Árboles B+

Clase 12

IIC 2133 - Sección 3

Prof. Eduardo Bustos

Sumario

Árboles de búsqueda 2-3

Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
   - 2 hijos árboles 2-3 si es un 2-nodo
   - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**

- Si es 2-nodo con llave k
  - las llaves k' del hijo izquierdo son $k' < k$
  - las llaves k'' del hijo derecho son $k < k''$
- Si es 3-nodo con llaves $k_1 < k_2$
  - las llaves k' del hijo izquierdo son $k' < k_1$
  - las llaves k'' del hijo central son $k_1 < k'' < k_2$
  - las llaves k''' del hijo derecho son $k_2 < k'''$

Árboles M-arios

Si consideramos un árbol M-ario

- Cada nodo tiene a lo más M hijos
- Si está lleno con n nodos, balanceado y con altura h

$$h \in \mathcal{O}(\log_M(n))$$

- Es decir, mientras mayor ramificación, menor altura (para n fijo)

Hoy veremos una versión más general de los árboles 2-3

Objetivos de la clase

- ☐ Comprender la estructura de árbol B+
- ☐ Conocer sus operaciones de inserción y eliminación
- ☐ Comprender su uso en el contexto de índices

Sumario

Árboles B+

Definición

Un **árbol B+ de orden** d es un árbol de búsqueda que almacena pares **(llave, valor)** y cumple con

1. Los nodos internos solo guardan llaves
2. La raíz tiene entre 1 y $2d$ hijos
3. Los nodos internos tienen entre d y $2d$ hijos
4. Las hojas están a la misma altura y guardan a lo más $2d$ pares (llave,valor) **ordenados por llave**
5. Las hojas están conectadas formando una lista doblemente ligada

¿Qué diferencias tiene este enfoque con los árboles 2-3?

Árboles B+

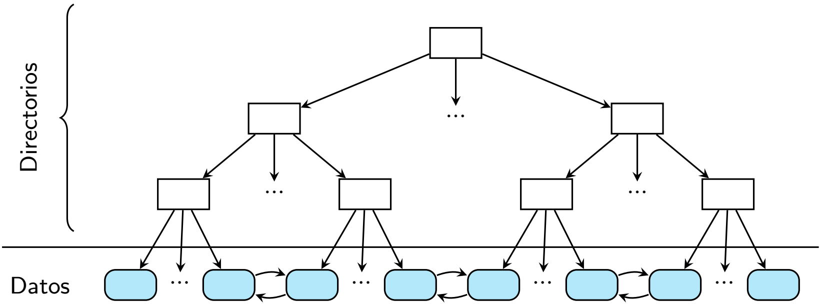
Algunas observaciones . . .

- Siempre esta **balanceado**.
- Mantiene la eficiencia de búsqueda: $\mathcal{O}\big(\log_{2d}(\#datos)\big)$.
- Procedimientos eficientes de insertar/eliminar elementos.
- Todos los nodos tienen un uso mínimo del 50% (menos el root).

"B+-trees are by far the most important access path structure in databases and file systems", Gray y Reuter (1993).

Estructura de B+-trees

Distinguimos entre nodos que solo permiten navegar (directorios) y aquellos que contienen los pares (datos)



¿Para qué sirve tener una lista doblemente ligada?

Paréntesis: ¿qué es un índice?

Definición

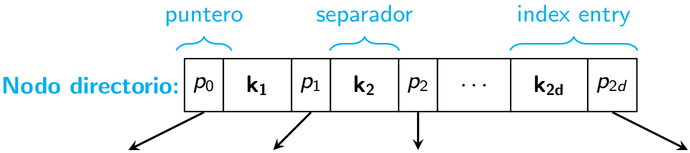
Método de acceso que **optimiza** el acceso a los datos
para **una consulta o conjunto de consultas** en particular.

Ejemplos

- Índice de un libro.
- Orden alfabético en un diccionario.
- Número de páginas de un libro.
- Secciones de un diario.

¡Los B+ se usan para almacenar índices en Bases de Datos!

