

# Unidad 8

## Árboles B

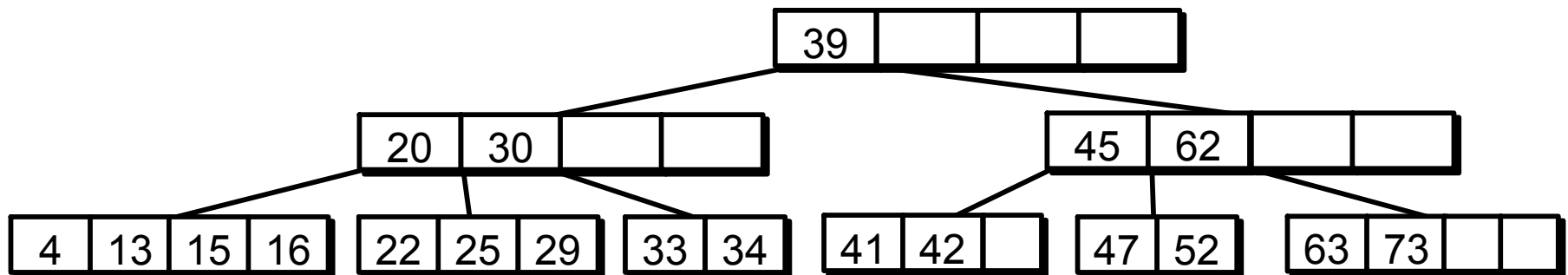
- Bibliografía: “Algoritmos y Estructuras de datos” de Aguilar y Martinez. Unidad 16
- Autor: Ing Rolando Simon Titiosky.

# Problemas de los AVL

- Los AVL tienen una Eficiencia  $F(n) = \log n$ .
  - *Su altura depende de la cantidad de nodos.*
  - A Nivel  $k$ , tendrá  $2^{k+1}-1$  nodos.
- *Cuando se tienen un conjunto masivo de datos (ej 1millon de Registros de Clientes de un banco equivalen  $k \cong 19$  niveles), los datos estarán ubicados en Discos.*
  - *El Tiempo de Acceso a disco es notablemente superior que el de RAM.*
  - *Es necesario minimizar estos accesos al disco y maximizar el uso de RAM*

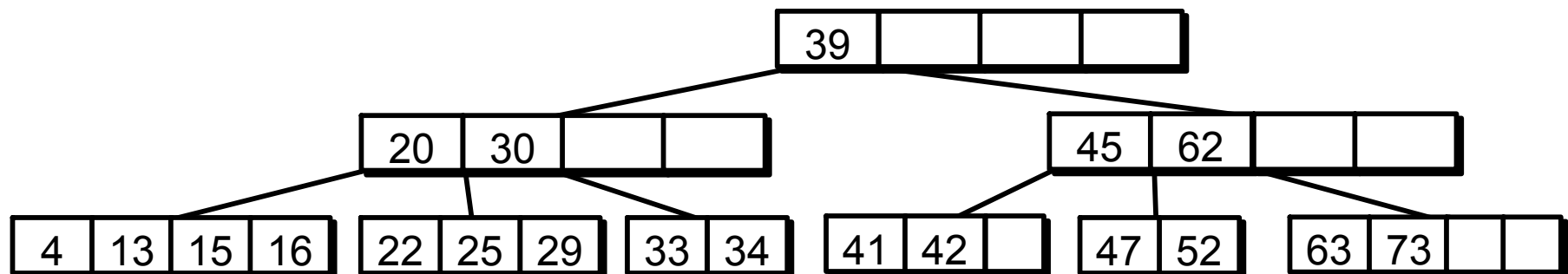
# Definición de Árbol B

- Solución: Árboles de Búsqueda m-arios.
- Cada Nodo puede tener hasta  $m$  subárboles.
- Las claves se organizan en AVL.
- *Objetivo:* Que la altura del árbol sea pequeña, pues las iteraciones y los acceso a disco dependerá de ello.
  - El Árbol B es una solución particular de esta tecnología



