Descripción de la asignatura

- Tema 1: Introducción
- Tema 2: Árboles genéricos
- Tema 3: Mapas y diccionarios
- Tema 4: Mapas y diccionarios ordenados
- Tema 5: Grafos
- **Tema 6: EEDD en memoria secundaria**

Bloque 1

Bloque 2

Bloque 3

Tema III.6
EEDD en memoria secundaria

jose.velez@urjc.es
abraham.duarte@urjc.es
raul.cabido@urjc.es
angel.sanchez@urjc.es
mariateresa.gonzalezdelena@urjc.es
juanjose.pantrigo@urjc.es

Resumen

- Introducción a las estructuras de la información
- Ficheros
- Índices
  - Basados en vectores, listas, ABB, AVL, árboles B

Objetivos

- Introducción a las estructuras de la información
- Índices como forma de acelerar el acceso a la información almacenada en memoria secundaria
- Implementación de índices en memoria principal y secundaria utilizando diferentes E.D.
- Serialización de estructuras de datos.

Introducción

Introducción

- Las estructuras de datos utilizadas se han almacenado y manipulado en memoria principal (MP)
- El uso de la memoria principal tiene ciertas limitaciones:
  - No permite la persistencia de los datos: Una vez que el programa se termina, los datos se pierden
  - Mayores limitaciones de espacio: la cantidad de datos que se puede manipular limitada a la cantidad de memoria
  - No facilita que diferentes programas compartan datos
  - Coste económico elevado
- El gran inconveniente de la memoria secundaria (MS) es la velocidad de acceso a los datos:
  - RAM: la tecnología PC133 necesita entre 10 y 7 ns
  - SSD: 0.25 a 0.05 ms (50 000 ns)
  - Disco duro magnético (HD): 30 a 7 ms (7 000 000 ns)

Introducción

	Memoria principal	Memoria Secundaria
Durabilidad	volátil	persistente
Capacidad	baja	alta
Tiempos de Acceso	corto	largo
Coste económico	elevado	bajo
Acceso CPU	directo	indirecto

Introducción

Un Fichero Informático (o archivo) es un conjunto de bits que son almacenados en un dispositivo. Un archivo es identificado por un nombre y la descripción de la carpeta o directorio que lo contiene.

Los archivos se organizan jerárquicamente en un árbol de directorios.

Introducción

- El Sistema Operativo del ordenador proporciona la abstracción de fichero
- El Sistema Operativo elije el tipo de sistema de ficheros más adecuado para el tipo de dispositivo
  - Cinta
  - Disco Magnético
  - Memoria SSD
  - Memoria RAM

Organización de ficheros

Organización de los ficheros

- Un campo (field) es una pieza de información sobre un objeto. Por ejemplo: el apellido (campo) de una persona (objeto).
- Un registro es una colección de campos. Por ejemplo: El registro empleado se compone de los siguientes campos: nombre, apellido y puesto.
- Un Fichero de Registros es una colección de registros.
  - El uso más habitual de los Ficheros de Registros es la consulta de datos, pues la inserción y borrado de datos es costosa.

Tipos de Ficheros de Registros

- Ficheros de registros de longitud variable
  - El número de campos es variable
  - La longitud de los campos es variable
- Ficheros de registros de longitud fija
  - El número de campos es fijo
  - La longitud de los campos está limitada

Persistencia en los lenguajes de programación

Los lenguajes de programación suelen ofrecer mecanismos de persistencia.

Estos mecanismos permiten almacenar y recuperar de memoria secundaria las estructuras de datos definidas por programa.

Las operaciones de guardar un objeto (por ejemplo una lista de objetos) en un fichero se denomina serialización.

Los lenguajes suelen proporcionar diferentes formas de serialización:

- A fichero en formato propio, en formato XML, en formato JSON.
- A una base de datos (hibernate).

Ejemplo de persistencia en Java

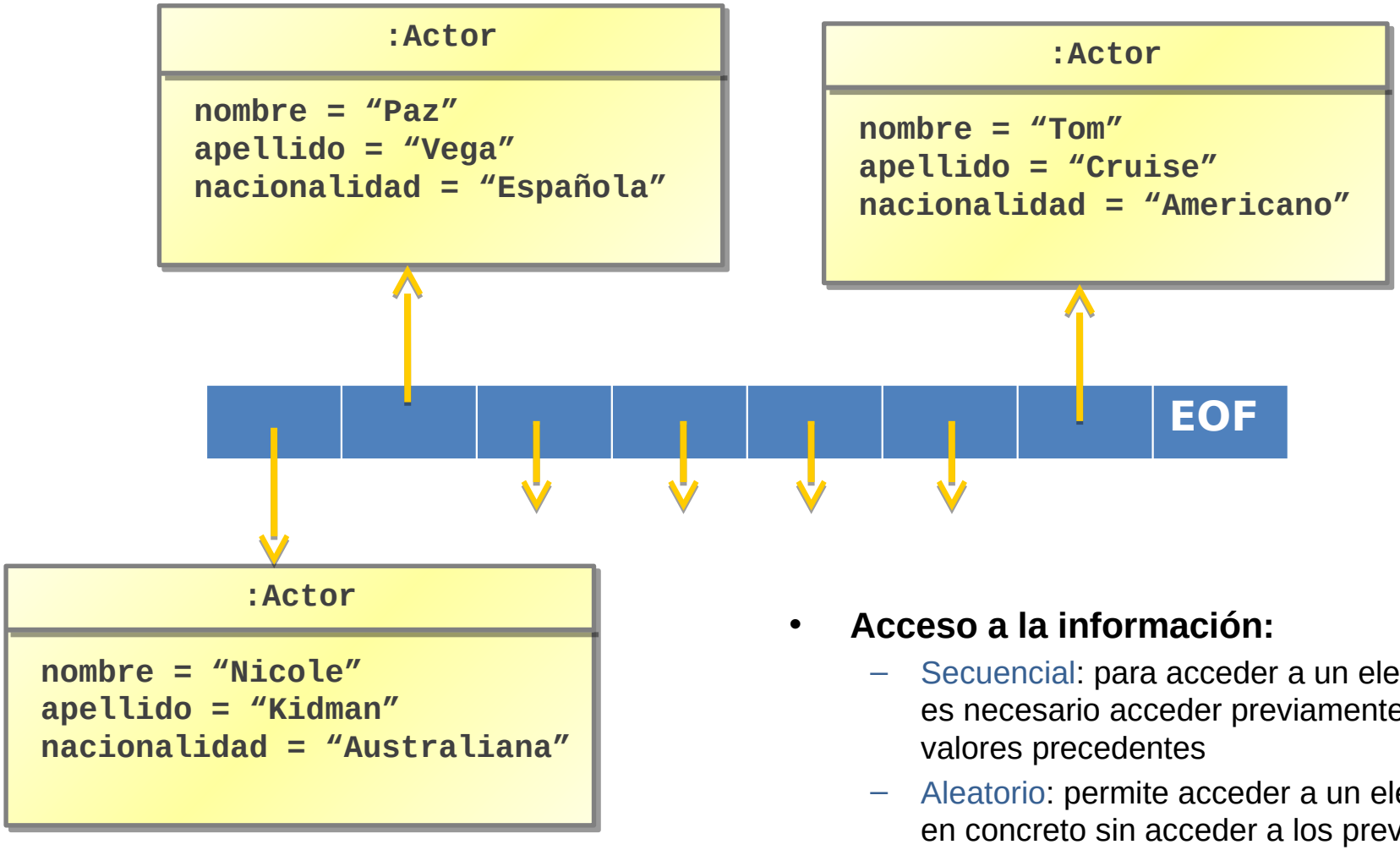
```
public class Movie {
        private int time;
        private String title;

        public void setTime(int time) {
                this.time = time;
                return this;
        }

        public void setTitle(String title) {
                this.title = title;
                return this;
        }

        public int getTime() {
                return time;
        }

        public String getTitle() {
                return title;
        }
}
```

Es importante tener setters y getters para la serializacion

Ejemplo de persistencia en Java

```
Movie p1 = new Movie();
p1.setTitle("Los Goonies")
p1.setTime(95);

Movie p2 = new Movie();
p2.setTitle("StarWars")
p2.setTime(120);

List moviesList=new LinkedList();
moviesList.add(p1);
moviesList.add(p2);

try{
     XMLEncoder encoder = new XMLEncoder(
                                  new BufferedOutputStream(
                                    new FileOutputStream("pelis.xml")));
     encoder.writeObject(moviesList);
     encoder.close();

}catch(FileNotFoundException fileNotFound){
     System.out.println("ERROR");
}
```

Registros de ficheros de longitud fija

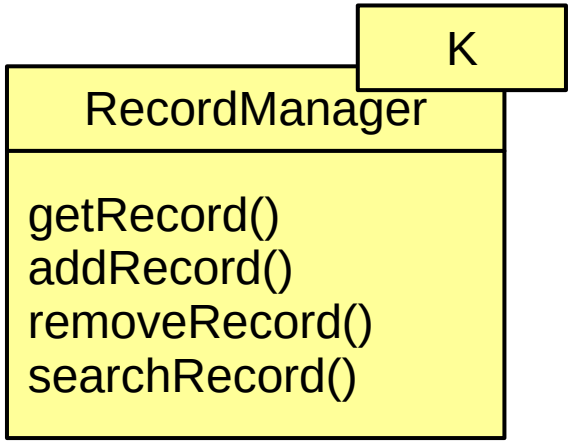


- **Acceso a la información:**
  - Secuencial: para acceder a un elemento es necesario acceder previamente a los valores precedentes
  - Aleatorio: permite acceder a un elemento en concreto sin acceder a los previos

