Técnico Informático

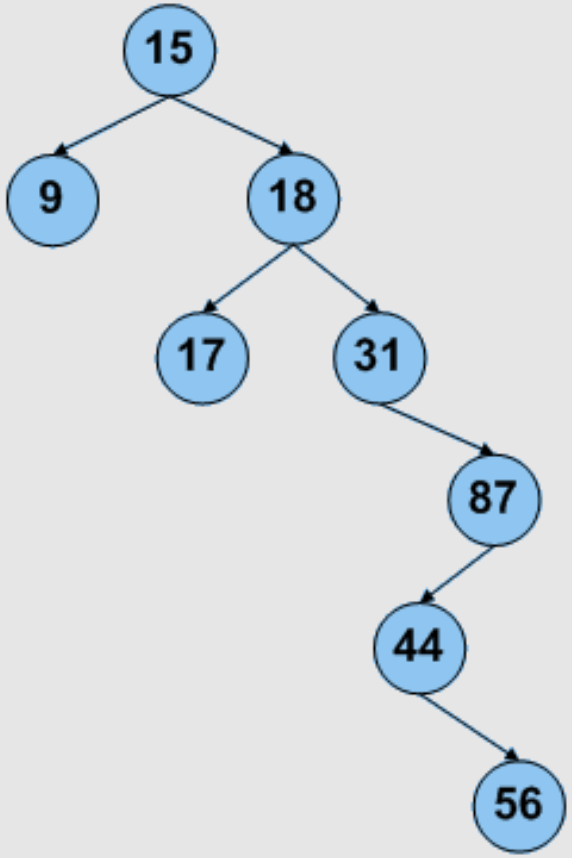
ESTRUCTURA DE DATOS

UNIDAD VI: ÁRBOLES AVL Y ÁRBOLES B

Escuela de Minas "Dr. Horacio Carrillo"
Universidad Nacional de Jujuy

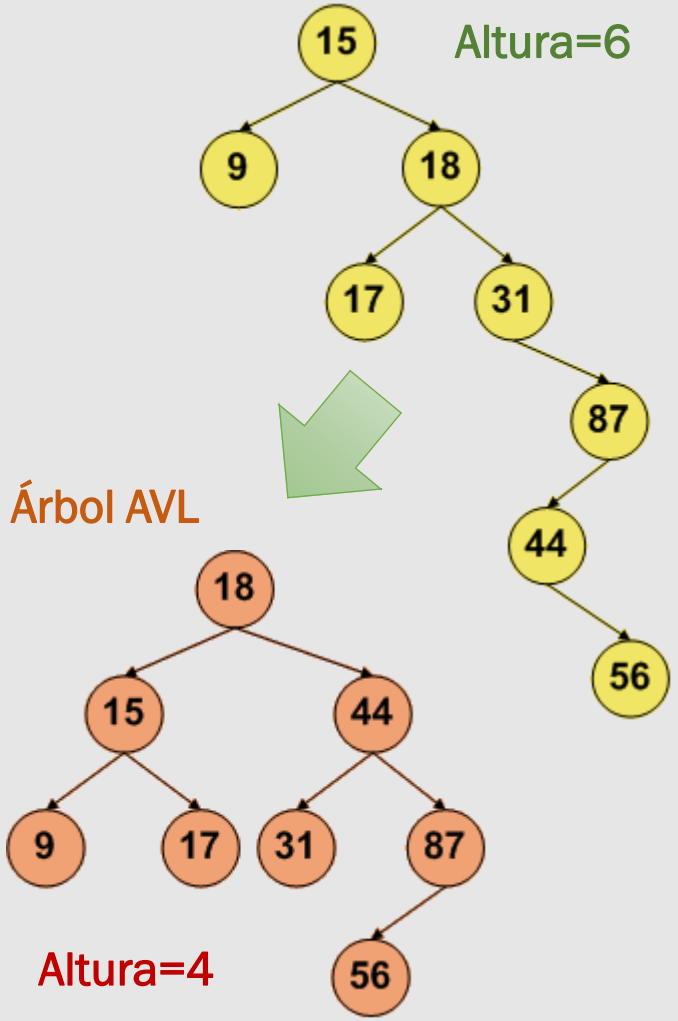
Árboles Binarios

- Los árboles binarios de búsqueda permiten organizar los datos de forma tal que las operaciones de inserción y búsqueda resultan relativamente sencillas.
- Cuando las ramas de un árbol crecen de forma desproporcionada, las operaciones sobre éste implican mayores recorridos de la estructura.



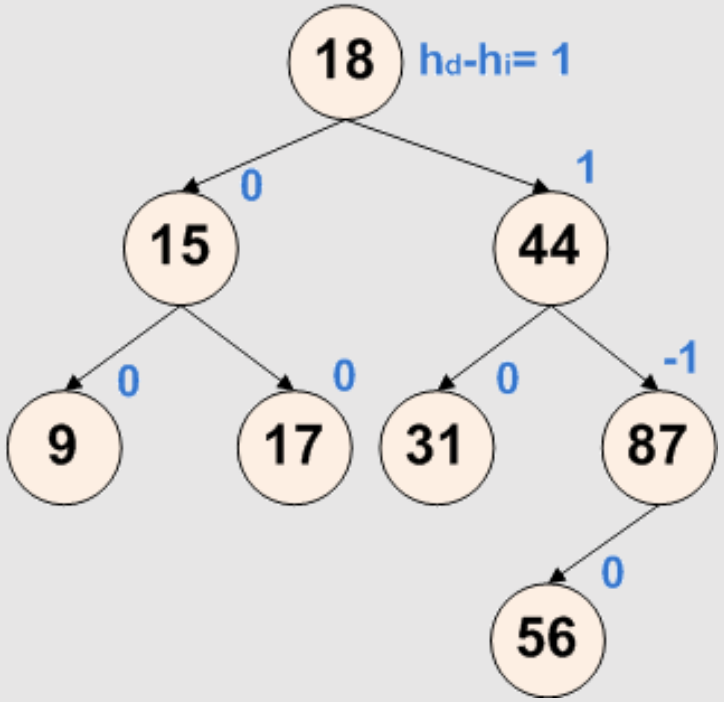
Árboles AVL (1)

- Adelson-Velski y Landis propusieron resolver el problema de crecimiento de los árboles binarios a través de los denominados árboles balanceados o AVL.



Árboles AVL (2)

- Un árbol binario se dice que está balanceado si y sólo si en cada nodo las alturas de sus 2 subárboles difieren como máximo en 1 (h_d-h_i pertenece al intervalo [-1,1]).