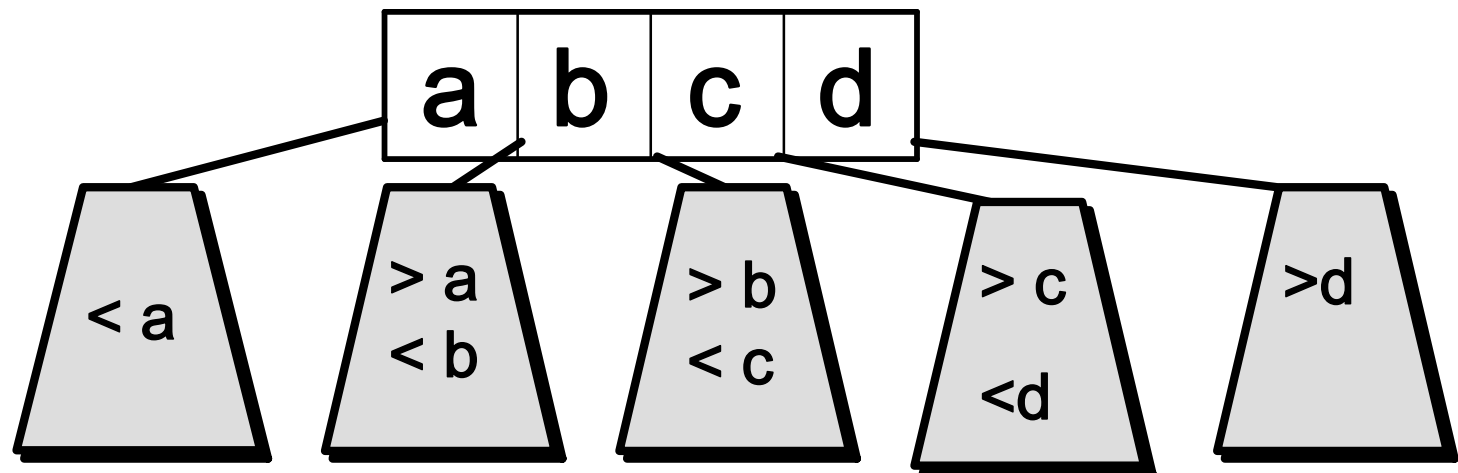
# Características del Árbol B

- Es  $m$ -arios y sin subárboles vacíos.
- Siempre está perfectamente equilibrado.
- **Página:** nombre de sus nodos. Se los accede en bloque.
  - Todas las **Páginas** están en el mismo nivel
  - Como Máximo:  $m$  Ramas y  $m-1$  Claves
  - Como Mínimo:  $(m/2)+1$  Ramas y  $(m/2)$  Claves
- La Raíz puede estar vacía o incluso tener 1 Clave, con sus 2 ramas.



# Características del Árbol B

- Las claves dividen el espacio de claves como en el AVL
- Los Árboles que estudiaremos serán de orden  $m=5$ 
  - Un orden mayor aumenta la complejidad de la Inserción y Borrado.
  - Un Orden menor disminuye la eficiencia de Búsqueda
  - Numero Máximo Por Nodo: 4 Claves y 5 Ramas
  - Numero Mínimo Por Nodo: 2 Claves y 3 Ramas
- Se rastrea el camino de búsqueda al igual que en el Árbol de Búsqueda.

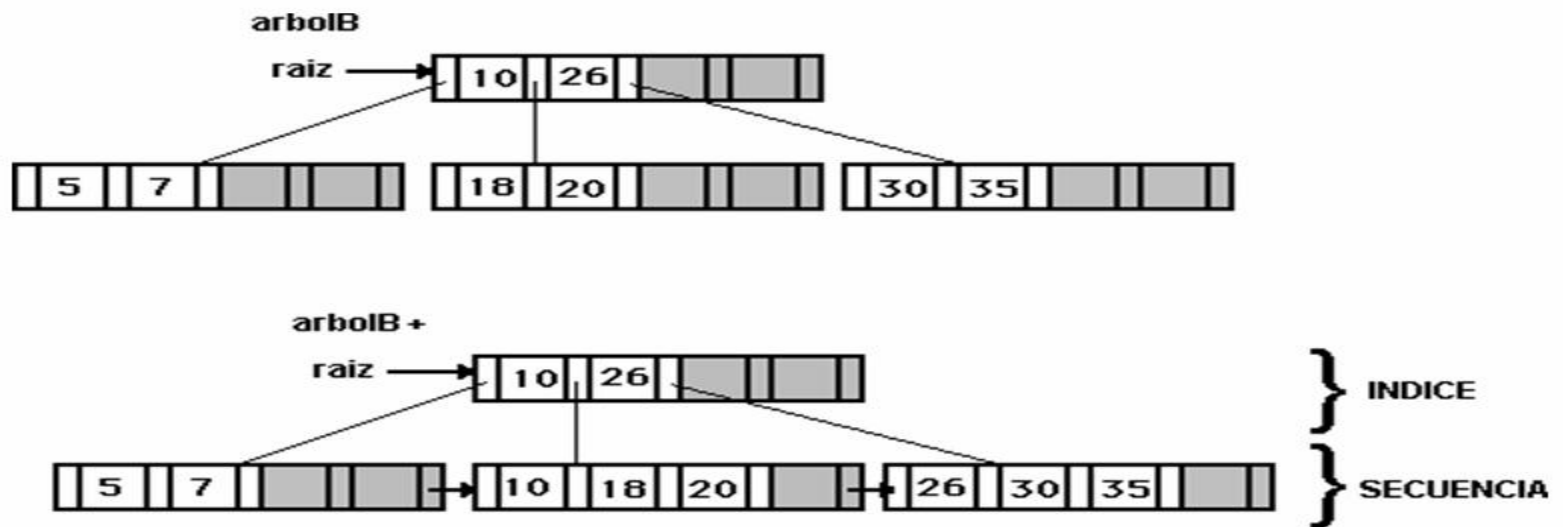


# Variantes: B+

- Los árboles B+ permiten un recorrido secuencial mas rápido que el B pues
  - Las claves se encuentran en el Índice Y en las hojas
  - Existe un puntero ProximaPagina.