Índices

Indexación

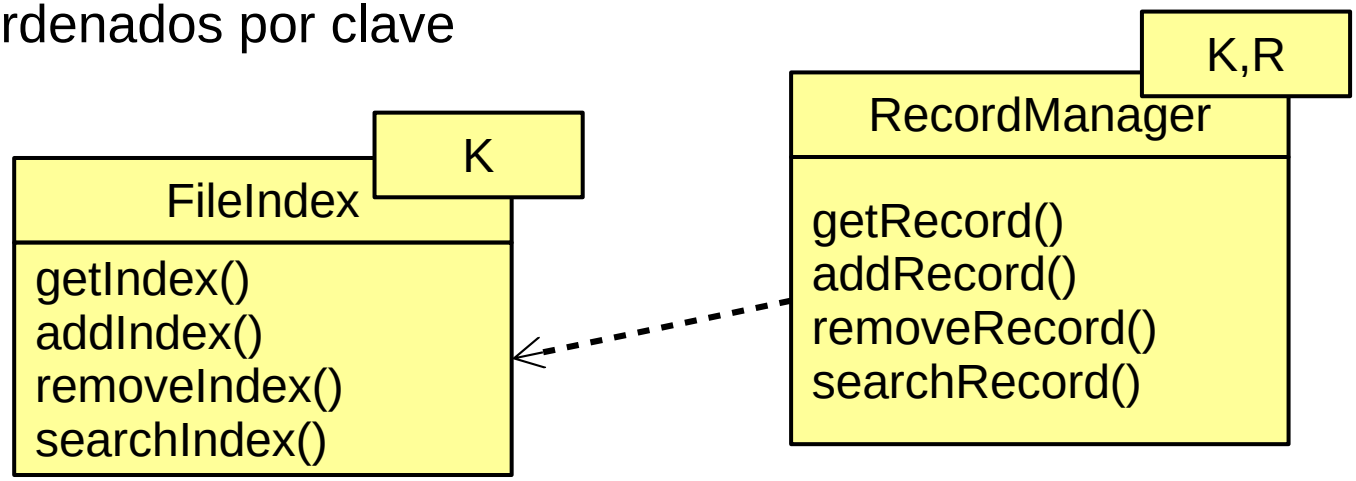
- Partimos de ficheros:
  - Con información
    - Estructurada en registros (objetos)
    - Desordenada
  - Permiten acceso directo
  - Almacenados en disco
- Tendremos la necesidad de:
  - mejorar las búsquedas
  - disminuir accesos a disco



Utilizar índices!!!

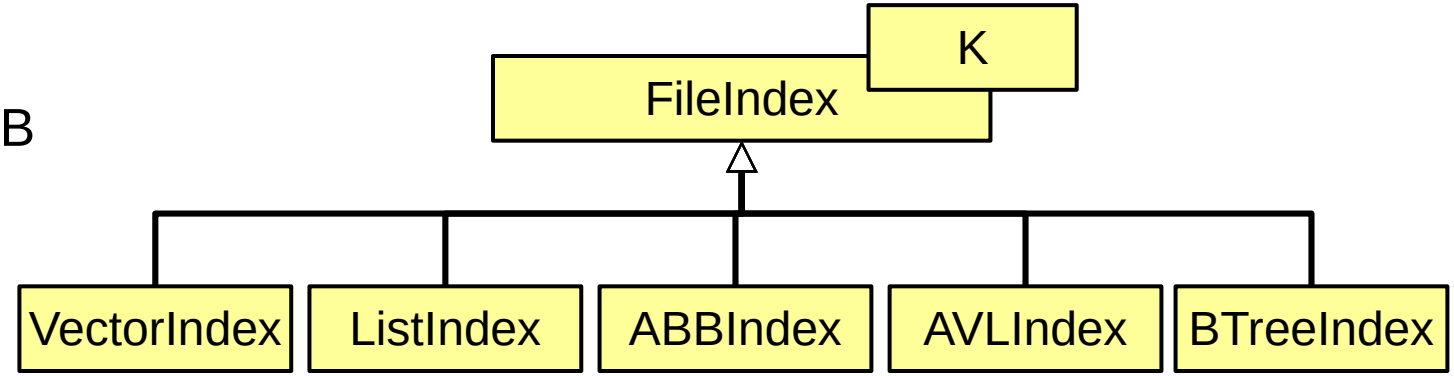
Definición de índice

- Un índice es una estructura de datos que acelera el acceso a la posición de un registro en un fichero.
- Ordenación virtual del fichero que permite encontrar la información más fácilmente
- Colección estructurada de claves y posiciones por pares
  - Clave: campo del fichero de datos que identifica al registro
  - Posición: posición relativa en la que se encuentra el registro (objeto) al que referencia la clave
  - Pares ordenados por clave



Implementaciones de índice

- Posibles implementaciones de índices que se verán en este tema:
  - Vector
  - Lista
  - ABB
  - AVL
  - Árbol B



- Los índices se pueden almacenar en memoria principal o en memoria secundaria

Implementaciones de índice (vector)

- Implementación de un índice como un vector
  - Vector en memoria principal

Índice Ordenado

0	1	2	3	4	5
<Belén,5>	<Eduardo,3>	<Nicole,2>	<Paz,4>	<Penélope,1>	<Tom,0>

Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

Implementaciones de índice (vector)

- Implementación de un índice como un vector
  - Vector en memoria secundaria

Índice Ordenado

	Clave	Posición
0	Belén	5
1	Eduardo	3
2	Nicole	2
3	Paz	4
4	Penélope	1
5	Tom	0

Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

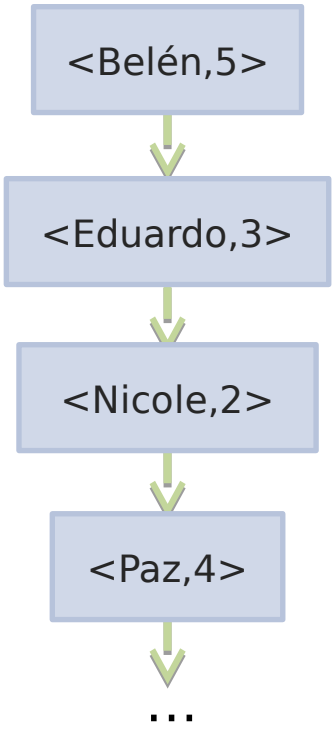
Implementaciones de índice (vector)

- Ventajas
  - Estructura de acceso directo
  - Permite hacer búsquedas binarias o dicotómicas con coste O(log(*n*))
- Inconvenientes
  - Limitación de memoria al tamaño del vector (sólo en MP)
  - Mantenimiento de nuevas claves: la inserción y el borrado son O(*n*) (sin contar la búsqueda de la posición)
    - Además, en memoria secundaria el mantenimiento del vector y la lectura de sus claves implica accesos a disco

Implementaciones de índice (lista)

- Implementación de un índice como una lista
  - Lista en memoria principal

Índice Ordenado



Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

Implementaciones de índice (lista)

- Implementación de un índice como una lista
  - Lista en memoria secundaria

Índice Ordenado

	Clave	Posición	Sig.
0	Tom	0	-1
1	Penélope	1	0
2	Nicole	2	4
3	Eduardo	3	2
4	Paz	4	1
5	Belén	5	3

Cabeza = 5

Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

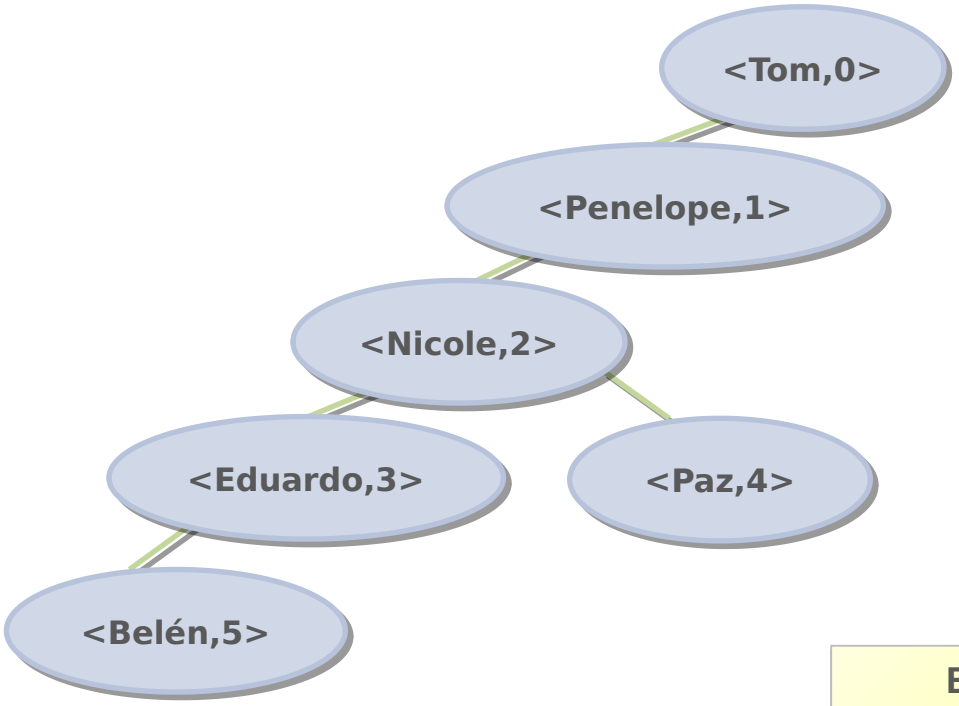
Implementaciones de índice (lista)

