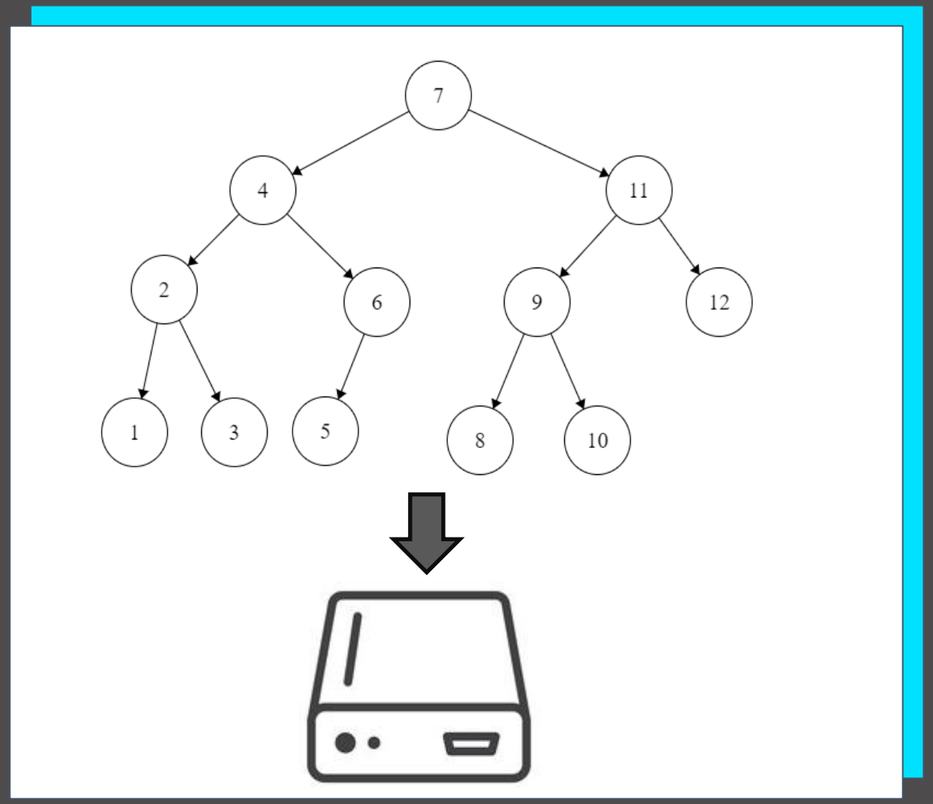
Árboles B
(semana 8)

Heider Sanchez
hsanchez@utec.edu.pe

TRANSFORMATEC

AVL: Limitaciones

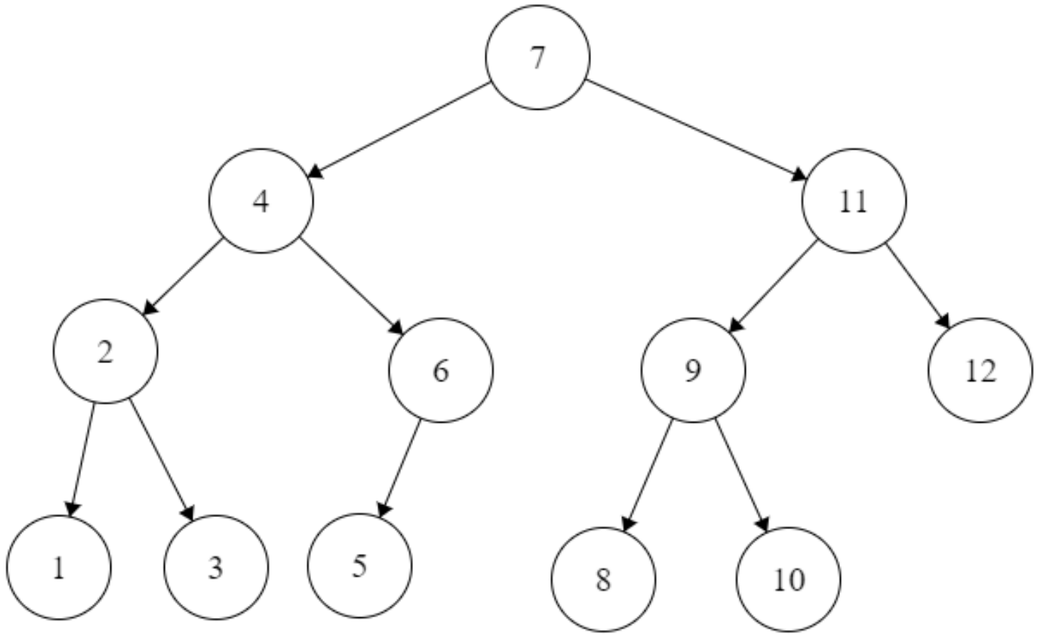


¿Cómo gestionamos una gran colección de datos con memoria RAM limitada?

