Tema 5
Árboles IV: Árboles equilibrados B y B+

1. Árboles B (representación, inserción y eliminación)
2. Tipos de árboles B: B^* y B^+
3. Árboles B^+ (representación, inserción y eliminación)

Problema:

- Estás escribiendo un programa para gestionar los datos de una guía telefónica de Madrid (digamos 2.000.000 teléfonos*).
- Asumamos que cada dato se guarda como un registro de 512 bytes.

Tamaño total = 2.000.000 * 512 bytes = 1 GByte

- Demasiado grande para guardarlo en memorial principal.
  - Necesitamos guardarlo en disco (memoria externa).

¿Qué estructura de datos elegirías?

**Realmente son muchos más, pero vamos a coger este número por conveniencia para las operaciones posteriores.*

Solución

- **¿Qué tamaño de ABB necesitaríamos?**
  - Usamos la estructura de bloques del disco para guardar la información.
  - Si el bloque de disco es de 8KB, el número de bloques que necesitamos es: 1 GB/ 8 KBytes por bloque ≈ 125.000 bloques

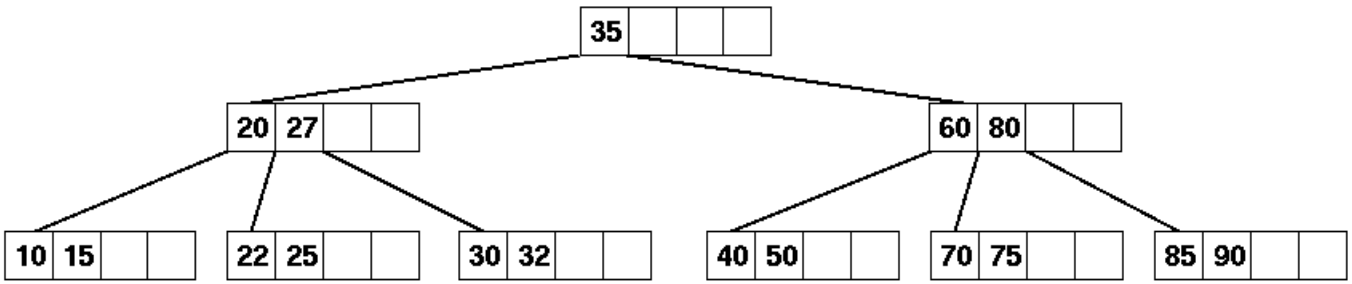
    que corresponde con 17 niveles en un ABB, ya que $2^{17}-1 \approx 125.000$ bloques
  - Si cada bloque se corresponde con un nodo, y este guarda aprox. 16 registros, tenemos 17 niveles en un ABB con 16 registros por nodo.
- **¿Qué tiempo llevaría en el peor caso encontrar un dato?**
  - En el peor caso, 17 niveles =>17 accesos a disco para encontrar un dato
  - 17 accesos* 70 ms/acceso ≈ 1,2 s (unas 10.000 veces más lento que el acceso a memoria principal, y peor si analizamos el coste de inserción o borrado)
- **¿Existe alguna forma de tener búsqueda, inserción y borrado rápidos teniendo los datos en disco? SÍ, CON UN ÁRBOL B**

Definición de árbol B de orden m

- Generalización de los árboles equilibrados para almacenar y recuperar información en medios externos.
- **Página**: unidad a la que se accede en bloque, almacena la información de un grupo de nodos identificados por claves.
  - Todas las páginas hoja están en el mismo nivel.
  - Número de ramas:
    - Todas las páginas internas tienen entre *m*/2 (redondeado al máximo entero) y *m* ramas (no vacías).
    - La raíz tiene entre 0 (si es la única página), 2 o un máximo de m ramas.
  - Número de claves:
    - en cada página interna es el número de sus ramas-1. Las claves de una página dividen las claves de sus ramas como un árbol de búsqueda.

Árbol B de orden 5
Página interna:
- Ramas: 3-4-5
- Claves: 2-3-4

Raíz:
- Ramas: 0, 2-3-4-5
- Claves: 1-2-3-4



1. Árboles B

Volvemos a nuestro problema de la guía telefónica, vamos a calcular qué altura de árbol B necesitamos para almacenar los 2 millones de datos.

- Si T es un árbol B de orden m que contiene un total de N datos y P páginas, y cada página contiene al menos m/2-1 datos (*número mínimo de claves=m/2*), entonces $P <= N/(m/2-1)+1$
- En cada nivel K tenemos al menos $2(m/2)^{(K-2)}$ páginas. Es decir, el número total de páginas P es:

$$P >= 1 + 2 + 2(m/2) + 2(m/2)^2 + 2(m/2)^3 + \ldots + 2(m/2)^H =$$
$$= 1+2((m/2)^{H+1} - 1)/(m/2 - 1))$$

- Despejando H (altura del árbol): $H <= \log_{(m/2)} (N + 1) - 1$
- Si m=16 y N=2.000.000, $H <= \log_8 (2.000.001) - 1 = 5,98$
  - Podemos guardar 2.000.000 de registros en 6 niveles de un árbol B de orden 16.
  - El tiempo de acceso a disco es 6accesos*70ms/acceso= 420ms
- Si m=2 y N=2.000.000, $H <= \log_2 (2.000.001) - 1 = 19,9$
  - Si tenemos un dato por nodo, necesitaremos 20 niveles.
  - El tiempo de acceso a disco es 20 accesos*70ms/acceso=1.400ms

1. 2. Representación de árboles B

- Representación de un árbol B de orden *m*:
  - Representar un nodo o página de un árbol B:
    - Almacenar las claves ⇒ Vector `Claves`
    - Almacenar las direcciones de las ramas que cuelgan de los nodos ⇒ Vector `Ramas`
    - Almacenar el nº de claves de la página ⇒ entero `Cuenta`.
  - Ejemplo: árbol B de orden 5

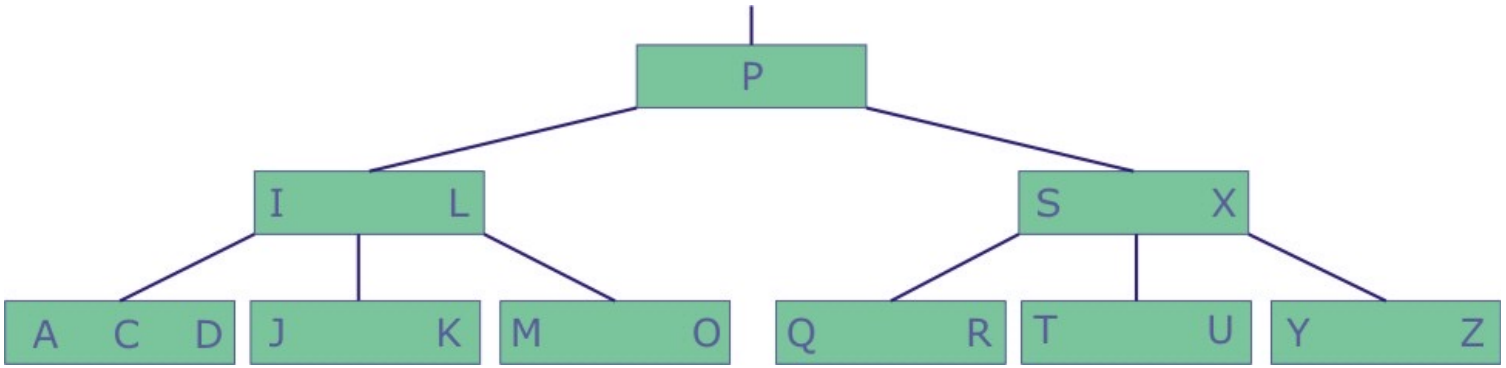
```
#define max 4 /*nº máx. de claves de 1 página.
              max=m-1, siendo m el orden del árbol*/
#define min 2 /*nº mín. de claves en página NO raíz; min=m/2-1*/

typedef int tipoclave;
typedef int tipoelem;
typedef struct Pagina{
     tipoelem Nodos[max];
     ArbolB Ramas[max+1];
     int Cuenta;
};
typedef struct Pagina * ArbolB;
```

1. 2. Representación de Árboles B

- Ejemplo: árbol B de orden 5 y 3 niveles
  - Claves=letras del alfabeto
  - Todas las páginas contienen 2, 3 ó 4 elementos excepto la raíz.
  - Todas las páginas que son hojas están en el nivel más bajo del árbol ⇒ comportamiento característico en árboles B:

    CRECEN "HACIA ARRIBA", CRECEN POR LA RAÍZ
  - Claves: ordenación izquierda a derecha dentro de cada página. Dividen a los nodos descendientes a la manera de un árbol de búsqueda: claves de nodo izquierdo menores y claves de nodo derecho mayores.