- Ventajas
  - No tiene limitación respecto al tamaño de los datos que puede guardar (ni en MP ni en MS)
  - La inserción y el borrado son O(1) en MP y MS (sin contar la búsqueda de la posición → O(*n*))
- Inconvenientes
  - Estructura de acceso secuencial (en MP y MS)
  - Las búsquedas son secuenciales con coste O(*n*)
  - En MS, el mantenimiento de la lista y la lectura de sus claves implica accesos a disco

Implementaciones de índice (ABB)

- Implementación de un índice como un ABB
  - ABB en memoria principal

Índice Ordenado



Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

El resultado depende del orden en la inserción de los elementos en el árbol

Implementaciones de índice (ABB)

- Implementación de un índice como un ABB
  - ABB en memoria secundaria

Índice Ordenado

	Clave	Posición	H.Izd	H. dcho.	Padre
0	Tom	0	1	-1	-1
1	Penélope	1	2	-1	0
2	Nicole	2	3	4	1
3	Eduardo	3	5	-1	2
4	Paz	4	-1	-1	2
5	Belén	5	-1	-1	3

Raiz = 0

Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

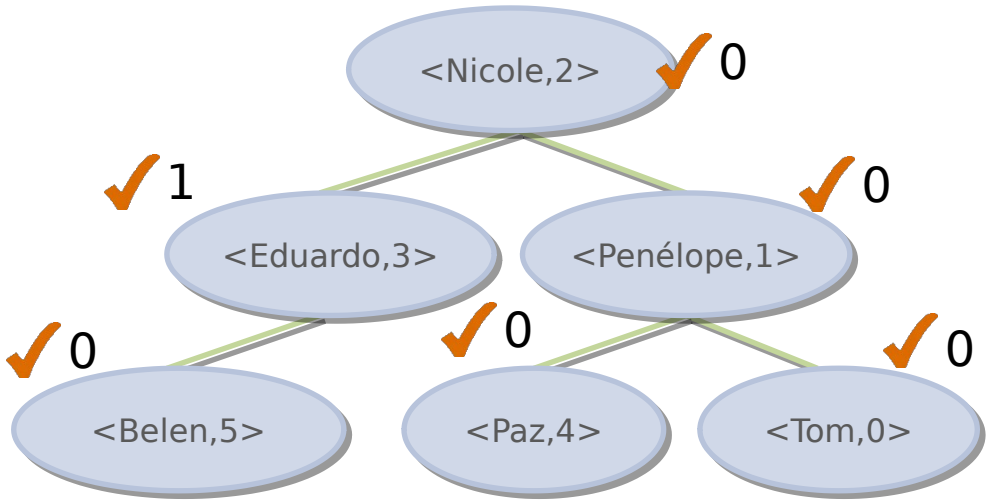
Implementaciones de índice (ABB)

- Ventajas
  - No limitación en el tamaño del árbol
  - Búsquedas eficientes: en promedio O(log(*n*))
  - Actualización del árbol menos costosa (comparado con la actualización de un vector o una lista ordenada)
- Inconvenientes
  - Pueden degenerar en una lista (ordenada)
  - [MS] El mantenimiento del árbol y la lectura de sus claves también implica accesos a disco

Implementaciones de índice (AVL)

- Implementación de un índice como un AVL
  - AVL en memoria principal

Índice Ordenado



Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

Implementaciones de índice (AVL)

- Implementación de un índice como un AVL
  - AVL en memoria secundaria

Índice Ordenado

	Clave	Pos	Height	H.Izd	H. dcho.	Padre
0	Tom	0	0	-1	-1	1
1	Penélope	1	1	4	0	2
2	Nicole	2	2	3	1	-1
3	Eduardo	3	1	5	-1	2
4	Paz	4	0	-1	-1	1
5	Belén	5	0	-1	-1	3

Raiz = 2

Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

Implementaciones de índice (AVL)

- Ventajas
  - No limitación en el tamaño del árbol
  - Búsquedas eficientes: O(log(*n*))
  - Actualización del árbol menos costosa (búsqueda + inserción)
  - No degeneran en una lista

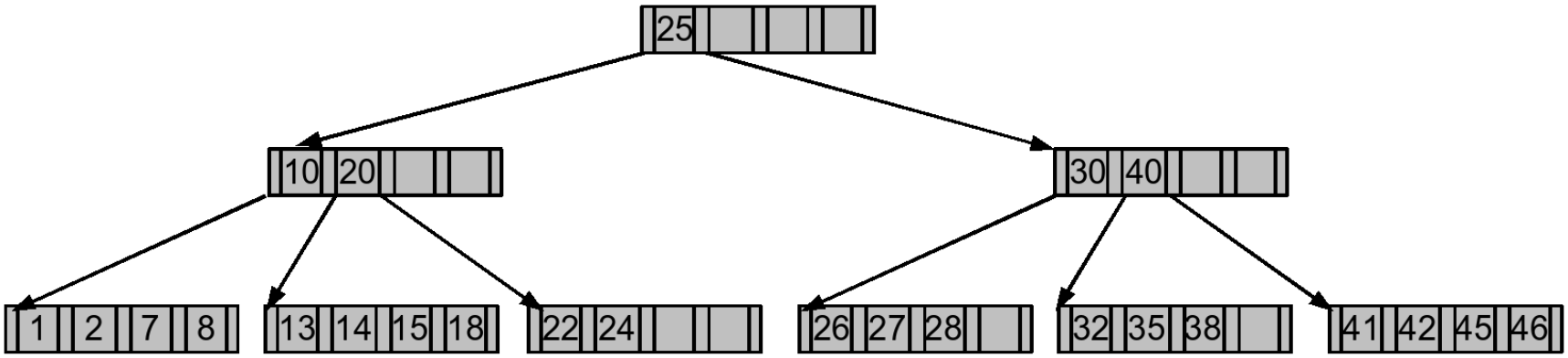
- Inconvenientes
  - Es necesario hacer rotaciones para mantener el factor de equilibrio (aunque habitualmente es O(1))
  - [MS] La lectura del índice y el mantenimiento del árbol implican accesos a disco
  - [MS] Las rotaciones para mantener el factor de equilibrio suponen correcciones de índices que necesitan acceso a disco, aunque la complejidad siga siendo O(log(*n*))

Árboles B

Implementaciones de índice (Árboles B)

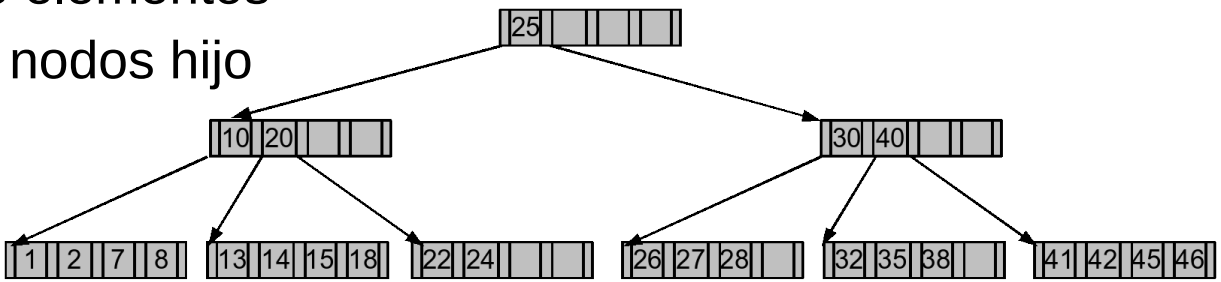
- Mejor método para mantener un índice en memoria externa → árbol B

¿Qué es un Árbol B?