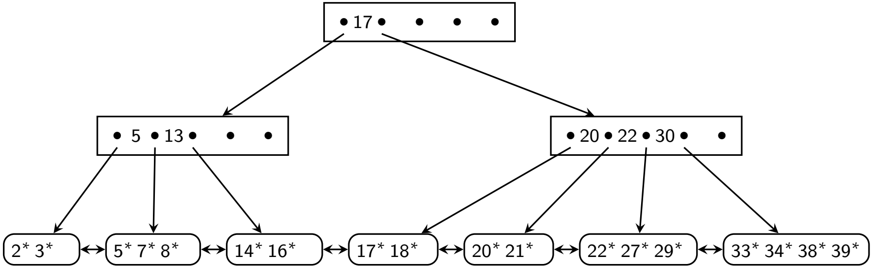
Estructura de B+-trees



- El mínimo y máximo número de llaves y punteros (n) viene dado por el **orden (d)** del B+-tree:

$$
\begin{array}{ll}
d \leq n \leq 2d & \text{para los intermedios} \\
1 \leq n \leq 2d & \text{para la raíz}
\end{array}
$$

Ejemplo de un $B+$-tree de orden 2



La lista ligada permite moverse **desde** un punto de interés

Búsqueda y range queries en B+-trees

Consideremos una consulta que use un índice sobre el atributo A:

```
SELECT  *
FROM    R
WHERE   A BETWEEN x AND y
```

1. Llamar P = `busquedaEnArbol`(x,raíz).
2. Realizar una búsqueda del mayor elemento k^* en P con $k^* \leq x$.
3. Hacer scan desde k^* sobre todos los valores menores o iguales a y.

¿Cómo cambia el desempeño de esta operación si x no está almacenado?

Sumario

Insertar un elemento en B+-trees (orden d)

Recordar: un B+-trees siempre debe estar balanceado.

Para **insertar** un valor k:

1. Llamar P = `busquedaEnArbol`(k, raíz).
2. Si la cantidad de llaves de P es menor a $2d$, insertar k en P.
3. En otro caso: debemos insertar k en P y hacer **split** de $[P + k]$!

   donde $[P + k] \ = \ k_1^* \, k_2^* \ldots k_{2d}^* \, k_{2d+1}^*$.

Insertar un elemento en B+-trees (orden d)

Para hacer **split** del nodo $[P + k]$ de tamaño $2d + 1$:

1. Asuma: $[P + k] = k_1^* \, k_2^* \ldots k_{2d}^* \, k_{2d+1}^*$ con $k_i < k_{i+1}$.

2. **Divida** $[P + k]$ en dos nodos:

$$P_1 = k_1^* \ldots k_d^* \quad \text{y} \quad P_2 = k_{d+1}^* \ldots k_{2d+1}^*$$

   y reemplace P por P_1, P_2 en la lista doble-ligada de los datos.

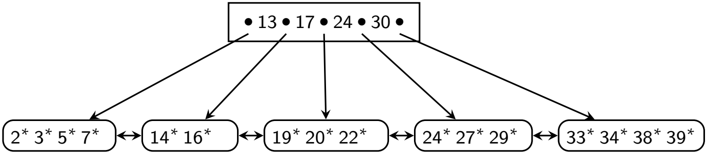
3. Seleccione el valor k_{d+1} como **divisor** de P_1 y P_2.

4. Reemplace el puntero p en el nodo P' que apuntaba a el nodo P por $p_1 \, k_{d+1} \, p_2$ donde p_1 apunta a P_1 y p_2 apunta a P_2.