Definición

```
typedef struct tnodo *pnodo
typedef struct tnodo
          {
           tipo_dato clave; //clave
           pnodo izq; //puntero a nodo
           pnodo der; //puntero a nodo
           int balance; //dif. hd y hi
           }
```

Inserción AVL (1)

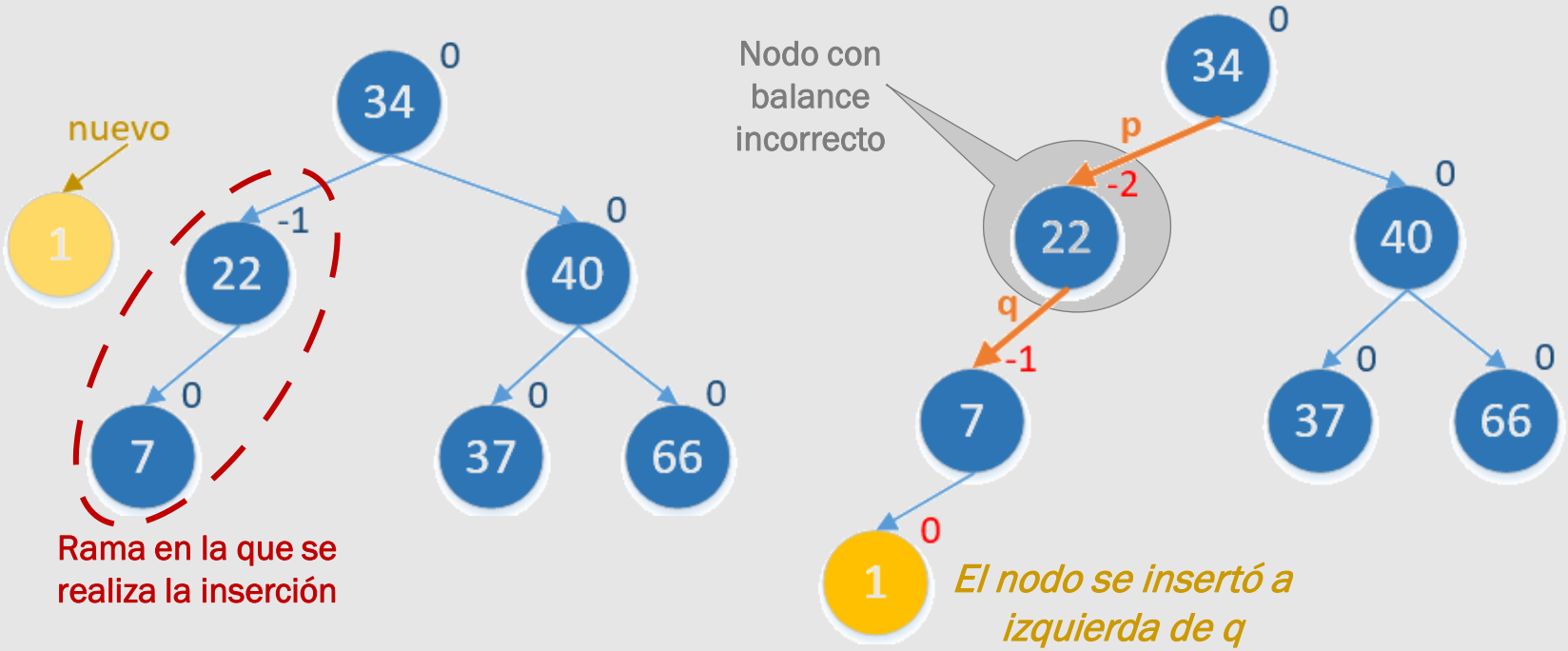
- La inserción en árboles AVL sigue la estrategia vista para árboles binarios de búsqueda, incorporando además un **criterio de equilibrio.**
- En base a este criterio, **tras cada inserción** se verifica el balance de los nodos del árbol, que debe ser **-1**, **0** o **1** para mantener un árbol AVL.
- Si se detecta que el balance no es correcto, el árbol se manipula reubicando nodos.

Inserción AVL (2)

- La inserción puede presentar 4 situaciones en las que es necesario **rebalancear** el árbol:
  - Left-Left (LL) o Izquierda-Izquierda
  - Left-Right (LR) o Izquierda-Derecha
  - Right-Right (RR) o Derecha-Derecha
  - Right-Left (RL) o Derecha-Izquierda

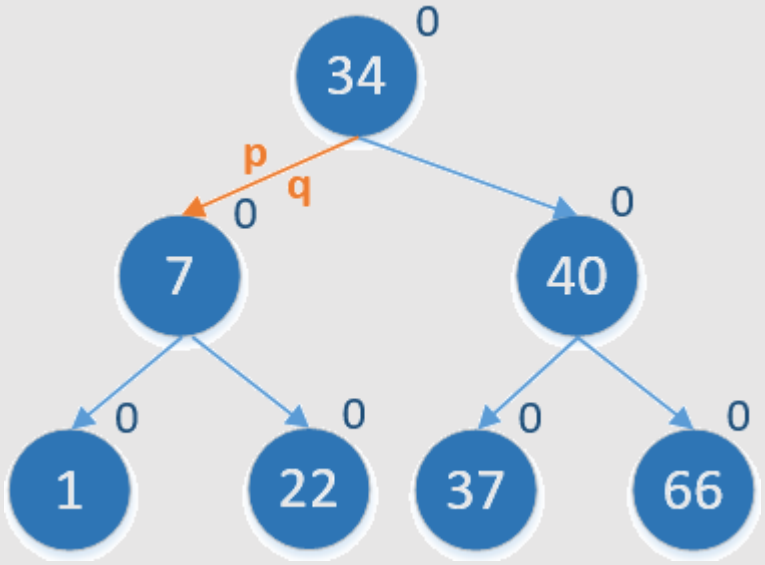
Inserción AVL (3)

- Rebalance LL (Izquierda-Izquierda)
  - Suponiendo que se inserta el valor 1 en el siguiente árbol AVL.



Inserción AVL (4)

- Rebalance LL (Izquierda-Izquierda)
  - Los nodos apuntados por ***p*** y ***q*** se reubican para restablecer el árbol AVL

