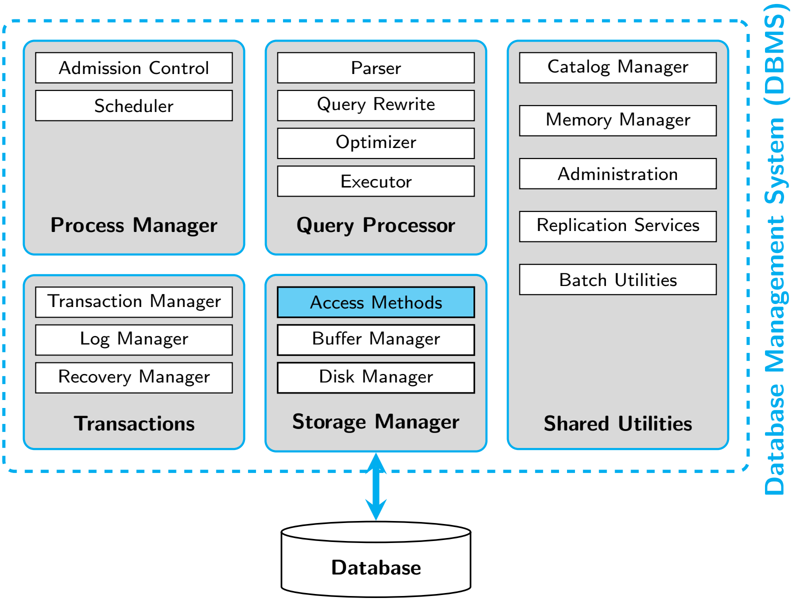
Índices basados en árboles

Clase 05

IIC 3413

Prof. Cristian Riveros

Índices basados en árboles



Outline

Outline

Archivos ordenados y búsqueda binaria

`SELECT`	*
`FROM`	Players
`WHERE`	pAge $\geq$ 31

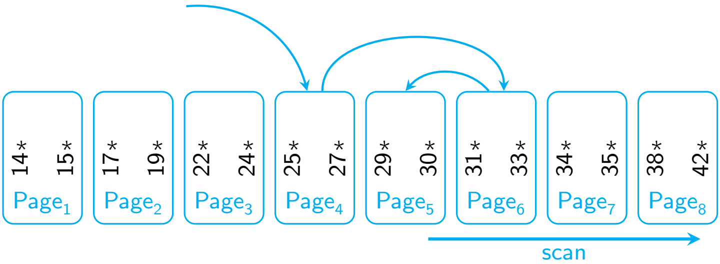
¿cómo hacemos esta búsqueda en un archivo ordenado?

- Búsqueda binaria al primer elemento tal que pAge $<$ 31.
- Retornamos todos los elementos mayores que 31.

14* 15* 17* 19* 22* 24* 25* 27* 29* 30* 31* 33* 34* 35* 38* 42*

scan

Archivos ordenados y búsqueda binaria

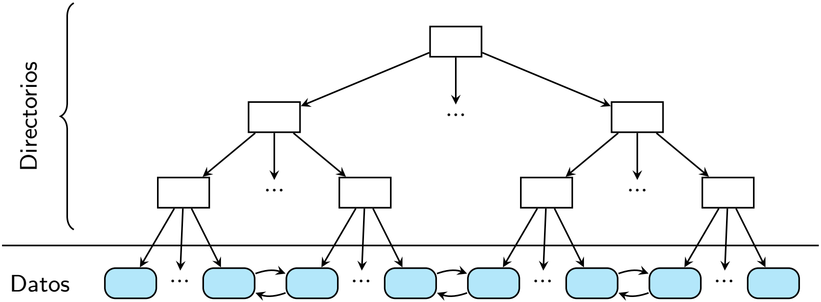


- Costo de la búsqueda: $\log_2(\#\mathit{pages})$.

¿cuál es el problema con este índice?

- Difícil de mantener el orden.