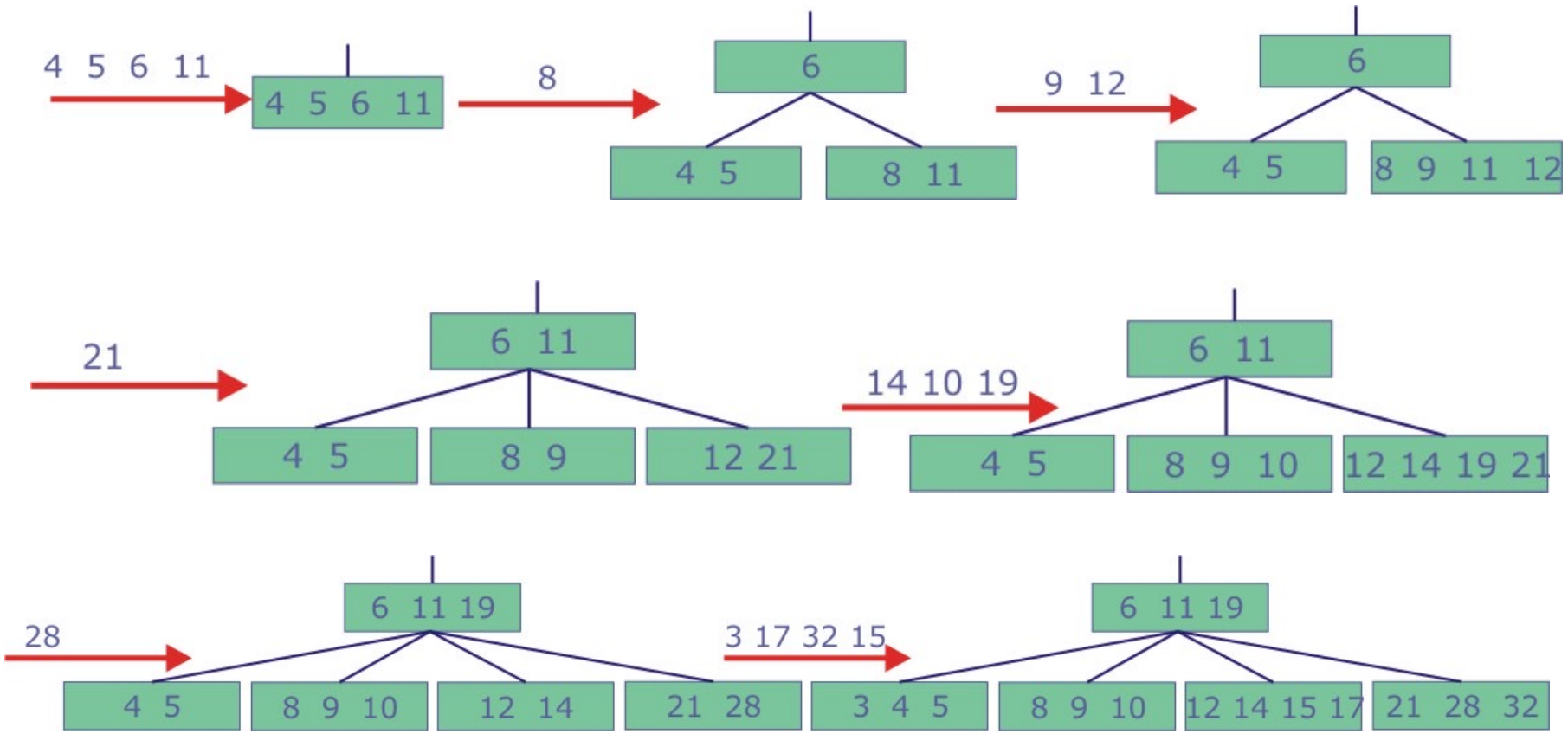
1. 3. Inserción en un árbol B

- **MÉTODO de inserción:**
  - Busco clave en árbol, siguiendo el camino de búsqueda.
  - Si clave no está en árbol, la búsqueda termina en una página hoja. La nueva clave se intenta insertar en la página hoja:
    - Si página hoja NO llena (nº claves < m-1), el proceso de inserción sólo involucra a esa página
    - Si página hoja LLENA, la inserción afecta a la estructura del árbol, ya que SE DIVIDE la página (incluyendo virtualmente la clave nueva) en 2 páginas en el mismo nivel del árbol, excepto la clave mediana que no se incluye en ninguna de las 2 páginas, sino que sube en el árbol por el camino de búsqueda para insertarla en la página antecedente.

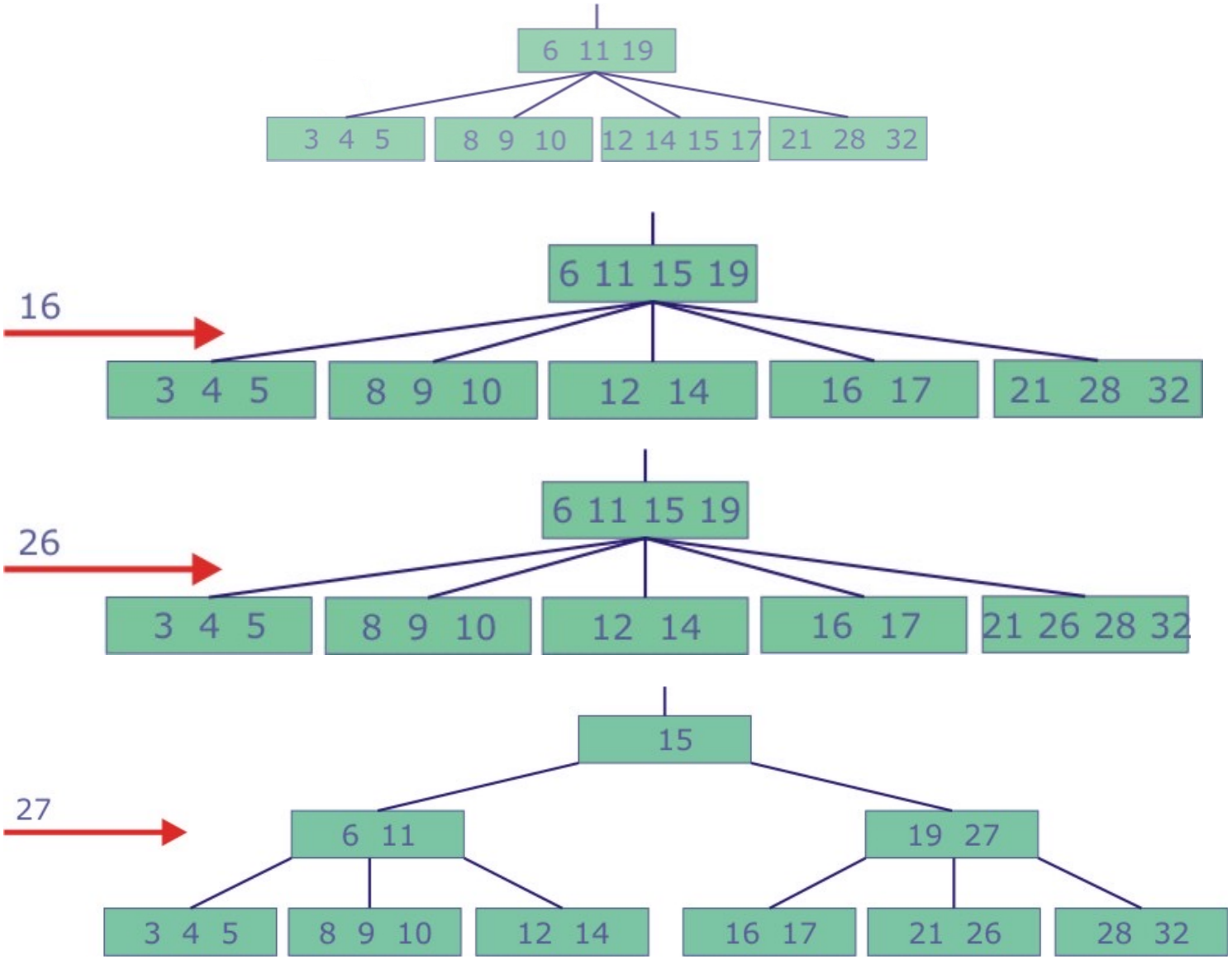
      Por eso se dice que el árbol crece hacia arriba. En esta ascensión de claves medianas puede ocurrir que llegue a la página raíz, entonces ésta se divide en 2 páginas y la clave enviada hacia arriba se convierte en una nueva raíz. Esta es la única forma de aumentar la altura del árbol B.