¿Cómo hacemos uso eficiente de la memoria secundaria para serializar el árbol?

Árboles B

Si necesitamos mantener los nodos del BST en memoria secundaria **¿Cuántos accesos se requieren?**

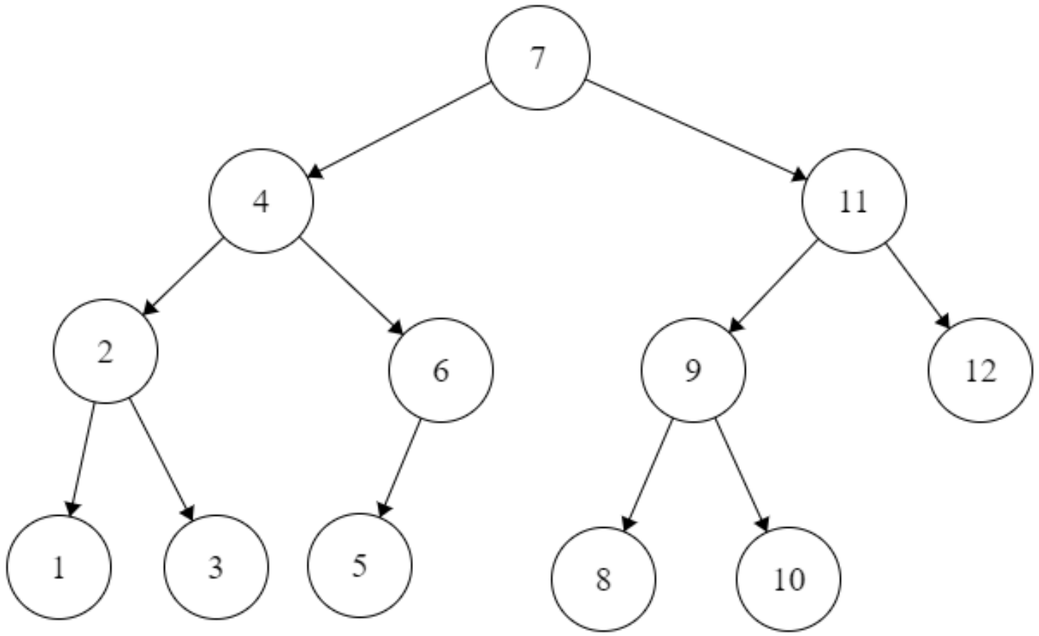


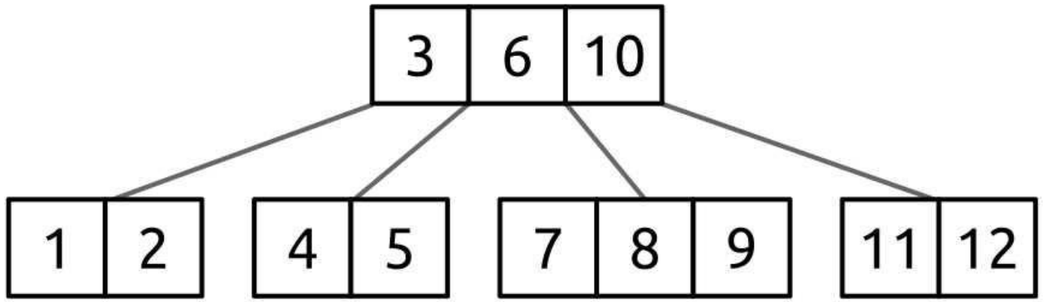
¿Un árbol ternario?