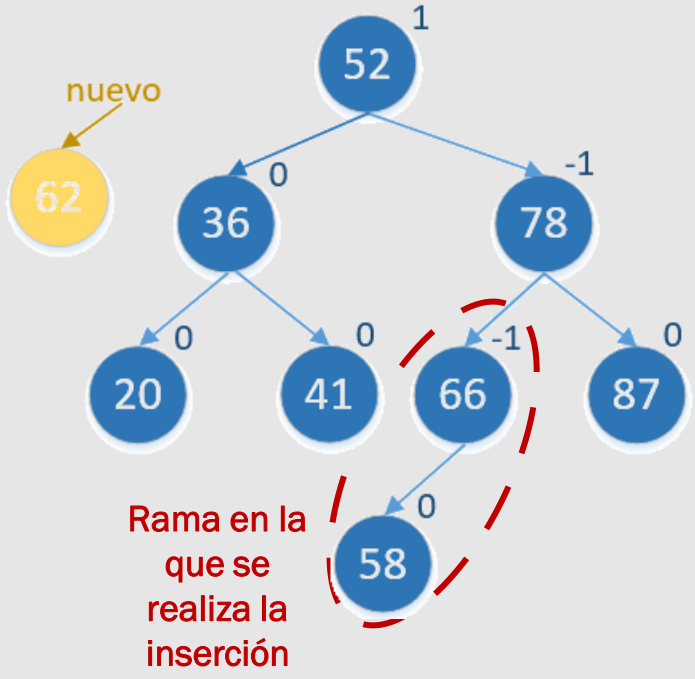


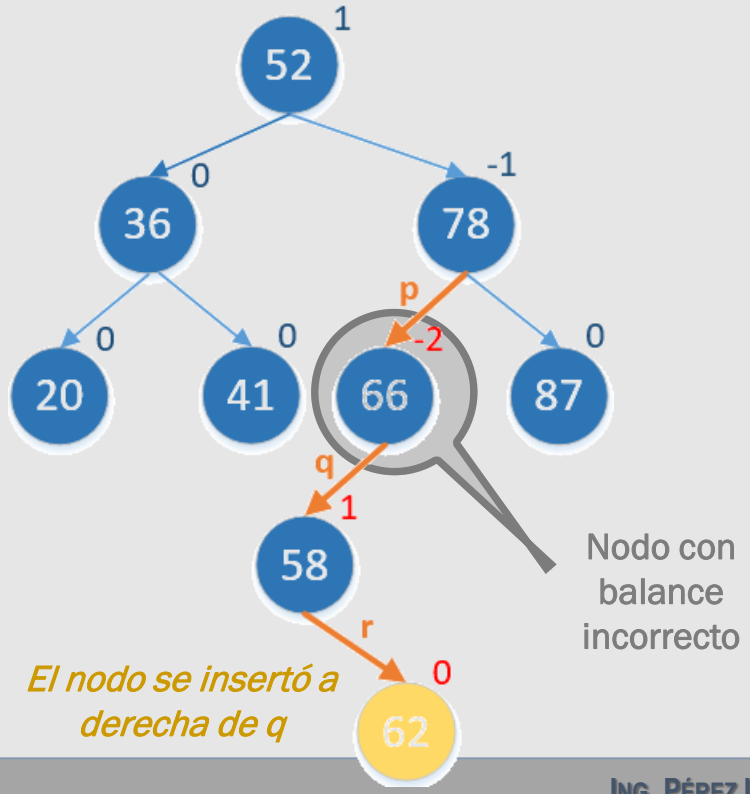
Reubicando nodos

```
p->izq=q->der;
q->der=p;
p=q;
```

Inserción AVL (5)

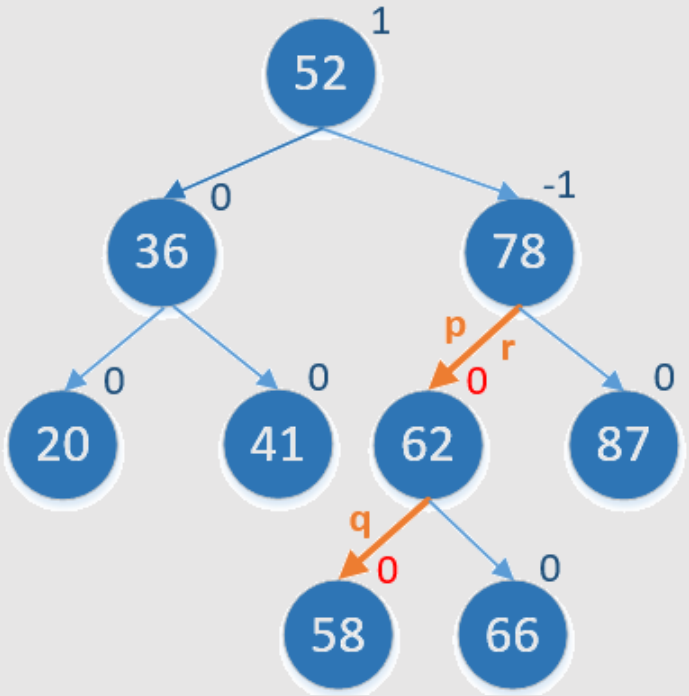
- Rebalance LR (Izquierda-Derecha)
  - Suponiendo que se inserta el valor 62 en el siguiente árbol AVL.





Inserción AVL (6)

- Rebalance LR (Izquierda-Derecha)
  - Los nodos apuntados por *p*, *q* y *r* se reubican para restablecer el árbol AVL



Reubicando nodos

```
q->der=r->izq;
p->izq=r->der;
r->izq=q;
r->der=p;
p=r;
```

Inserción AVL (7)

- Rebalance RR (Derecha-Derecha) y RL (Derecha-Izquierda)
  - Los casos de rebalanceo por derecha son análogos a los de rebalanceo por izquierda.

Caso RR

```
p->der=q->izq;
q->izq=p;
p=q;
```

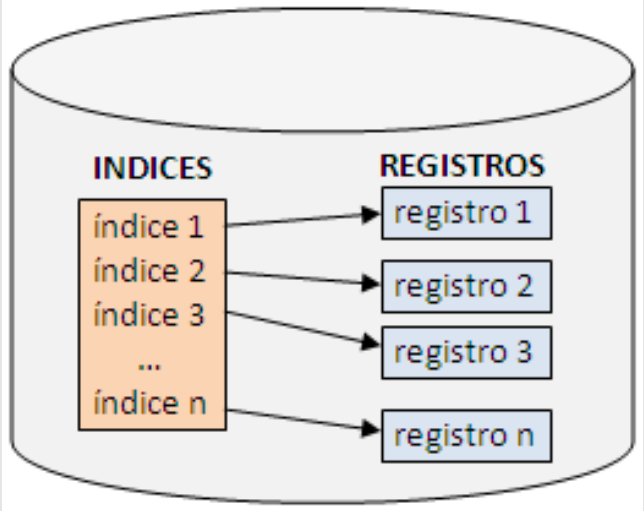
Caso RL

```
p->der=r->izq;
q->izq=r->der;
r->izq=p;
r->der=q;
p=r;
```

Árboles B (1)

- ***En el manejo de grandes volúmenes de información, siempre estuvo presente la necesidad de HACER EFICIENTE EL PROCESO DE BÚSQUEDA.***

- ***Problema***: acceso rápido y eficiente a grandes volúmenes de datos indizados almacenados en memoria secundaria. Por ejemplo, las bases de datos de los buscadores como *Google*.



Árboles B (2)

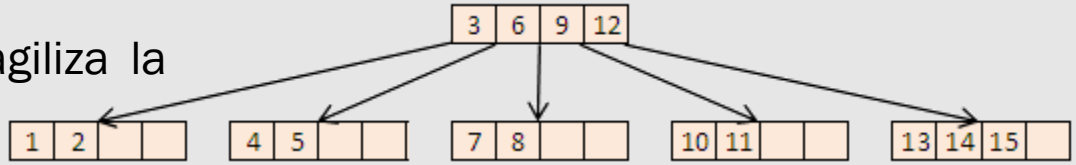
- *Solución 1:* los árboles binarios de búsqueda estructuran la información de modo que la búsqueda se realice rápidamente .

  ¿qué ocurre si el volumen de datos es muy grande?

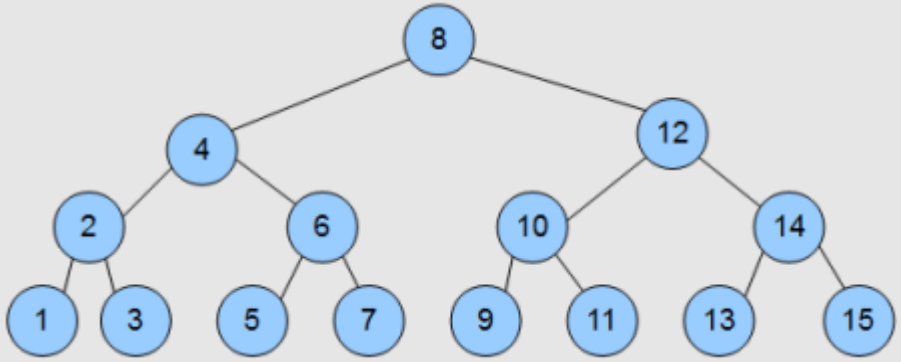
- *Solución 2*: a principios de los años 70, Rudolf Bayer y Edward McCreight propusieron los árboles B (estructuras no lineales) como solución al mantenimiento de índices en almacenamiento secundario.