## Árboles B+

### ■ Ejemplo



# Variante: B\*

- Propone nuevas reglas para el mantenimiento.
  - Los nodos deben estar  $2/3$  llenos siempre.
  - La nueva construcción logra una búsqueda más rápida que el B+ pero una inserción más costosa.
- Si cada nodo tiene un máximo de  $m$  descendientes.
  - C/nodo menos la raíz tiene al menos  $(2m-1)/3$  hijos.
  - Si es de orden  $m=5$ 
    - Numero Máximo Por Nodo: 4 Claves y 5 Ramas
    - Numero Mínimo Por Nodo: 3 Claves y 4 Ramas
- Recuerden: En Arbol B
  - C/nodo menos la raíz tiene al menos  $(m/2)+1$  hijos.
    - Numero Máximo Por Nodo: 4 Claves y 5 Ramas
    - Numero Mínimo Por Nodo: 2 Claves y 3 Ramas

# TAD arbolB: *ArbolB.h*

## **/\*Definición de los Datos del TAD\*/**

```
#define m 5                      /*Orden del Árbol B: Como Máximo: m Ramas y m-1 Claves*/
typedef int tipoClave;
typedef struct pagina
{
    tipoClave claves[m];          /* m{0..4}Numero de claves será for (k=1; k ≤m; k++) */
    struct pagina* ramas[m];
    int cuenta;                  /*Numero de claves de la pagina*/
} Pagina;
```

## **/\*Definición de las Operaciones del TAD\*/**

```
void escribeNodo(Pagina* actual);
int nodoLLeno(Pagina* actual); /*Devuelve verdadero si el numero de claves es m-1*/
int nodoSemiVacio(Pagina* actual); /*Devuelve .V. si el numero de claves es menor a m/2*/
void crearArbolB(Pagina **raiz);
Pagina *buscar (Pagina *actual, tipoClave cl, int * indice);
Pagina *buscarNodo(Pagina *actual, tipoClave cl, int * k);
void insertar (Pagina **raiz, tipoClave cl);
....
```



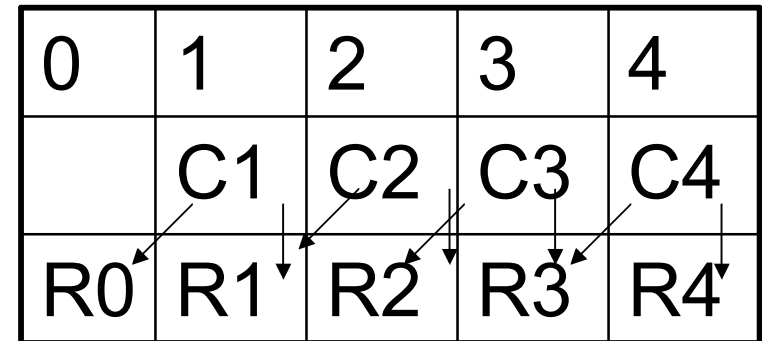
# TAD arbolB: *ArbolB.c*

```
int nodoLLeno(Pagina* actual)
{
    return (actual->cuenta == m - 1);
}
```

```
int nodoSemiVacio(Pagina* actual)
{
    return (actual->cuenta < m/2);
}
```

```
void escribeNodo(Pagina* actual)
{
    int k;
    printf("\n Nodo: ");
    for (k = 1; k <= actual->cuenta; k++)
        printf(" %d ", actual->claves[k]);
    printf("\n");
}
```

0	1	2	3	4
	C1	C2	C3	C4
R0	R1	R2	R3	R4



# Búsqueda de una Clave en Árbol B

- Hay que inspeccionar en c/Página todas las claves de que consta para definir:
  - La Posición propia de la clave
  - El ptr a la rama que nos llevará a la clave
- **buscar()**: desciende por el árbol por la ruta determinada por la clave y los nodos.
- **buscarNodo()**: función auxiliar que realiza la inspección interna de c/página.