1.3. Inserción en un árbol B

- Ejemplo: Creación de árbol B de orden 5 insertando claves de enteros sin signo: 4, 5, 6, 11, 8, 9, 12, 21, 14, 10, 19, 28, 3, 17, 32, 15, 16, 26, 27



1.3. Inserción en un árbol B

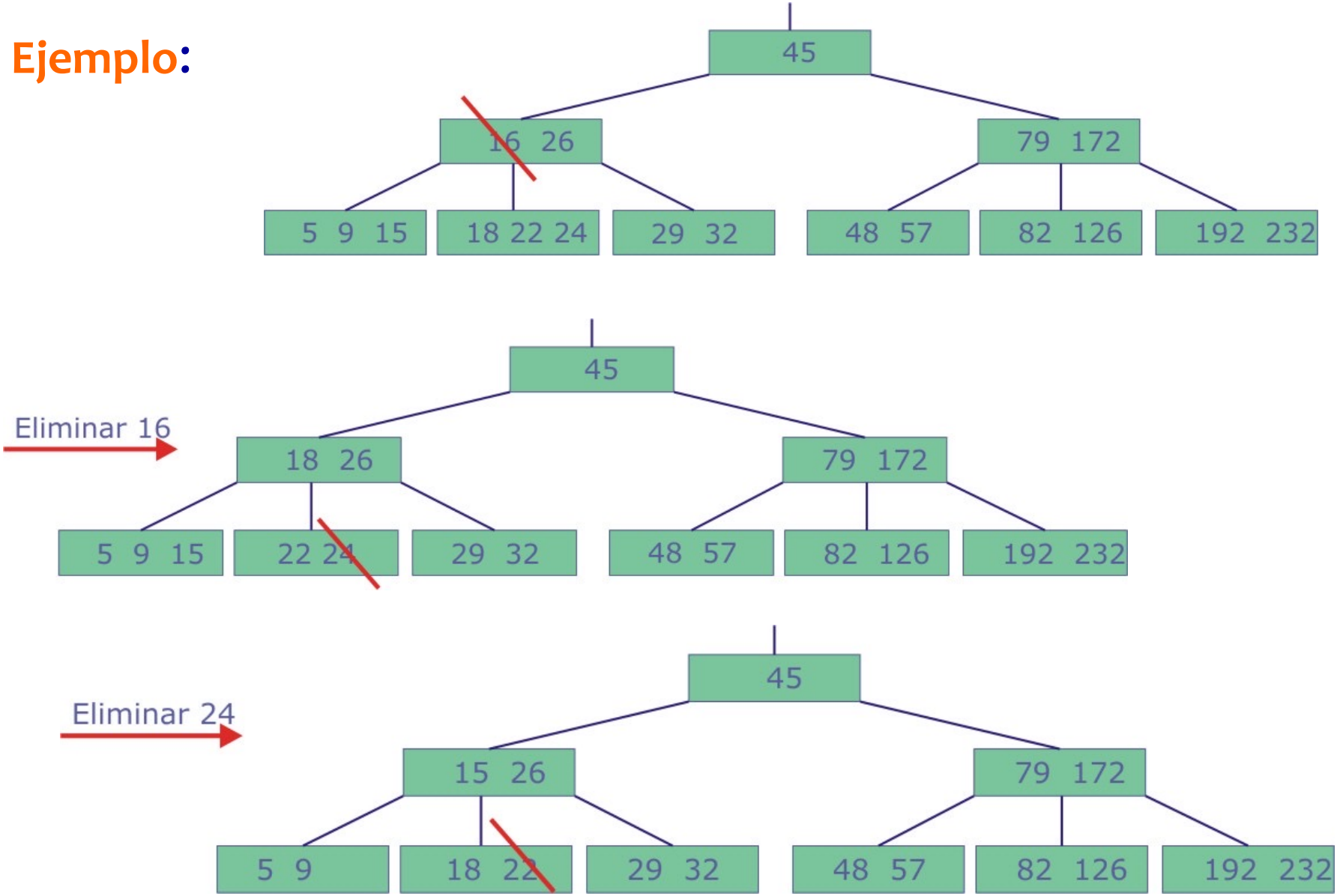


1.4. Eliminación en un árbol B de orden *m*

- **MÉTODO de eliminación** (siempre en una página hoja).
  - Si la clave a eliminar no está en una hoja, su clave antecesora o sucesora en orden natural lo estará.
  - Caso 1: La hoja donde se va a eliminar tiene nºclaves>mínimo
    - Si la clave a eliminar no está en una hoja, se pondrá a la clave predecesora/sucesora en la posición ocupada por la clave eliminada y se suprimirá la clave en la hoja.
  - Caso 2: Si la hoja donde se va a eliminar tiene nºclaves=mínimo
    - Realizar movimiento de claves para que siga siendo árbol B.
    - Examinamos hojas adyacentes al nodo con mismo antecedente.
    - Caso 2a: Si una de las hojas tiene nº claves > mínimo
      - Subir una clave al nodo padre para que a su vez descienda de éste otra clave al nodo que se quiere restaurar.
    - Caso 2b: Si hojas contiguas tienen nº claves = mínimo
      - Se coge la hoja a eliminar, su contigua y la mediana de ambas, procedente del nodo antecedente, y se combinan como un nuevo nodo hoja. El proceso se puede propagar hacia arriba hasta suprimir la raíz y disminuir la altura del árbol B.

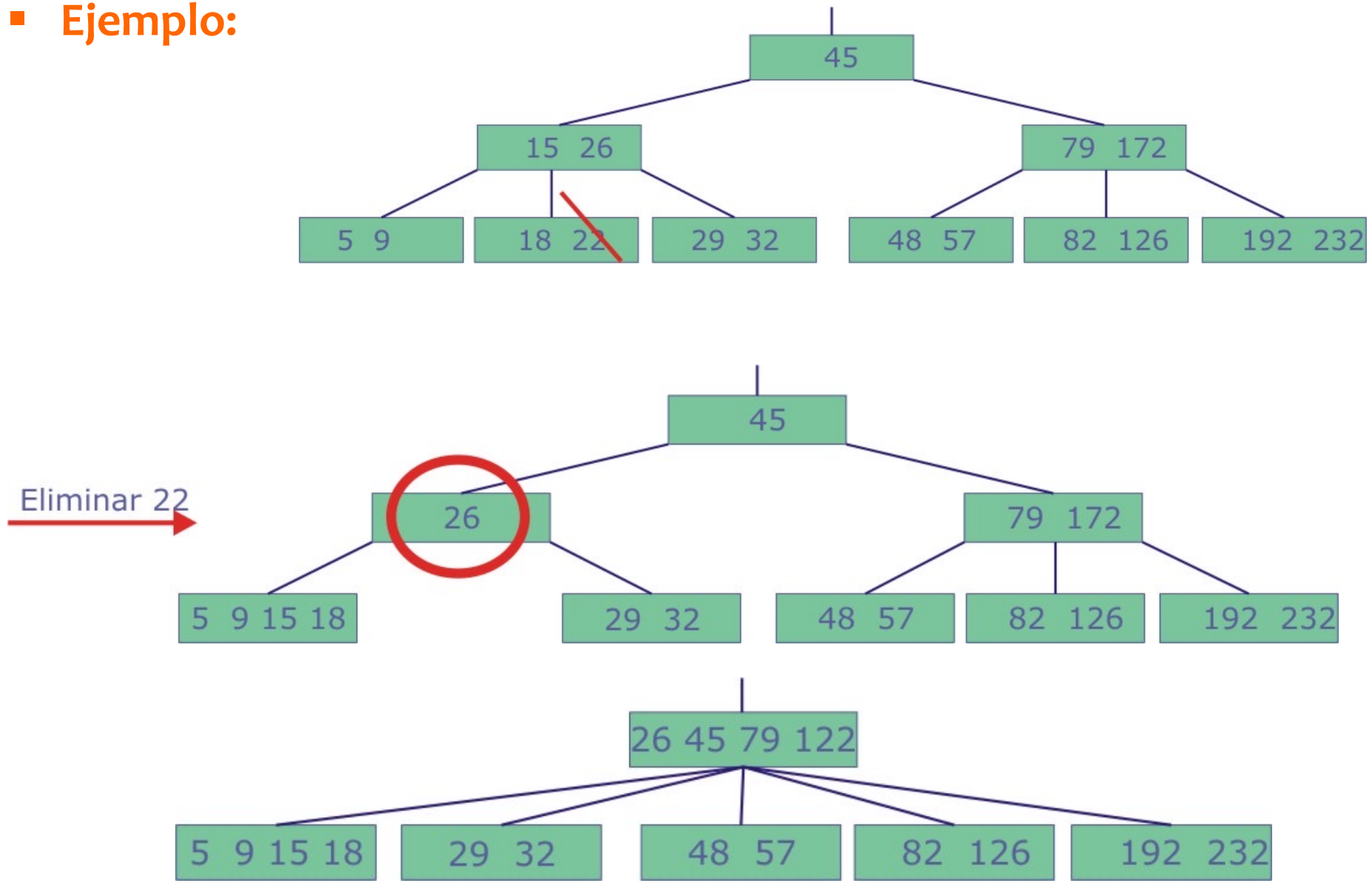
1.4. Eliminación en un árbol B de orden *m*