Índices basados en árboles



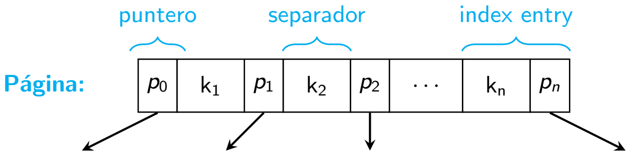
Dos tipos fundamentales de índices con estructura de árbol:

- ISAM
- B+ trees

Outline

ISAM: Indexed Sequential Access Method

- Índice con estructura de directorio *estático*.
- Cada página del directorio es de la forma:

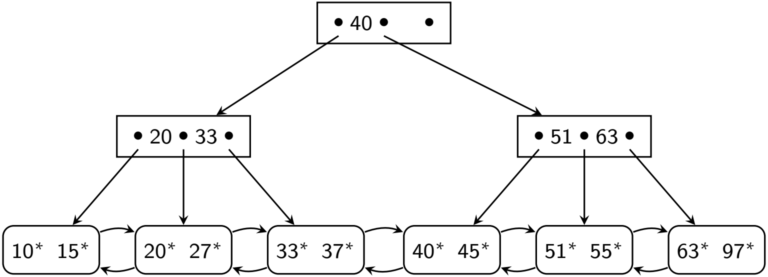


- Se garantiza que los valores k apuntados por p_i satisfacen:

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

- Datos (hojas) estan ordenados y sus páginas están doblemente ligadas.
- Dependiendo la altura H del directorio, se le llama "H-level ISAM".

Ejemplo 2-level ISAM



La altura del árbol es **estática** y fijada por el usuario.

Busqueda en ISAM

Primero necesitamos la siguiente función de búsqueda:

input : Un search key k y una página P del índice.
output: Una página de datos que puede contener a k.

```
Function busquedaEnArbol (k, P)
    if P es una página de datos then
        return P
    let P = p0 k1 p1 ... kn pn
    switch k do
        case k < k1 do
            return busquedaEnArbol (k, p0)
        case ki ≤ k < ki+1 do
            return busquedaEnArbol (k, pi)
        case kn ≤ k do
            return busquedaEnArbol (k, pn)
```

Range queries en ISAM

Para responder una consulta de la forma:

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

1. Llamar $P = \texttt{busquedaEnArbol}(x, root)$.
2. Realizar una búsqueda binaria del mayor elemento k^* en P tal que

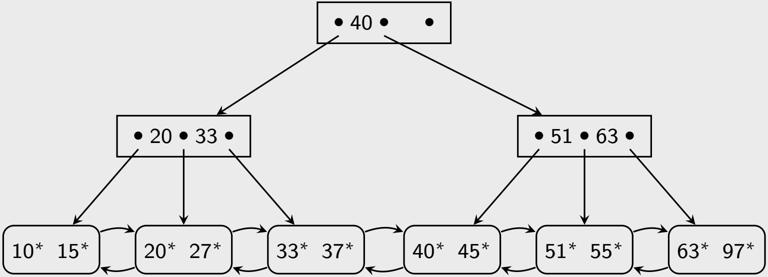
$$k^* \leq x$$

3. Hacer scan desde k^* sobre todos los valores menores o iguales a y.

Range queries en ISAM

Ejemplo

```
SELECT   *
FROM     R
WHERE    A BETWEEN 46 AND 63
```



Insertar un elemento en ISAM

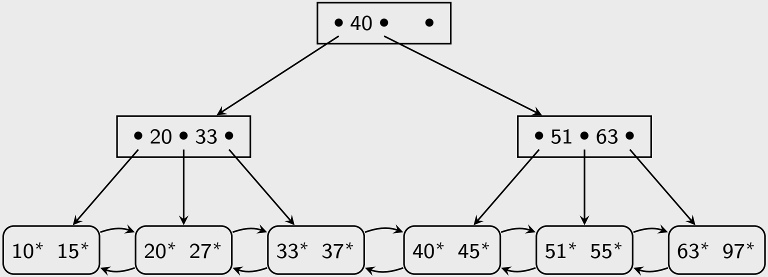
Para **insertar** un valor k:

1. Llamar $P = \texttt{busquedaEnArbol}(k, root)$.
2. Si P tiene espacio para k, insertar k en P.
3. Si P no tiene espacio para k, insertar k en una página de **overflow**.

Insertar un elemento en ISAM

Ejemplo

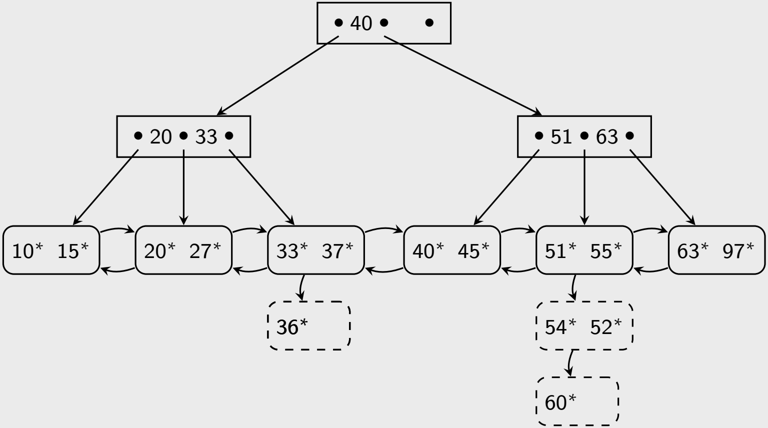
Inserte los valores 54^*, 36^*, 52^* y 60^*.