# Codificación de la Búsqueda en Árbol B

```
Pagina* buscar(Pagina* actual, tipoClave cl, int* indice)
{
    if (actual == NULL) return NULL; /*No lo encontré*/
    else { int esta = buscarNodo(actual,cl,indice);
          if (esta) return actual;
          else return buscar(actual -> ramas[*indice],cl,indice);
        }
}

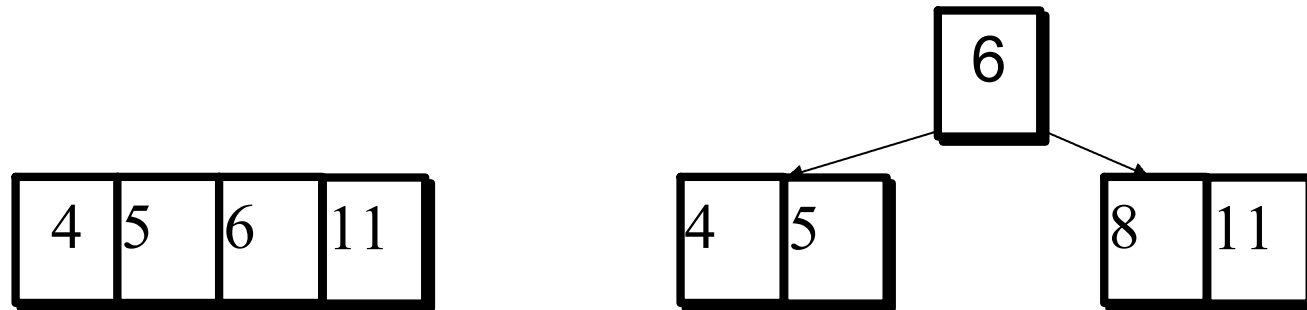
int buscarNodo(Pagina* actual, tipoClave cl, int* k)
{
    int encontrado; /*K indica la posición dentro de las Claves*/
    if (cl < actual->claves[1]) /*La clave es menor que la menor clave de esta pagina*/
    {
        encontrado = 0;
        *k = 0; /*Indica que debe seguir por el ptr a la pagina con claves mas chicas*/
    }
    else { /* examinan las claves del nodo en orden descendente */
          *k = actual->cuenta;
          while ((cl < actual->claves[*k]) && (*k > 1)) (*k)--;
          encontrado = (cl == actual->claves[*k]);
        }
    return encontrado;
}
```

# Proceso de Formación de un Árbol B

- Un Árbol B crecen “Hacia Arriba”, hacia la raíz
  - Las claves que se insertan, siempre van en un nodo hoja.
    - Por ser perfectamente equilibrado, toda hoja está al mismo nivel.
- Pasos *del algoritmo para Insertar* una nueva clave:
  1. Se **Busca** la clave a insertar en el Árbol. Para lo cual se desciende por el camino de búsqueda hasta una hoja.
  2. Si no está en el árbol Entonces Empieza la **Inserción**.
  3. ¿Está Llena la Página? (actual->cuenta == m -1)
    - Hay Lugar (cuenta < m -1) :
      - Inserta en ese Nodo. Actualiza Cuenta. Fin proceso
    - Se llenó: No se puede insertar allí:
      - Se **divide** la Página en 2 Paginas al mismo nivel que todas las demás, extrayendo la clave mediana (para m=5 es la clave[3])
      - Con esta Mediana, se sube por el camino de Búsqueda y se comienza el proceso desde el paso 1 nuevamente.
- Esta proceso de ascensión de la clave mediana puede llegar hasta el nodo raíz, que también se partirá y su Mediana, subiendo, será la nueva raíz de todo el árbol B.

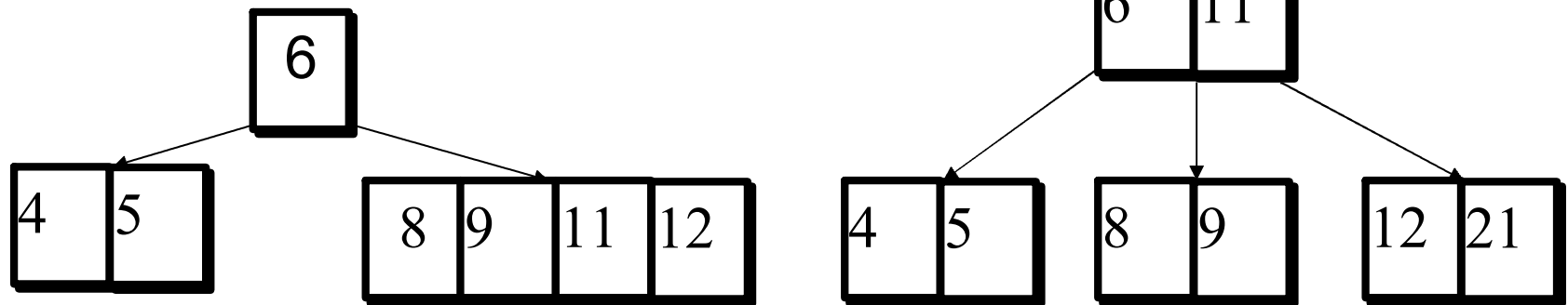
# Ejemplo de Inserción en Árbol B m=5

Secuencia de Inserción: 6, 11, 5, 4, 8, 9, 12, 21



1ro: 6, 11, 5, 4

2do: 8



3ro: 9, 12

4to: 21

Continuar insertando: 14, 10, 19, 28, 3, 17, 32, 15, 16, 26, 27. (Respuesta en pag 487)

# Codificación de la Inserción en Árbol B

EL Algoritmo de Inserción se implementa con varias funciones Auxiliares.