Árboles B (3)

- *Ventajas*
  - Poseen una estructura que agiliza la búsqueda.
  - Aprovechan mejor el espacio de almacenamiento secundario.
  - Tienen menor profundidad que sus equivalentes binarios.
  - Reducen el número de accesos al dispositivo.



Árbol B de orden 2 con 2 niveles

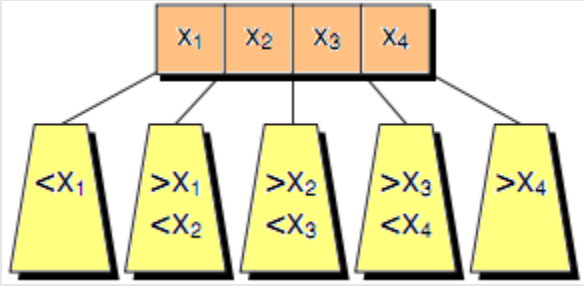


Árbol binario de búsqueda con 4 niveles

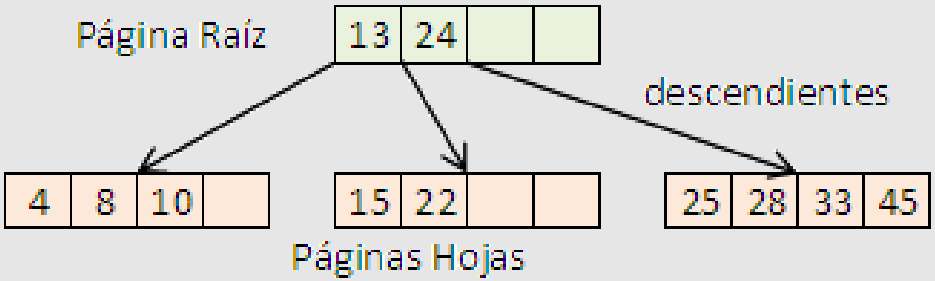
Definición (1)

Un árbol B de **orden** *n* es un *árbol de búsqueda* que satisface las siguientes propiedades:

- Cada página tiene como máximo *2 n* claves.
- Cada página tiene como mínimo *n* claves, salvo la raíz que puede tener sólo 1.
- Una página con *m* claves tiene *m+1* descendientes o *ninguno* (página hoja).
- Todas las **páginas hojas** aparecen al **mismo nivel** (condición de balance).

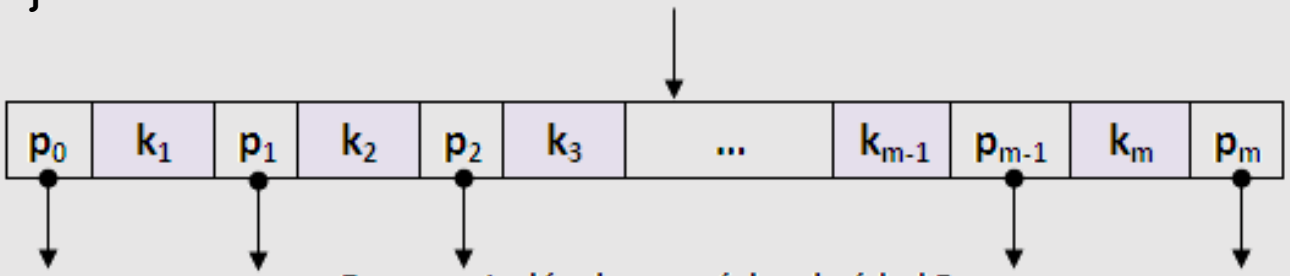


ÁRBOL B DE ORDEN 2



Definición (2)

```
typedef struct pagina *ppagina
typedef struct pagina
        {
         tipoClave claves[k]; //claves (k=2*n)
         ppagina ramas[k+1]; //punteros a páginas
         int cuenta;  //número de claves de la página
        }
```



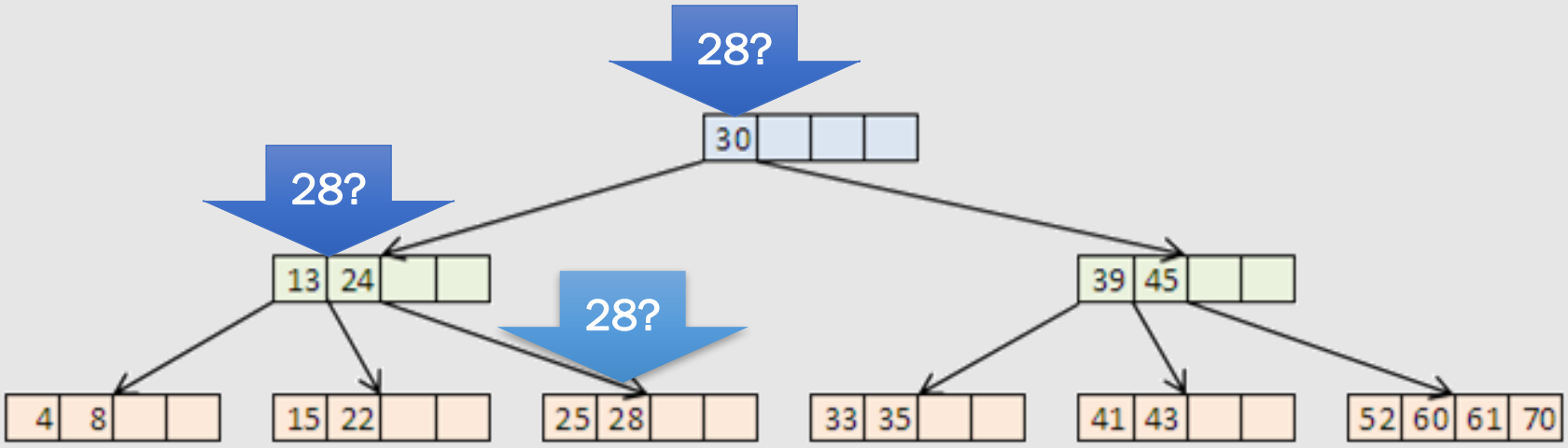
Representación de una página de árbol B

Operaciones Básicas

- Búsqueda
- Inserción
- Eliminación
  - Caso 1: Eliminar un elemento de una página hoja.
    A. La hoja tiene más de *n* claves
    B. La hoja tiene *n* claves
  - Caso 2: Eliminar un elemento de una página interna.
    - ✓ Estrategia Mayor de los Menores
    - ✓ Estrategia Menor de los Mayores