Insertar un elemento en ISAM

Ejemplo

Inserte los valores 54^*, 36^*, 52^* y 60^*.



Eliminar un elemento en ISAM

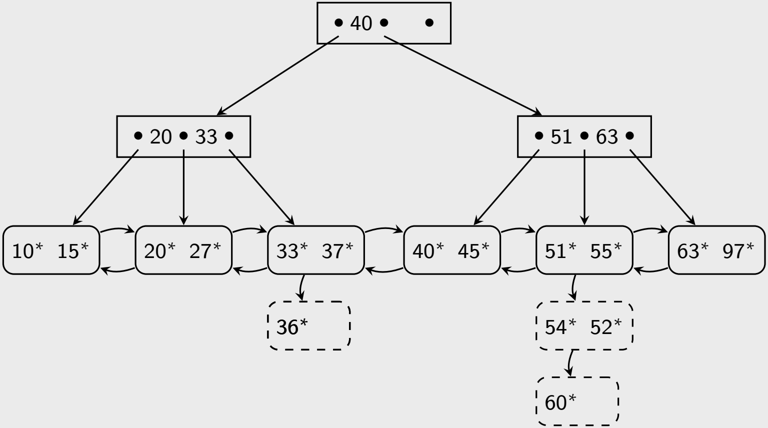
Para **eliminar** un valor k:

1. Llamar $P = \texttt{busquedaEnArbol}(k, root)$.
2. Si P contiene a k, eliminar k en todas las páginas de overflow que contengan a k (duplicados).

Eliminar un elemento en ISAM

Ejemplo

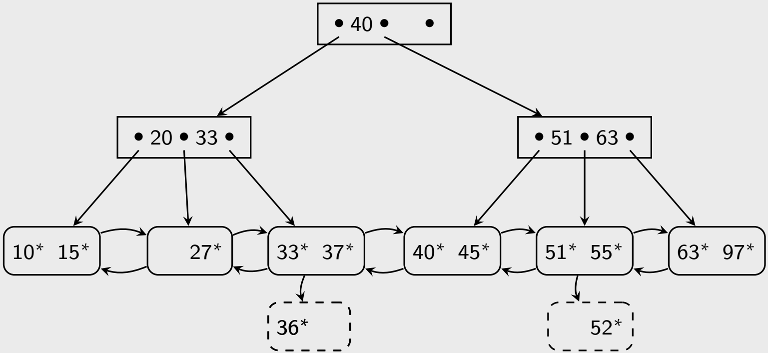
Eliminar los valores 20*, 54* y 60*.



Eliminar un elemento en ISAM

Ejemplo

Eliminar los valores 20*, 54* y 60*.



Eficiencia de ISAM

Considere:

- H: la cantidad de niveles.
- V: largo máximo las cadenas de páginas de overflow.

El costo de cada operación (en I/O):

Busqueda:	$\mathcal{O}(H + V)$
Insertar:	$\mathcal{O}(H + V)$
Eliminar:	$\mathcal{O}(H + V)$

Costo de las operaciones fuertemente influenciado por páginas de **overflow**.

¿por qué usar ISAM?

Desventajas:

- Deficiente si la cantidad de datos aumenta.
- Rendimiento pobre si hay muchas modificaciones.

Pero...

- Eficiente en la busquedad (para pocas modificaciones).
  - Optimización: iniciar páginas con un 20% de espacio libre.
- Muy sencillo de implementar.
- Útil para acceso concurrente.

Outline

B+-trees: Índice dinámico

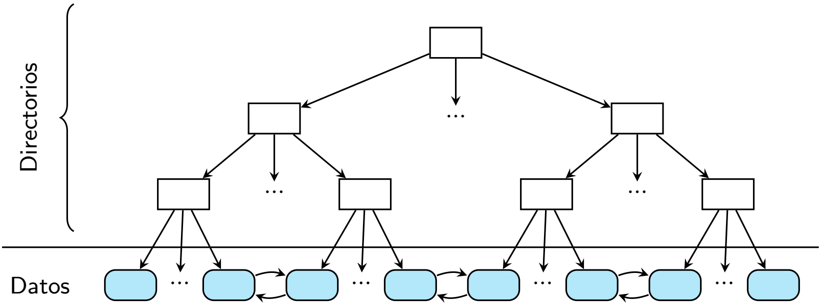
Estructura similar a ISAM, pero ...

