nuevo el nodo, haciendo subir al 19 a la raíz:



7. Añadimos el 29:



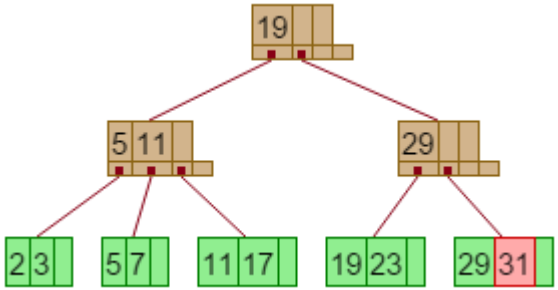
8. Al añadir el 31, el nodo raíz se divide en dos nodos, pasando a tener un nuevo nodo raíz con el valor 19, que sube después de hacer la división:



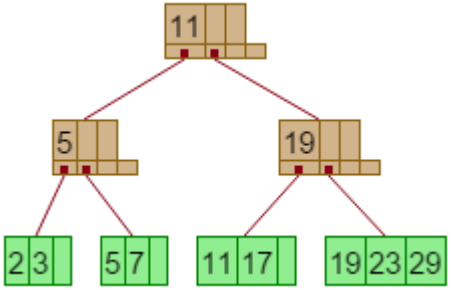
5.2.1 Árboles (índices multinivel): Árbol B+

Borrado de un árbol B+ de orden 4 (cada nodo entre 2 y 4 hijos; cada nodo hoja entre 1 y 3 valores, 4 punteros)

Partiendo del árbol anterior

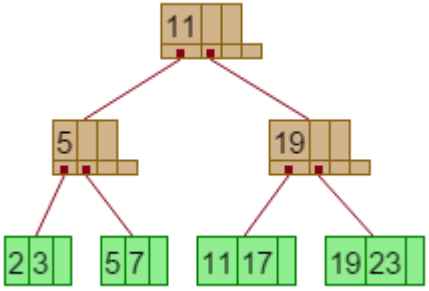


1. vamos a borrar el valor 31:



Si al eliminar un valor, la cantidad de valores restantes en el nodo es menor que la mitad de su nodo padre, entonces debe realizarse una redistribución de valores, tanto en el índice como en los nodos hijos.

2. Borramos el valor 29:



Índice

6. Especificación de índices en SQL

- En muchos SGBD comerciales, el índice ...
  - es una **estructura de datos independiente**
  - se crea y destruye dinámicamente, sin afectar al fichero principal.
- Conviene **crear un índice** sobre cierto campo **si** se va a acceder con **frecuencia** al fichero según una **condición de selección o reunión en la que aparezca dicho campo.**
  - ✓ El índice incrementará la velocidad de esas consultas o accesos
  - X Cada índice sobre una columna hace que inserción, borrado y actualización sean más complejas y consuman más tiempo
- Normalmente, el SGBD creará un **Índice Secundario**
  - Independiente de la ordenación física de los registros
  - Puede crearse con casi cualquier organización primaria de ficheros
  - Sobre un mismo fichero puede crearse varios índices secundarios

6. Especificación de índices en SQL

- Los índices son construidos sobre árboles **B, B$^+$, B*** o sobre una mezcla de ellos, funciones de cálculo u otros métodos.
- Una fila en una *tabla ordinaria* tiene una ubicación física estable. Una vez establecida esta ubicación, la fila nunca se mueve completamente. Por defecto, Oracle almacena las filas de una tabla como una colección desordenada de filas
  - No cabe hablar de **índices primarios** o de **agrupamiento**, tal y como los hemos definido en este tema
- Tipos de índices, según su implementación
  - **Índices de árbol B**, los creados por defecto
  - **Índices basados en funciones**
  - **Índices bitmap**
  - **Índices particionados**
  - **Índices de dominio**

http://blogdejuls.blogspot.com.es/2009/06/algunos-indices-en-oracle.html

- Aunque es posible almacenar una tabla con una estructura de índice (árbol **B***): *Index Organized Table (IOT)*. Una fila de *IOT* no tiene una ubicación física estable. Mantiene los datos en orden, en las hojas de un índice B * construido sobre la clave principal de la tabla. Estas filas pueden moverse para preservar el orden ordenado. Por ejemplo, una inserción puede hacer que una fila existente se mueva a una ranura diferente, o incluso a un bloque diferente.

6. Especificación de índices en SQL

ⓘEn Oracle el significado de los conceptos índice primario e índice secundario es algo diferente a todo lo anteriormente explicado

- Tipos de índices, según la columna (o campo) de indexación
  - **Índice Único**
    - Basado en una columna de tabla cuyos valores son **únicos**
  - **Índice Primario** (un caso particular de Índice Único)
    - Basado en una clave primaria de tabla
  - **Índice Secundario**
    - Basado en una columna con valores **no únicos**
  - **Índice Compuesto (o Índice Concatenado)**
    - Basado en 2 o más columnas de una misma tabla
    - Puede ser Único, Primario o Secundario
      - Un Índice Compuesto Único es útil para imponer la unicidad de la combinación de valores de varias columnas

UNA columna

VARIAS columna

- **Oracle**, crea automáticamente un **índice sobre cada clave** (primaria o alternativa):
  - Campo de indexación: columna (o conjunto de columnas) especificada como **UNIQUE** o **PRIMARY KEY** en el esquema de BD
  - Oracle recomienda la creación explícita del índice, y no sólo la declaración de la clave como UNIQUE en la especificación de la tabla.