5. Itere sobre el nodo de directorio P' que apuntaba a P (**split** de P').

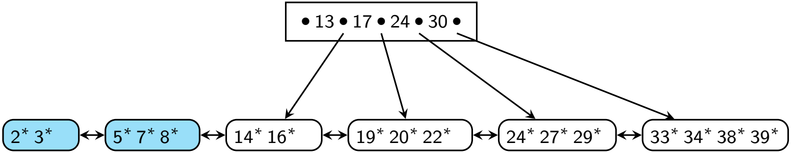
Ejemplo de como insertar un elemento en B+-trees

Inserte el elemento 8^* en el siguiente B+-tree (de orden $d = 2$):



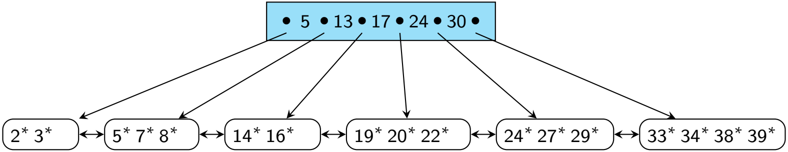
Ejemplo de como insertar un elemento en B+-trees

Inserte el elemento 8^* en el siguiente B+-tree (de orden $d = 2$):



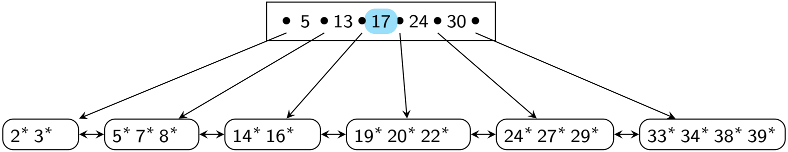
Ejemplo de como insertar un elemento en B+-trees

Inserte el elemento 8^* en el siguiente B+-tree (de orden $d = 2$):



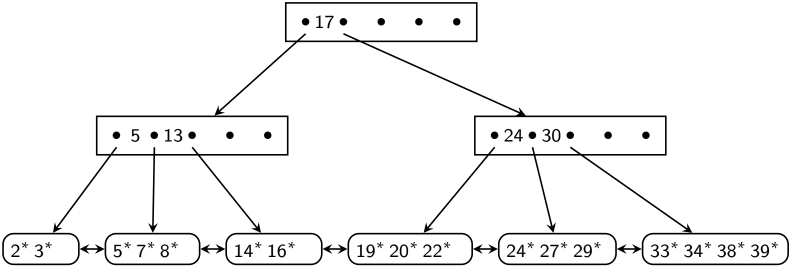
Ejemplo de como insertar un elemento en B+-trees

Inserte el elemento 8^* en el siguiente B+-tree (de orden $d = 2$):



Ejemplo de como insertar un elemento en B+-trees

Inserte el elemento 8^* en el siguiente B+-tree (de orden $d = 2$):



Nuestro B+tree queda "balanceado"

Split del nodo raíz de un B+-tree

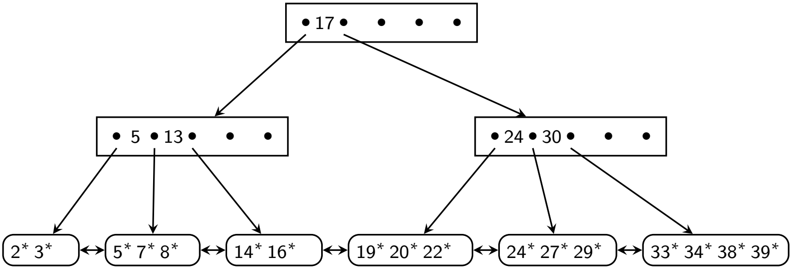
- Split de los nodos parte desde las hojas y continua hasta la raíz.
- Si:
  - es necesario hacer split de todos los nodos y
  - el nodo raíz esta lleno,

  entonces será necesario crear un nuevo nodo raíz.
- El nodo raíz es el único que se le permite estar lleno al menos del 50%.

¿qué ocurre con la altura del árbol al hacer split de la raíz?

Otro ejemplo de como insertar un elemento en B+-trees

Inserta los elementos 23^* y 40^* en el siguiente B+-tree (de orden $d = 2$):

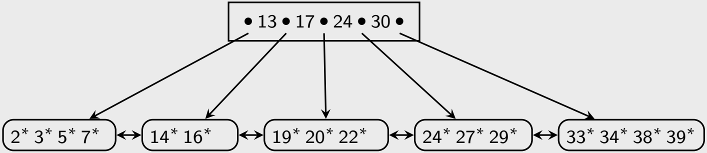


Alternativa a split: redistribución de elementos

Es posible evitar el crecimiento del árbol
usando redistribución en los nodos vecinos.