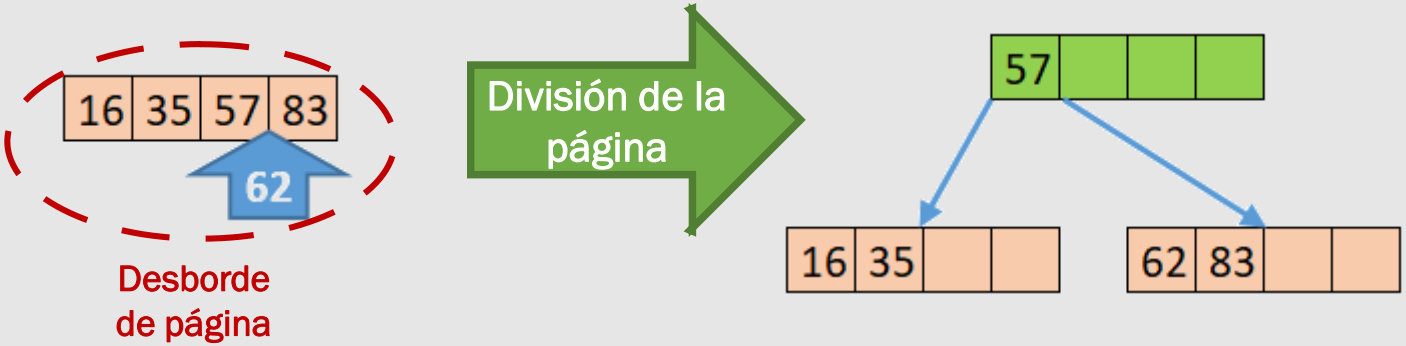
Búsqueda

- El proceso de búsqueda se inicia en la raíz del árbol, explorando las páginas de los siguientes niveles según el valor de las claves almacenadas. La búsqueda finaliza en una página hoja.
- Por ej., si se busca el valor 28 en el siguiente árbol B.

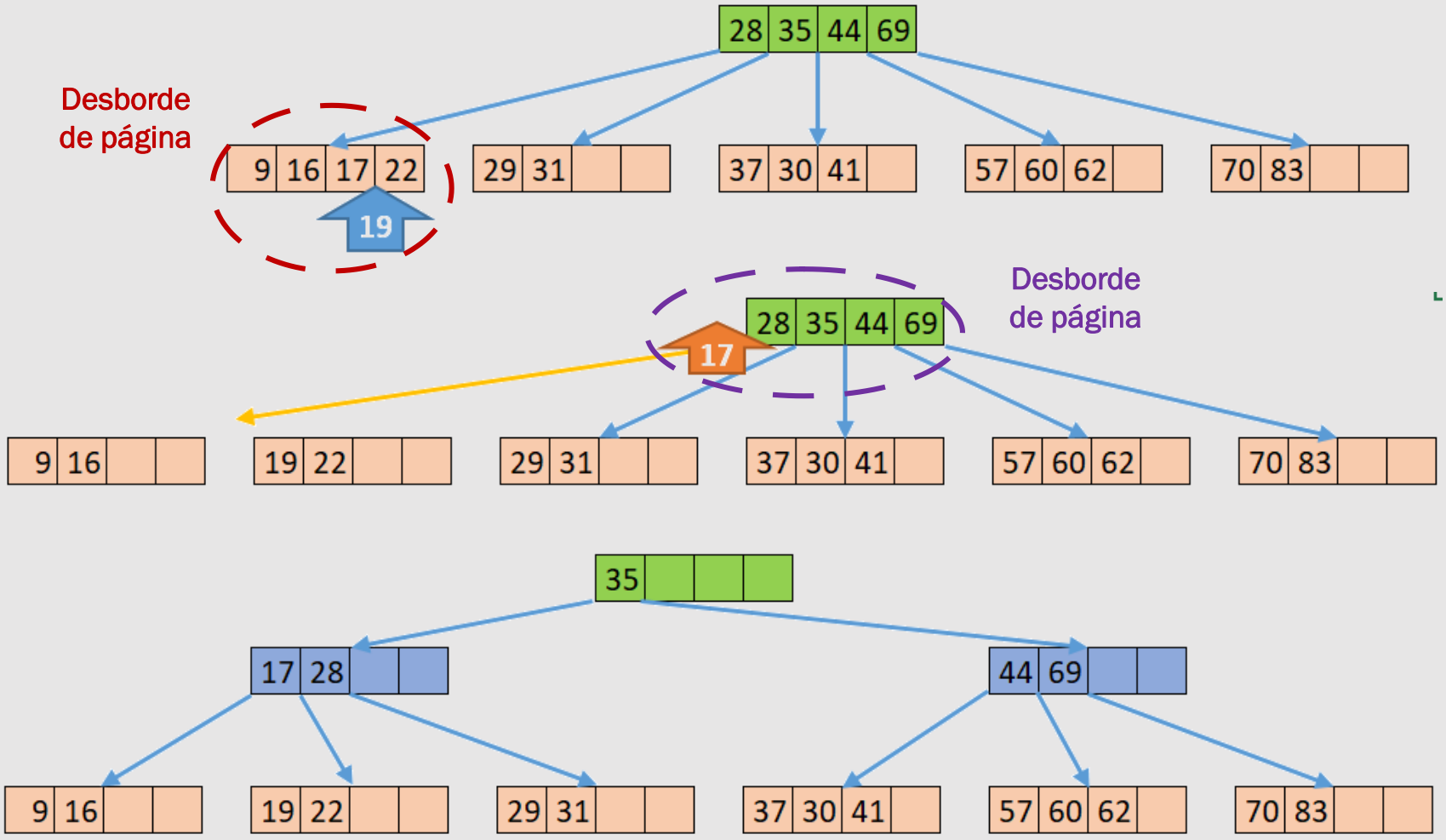


Inserción (1)

- Para insertar un nuevo elemento en un árbol B, éste se recorre hasta encontrar la página en la que debe almacenarse.
- Si la página está completa se produce un **desborde de página** y es necesario dividir ésta en 2 nuevas páginas, llevando la clave central a la página inmediata de nivel superior.



Inserción (2)

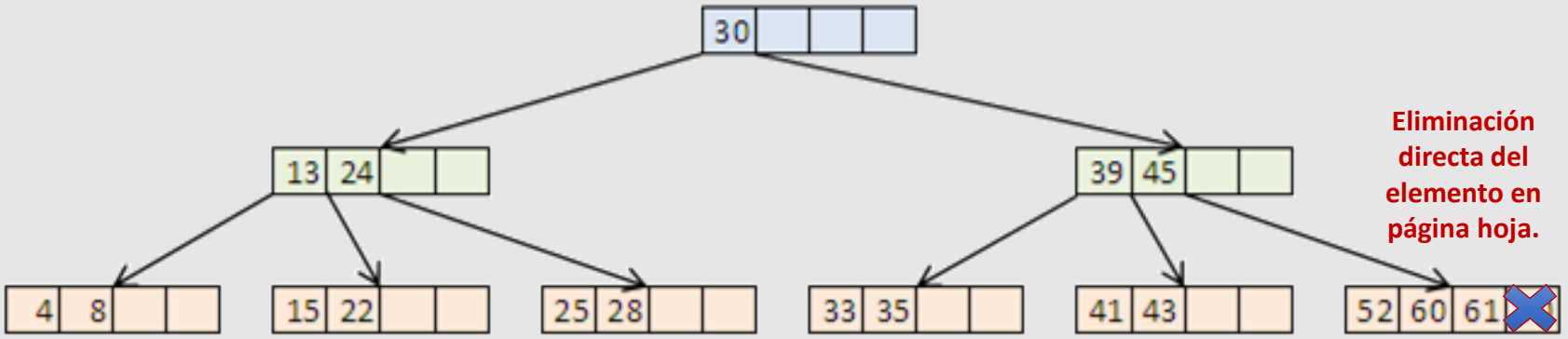

28 35 44 69
Desborde de página
9 16 17 22
19
29 31
37 30 41
57 60 62
70 83
Desborde de página
28 35 44 69
17
9 16
19 22
29 31
37 30 41
57 60 62
70 83
35
17 28
44 69
9 16
19 22
29 31
37 30 41
57 60 62
70 83