- *Insertar()*: es la interfaz de operación.
- *Empujar()*: es la encargada de realizar efectivamente la inserción. Bajaré y Subiré por el camino de Búsqueda localizando el Punto de Inserción. Empujaré, cuando necesite, la Mediana hacia arriba.
- *buscarNodo()*: Determina la rama por donde bajar para encontrar la clave.
- *meterHoja()*: en caso de insertar una clave sin división de Página.
- *dividirNodo()*: Se crea un nuevo Nodo al que se desplazan las claves mayores de la Mediana y sus ramas, dejando en el original las claves menores.
  - Ascenderá la Mediana mediante el Argumento “&mediana”, que al retornar las llamadas recursivas, la inserta en el Nodo Antecedente.

## Codificación de Inserción en Árbol B: *insertar()*

- Es la Interface de Inserción.
- En caso de que la propagación llegue a la Raiz, creará una nueva.

```
void insertar(Pagina**raiz, tipoClave cl)
{
    int subeArriba;
    tipoClave mediana;
    Pagina *p,*nd;
    empujar(*raiz, cl, &subeArriba, &mediana, &nd);
    if (subeArriba) /*Crece de Nivel x la raiz, si la propagación llega a la cima*/
    {
        p = (Pagina*) malloc(sizeof(Pagina));
        p -> cuenta = 1;
        p -> claves[1] = mediana;
        p -> ramas[0] = *raiz;
        p -> ramas[1] = nd;
        *raiz = p;
    }
}
```

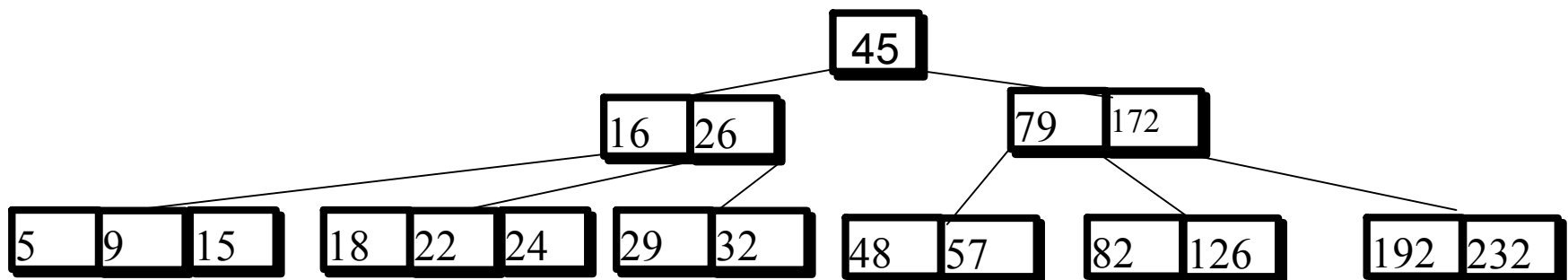
# Codificación de Inserción en Árbol B: *empujar()*

```
void empujar(Pagina* actual, tipoClave cl, int* subeArriba, tipoClave* mediana,
    Pagina** nuevo)
{
    int k; /*Posición dentro del espacio de claves de la pagina*/
    if (actual == NULL) /*Encontró una rama vacía en el camino de búsqueda, Activa los indicadores*/
    {
        *subeArriba = 1;
        *mediana = cl;
        *nuevo = NULL;
    }
    else {int esta= buscarNodo(actual,cl,&k); /*Baja hasta una rama vacía*/
        if (esta) {
            puts("\nClave duplicada");
            *subeArriba = 0; return;
        }
        empujar(actual->ramas[k], cl, subeArriba, mediana, nuevo);
        /* La Recursión devuelve el control; vuelve por el camino de búsqueda */
        if (*subeArriba) /*Será 1 cuando su rama=null => insertar pues no podemos seguir bajando*/
        {
            if (nodoLLeno(actual))
                dividirNodo(actual, *mediana, *nuevo, k, mediana, nuevo);
            else{
                *subeArriba = 0;
                meterHoja(actual, *mediana, *nuevo, k);
            }
        }
        /*Dejará de elevar Medianas cuando pueda, al menos una vez meterHoja*/
    }
}
```



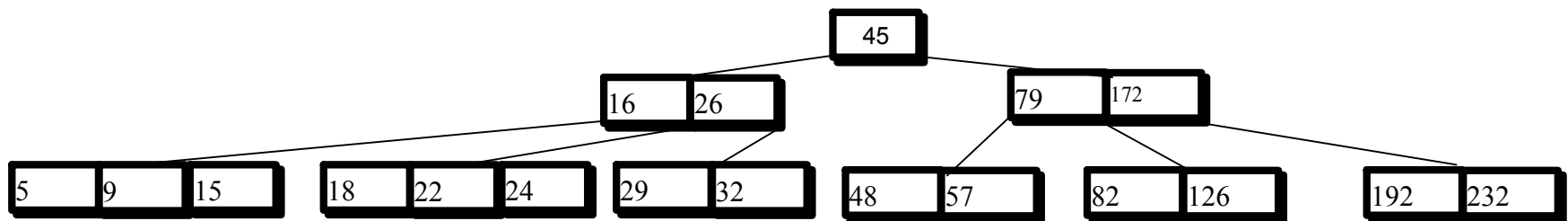
# Eliminación de una clave en Árbol B

- **Propiedad de los Árbol B:** Si una clave no está en una hoja, la clave **predecesora** o **sucesora** en el orden natural del Árbol, están en hojas.
  - Ej: 45 no es Hoja, pero el 32 y el 48 si.
  - Se procede a sustituirla por la clave sucesora o predecesora que si está en una hoja.
  - Pero si la hoja queda con menos del mínimo de claves, hay que mover claves para restablecer la estructura del Árbol B