La memoria RAM es entre 1,000 y 100,000 veces más rápida que la memoria secundaria (como discos duros o SSDs)

Árboles B

Si necesitamos mantener los datos del BST en memoria secundaria **¿Cuántos accesos se requieren?**





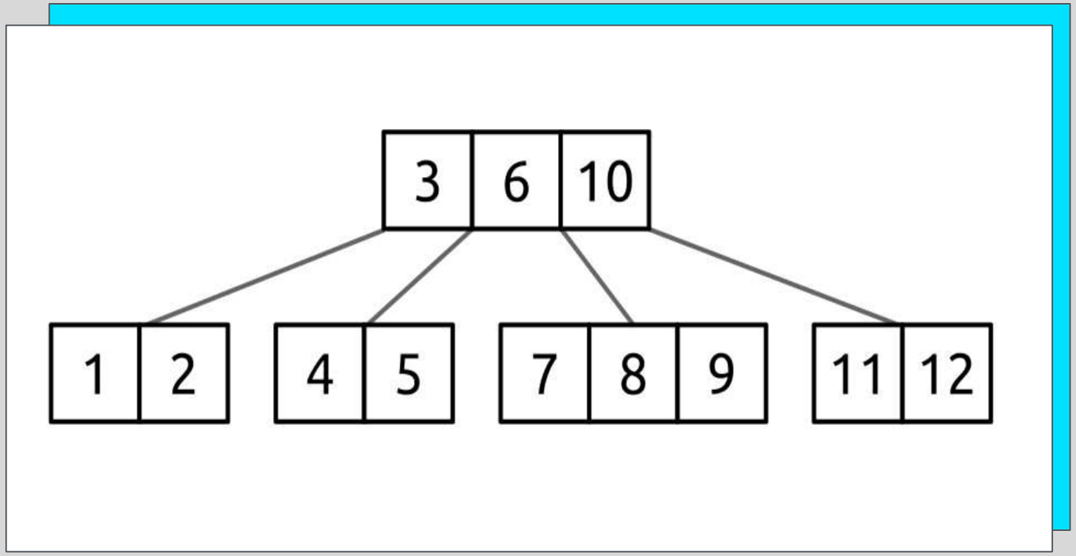
B Tree

¿Por qué utilizar árboles B? ¿En qué casos sería mejor utilizar árboles B que ABB?

Básicamente cuando tenemos grandes cantidades de datos y poca capacidad de memoria principal, por eso son un estándar para organizar índices en sistemas de bases de datos en memoria secundaria.

Viendo la forma que tiene un árbol B a la derecha, cuál será la relación entre keys (elementos de un nodo) e hijos del nodo?

En un árbol B, los nodos tienen *"M"* hijos y *"M-1"* keys. La profundidad del árbol sigue siendo $\log_M(n)$, pero en base M.