- **Ejemplo:**



1.4. Eliminación en un árbol B de orden *m*

- **Ejemplo:**



1.5. Propiedades de un árbol B

- Observaciones:
  - Siempre es la clave mediana la que "sube" al nodo antecedente. La mediana no tiene porqué coincidir con la clave que se está insertando, por lo que podemos afirmar que no importa el orden en que lleguen las claves en el balanceo del árbol.
  - Usando el ejemplo de la guía telefónica, si tenemos un árbol B con nodos que están al menos medio llenos entonces la altura es aproximadamente 6.
    - Para insertar sin subdivisiones de nodos, necesitamos 6 accesos a disco para buscar el nodo, más un acceso a disco para escribir el nodo actualizado. Total: 7 accesos a disco.
    - Para insertar con una subdivisión de nodo, necesitamos 5 accesos más a disco (más los 6 de búsqueda), total: 11 accesos a disco **PERO** la división de un nodo prepara la estructura para inserciones simples de nuevas claves.

1.6. Aplicaciones de los árboles B

- Creación de bases de datos
  - El árbol B es una forma de implementar los índices de una base de datos relacional.
- Gestión del sistema de archivos del SO OS/2:
  - Para aumentar la eficacia en la búsqueda de archivos por los subdirectorios.
- Sistemas de compresión de datos
  - Árboles B para la búsqueda por clave de datos comprimidos.
- Se utilizan en búsquedas externas con el objeto de minimizar el número de accesos al disco.
  - Acceso a disco => lectura de bloque de información.
  - Hasta ahora: Árboles funcionando en memoria principal
  - Si volumen de datos grande: manejar los datos directamente sobre un dispositivo de almacenamiento externo => ÁRBOLES B.

2. Tipos de árboles B

- Árboles B: datos en nodos internos y hojas

- Árboles B*
  - optimización de los árboles B para aumentar el promedio de utilización de los nodos
  - Inserción: si nodo lleno, mueve las claves a uno de sus hermanos.
    - Se pospone la división del nodo hasta que ambos hermanos están completamente llenos.
      - Cuando esto sucede, éstos pueden dividirse en 3 nodos, y cada uno estará lleno en sus 2/3 partes.

2. Tipos de árboles B

- **Árboles B^+**
  - Permiten recorrido secuencial rápido
  - Todas las claves se encuentran en hojas, duplicándose en la raíz y nodos interiores aquéllas que resulten necesarias para definir los caminos de búsqueda.
  - Para facilitar el recorrido secuencial rápido las hojas se pueden vincular, obteniéndose una trayectoria secuencial para recorrer las claves del árbol.
  - **Las claves de las páginas raíz e interiores se utilizan únicamente como índices.**
  - Todos los caminos desde la raíz hasta cualquiera de los datos tienen la misma longitud.
  - Ocupan algo más de espacio que los árboles B, pues existe duplicidad de algunas claves. Esto es aceptable si el archivo se modifica frecuentemente, pues evita la operación de reorganización del árbol (tan costosa en los árboles B).

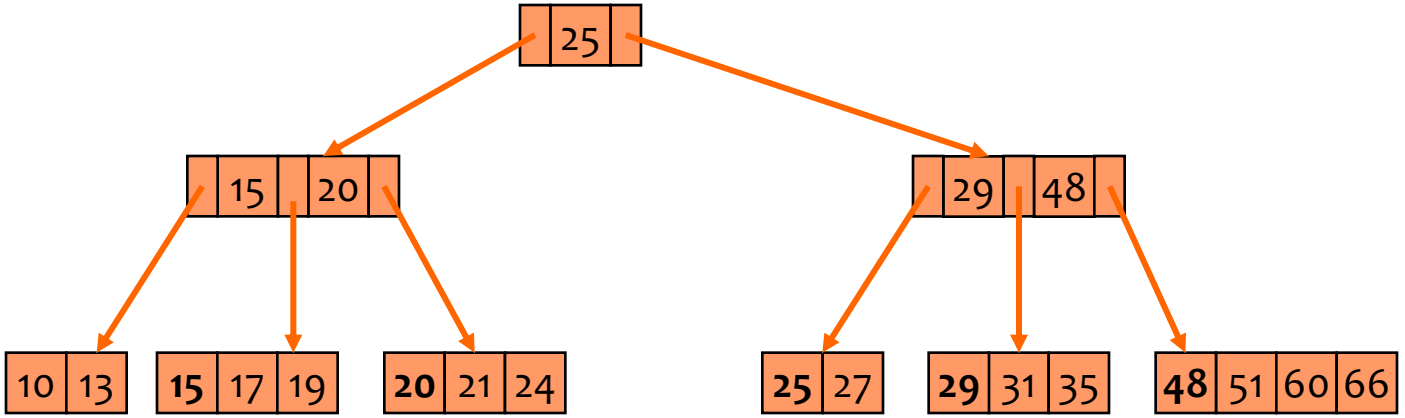
3. Árboles B+

- **Árbol B+ de orden m:**
  - Cada página, excepto la raíz, contiene n elementos, siendo [m/2]<=n<=m-1.
  - La raíz contiene de 1 a m-1 elementos.
  - Cada página, salvo la raíz, tiene entre [m/2] y m descendientes.
  - La página raíz tiene al menos 2 descendientes.
  - Las páginas hojas están todas al mismo nivel.
  - Toda la información, con las claves que las identifican, está en las hojas, ya que el árbol crece "hacia arriba".
  - Las claves almacenadas en la raíz y en las páginas interiores se utilizan como índices.
    - Por tanto sólo tienen un vector de claves, no un vector de nodos

```
typedef struct PaginaHoja{
        tipoelem Nodos[max];
        int Cuenta;
};
```

```
typedef struct PaginaInterna{
        tipoclave Claves[max];
        ArbolBmas Ramas[max+1];
        int Cuenta;
};
```

- **Ejemplo: árbol B+ de orden 5**

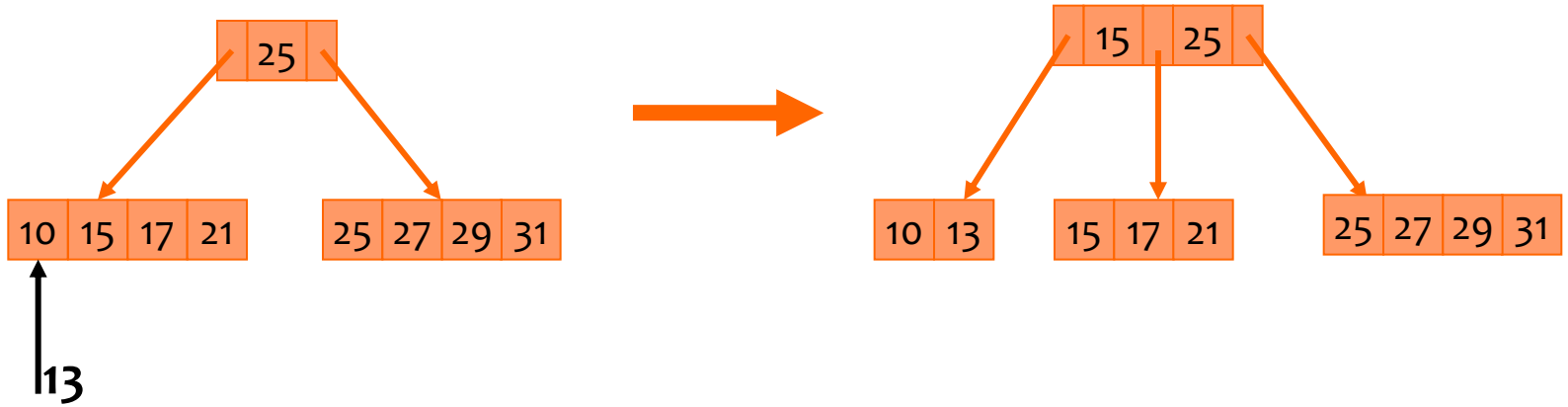


3.1. Inserción en árboles B^+

- Similar al proceso de inserción en árboles B.
- Dificultad: insertar una clave en una página llena (n=m-1)
- La página afectada se divide en 2, distribuyéndose las n+1 claves de la siguiente forma:
  - Las n/2 primeras claves en la página de la izquierda, y las (n/2)+1 restantes en la de la derecha.
  - Una **copia** de la clave MEDIANA sube a la página antecesora.
- Si la página antecesora se desborda, se repite el proceso.
- El desbordamiento de una página que no es hoja no produce duplicidad de claves.