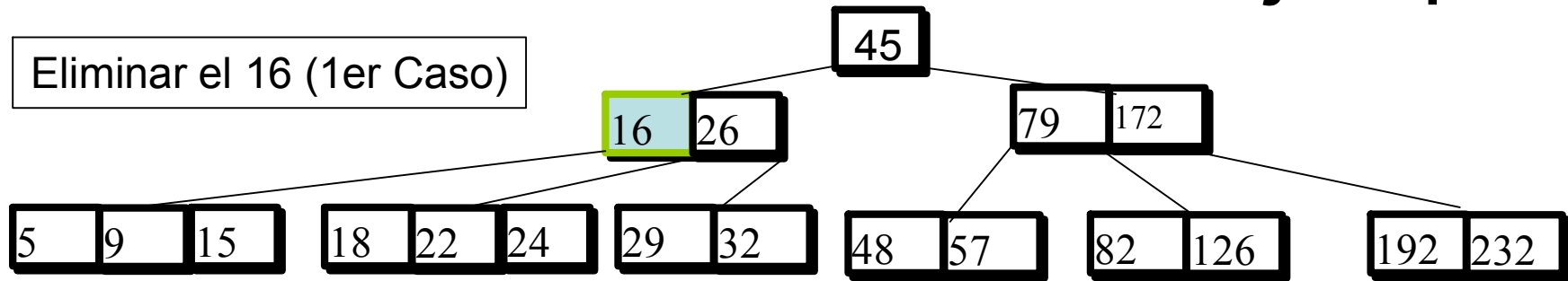
# Algoritmo de Eliminación de una clave en Árbol B

- 1er Caso: La Hoja del **Sucesor** tiene mas del mínimo
  - Buscar la Clave y su Sucesor.
  - Sustituir la clave de búsqueda con la clave sucesora que está en una hoja.
- 2do Caso: La Hoja del **Sucesor** tiene menos del mínimo, pero los Adyacentes mas.
  - Examinar los nodos adyacentes al Nodo a Borrar
  - Si un nodo Adyacente tiene mas claves que la mínima, se puede subir la clave elegida al nodo Antecedente.
  - Se baja del Nodo Antecedente la clave hacia el Nodo Problema
- 3er Caso: La Hoja de **Sucesor y Adyacente** tiene menos del mínimo
  - Tomar el Nodo Problema, el Nodo contiguo y la clave mediana de ambos (procedente del nodo Antecedente) y se los combina para formar un único Nodo.
  - Si deja al Nodo Antecedente menos del mínimo del claves, el proceso se propaga hacia arriba: *El limite este caso es bajar la Raiz: la altura del Árbol B disminuye.*



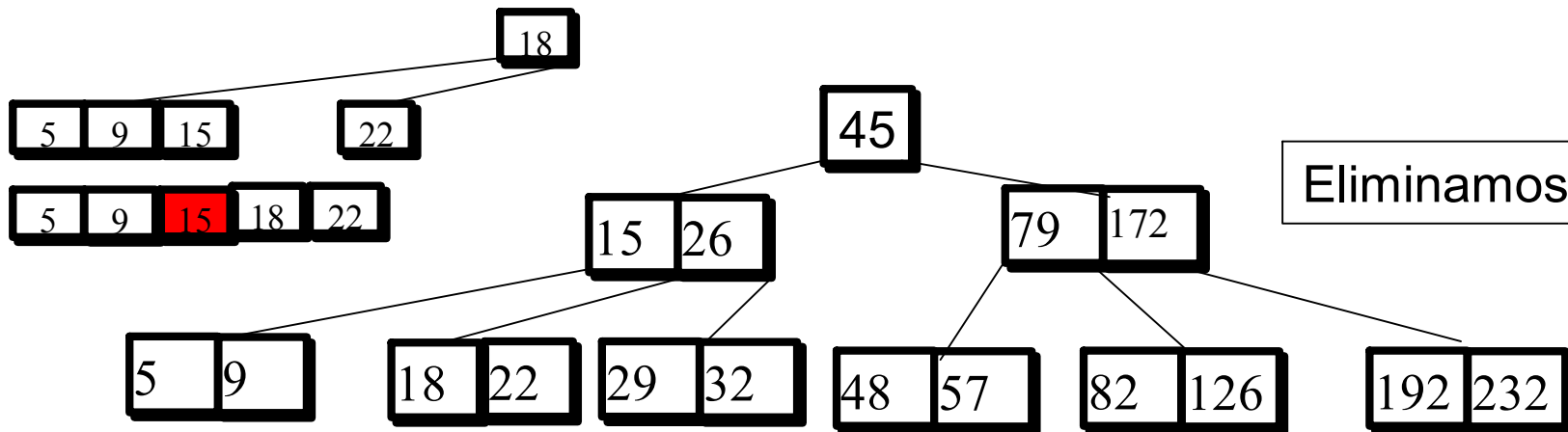
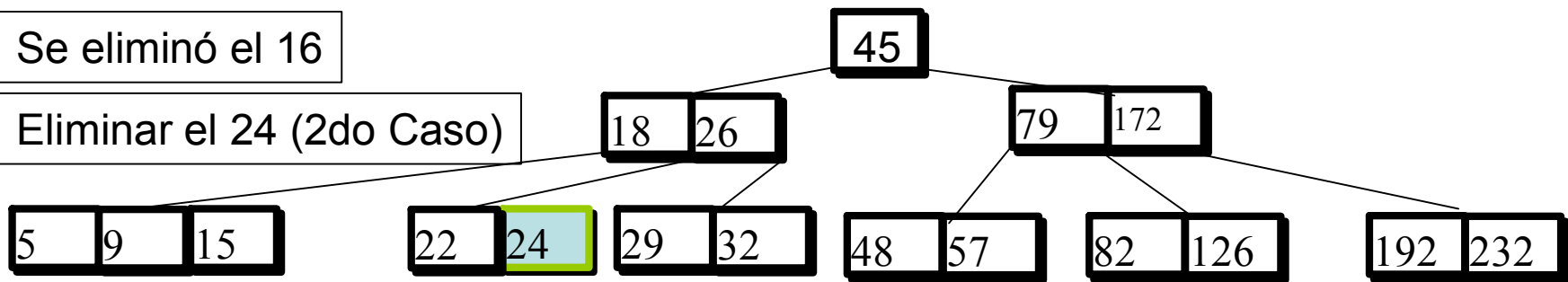
# Eliminación de un Árbol B. Ejemplo