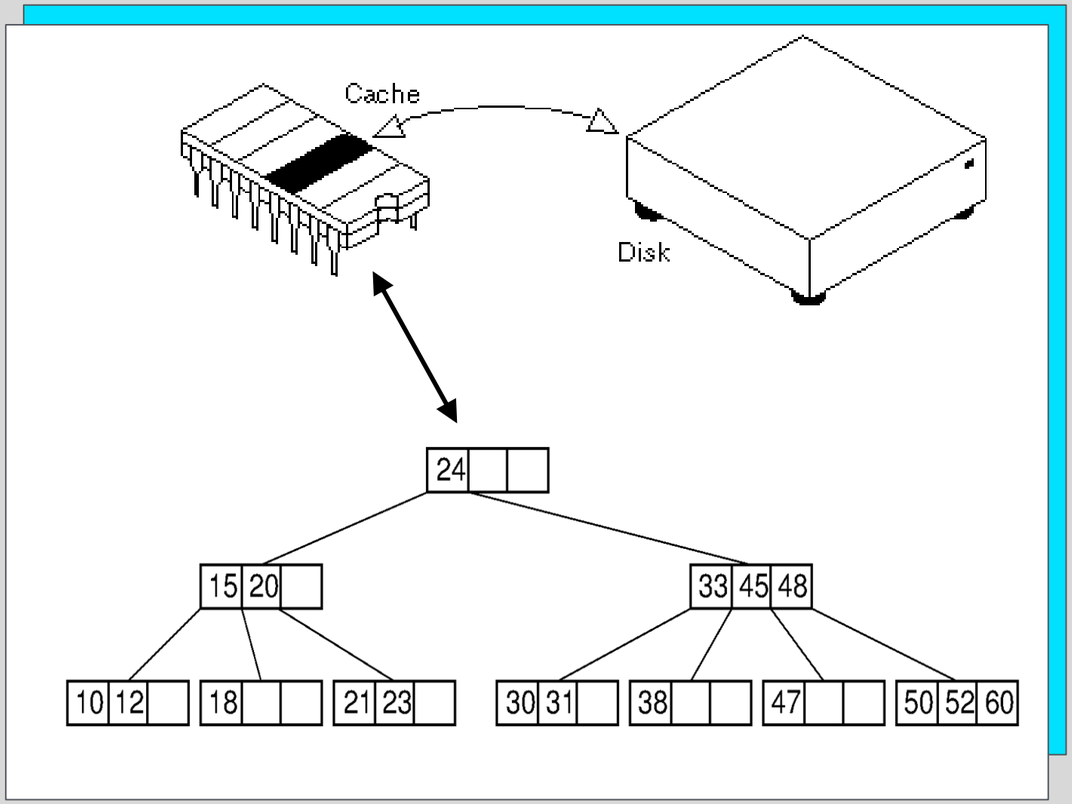


B Tree

Al trabajar con memoria secundaria, se suele aprovechar el tamaño de la cache para minimizar los acceso a disco, ya que estas **suelen ser muy caras**.

Por ello, los **nodos de un árbol B tienden a ser grandes,** del tamaño de la cache, para reducir los accesos a disco.

De esta manera, podemos manejar más elementos en la caché, evitando acceder al disco cada vez que operamos con un solo elemento del árbol (el acceso a disco ocurre solo al descender un nivel en el árbol).



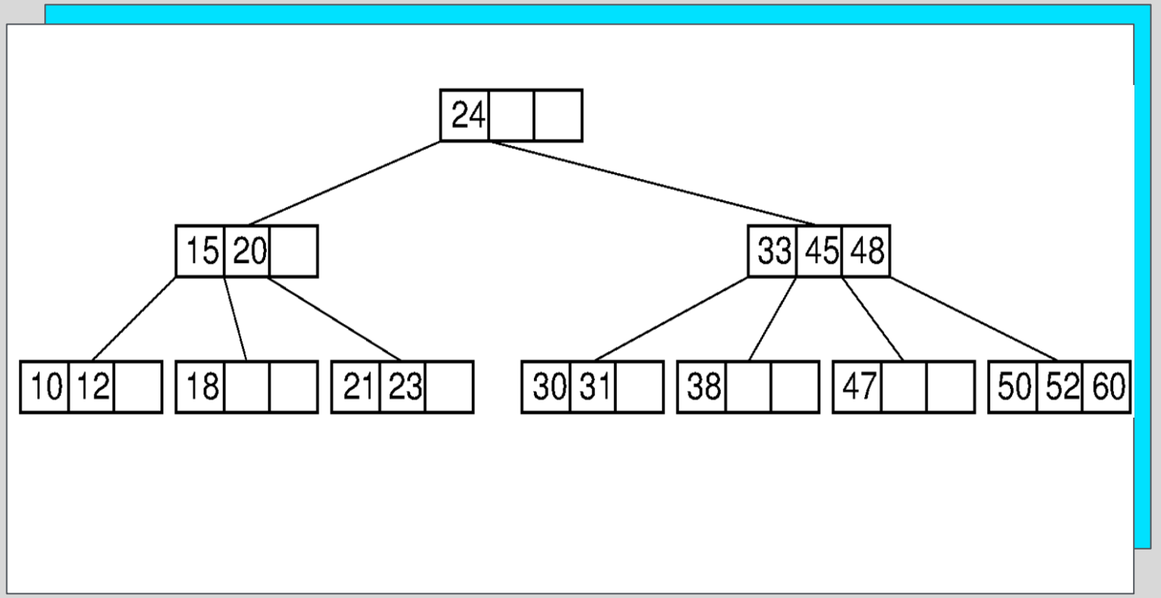
B Tree

Características:

- La raíz tiene al menos 2 hijos y contiene al menos un key
- Todas las hojas están al mismo nivel
- En cada nodo, todas las keys están ordenadas
- Si M es el orden del árbol, entonces:
  - Cada nodo excepto la raíz, [M/2] - 1<= numero de keys <= M - 1
  - Cada nodo interno excepto la raiz, [M/2]<= número de hijos <= M