Ejemplo

Insertar el elemento 6^* en:

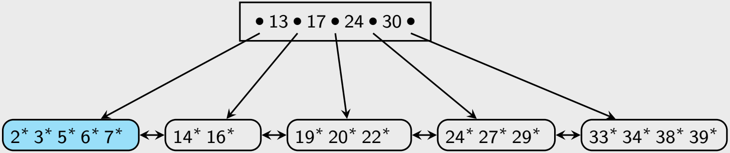


Alternativa a split: redistribución de elementos

Es posible evitar el crecimiento del árbol
usando redistribución en los nodos vecinos.

Ejemplo

Insertar el elemento 6^* en:

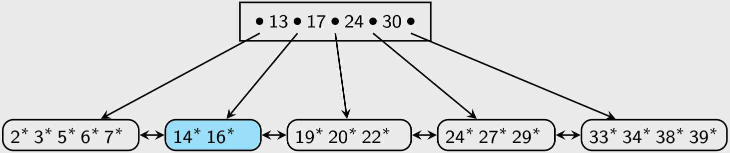


Alternativa a split: redistribución de elementos

Es posible evitar el crecimiento del árbol
usando redistribución en los nodos vecinos.

Ejemplo

Insertar el elemento 6^* en:

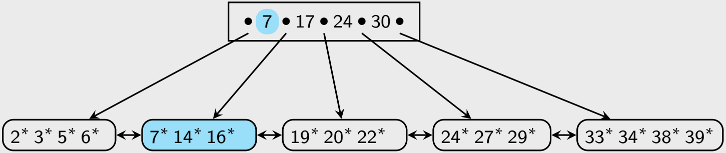


Alternativa a split: redistribución de elementos

Es posible evitar el crecimiento del árbol
usando redistribución en los nodos vecinos.

Ejemplo

Insertar el elemento 6^* en:



- Redistribución es posible también a nivel de directorio.

Sumario

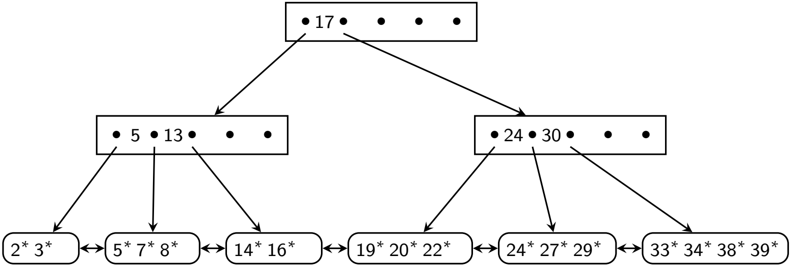
Eliminar un elemento en B+-trees (orden d)

Para **eliminar** un valor k:

1. Llamar P = `busquedaEnArbol`(k, root).
2. Si el espacio usado en P es mayor o igual a $d + 1$, eliminar k en P.
3. En otro caso: debemos eliminar k en P y **rebalancear** $[P - k]$!

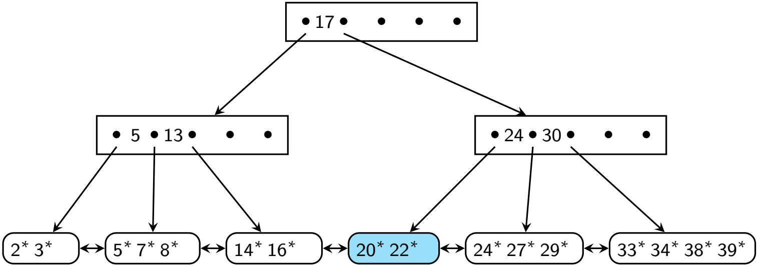
Ejemplo de como eliminar un elemento en B+-trees