Eliminación

- Eliminación
  - Caso 1: Eliminar un elemento de una página hoja.
    A. La hoja tiene más de *n* claves
    B. La hoja tiene *n* claves
  - Caso 2: Eliminar un elemento de una página interna.
    - ✓ Estrategia Mayor de los Menores
    - ✓ Estrategia Menor de los Mayores

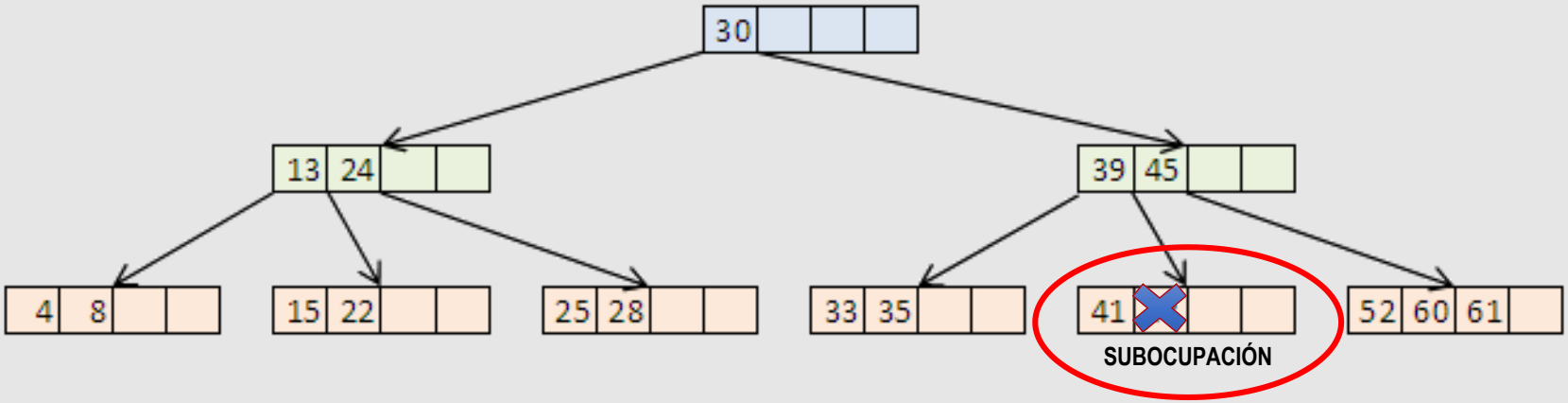
Eliminación. Caso 1A

- Caso 1.A: Eliminar un elemento que pertenece a una página hoja, con más de *n* elementos.



Eliminación. Caso 1B (1)

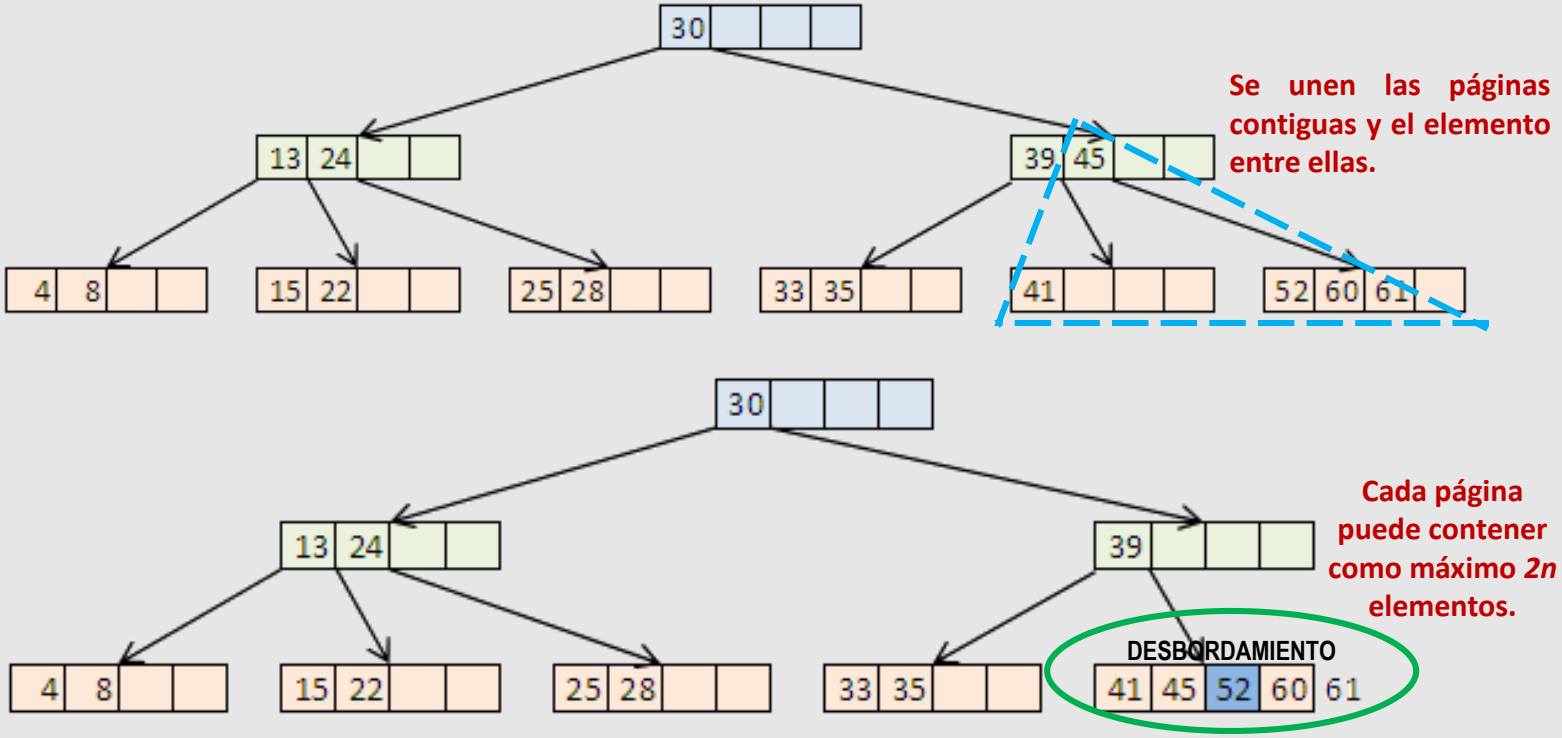
- Caso 1.B: Eliminar un elemento que pertenece a una página hoja, con *n* elementos.