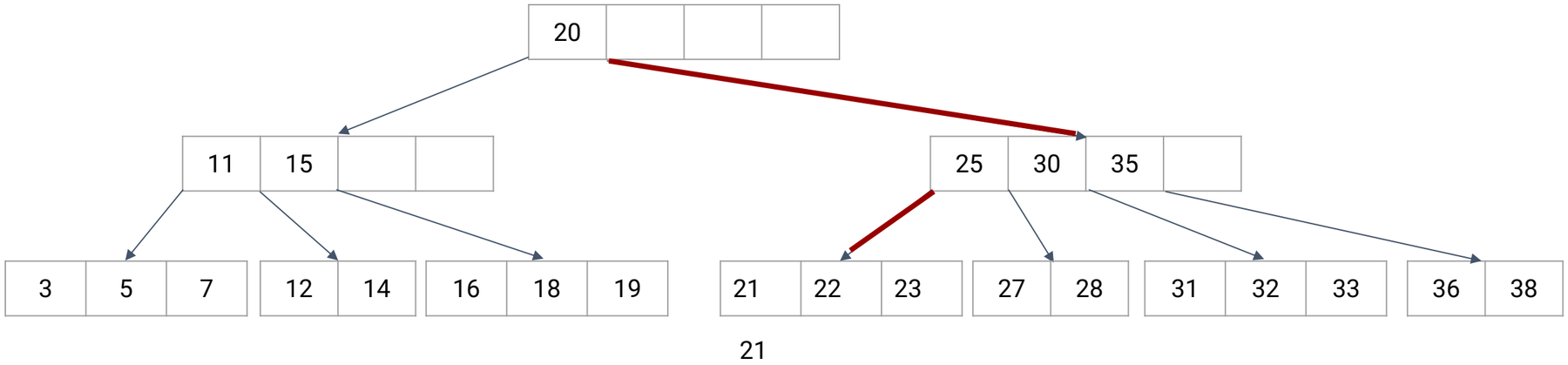
Entonces, en el ejemplo de la derecha:

➔ ¿Qué valor tomaría M?



Búsquedas en árboles B

Buscar el 21:



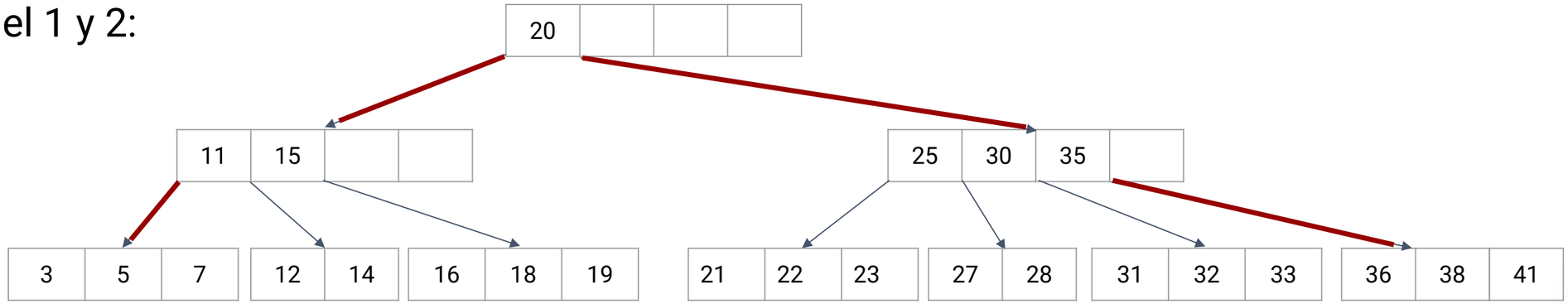
La búsqueda es log(n), al igual que un BST:

1. Primero buscamos en el nodo actual si está la key, si no vamos al hijo correspondiente
2. Derecha para mayores, izquierda para menores. Igual que en el BST
3. Si no se encuentra en las hojas, entonces no está en el árbol

Inserciones en árboles B

Insertar el 41:

Insertar el 1 y 2:



Se realiza una búsqueda en el árbol para encontrar la posición donde se debe insertar

Si hay espacio disponible, se inserta

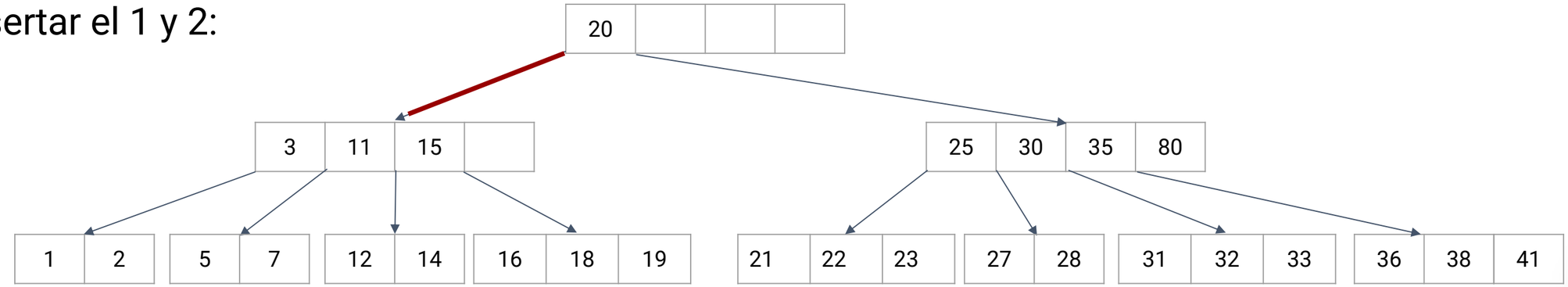
Si no hay espacio, se divide en dos nodos y se sube el elemento central