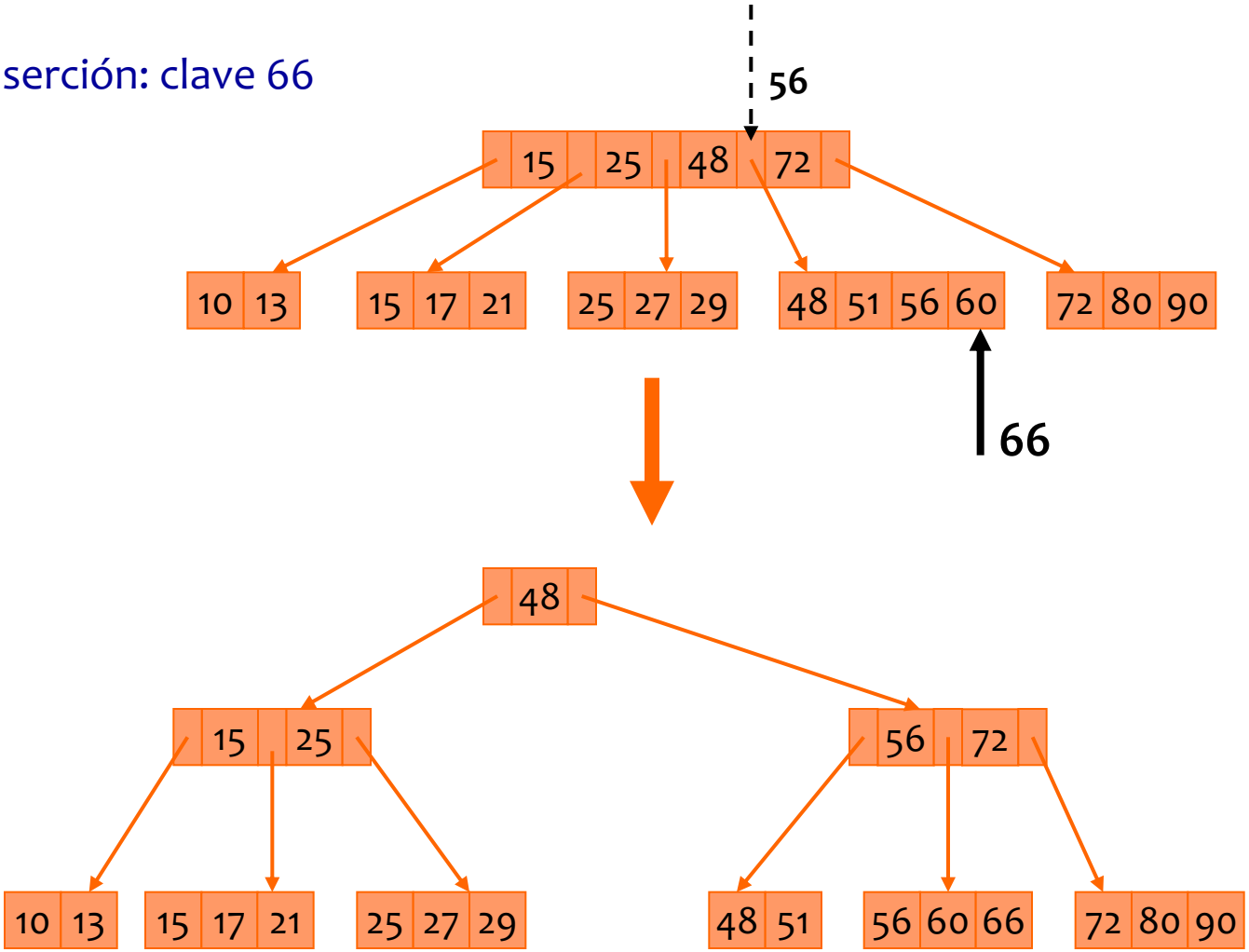
3.1. Inserción en árboles B+

Inserción: clave 13



3.1. Inserción en árboles B+

Inserción: clave 66

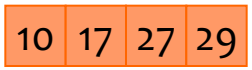


3.1. Inserción en árboles B+

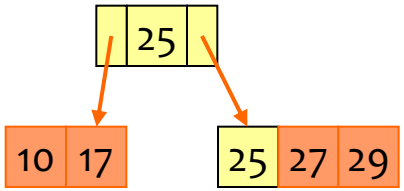
Claves:

10 – 27 – 29 – 17 – 25 – 21 – 15 – 31 – 13 – 51 – 20 – 24 – 48 – 19 – 60 – 35 – 66

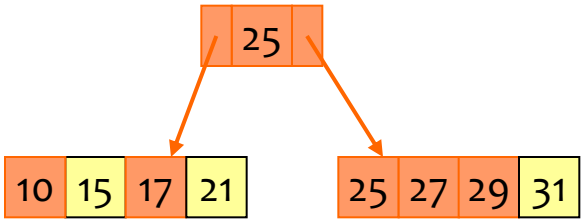
a) Claves 10–27-29-17



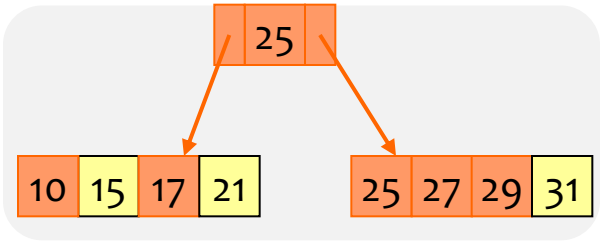
b) Clave 25



c) Claves 21-15-31



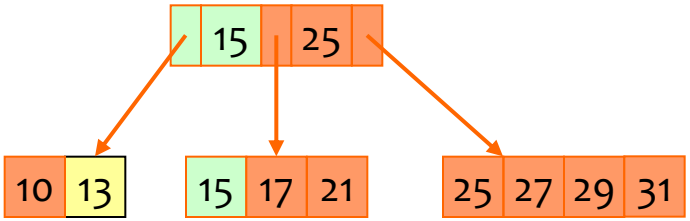
3.1. Inserción en árboles B+



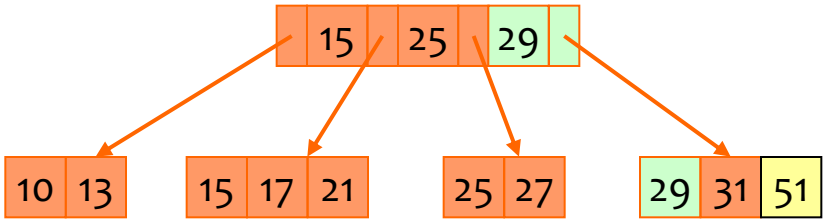
Claves:

10 – 27 – 29 – 17 – 25 – 21 – 15 – 31 – 13 – 51 – 20 – 24 – 48 – 19 – 60 – 35 – 66