En cada página debe haber al menos *n* elementos, de lo contrario se produce subocupación.

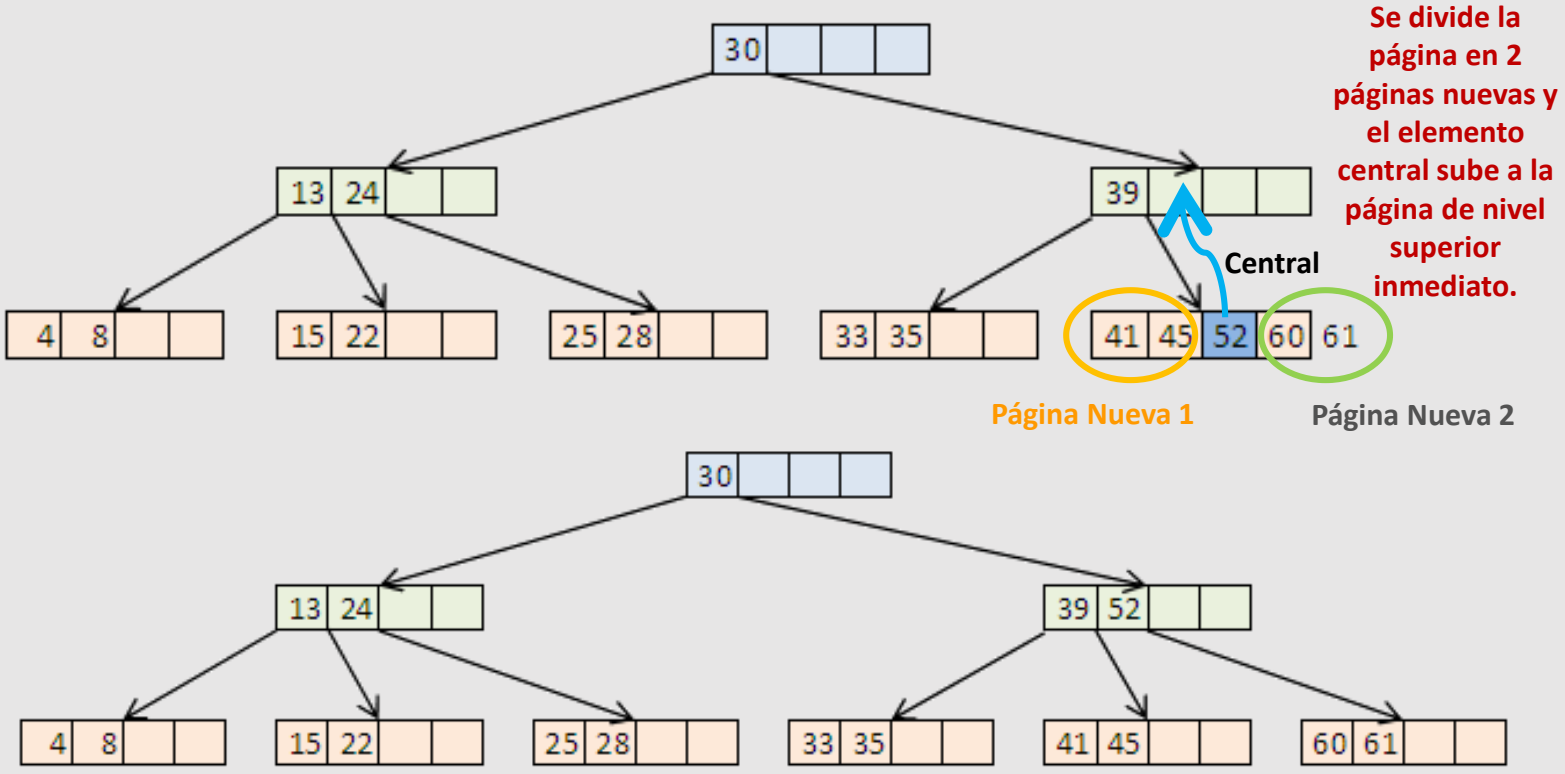
Eliminación. Caso 1B (2)

- ¿Cómo se resuelve la subocupación?



Eliminación. Caso 1B (3)

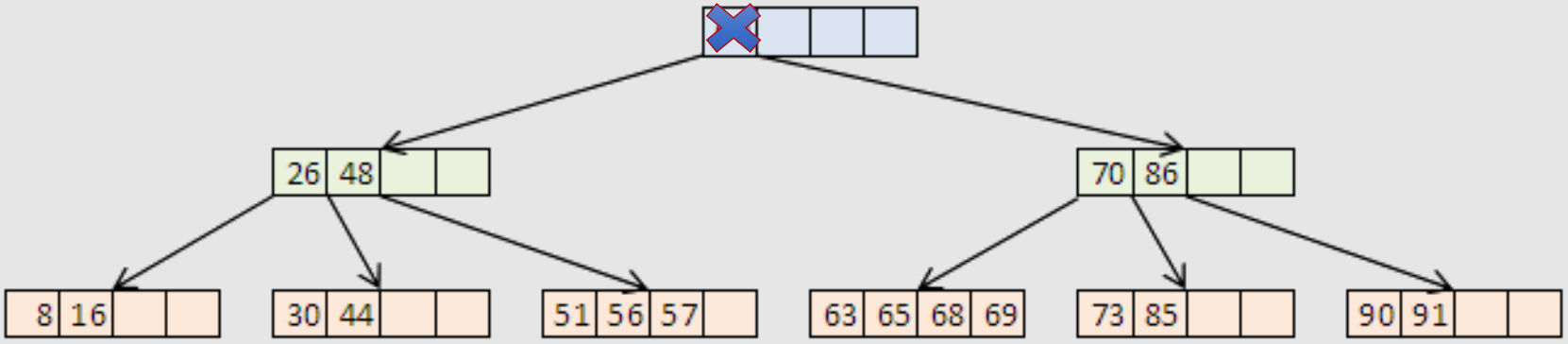
- ¿Cómo se resuelve el desbordamiento?



Eliminación. Caso 2 (1)