Se valida que las propiedades se cumplan

Inserciones en árboles B

Insertar el 41:

Insertar el 1 y 2:



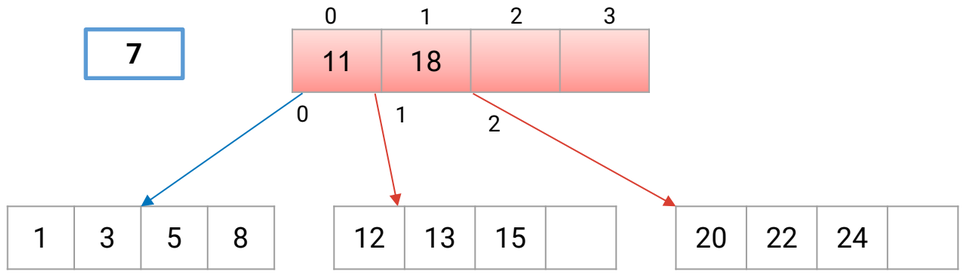
Se realiza una búsqueda en el árbol para encontrar la posición donde se debe insertar

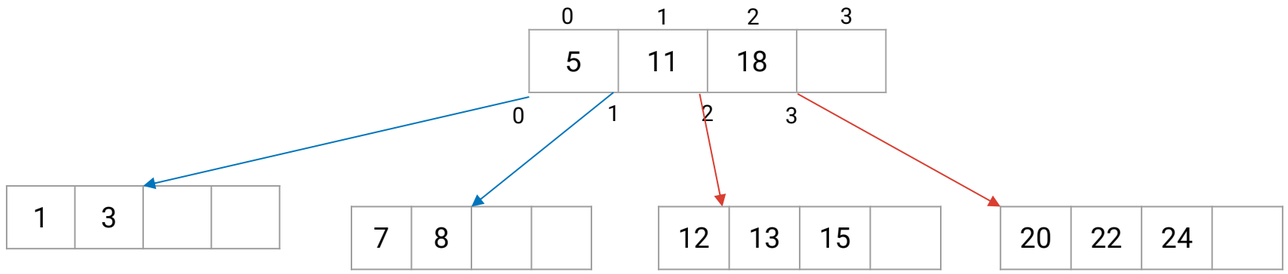
Si hay espacio disponible, se inserta

Si no hay espacio, se divide en dos nodos y se sube el elemento central

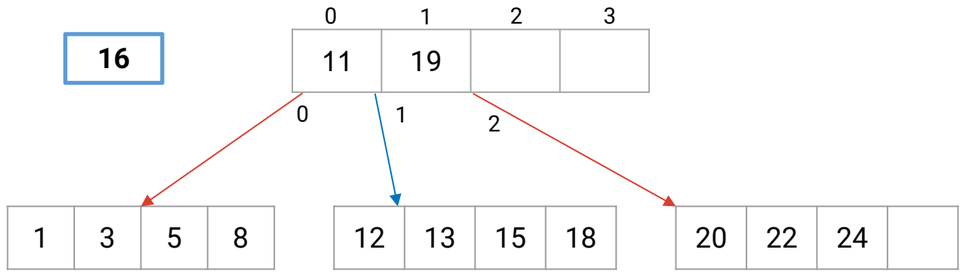
Se valida que las propiedades se cumplan