Para eliminar el elemento 19*:



Ejemplo de como eliminar un elemento en B+-trees

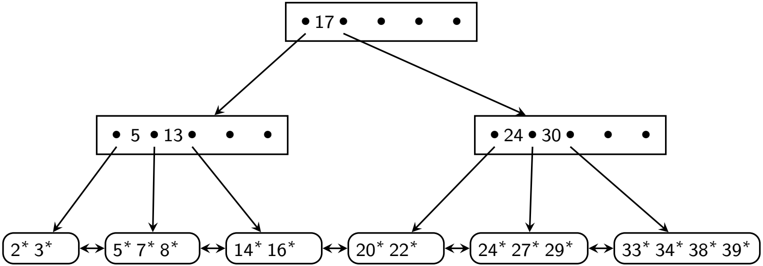
Para eliminar el elemento 19*:



Podemos eliminar 19∗ sin problemas (#-data entries > 2).

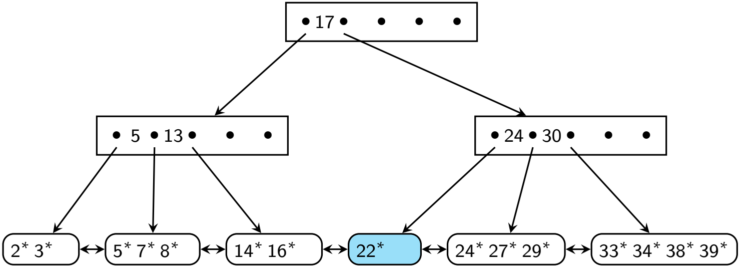
Ejemplo de como eliminar un elemento en B+-trees

Para eliminar el elemento 20*:



Ejemplo de como eliminar un elemento en B+-trees

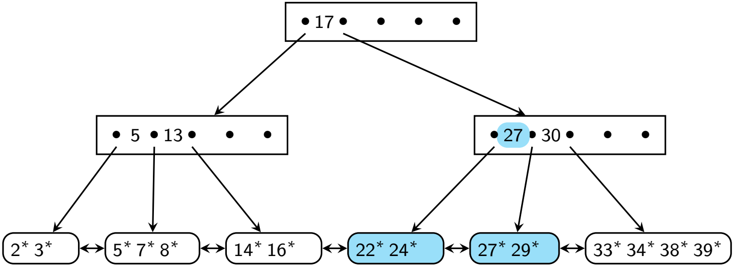
Para eliminar el elemento 20*:



Debemos hacer redistribución o "unsplit" de las nodos.

Ejemplo de como eliminar un elemento en B+-trees

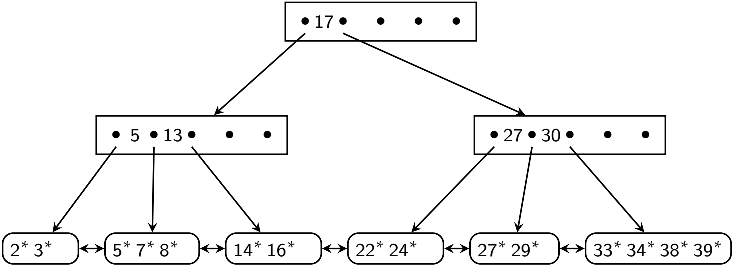
Para eliminar el elemento 20^*:



Debemos hacer redistribución o "unsplit" de las nodos.