6. Especificación de índices en SQL

- Sentencia LDD para **crear índices**

create [bitmap | unique] index nombre_indice on nombre_tabla (nombre_columna [, nombre_columna2] ...) [reverse];

- *bitmap* indica que se cree un índice de mapa de bits que permite crear índices en columnas con muy pocos valores diferentes.
- *unique* indica que el valor de la o las columnas indexadas debe ser único, no puede haber duplicidades.
- *nombre_indice* debe ser un nombre unívoco (no debe existir otro nombre de objeto en Oracle) que siga los convenios de denominación de Oracle para nombrar columnas.
- *nombre_tabla* será el nombre de la tabla donde se creará el índice.
- *nombre_columna* (o columnas) será la columna de la tabla *nombre_tabla* en la que se creará el índice. Se puede crear un índice para varias columnas.
- *reverse* indica a Oracle que invierta los bytes del valor indexado, lo que puede mejorar la distribución del procesamiento y de los datos cuando se insertan muchos valores de datos secuenciales.

6. Especificación de índices en SQL

- Sentencia LDD para **crear índices**

```
CREATE INDEX FACTURACION_NOMBRECLIENTE ON FACTURACION (NOMBRECLIENTE);
CREATE INDEX Indice_Genero_Pelicula ON Pelicula (genero) ;
CREATE INDEX product_name_sub_idx ON product substr(name,1,3);
```

(en último ejemplo, el parámetro QUERY_REWRITE_ENABLED debe estar establecido en TRUE. Costoso: name **like** 'MUE%', Rápido: substr(name,1,3) = 'MUE';)

- Especificación del **orden** ascendente o descendente **de las entradas** del índice

```
CREATE INDEX Indice_Año_Pelicula ON Pelicula ( año DESC ) ;
```

- **Índice compuesto**

```
CREATE UNIQUE INDEX FACT_CODCLI_FEC ON FACTURACION (CODCLI, FECHA);
CREATE INDEX Indice_Nombres_Director ON Director ( apellido1 ASC, apellido2 ASC, nombre DESC ) ;
```

  - El índice **incrementa** la **velocidad de** las **consultas** que acceden por...
    - **apellido1**
    - **apellido1 y apellido2**
    - **apellido1 y apellido2 y nombre**
  - Pero el SGBD **no usará el índice** para consultas que acceden por...
    - **apellido2**
    - **nombre**
    - **apellido2 y nombre**

6. Especificación de índices en SQL

- Sentencia LDD para **destruir índices**

  DROP INDEX Índice_Genero_Película ;

  – Conviene eliminar un índice cuando ya no se espera realizar consultas basadas en el campo de indexación

  – Desaparece el **coste de mantenimiento** del índice

  – Se recupera el **espacio** de almacenamiento ocupado por el índice

- No se puede eliminar un índice creado automáticamente por el sistema (debido a la especificación de una restricción de integridad UNIQUE o PRIMARY KEY sobre una o varias columnas), sino que se debe eliminar (DROP) o desactivar (DISABLE) la restricción de integridad correspondiente

6. Especificación de índices en SQL

Cuándo crear un índice

- **Indexar** una columna si es **Muy usada** en **condiciones de reunión** o de **selección**
  - Suele aparecer en cláusulas WHERE en comparaciones de **igualdad o rango de valores**: =, >, <, ≥, ≤, BETWEEN
  - Suele ser columna de reunión (join) de tablas (ej. claves extranjeras)
  - Tiene gran **variedad** de valores (⏵ más discriminación en búsquedas)
  - **No** es **modificada** muy **a menudo**

- Emplear un **índice compuesto** si habitualmente se accede por medio de varias columnas utilizadas conjuntamente:

  Sea la tabla PERSONA(nif, nombre, apellido, fecha_nacim, ciudad, telef)

  CREATE INDEX idx_persona ON PERSONA (**apellido, nombre**);

- Al tener un índice ordenado se puede utilizar para operaciones que impliquen ordenación (ORDER BY, GROUP BY, UNION, DISTINCT, JOIN).

6. Especificación de índices en SQL: Índices en Oracle

Cuándo evitar un índice

- La tabla es muy pequeña: cargarla en memoria
- Sobre columnas...
  - **modificadas** muy a menudo
  - Si el resultado de la consulta incluye una proporción grande de la tabla (ej: el 70 % de las tuplas) → exige recorrer gran parte de la tabla
  - que **sólo** aparecen **en funciones/operadores** (distintos de MAX o MIN): pueden hacer que el índice no se use
  - Si el atributo contiene cadenas de caracteres muy largas (ejemplo: párrafos) → entrarían pocos valores en cada nodo del índice disminuyendo su eficiencia
- Se degradan los requisitos de procesamiento crítico: si se le da al SGBD muchos índices a elegir
- El **costo(almacenamiento+mantenimiento)** > **beneficio**
  - INSERT, UPDATE, DELETE sobre tablas indexadas
    - mantenimiento del índice (automático, por parte del SGBD)
  - A veces conviene...
    - **crear** la **tabla**, **rellenar**la, y luego **crear** los **índices**
    - **eliminar índices**, **modificar** datos y **crear** de nuevo los **índices**

6. Especificación de índices en SQL: Índices en Oracle

Consejos para la utilización de índices

- Máximo 4 índices por tabla
  - ⬆ índices ⇨ ⬆ necesidad de espacio y ⬇ velocidad al modificar
  - 👁 el SGBD **puede crear** implícitamente **índices sobre** las **claves**
- Se puede crear más si es raro actualizar la tabla (histórico)

- En todos los casos: lo mejor es **hacer pruebas** sobre el resultado que ofrecen los *IS* frente a no tenerlos.
- Es útil usar las herramientas del SGBD que muestran el **plan de ejecución**.
- Hay que tener en cuenta que el SGBD crea índices para garantizar la unicidad de los atributos clave (primary key y unique)