- No usa páginas de overflow y siempre esta **balanceado**.
- Mantiene la eficiencia de búsqueda: $\mathcal{O}\big(\log_B(\#\mathit{tuplas})\big)$.
- 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

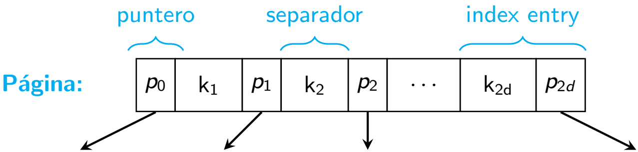
Misma estructura que ISAM:



Hojas contienen los data entry (pueden ser del tipo 1, 2, o 3).

Estructura de B+-trees

Nodos internos tienen la misma estructura que ISAM:



pero...

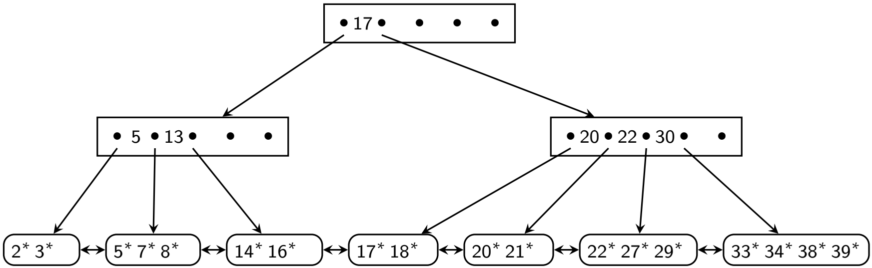
- El minimo y máximo número de llaves y punteros (n) viene dado por el orden (d) del B+-tree:

$$d \leq n \leq 2d$$
$$1 \leq n \leq 2d \qquad \text{para el root}$$

- Orden d del árbol es siempre del orden del tamaño de una página

$$d \approx \frac{B}{2}$$

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



Búsqueda y range queries en B+-trees

Misma historia que búsqueda en ISAM (con una diferencia):

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

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

Suposición: data keys son todos distintos (sin duplicados)

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

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

Para **insertar** un valor k:

1. Llamar $P = \texttt{busquedaEnArbol}(k, root)$.
2. Si el espacio libre de P es menor o igual 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** de una página de datos $[P + k]$ de tamaño $2d + 1$:

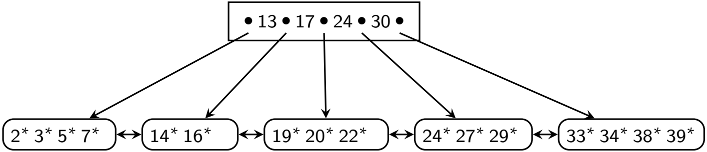
1. Asuma: $[P + k] \;=\; k_1^* \, k_2^* \ldots k_{2d}^* \, k_{2d+1}^*$ con $k_i < k_{i+1}$.
2. **Divida** $[P + k]$ en dos páginas:

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

   y reemplaze 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 la página P' que apuntaba a la página 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 la página de directorio P' que apuntaba a P (**split** de P').

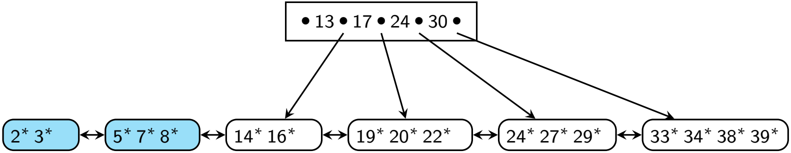
Ejemplo de como insertar un elemento en B+-trees

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



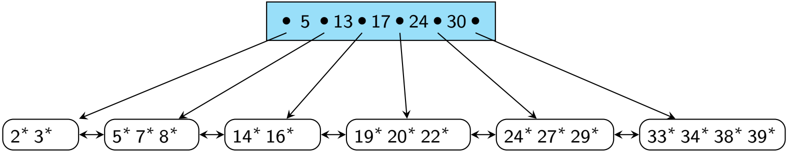
Ejemplo de como insertar un elemento en B+-trees