Eliminar el 16 (1er Caso)



Se eliminó el 16

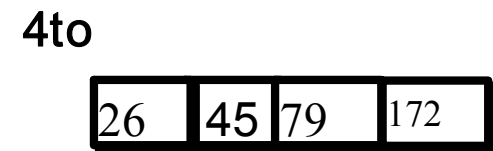
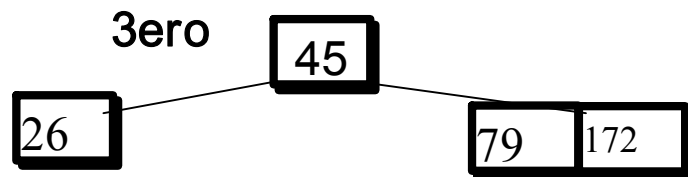
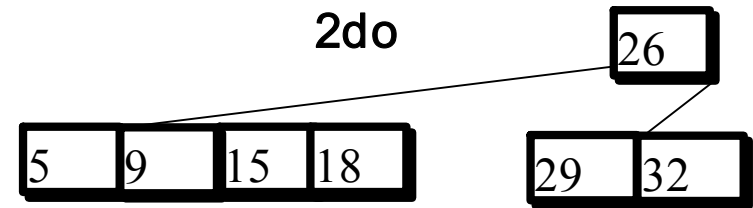
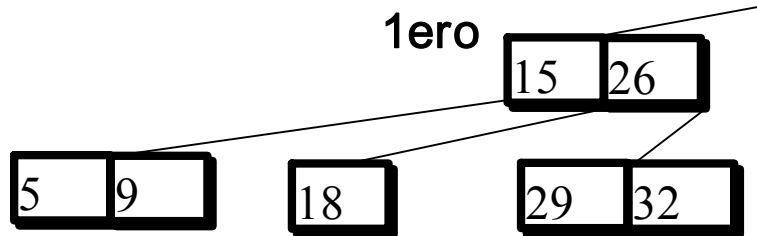
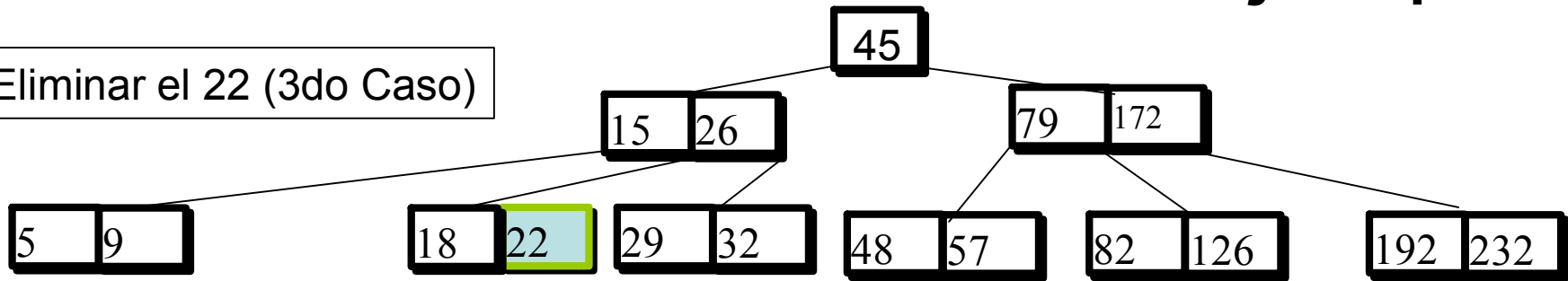
Eliminar el 24 (2do Caso)