Árboles B

- Árbol de búsqueda *m*-ario (grado máximo de un nodo)
  - Cada nodo puede tener hasta *m* hijos
- Todas las hojas se encuentran a la misma altura
  - Equilibrado en altura
- El árbol crece hacia arriba
  - La raíz emerge en lugar de ponerla y luego intentar cambiarla
  - No hay rotaciones
- Cada nodo contiene:
  - Lista ordenada de elementos
  - Referencias a los nodos hijo



Árboles B

- El número de valores que puede contener un nodo (*NCN*) depende del orden del árbol
  - Para un nodo que no es la raíz

$$\frac{m-1}{2} \leq NCN \leq m-1$$

  - Para la raíz

$$1 \leq NCN \leq m-1$$

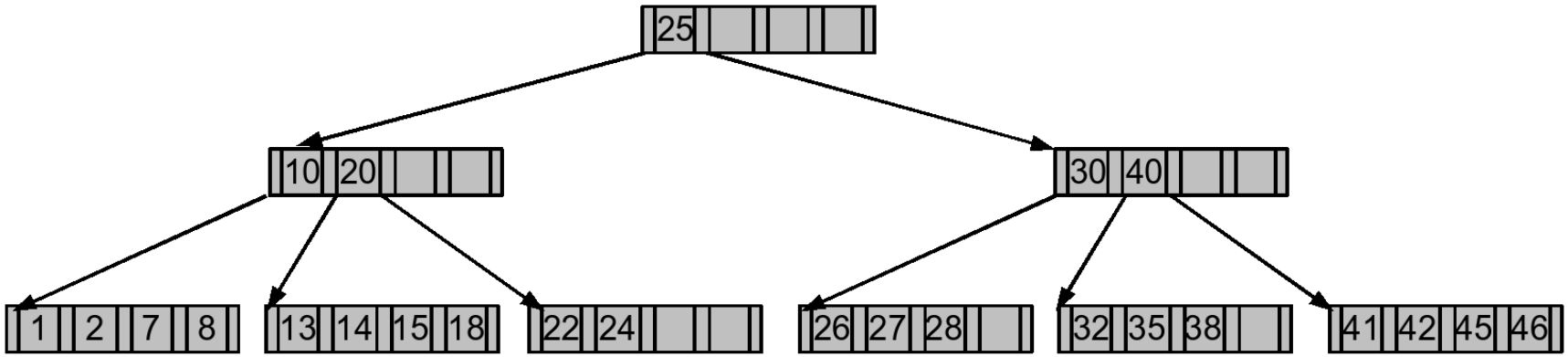
- Número de referencias a los hijos (descendientes) dependen del número de claves

$$NCN+1$$

- La raíz de un árbol por lo menos tiene dos hijos (descendientes), salvo que él mismo sea una hoja

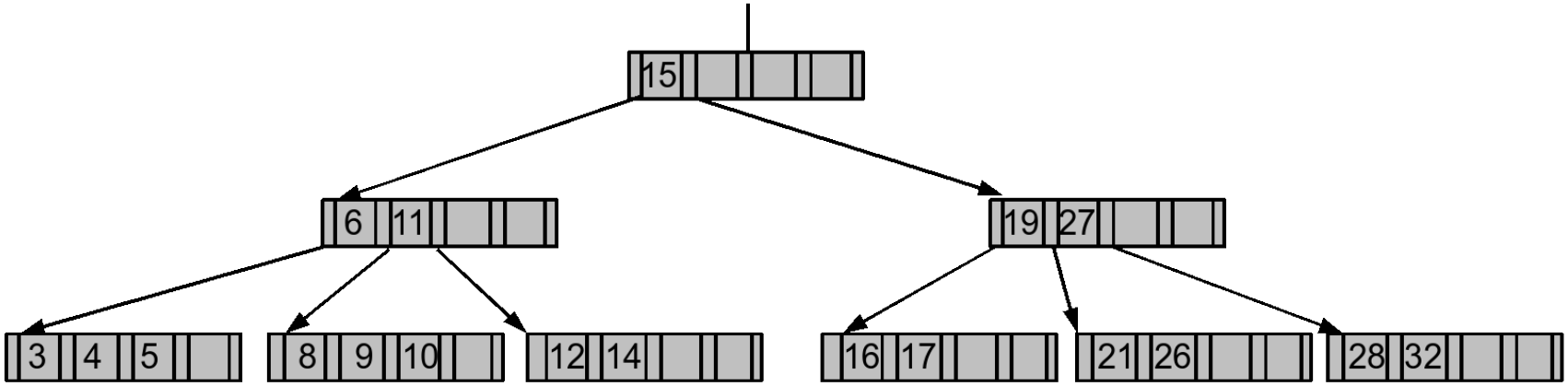
Árboles B

- Árbol de orden 5



Árboles B (búsqueda)

- Igual que en un árbol binario de búsqueda, pero inspeccionando todas las claves de cada nodo
  - Ejemplo: buscar el valor 26



Árboles B (inserción)

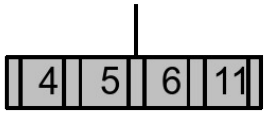
1. Búsqueda del valor en el nodo actual (1ª vez, en el raíz)
   - Si existe la clave → No se hace nada
   - Si no existe → Ir a paso 2
2. Búsqueda en los hijos, siguiendo el camino de búsqueda
3. Se ha encontrado la hoja donde insertar la clave:
   - Si no está llena → inserción en el orden adecuado
   - Si está llena
     - Inserción y División: Creamos un nodo nuevo y repartimos las *m* claves
     - Promoción: Promocionamos la clave central del nodo al nodo padre

Árboles B (inserción)

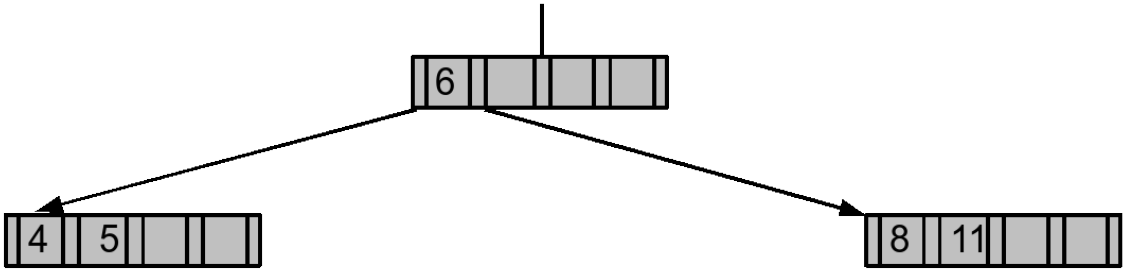
- **Ejemplo**: insertar en un árbol B la siguiente secuencia de elementos:

  6 11 5 4 8 9 12 21 14 10 19 28 3 17 32 15 16 26 27

  – Insertar 6, 11, 5, 4

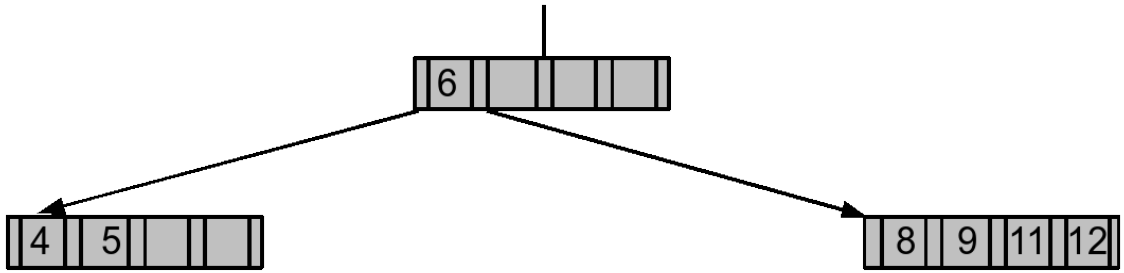


  – Insertar 8

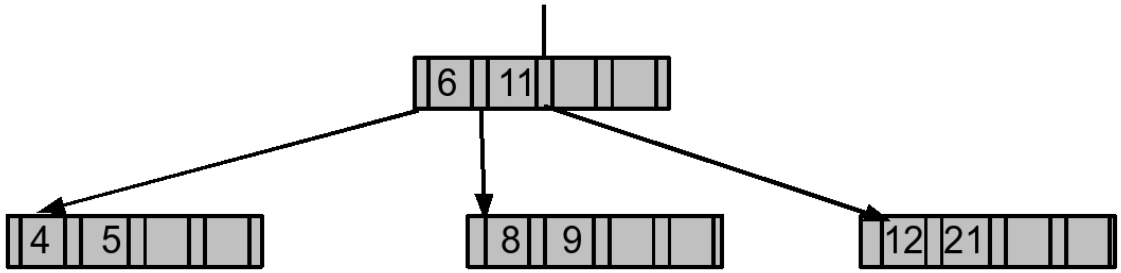


Árboles B (inserción)

– Insertar 9, 12

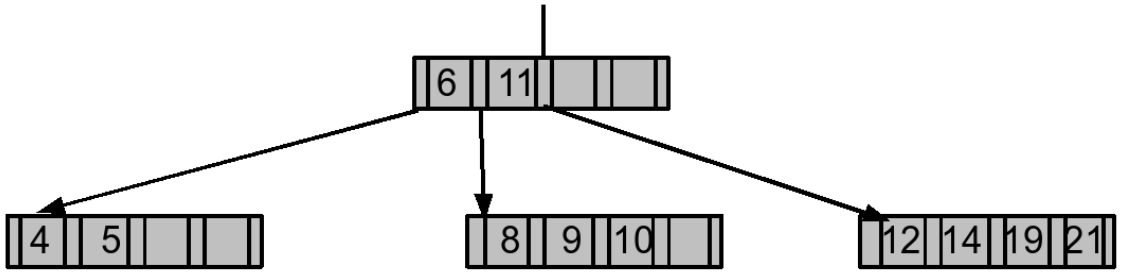


– Insertar 21

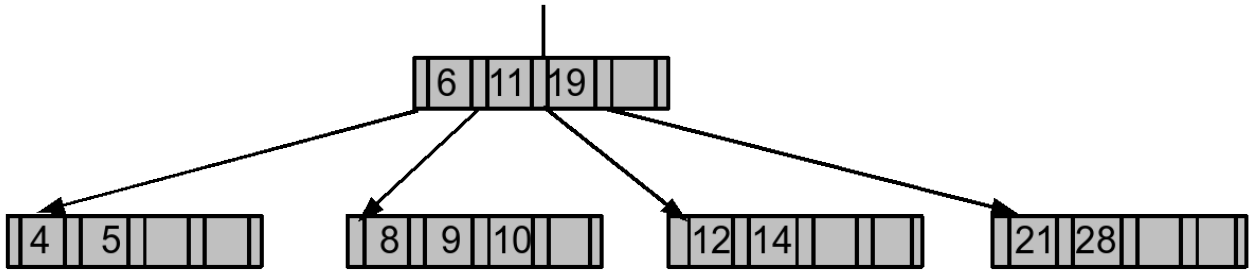


Árboles B (inserción)