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



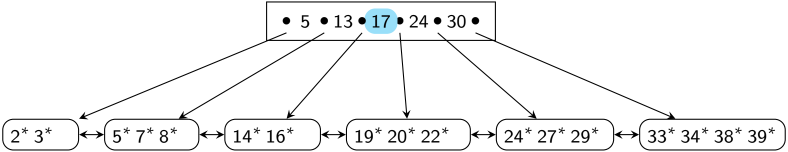
Ejemplo de como insertar un elemento en B+-trees

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



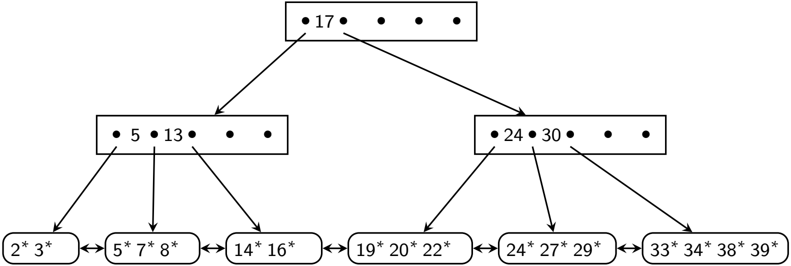
Ejemplo de como insertar un elemento en B+-trees

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.

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

Para hacer **split** de una página de **directorio** P de tamaño $2d+1$:

1. Asuma:

$$P \ = \ p_0 \, k_1 \, p_1 \, k_2 \, p_2 \ldots p_{2d} \, k_{2d+1} \, p_{2d+1}$$

   con $k_i < k_{i+1}$.

2. **Divida** P en dos páginas:

$$P_1 \ = \ p_0 \, k_1 \, \ldots \, k_d \, p_d \qquad \text{y} \qquad P_2 \ = \ p_{d+1} \, k_{d+2} \, \ldots \, k_{2d+1} \, p_{2d+1}$$

   y reemplaze P por P_1, P_2 en el directorio.

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

4. Reemplace el puntero p en la página P' que apuntaba a la página 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 la página de directorio P' que apuntaba a P (split de P').

Split del nodo raíz de un B+-tree

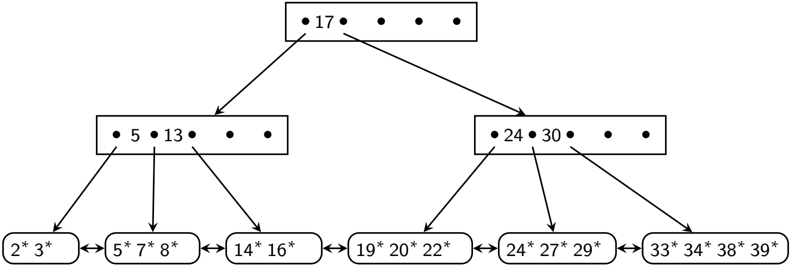
- Split de los nodos parte desde las hojas y continua hasta la raíz.
- Si:
  - es necesario hacer split de todos los nodos y
  - el nodo raíz esta lleno,

  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 $d = 2$):

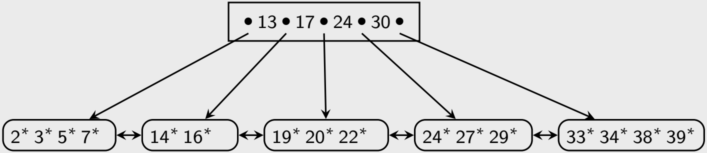


Alternativa a split: redistribución de elementos

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

Ejemplo

Insertar el elemento 6^* en:

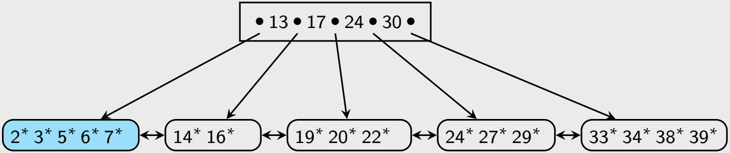


Alternativa a split: redistribución de elementos

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

Ejemplo

Insertar el elemento 6^* en:

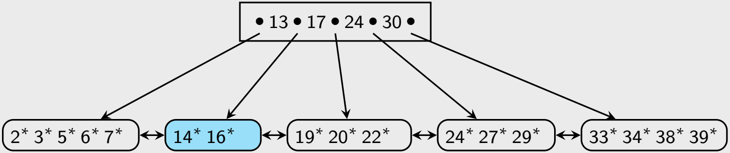


Alternativa a split: redistribución de elementos

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

Ejemplo

Insertar el elemento 6^* en:

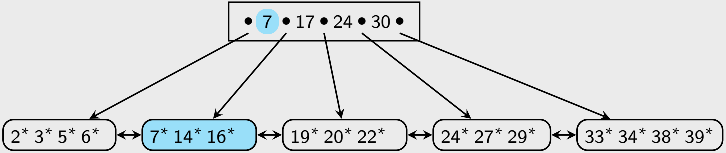


Alternativa a split: redistribución de elementos

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.