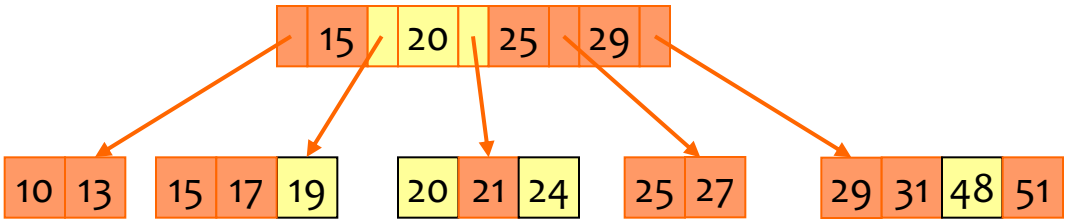
d) Clave 13



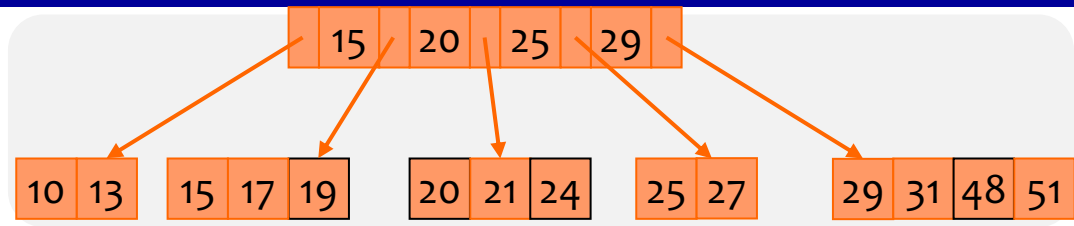
e) Clave 51



f) Claves 20-24-48-19



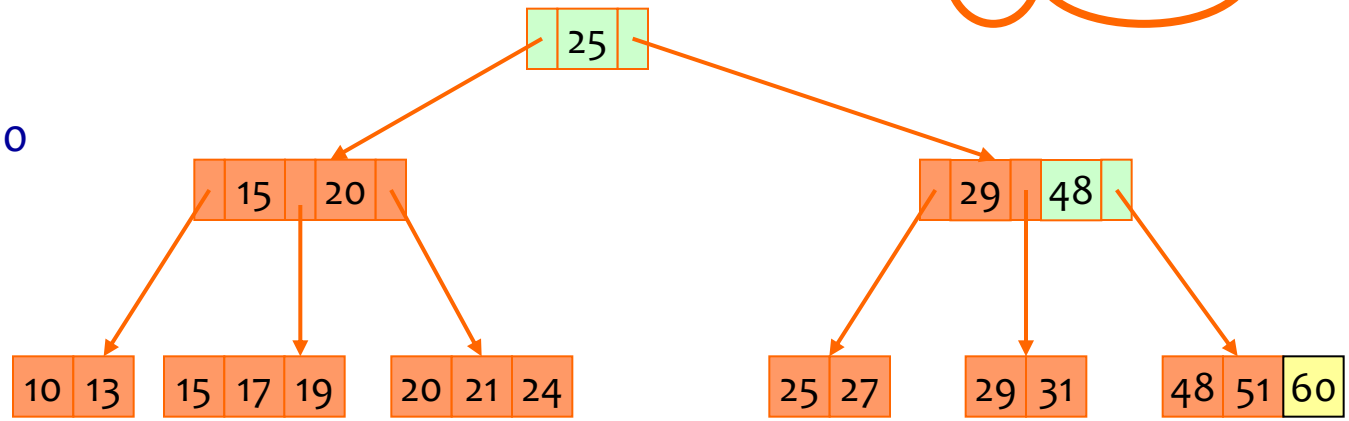
3.1. Inserción en árboles B+



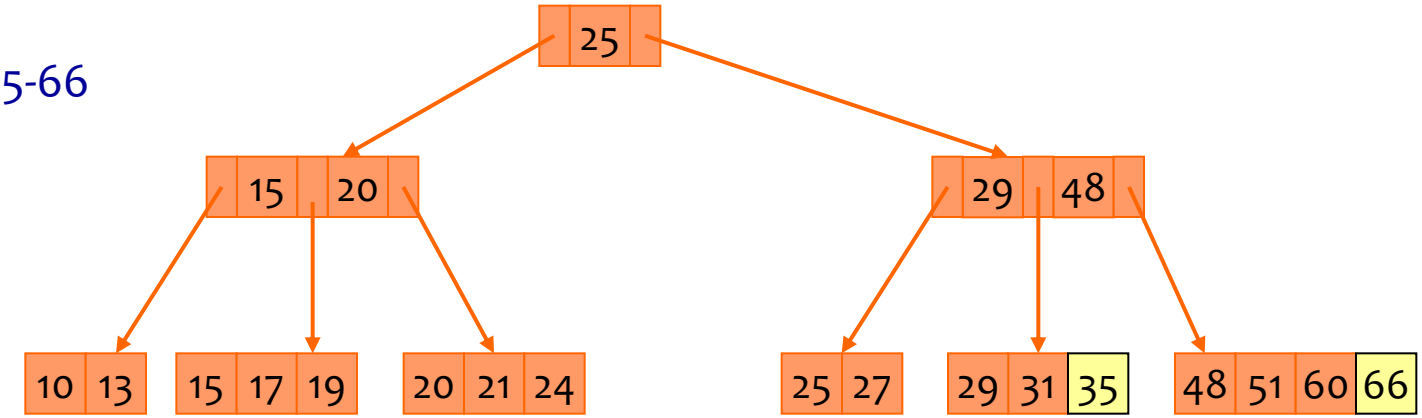
Claves:

10 – 27 – 29 – 17 – 25 – 21 – 15 – 31 – 13 – 51 – 20 – 24 – 48 – 19 – 60 – 35 – 66

g) Clave 60



h) Claves 35-66



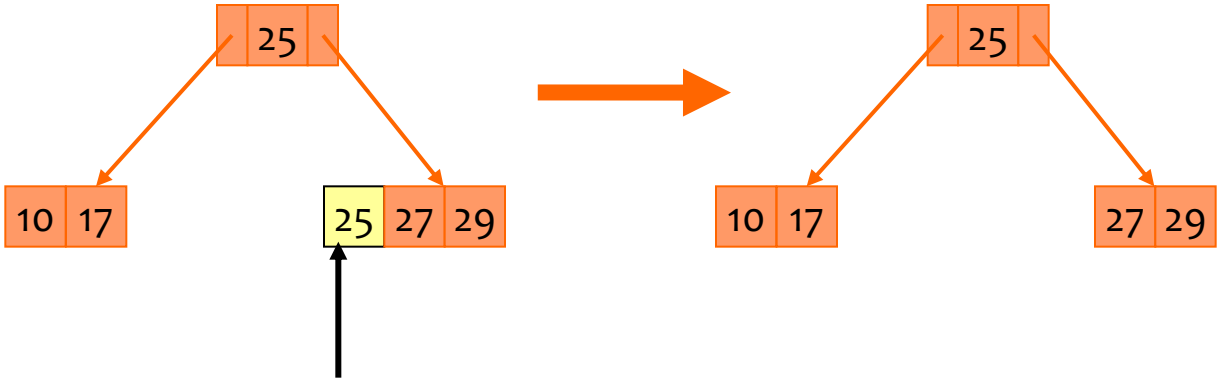
3.2. Eliminación en árboles B+

- Más simple que en árboles B.
- Las claves a eliminar siempre están en las páginas hoja.
- Dos casos:
  - Si al eliminar una clave n queda mayor o igual a [m/2], termina la operación. **Las claves en la raíz o páginas internas no se modifican pues siguen siendo un separador válido.**
  - Si al eliminar una clave, n queda menor que [m/2], deben redistribuirse las claves, tanto en el índice como en las hojas.

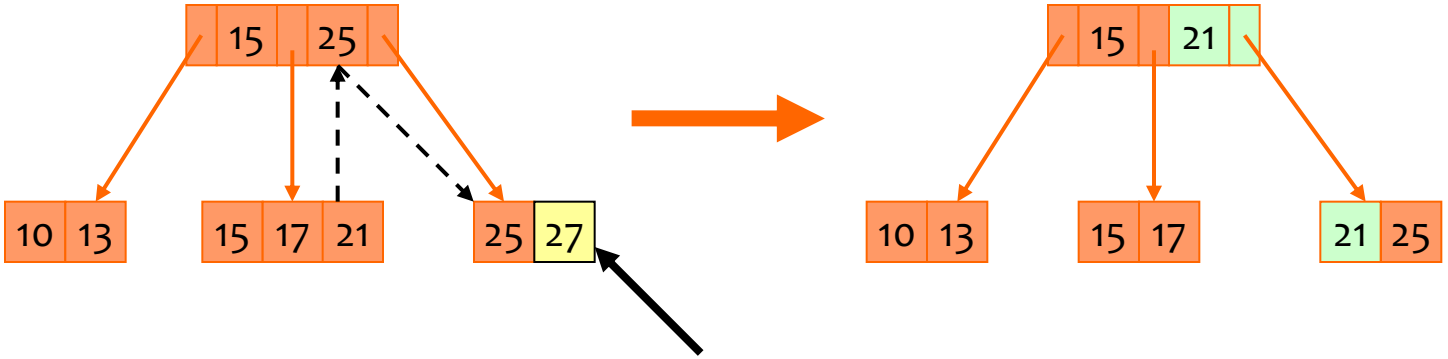
  Cuando se cambia la estructura del árbol, se quitan las claves de los nodos interiores después de haber eliminado la información de sus nodos hoja.
- Al realizar la distribución de claves es posible que la altura del árbol disminuya.