- Insertar 14, 10, 19

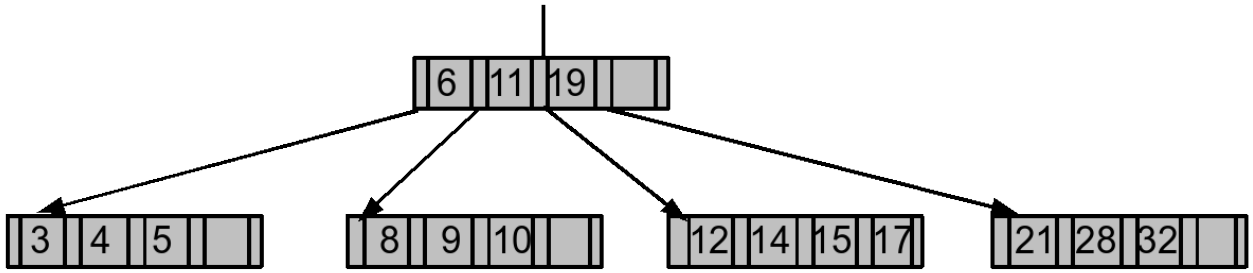


- Insertar 28

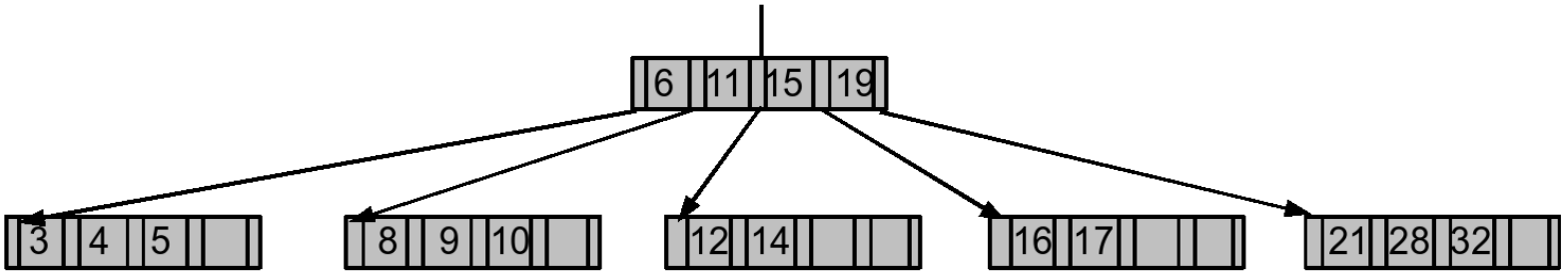


Árboles B (inserción)

- Insertar 3, 17, 32, 15

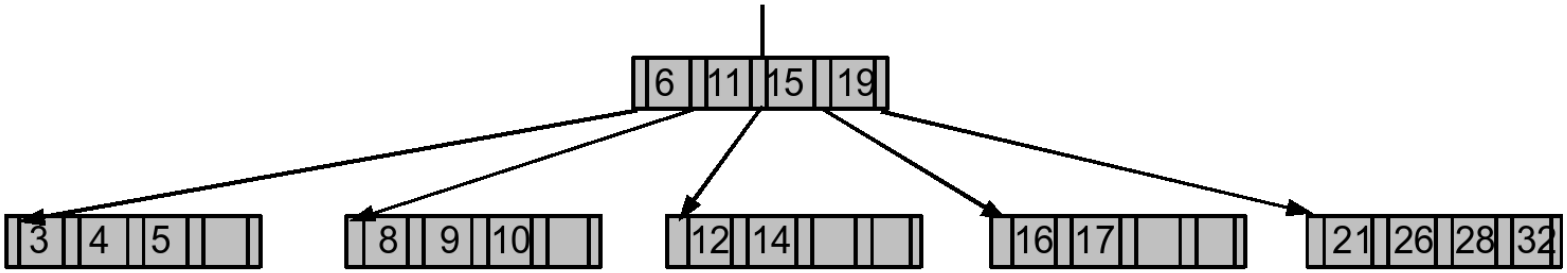


- Insertar 16

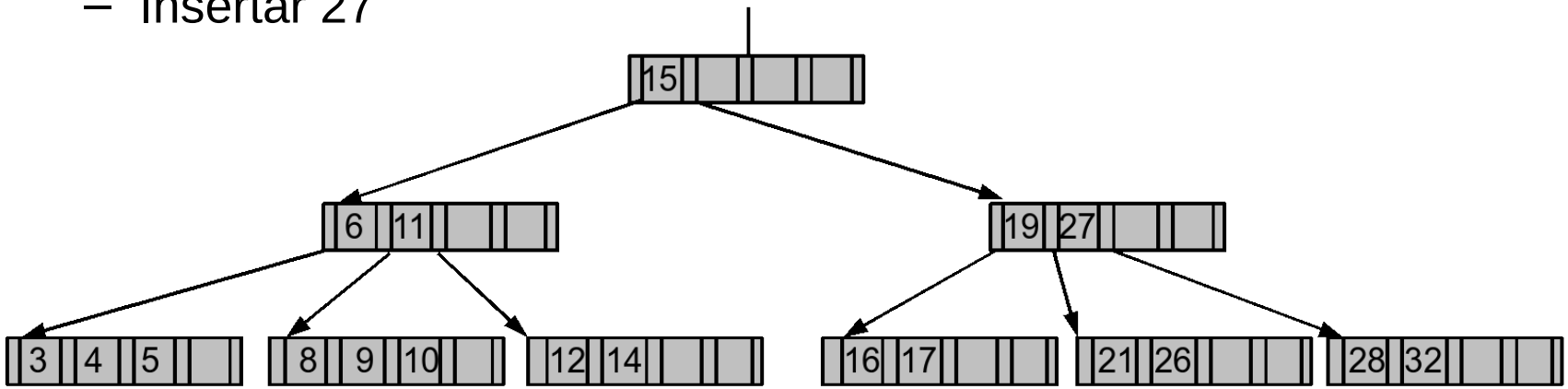


Árboles B (inserción)

- Insertar 26

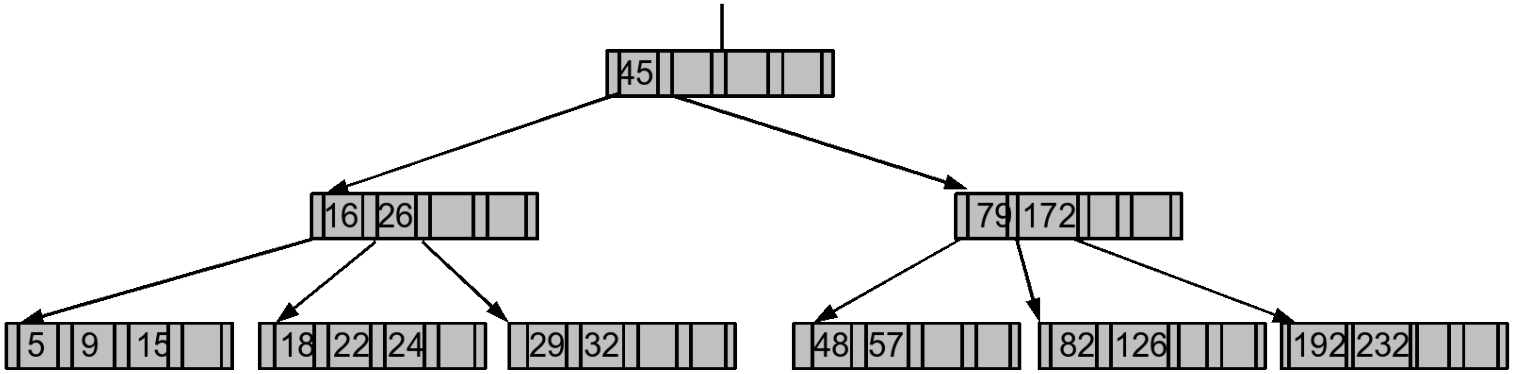


- Insertar 27



Árboles B (borrado)