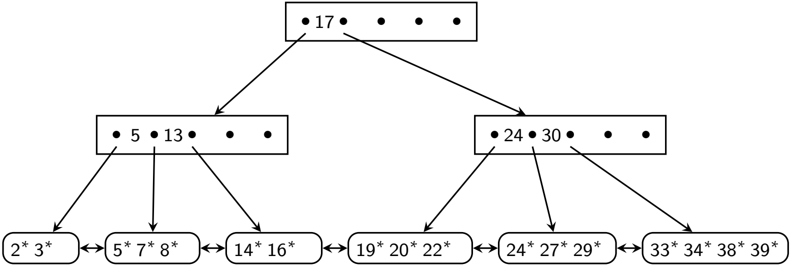
Eliminar un elemento en B+-trees (orden d)

Para **eliminar** un valor k:

1. Llamar $P = \texttt{busquedaEnArbol}(k, root)$.
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 $[P - k]$!

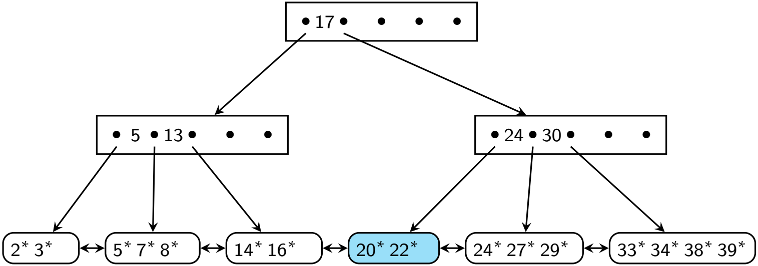
Ejemplo de como eliminar un elemento en B+-trees

Para eliminar el elemento 19*:



Ejemplo de como eliminar un elemento en B+-trees

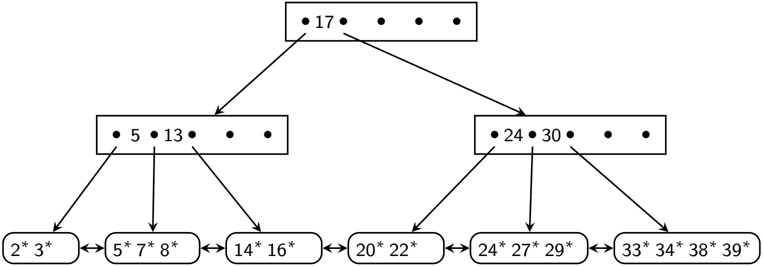
Para eliminar el elemento 19^*:



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

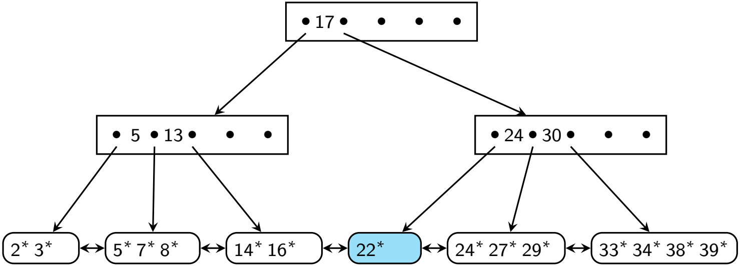
Ejemplo de como eliminar un elemento en B+-trees

Para eliminar el elemento 20*:



Ejemplo de como eliminar un elemento en B+-trees

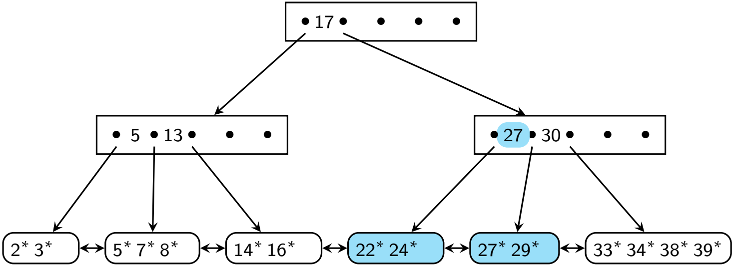
Para eliminar el elemento 20^*:



Debemos hacer redistribución o "unsplit" de las páginas.