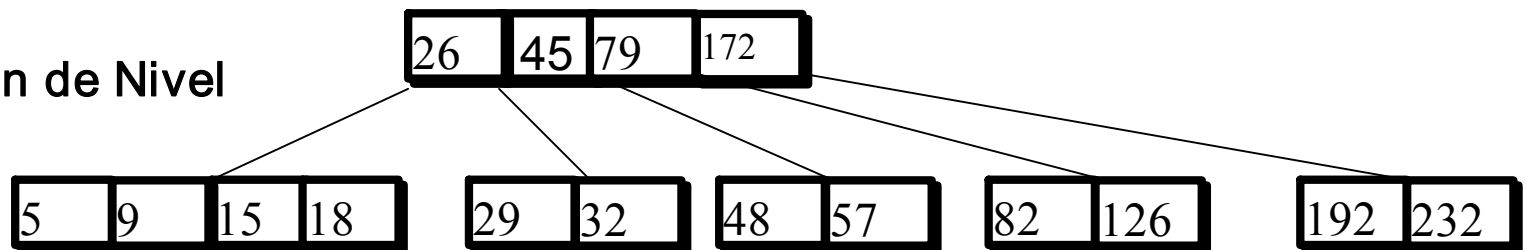
Eliminamos el 24

# Eliminación de un Árbol B. Ejemplo

Eliminar el 22 (3do Caso)



Resultado:  
Disminución de Nivel



# Codificación Eliminación Árbol B

- EL Algoritmo de Eliminación implementa varias funciones Auxiliares.
  - Eliminar(): interfaz de operación, pasa el control a ...
  - eliminarRegistro(): controla todo el proceso de borrado.
  - buscarNodo(): Determina la posición de la clave en el arbol.
  - sucesor(): encuentra la clave Menor, dentro del SubConjunto de Claves Mayores a la Clave a borrar, y hace la sustitución.
  - quitar(): si la clave esta en un nodo hoja, quita la clave de allí y termina el proceso.
  - restablecer(): restaura el orden del árbol B, de acuerdo al estado de sus Hermanos Derechos e Izquierdo.
  - moverIZQ() y moverDER(): apoya a restablecer en caso de los hermanos (IZQ o DER) tengan menos del mínimo.
  - combina(): si las claves son menores que el mínimo junta los nodos implicados.

# Codificación de la Eliminación en Árbol B: eliminar()

```
void eliminar(Pagina**raiz, tipoClave cl)
{
    int encontrado;
    eliminarRegistro(*raiz, cl, &encontrado);
    if (encontrado)
    {
        printf("Clave %d eliminada\n",cl);
        if ((*raiz) -> cuenta == 0)
        {
            /* La raiz está vacía, libera el nodo y se establece la nueva raíz */
            Pagina* p = *raiz;
            *raiz = (*raiz) -> ramas[0];
            free(p);
        }
    }
    else
        puts("La clave no se encuentra en el árbol\n");
}
```

## Codificación de la Eliminación en Árbol B: eliminaRegistro()

```
void eliminarRegistro(Pagina* actual, tipoClave cl, int* encontrado)
{
    int k;
    if (actual != NULL)
    {
        *encontrado = buscarNodo(actual, cl, &k); /* busca la hoja con la clave*/
        if (*encontrado)
        {
            if (actual->ramas[k-1] == NULL) /* es un nodo hoja */
                quitar(actual, k); /* quita la clave de la hoja */
            else {
                sucesor(actual, k); /* encuentra la clave sucesora y hace el swap*/
                eliminarRegistro(actual->ramas[k], actual->claves[k], encontrado);
                /* Como va con la Pagina Descendente, buscará hasta encontrar la pagina hoja
                   con el que dato Sucesor (que ya se cambio) y lo quitará*/
            }
        }
    }
    else eliminarRegistro(actual->ramas[k], cl, encontrado);
    /* Las llamadas recursivas devuelven control en este punto. Se comprueba el número de claves del nodo
       descendiente, desde el nodo actual en la ruta de búsqueda seguida. */
    if (actual->ramas[k] != NULL)
        if (actual->ramas[k]->cuenta < m/2) restablecer(actual, k);
}
else *encontrado = 0;
}
```

# Trabajos Prácticos Unidad 8

## COMO MINIMO, REALIZAR:

- De la Bibliografía
  - Del Capitulo 16:
    - Ejercicios: 16.1, 16.2, 16.8
- Problemas:
  - Implementar con Árboles B los problemas: 14.2, 14.3, 14.7
- Complementarios
  - Diseñar un Árbol B, sin utilizar las estructuras estudiadas.
    - Proponer un TAD de al menos 5 operaciones .
    - Implementar al menos 2.