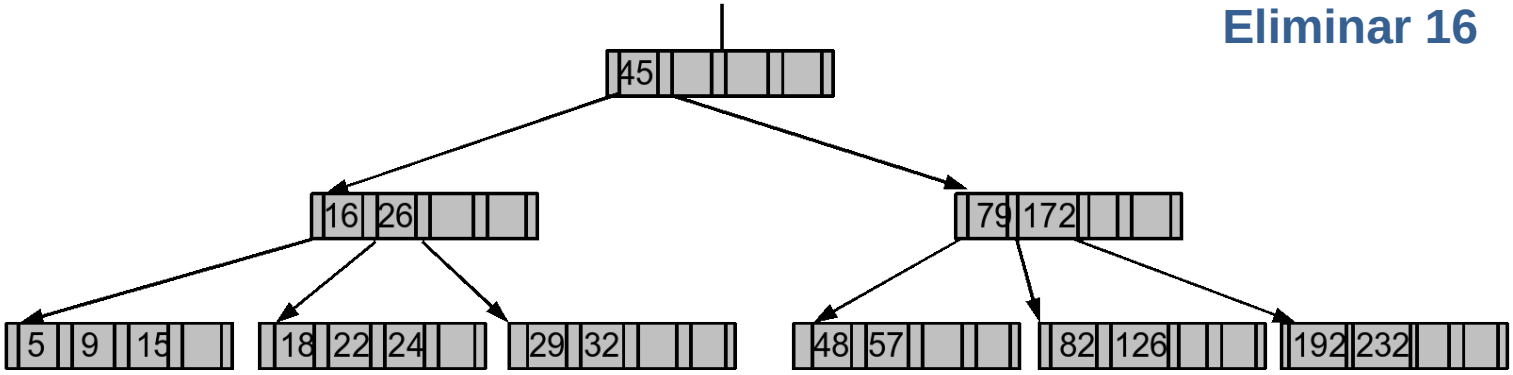
- Inicialmente, borrar un valor en un árbol B es igual que aplicar una operación de borrado en un ABB
  - Recordatorio
    - Aplicable únicamente si el elemento a borrar tiene un hijo hoja
    - Si no tiene hijo hoja, se intercambia por su sucesor
- Tras el borrado → tres posibles casos



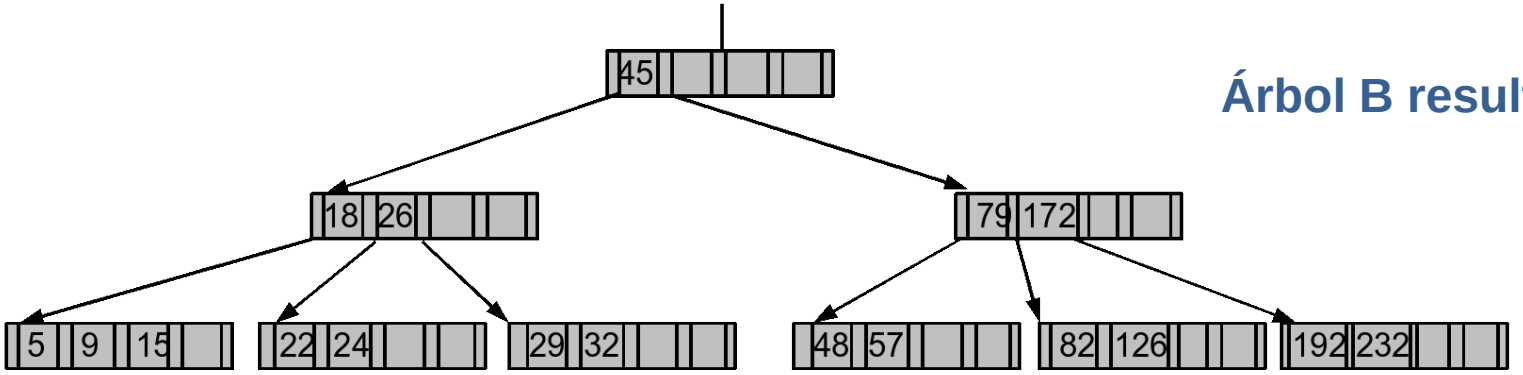
Árboles B (borrado)

- Caso 1: tras el borrado el árbol continua siendo árbol B
  - No requiere restructuración

Eliminar 16



Árbol B resultante

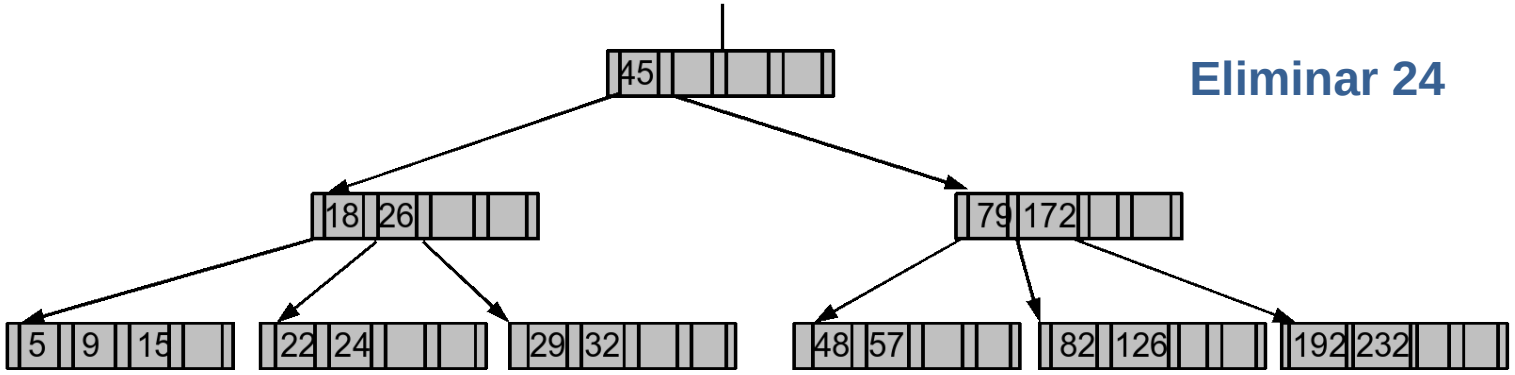


Árboles B (borrado)

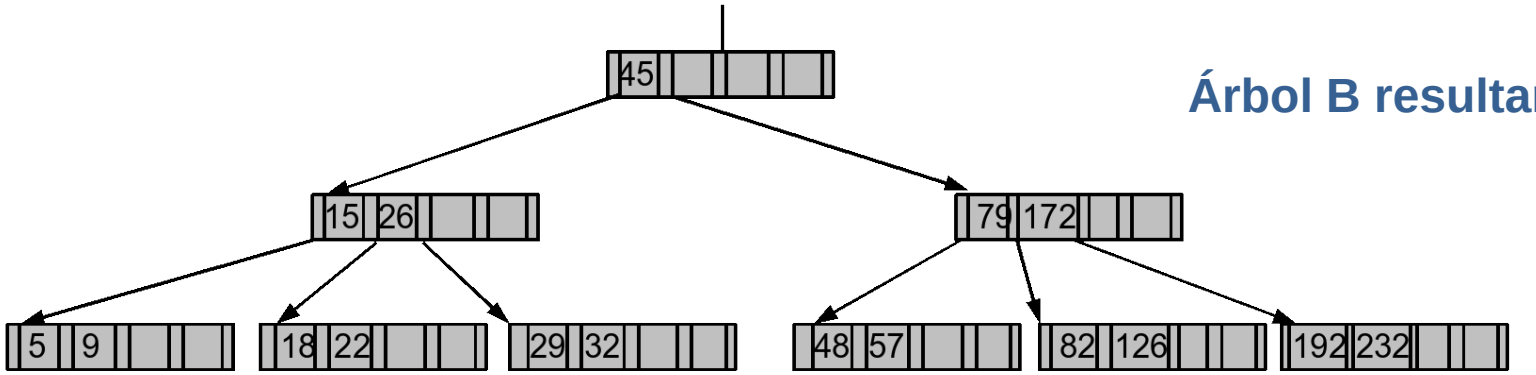
- Caso 2: tras el borrado el árbol deja de ser árbol B
  - Una hoja contiene menos de $(m\text{-}1)/2$ claves: sub-ocupación
- Se requiere una reestructuración:
  - Se examinan las dos hojas hermanas inmediatamente adyacentes a la hoja afectada
  - Si la hoja a su izquierda tiene más de $(m\text{-}1)/2$ claves, se mueve el último al nodo padre, para que descienda el del nodo padre a la hoja afectada
  - Si no, si la hoja a su derecha tiene más de $(m\text{-}1)/2$ claves, se mueve la primera al nodo padre, para que descienda la del nodo padre a la hoja afectada

Árboles B (borrado)

- Caso 2: tras el borrado el árbol deja de ser árbol B
  - La hoja se queda con menos de $(m\text{-}1)/2$ valores