Ejemplo de como eliminar un elemento en B+-trees

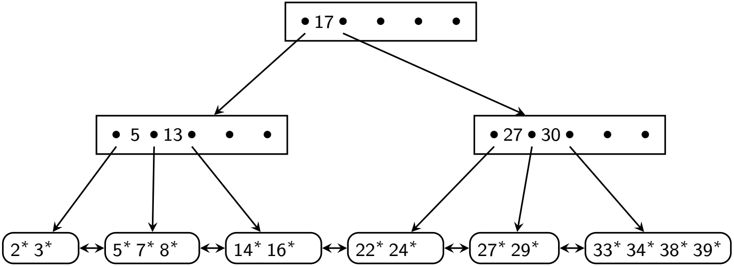
Para eliminar el elemento 20^*:



Debemos hacer redistribución o “unsplit” de las páginas.

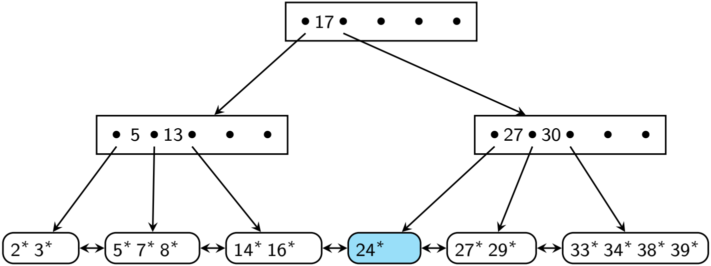
Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22^*:



Ejemplo de como eliminar un elemento en B+-trees

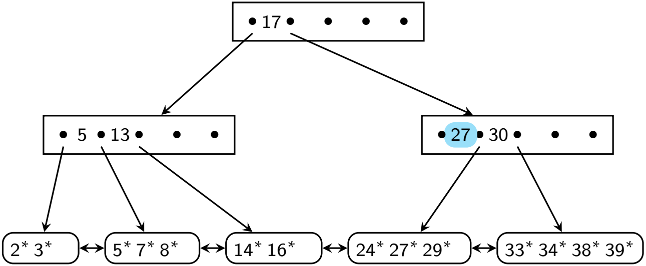
Ahora, si deseamos eliminar el elemento 22*:



Necesitamos hacer un "unsplit" de las páginas.

Ejemplo de como eliminar un elemento en B+-trees

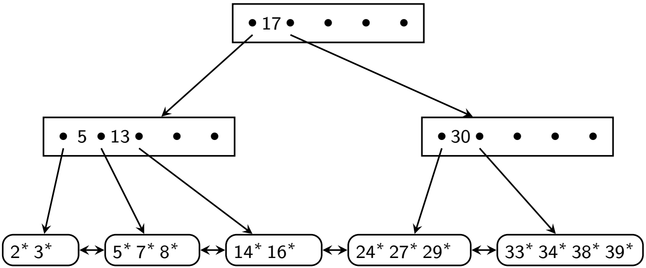
Ahora, si deseamos eliminar el elemento 22^*:



Necesitamos hacer un "unsplit" de las páginas.

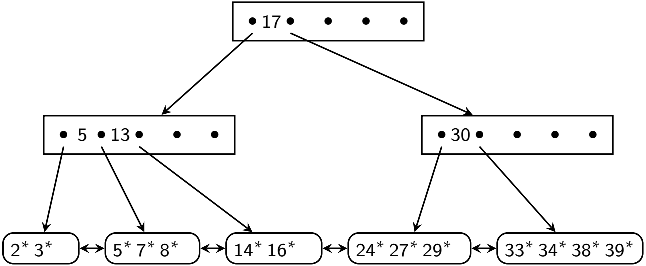
Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22*:



Ejemplo de como eliminar un elemento en B+-trees

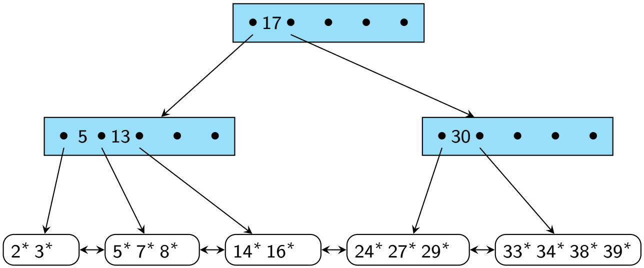
Ahora, si deseamos eliminar el elemento 22^*:



Otro "unsplit", pero ahora del directorio.