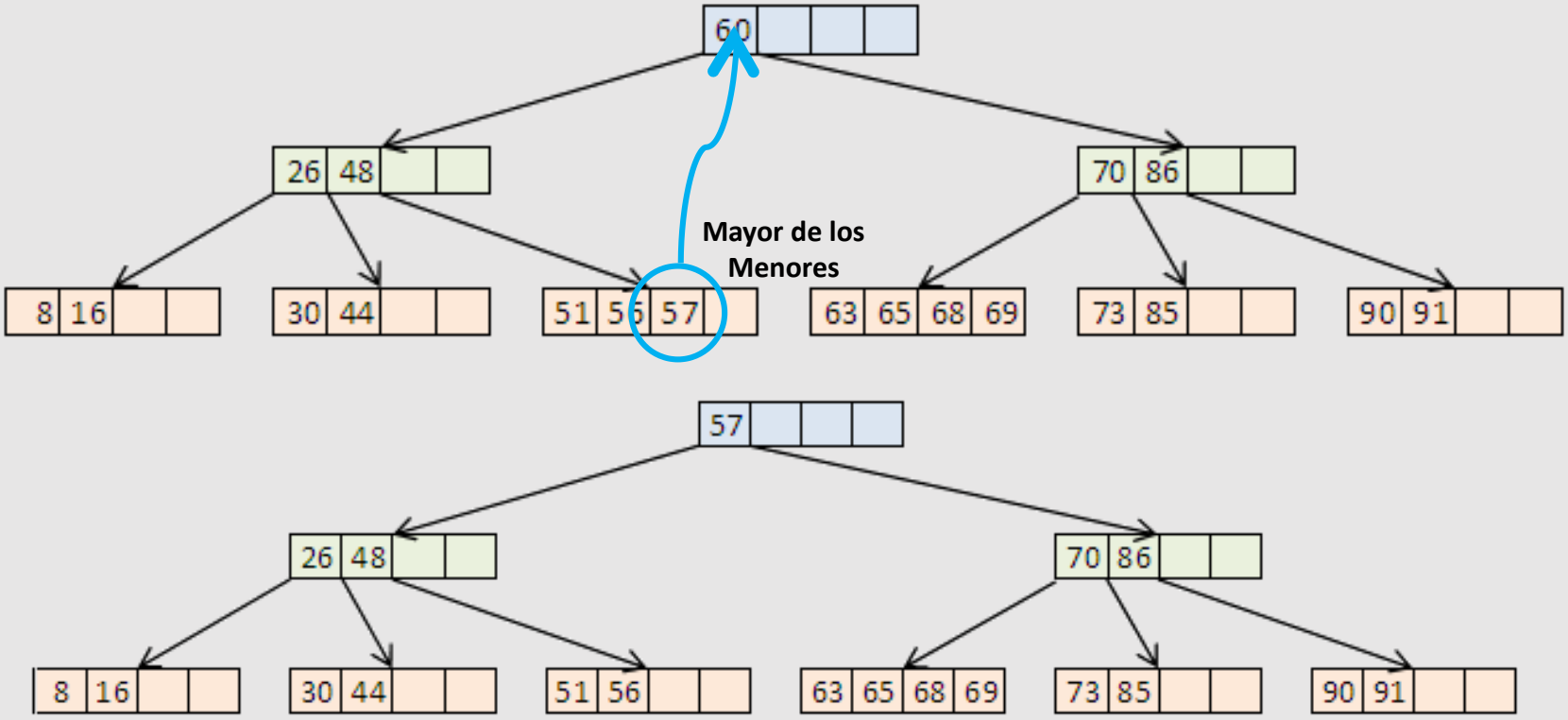
- Caso 2: Eliminar un elemento que pertenece a una página interior.

  En este caso el elemento a eliminar se sustituye por otro elemento del árbol.



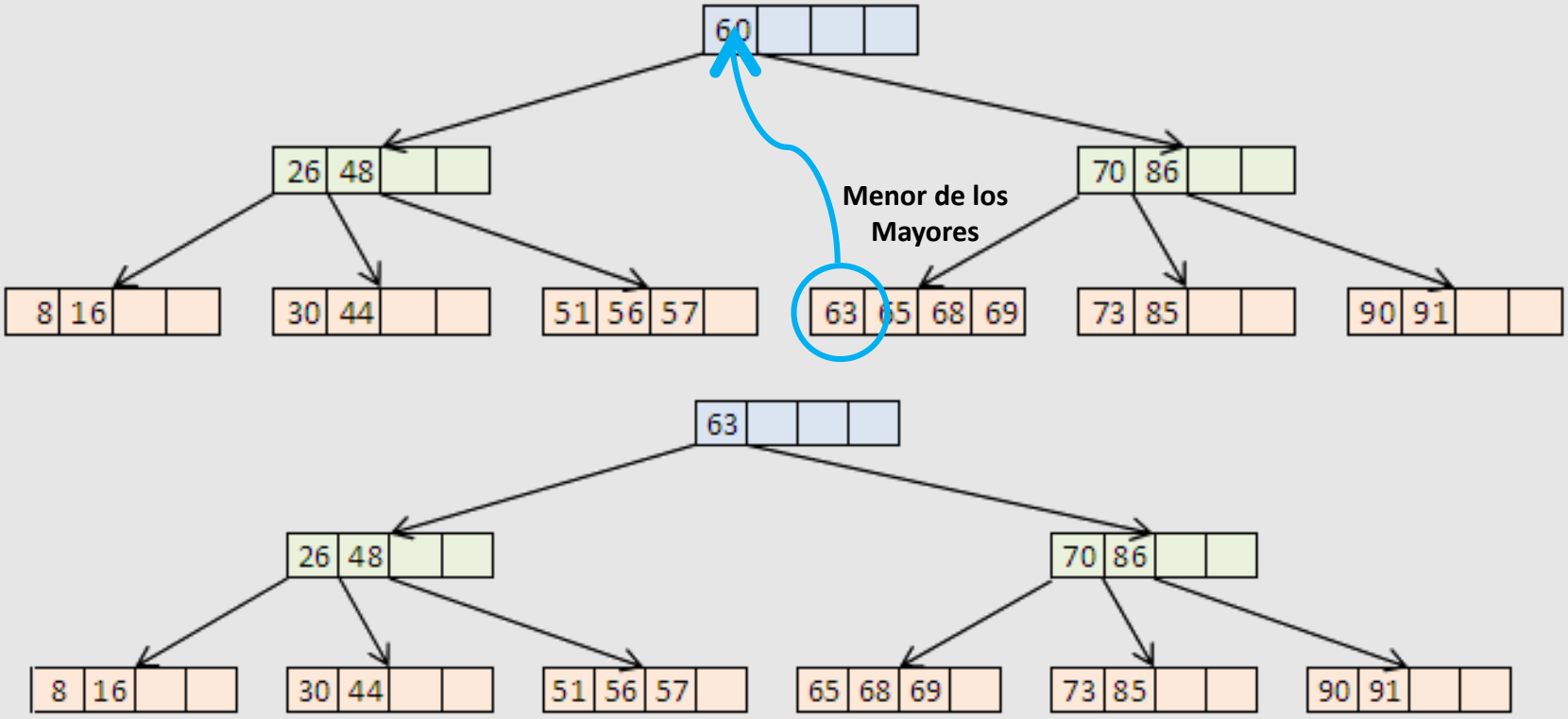
Eliminación. Caso 2 (2)

- Sustituir por el Mayor de los Menores



Eliminación. Caso 2 (3)

- Sustituir por el Menor de los Mayores



Resumen

- Un árbol B cumple con el **criterio de crecimiento** establecido por Bayer y McCreight .
- Las operaciones definidas mantienen las **propiedades del árbol B**.
- Las estrategias de **Mayor de los Menores** y **Menor de los Mayores** vistas para árboles binarios también se aplican a árboles B.
- La **subocupación** y el **desbordamiento** se resuelven rebalanceando.
- Cada página aprovecha al menos el 50% del espacio de almacenamiento, con lo que se requieren **menos accesos a páginas** para ubicar un elemento.

Bibliografía

- Hernández, Roberto *et al*. Estructuras de Datos y Algoritmos. Prentice Hall. 2001.
- Joyanes Aguilar *et al*. Estructuras de Datos en C++. Mc Graw Hill. 2007.
- De Giusti, Armando *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta. 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.