Eliminar 24



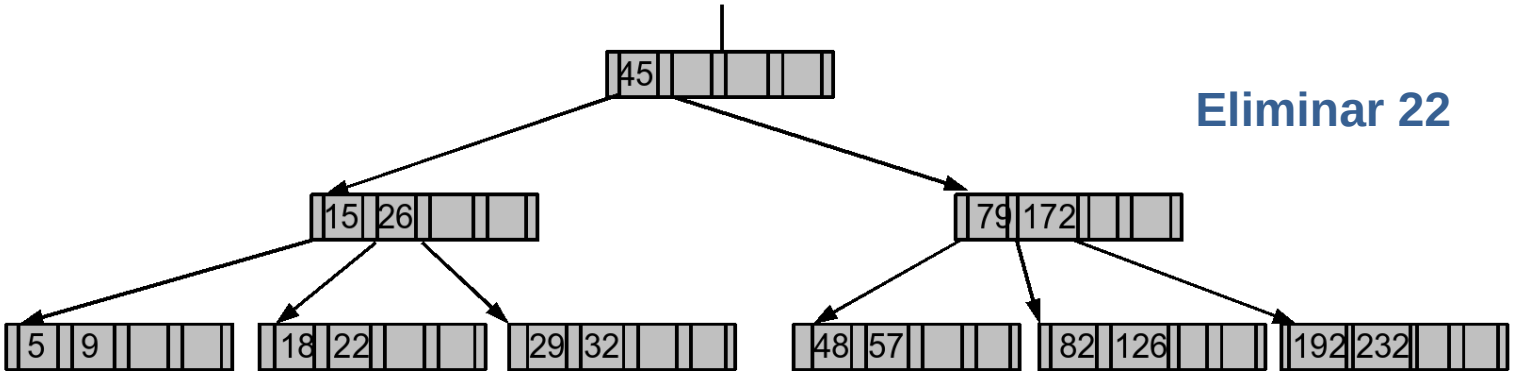
Árbol B resultante

Árboles B (borrado)

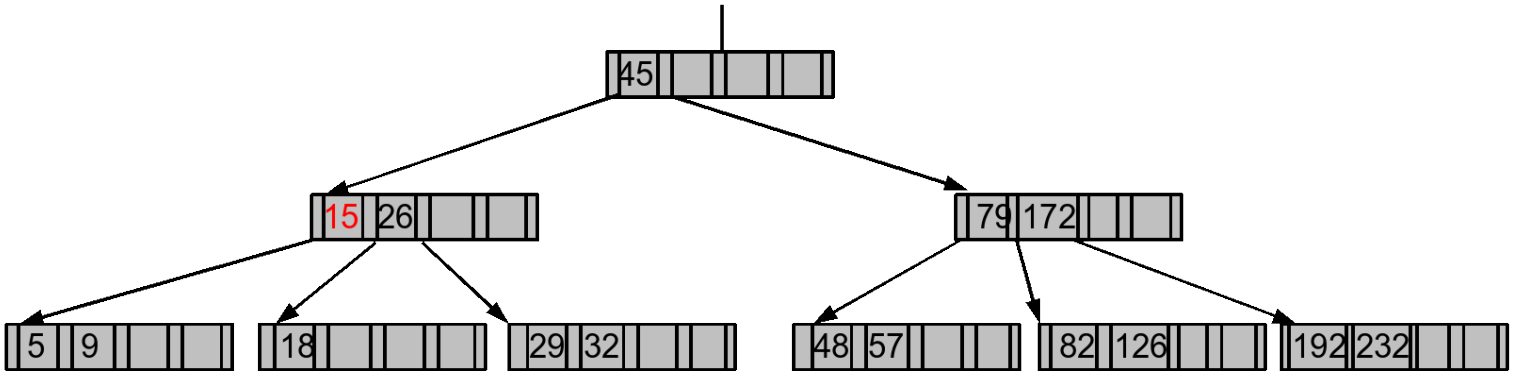
- Caso 3:
  - Es posible que los dos hermanos tengan $(m\text{-}1)/2$ claves
  - Se requiere una reestructuración del árbol que cambia el número de nodos
  - Estrategia de reestructuración
    - Como la unión de la hoja con la contigua sigue resultando en un nodo no completamente lleno, se crea un nuevo nodo que combina:
      - El nodo afectado por el borrado
      - El nodo adyacente que se desea fusionar (puede ser el izquierdo o el derecho)
      - La clave mediana de ambos nodos (que se encuentra en el nodo padre)
    - Repetir el proceso si la restructuración deja al nodo padre con un número de claves menor que $(m\text{-}1)/2$
    - En el caso límite, el proceso se extiende hasta la raíz, quitándole su clave y disminuyendo en uno la altura del árbol.

Árboles B (borrado)

- Caso 3

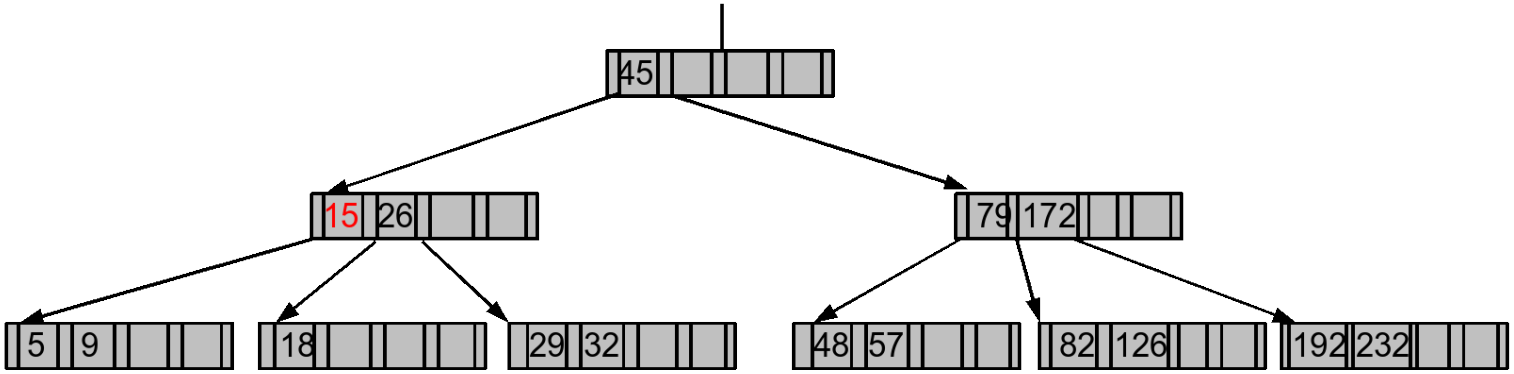


  - Seleccionar la mediana de las hojas afectadas

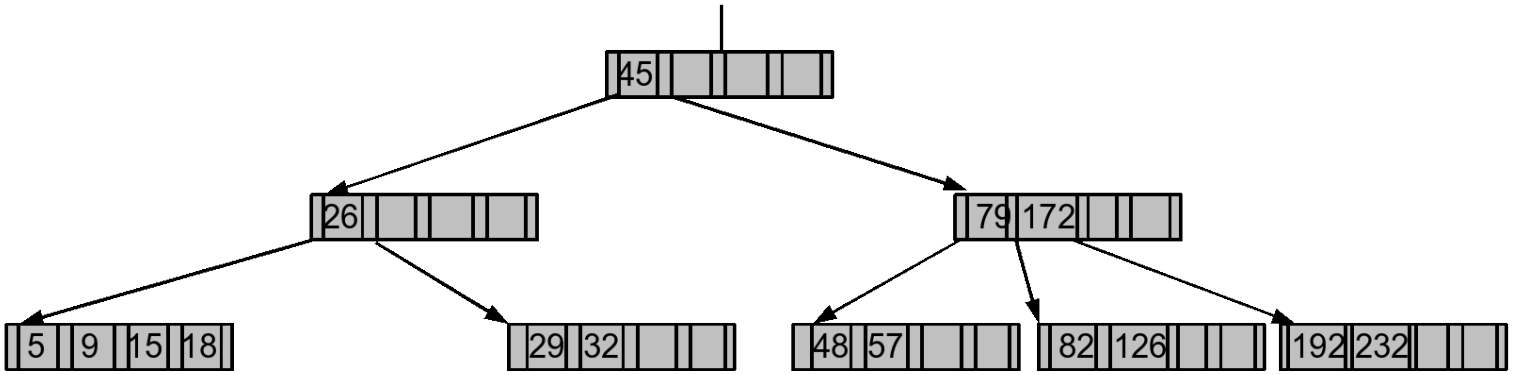


Árboles B (borrado)