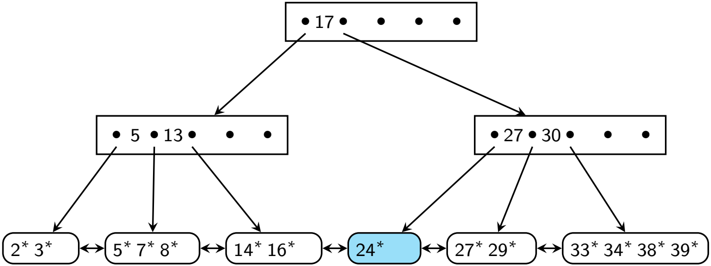
Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22*:



Ejemplo de como eliminar un elemento en B+-trees

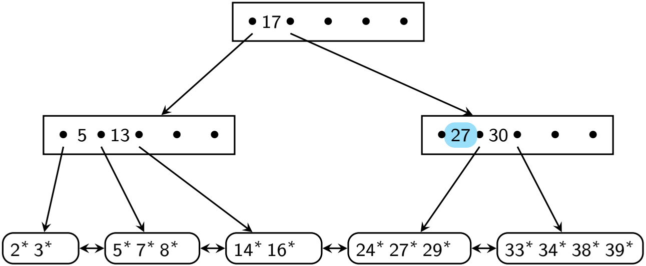
Ahora, si deseamos eliminar el elemento 22*:



Necesitamos hacer un "unsplit" de los nodos.

Ejemplo de como eliminar un elemento en B+-trees

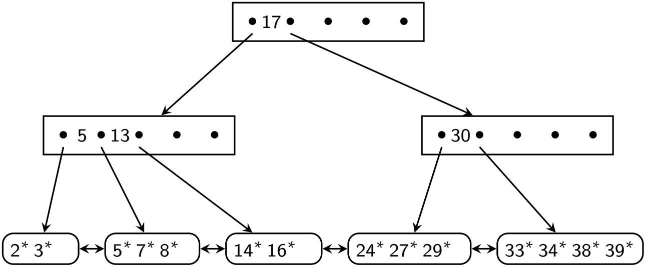
Ahora, si deseamos eliminar el elemento 22*:



Necesitamos hacer un “unsplit” de los nodos.

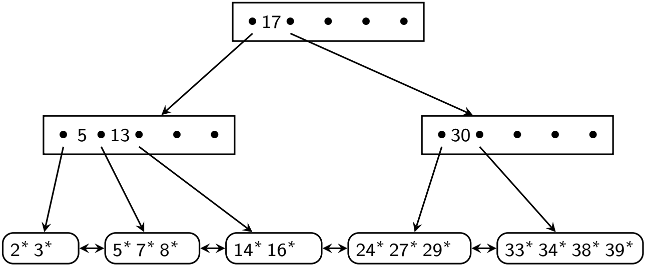
Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22*:



Ejemplo de como eliminar un elemento en B+-trees

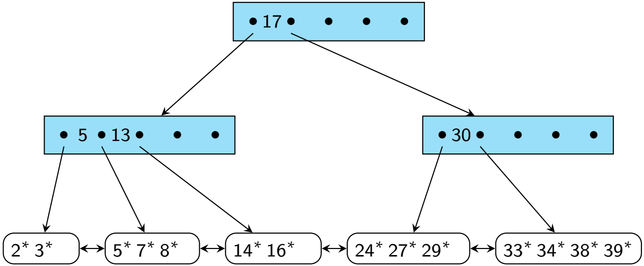
Ahora, si deseamos eliminar el elemento 22*:



Otro "unsplit", pero ahora del directorio.

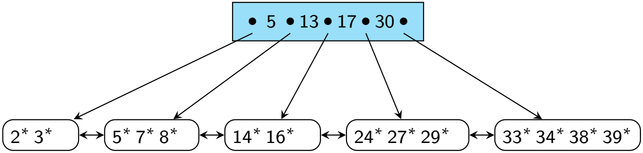
Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22^*:



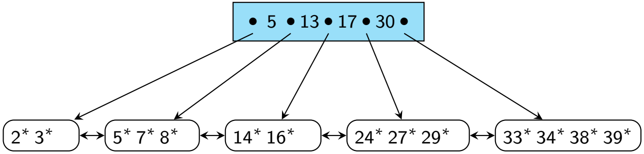
Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22^*:



Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22^*:

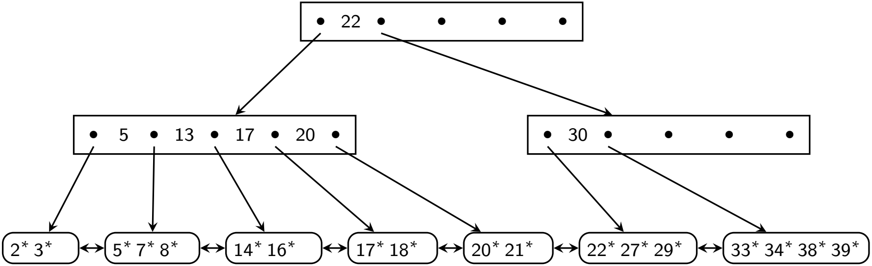


Si el root tiene un solo elemento,
esta es la única manera de decrementar la altura H del árbol.

Eliminación y redistribución de elementos

En algunos casos es necesario **redistribuir**
los elementos de un directorio en una **eliminación**.

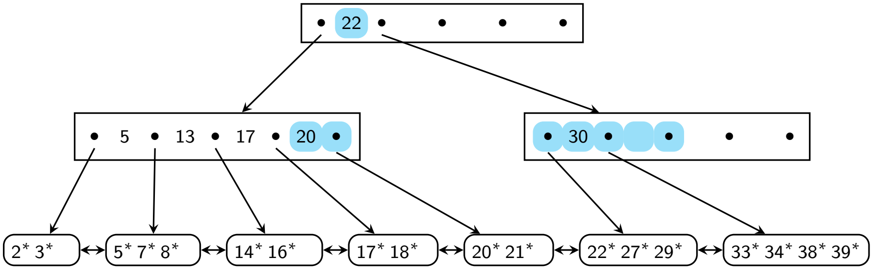
Ejemplo



Eliminación y redistribución de elementos

En algunos casos es necesario **redistribuir**
los elementos de un directorio en una **eliminación**.

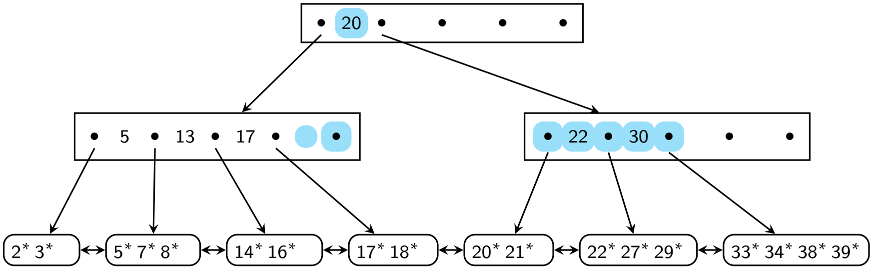
Ejemplo



Eliminación y redistribución de elementos

En algunos casos es necesario **redistribuir**
los elementos de un directorio en una **eliminación**.

Ejemplo



Eficiencia de B+-trees

Considere:

- n: el número de nodos.
- $2d$: cantidad (máxima) de datos en una hoja.

El costo de cada operación (en I/O):

Búsqueda:	$\mathcal{O}(\log_{2d}(n))$
Insertar:	$\mathcal{O}(\log_{2d}(n))$
Eliminar:	$\mathcal{O}(\log_{2d}(n))$

Costo depende logaritmicamente en base d!

Duplicados en B+-trees

Suposición: **NO** hay data-entries duplicados.

Si consideramos **duplicados**, tenemos varias opciones ...

- usar páginas con **overflow** , o
- data entries extendidos con una llave compuesta $(k, \mathtt{id})$, o
- permitir duplicados y flexibilizar los intervalos del directorio:

$$k_i \leq k < k_{i+1} \quad \Rightarrow \quad k_i \leq k \leq k_{i+1}$$

Optimizaciones a B+-trees y otros

Varias posibles optimizaciones para B+-trees:

- **Compresión** de index keys (o directorio).
- **Bulk** loading.

Sumario

Objetivos de la clase

- Comprender la estructura de árbol B+
- Conocer sus operaciones de inserción y eliminación
- Comprender su uso en el contexto de índices