3.2. Eliminación en árboles B+

Eliminación: clave 25

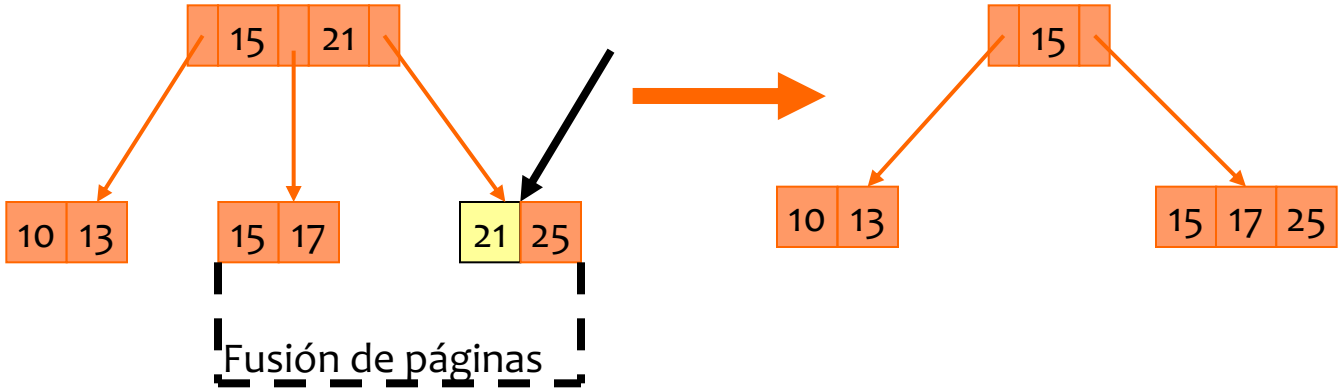


Eliminación: clave 27



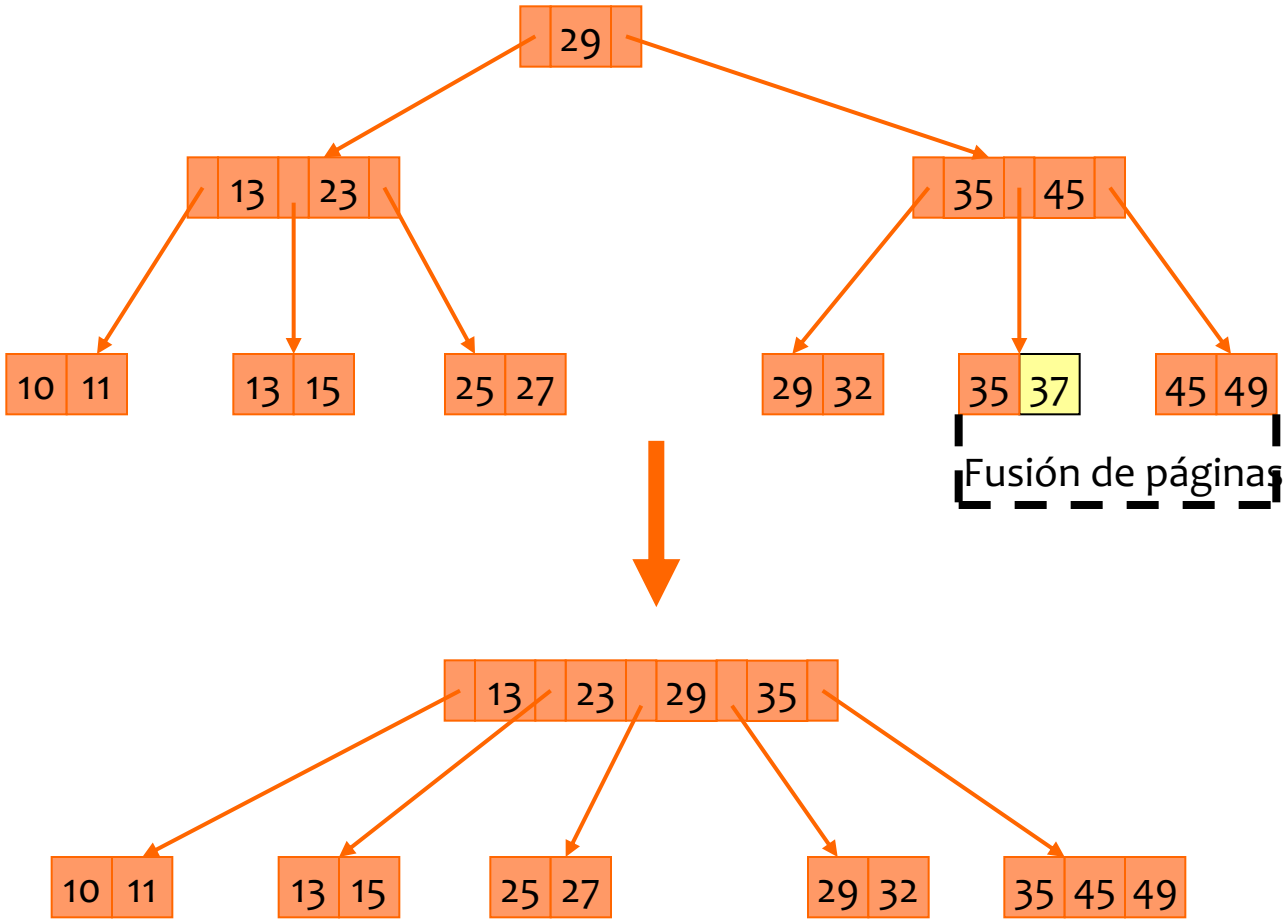
3.2. Eliminación en árboles B+

Eliminación: clave 21



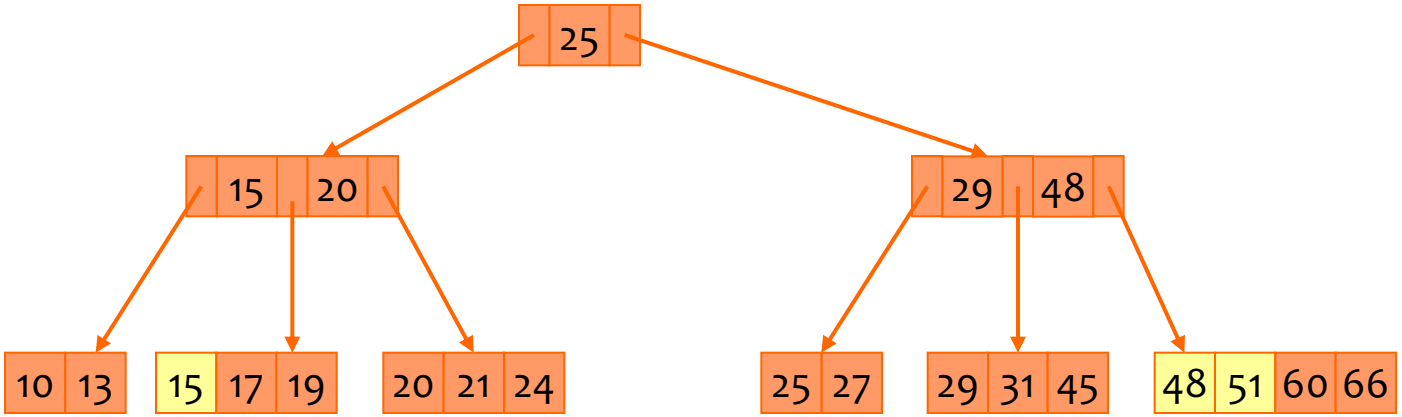
3.2. Eliminación en árboles B+

Eliminación: clave 37



3.2. Eliminación en árboles B+

- Ejercicio:

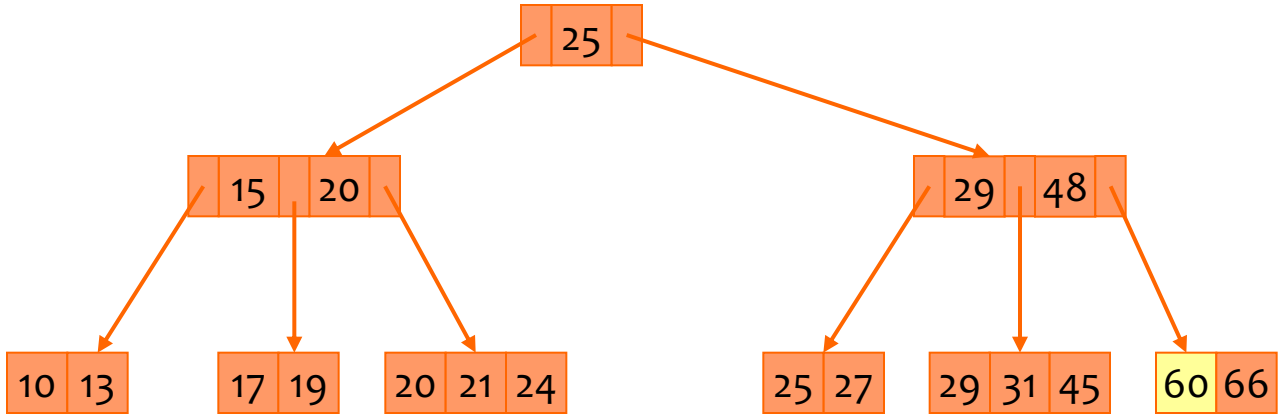


Se desea eliminar las siguientes claves:

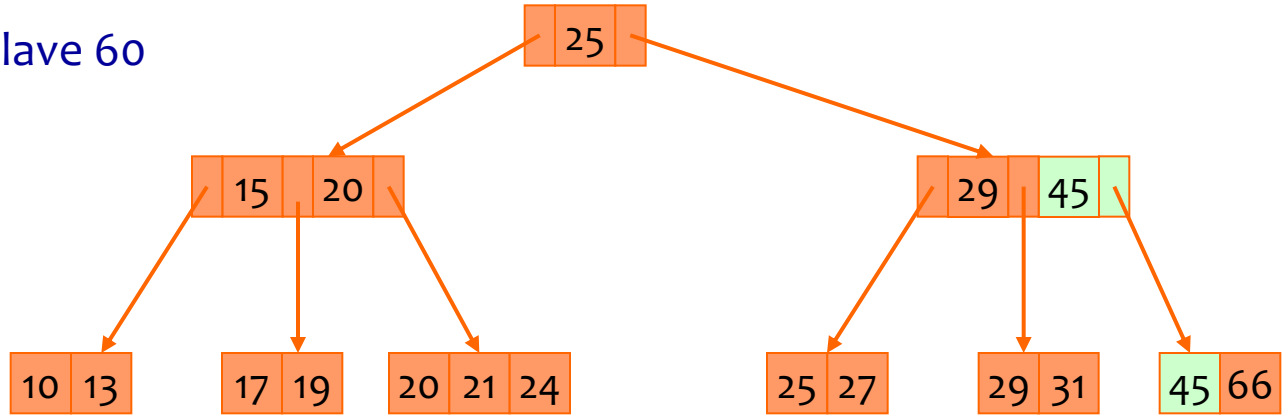
15 – 51 – 48 – 60 – 31 – 20 – 66 – 29 – 10 – 25 - 17 - 24

- Ejercicio:

a) Claves 15-51-48

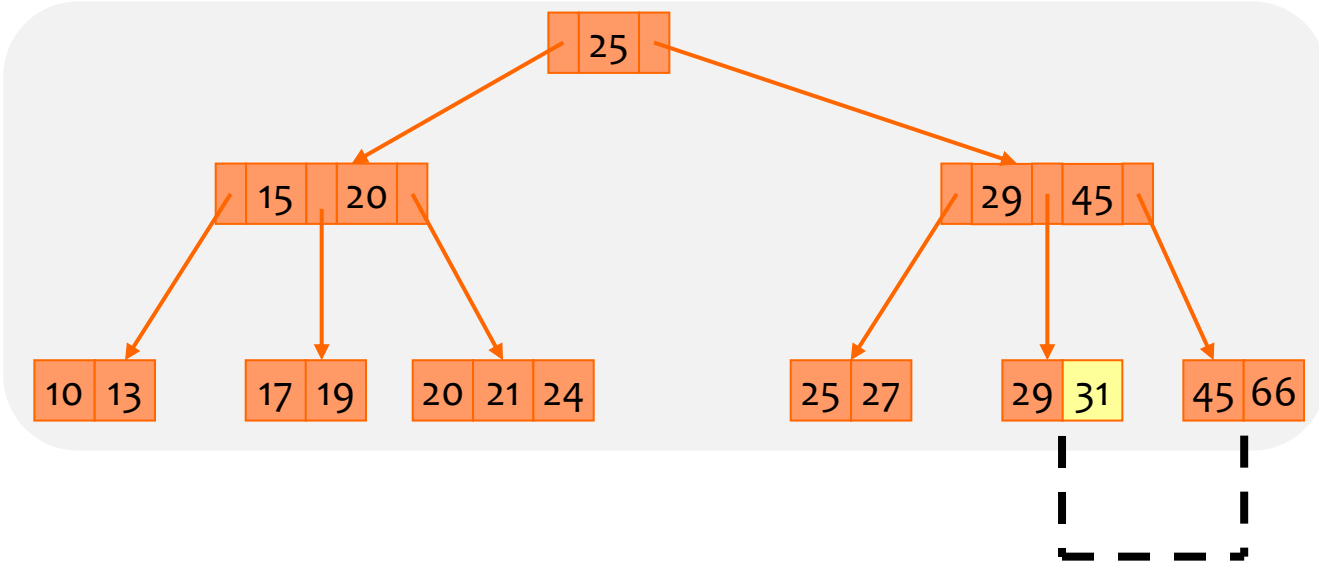


b) Clave 60

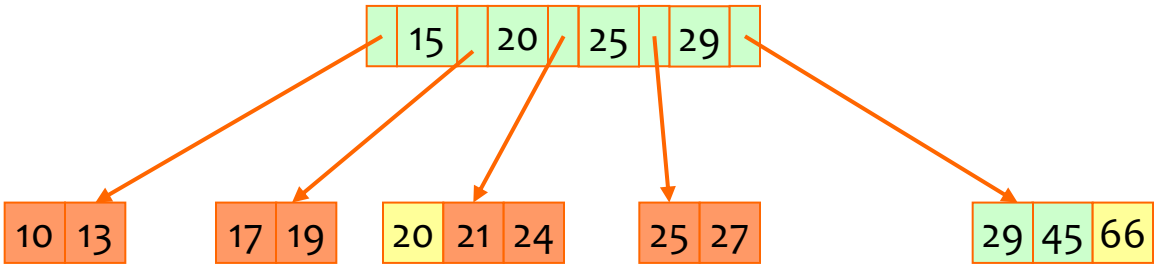


3.2. Eliminación en árboles B+

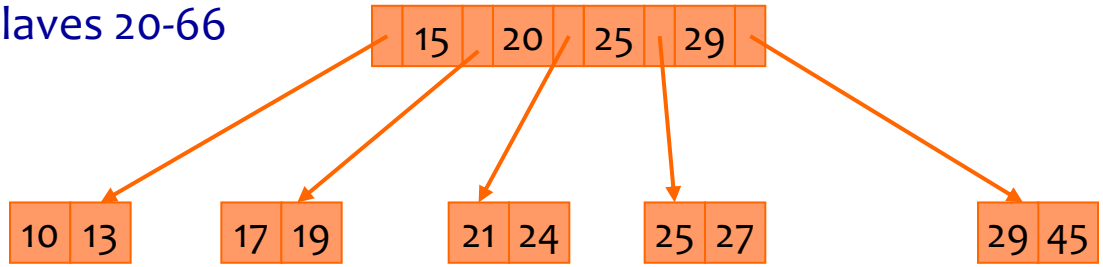
- Ejercicio:



c) Clave 31

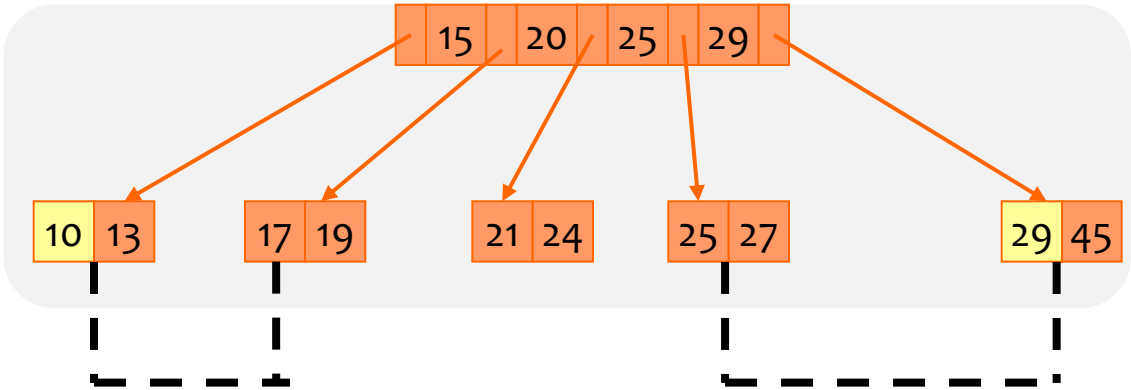


d) Claves 20-66

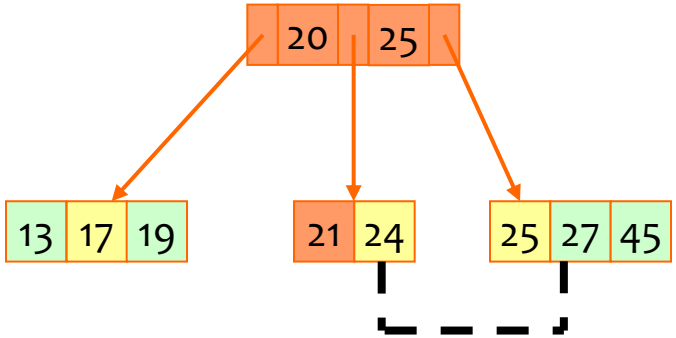


3.2. Eliminación en árboles B+

- Ejercicio:



e) Claves 29-10



f) Claves 25-17-24