- Caso 3

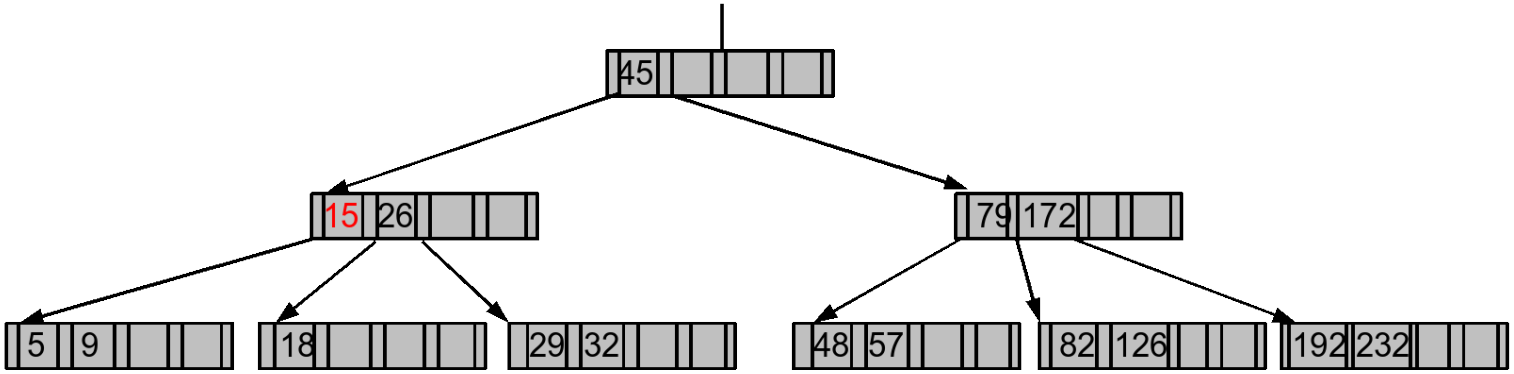


  - Fundir hojas afectadas, incluyendo la mediana

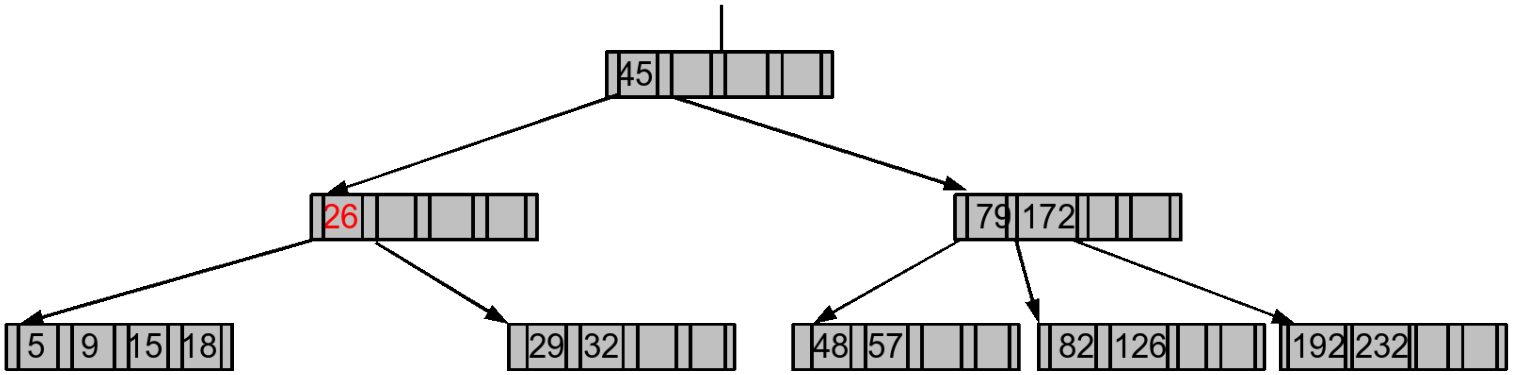


Árboles B (borrado)

- Caso 3

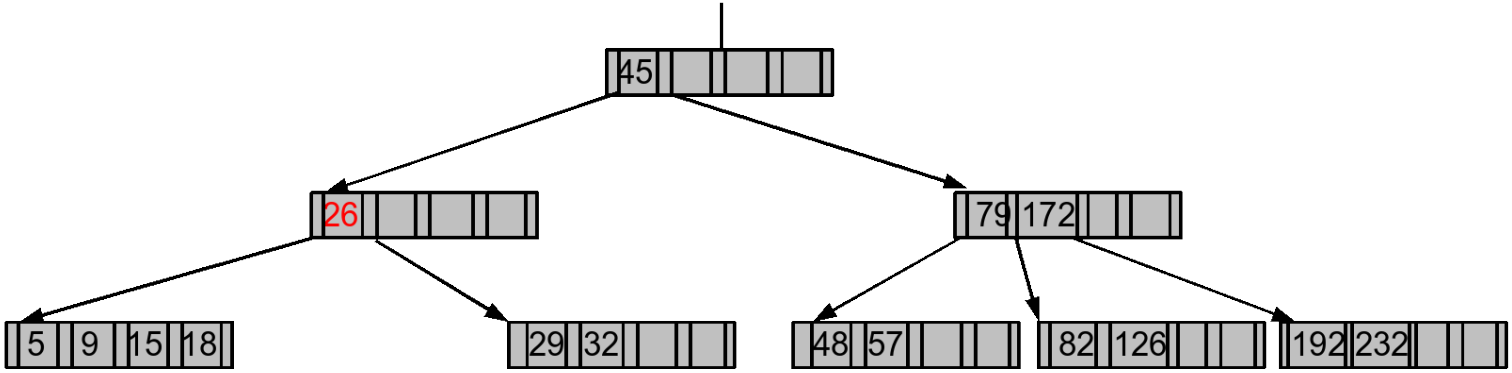


  - El problema de la desocupación se propaga al padre (26)

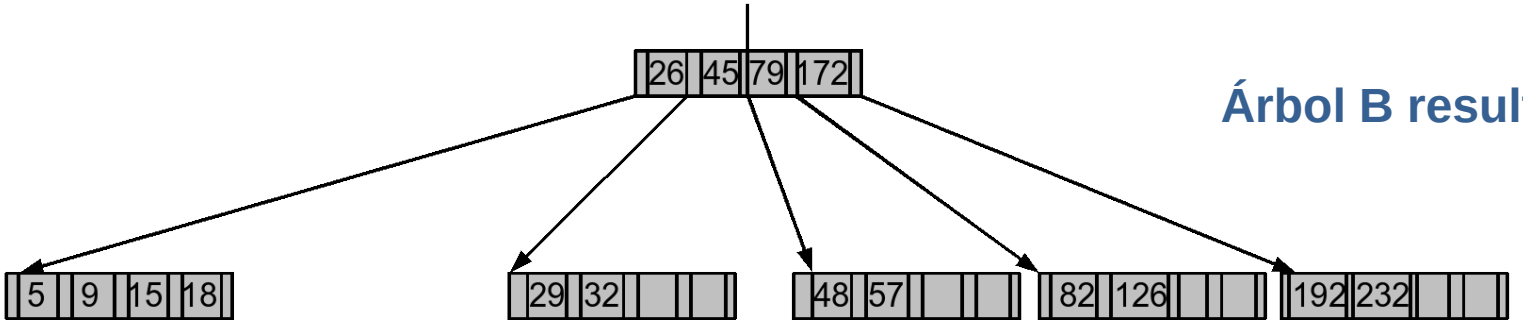


Árboles B (borrado)

- Caso 3



  - Repetir el proceso con el nodo afectado, su hermano derecho y su padre



Árbol B resultante

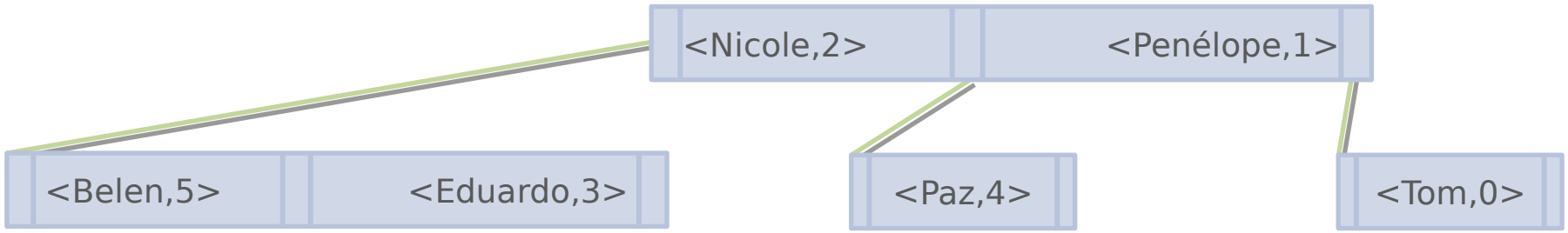
Implementaciones de índice (Árboles B)

- Implementación de un índice como un árbol B
  - Árbol B en memoria principal

Archivo desordenado

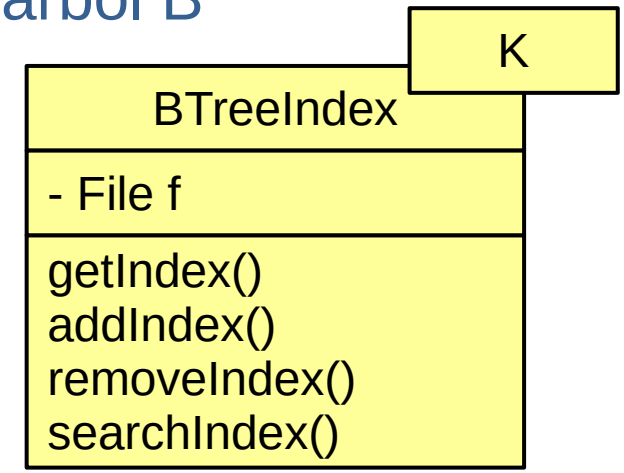
	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	Española

Índice Ordenado



Implementaciones de índice (Árboles B)

- Implementación de un índice como un árbol B
  - Árbol B en memoria secundaria



Archivo desordenado

	Nombre	Apellidos	Nacionalidad
0	Tom	Cruise	Americano
1	Penélope	Cruz	Española
2	Nicole	Kidman	Australiana
3	Eduardo	Noriega	Español
4	Paz	Vega	Española
5	Belén	Rueda	

Índice Ordenado

	H1	Clave1	Pos1	H2	Clave2	Pos2	H3	Padre
0	-1	Tom	0	-1			-1	1
1	2	Nicole	2	3	Penélope	1	0	-1
2	-1	Belén	5	-1	Eduardo	3	-1	1
3	-1	Paz	4	-1			-1	1

Raiz = 1

Referencias

https://www.cs.usfca.edu/~galles/visualization/BTree.html