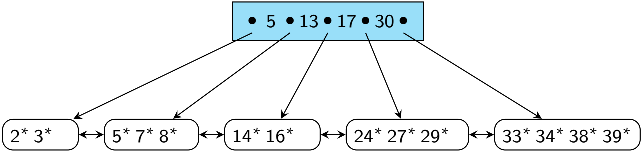
Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22^*:



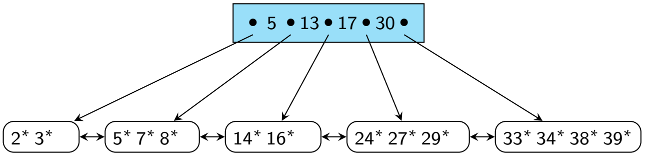
Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22*:



Ejemplo de como eliminar un elemento en B+-trees

Ahora, si deseamos eliminar el elemento 22^*:

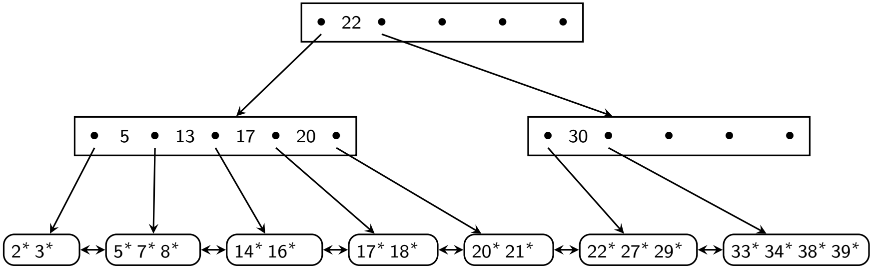


Si el root tiene un solo elemento,
esta es la única manera de decrementar la altura H del árbol.

Eliminación y redistribución de elementos

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

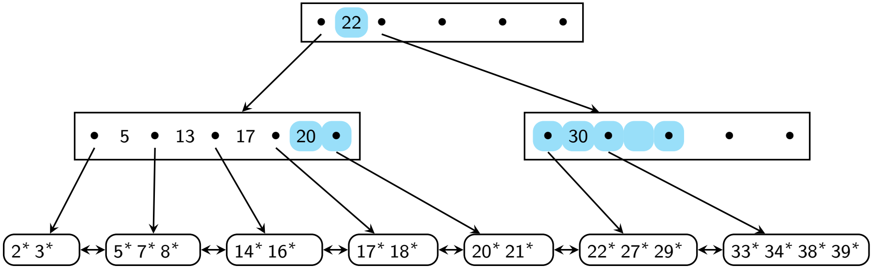
Ejemplo



Eliminación y redistribución de elementos

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

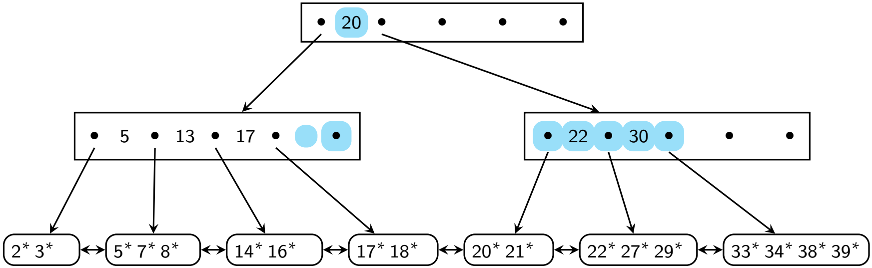
Ejemplo



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:

- T: el número de tuplas.
- B: el tamaño de una página.

El costo de cada operación (en I/O):

$$\begin{array}{ll} \text{Busqueda:} & \mathcal{O}(\log_{\frac{B}{2}}(\frac{2 \cdot T}{B})) \\ \text{Insertar:} & \mathcal{O}(\log_{\frac{B}{2}}(\frac{2 \cdot T}{B})) \\ \text{Eliminar:} & \mathcal{O}(\log_{\frac{B}{2}}(\frac{2 \cdot T}{B})) \end{array}$$

Costo depende logaritmicamente en base B!

Duplicados en B+-trees

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

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

- usar páginas con **overflow** , o
- data entries extendidos con una llave compuesta $(k, \texttt{id})$, 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.