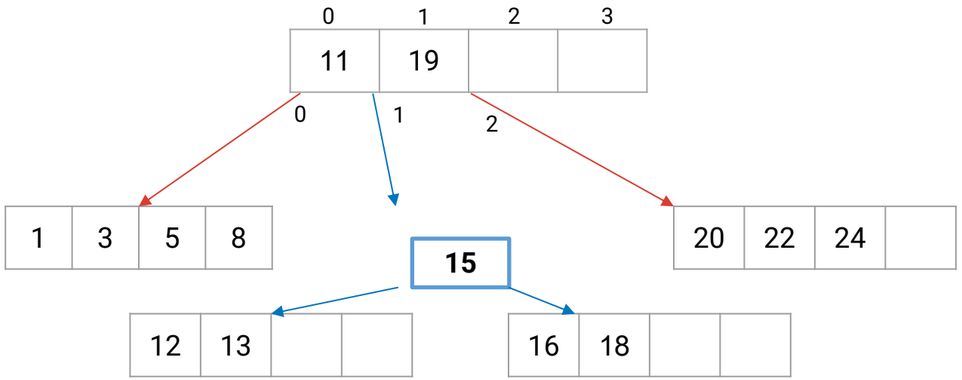
Insertar 7:

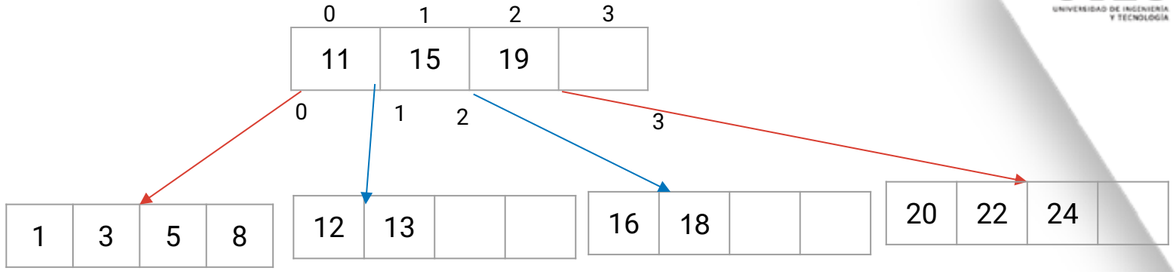

7
0
1
2
3
11
18
0
1
2
1
3
5
8
12
13
15
20
22
24


0
1
2
3
5
11
18
0
1
2
3
1
3
7
8
12
13
15
20
22
24

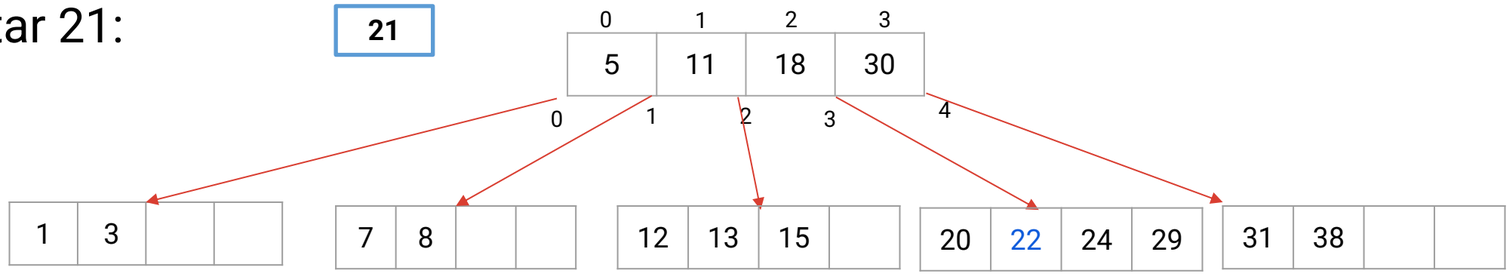
Insertar 16:

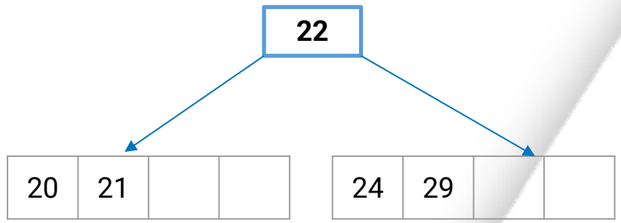

16
0 1 2 3
11 19
0 1 2
1 3 5 8
12 13 15 18
20 22 24

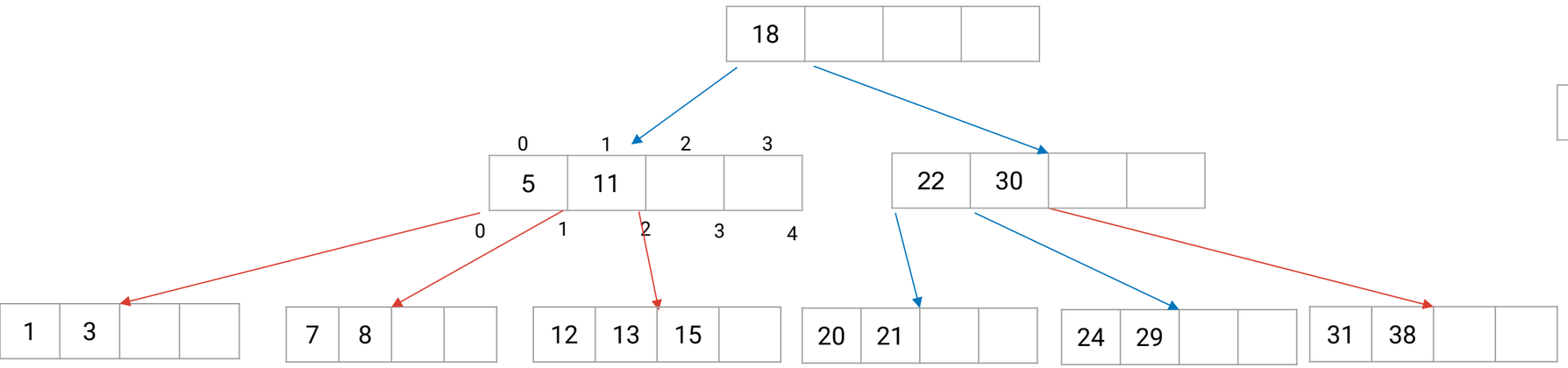

0 1 2 3
11 19
0 1 2
1 3 5 8
15
12 13
16 18
20 22 24

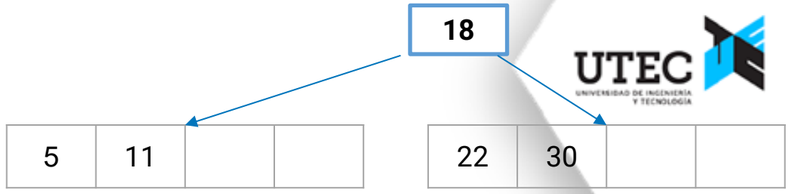

0 1 2 3
11 15 19
0 1 2 3
1 3 5 8
12 13
16 18
20 22 24

Insertar 21:


21
0 1 2 3
5 11 18 30
0 1 2 3 4
1 3
7 8
12 13 15
20 22 24 29
31 38

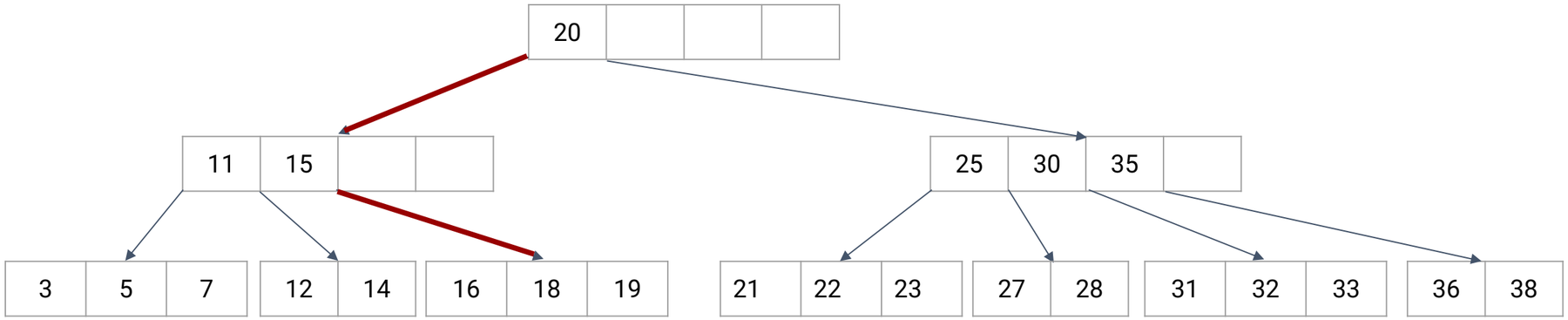

22
20 21
24 29


18
0 1 2 3
5 11
22 30
0 1 2 3 4
1 3
7 8
12 13 15
20 21
24 29
31 38


18
5 11
22 30

Borrado en árboles B (caso 0)

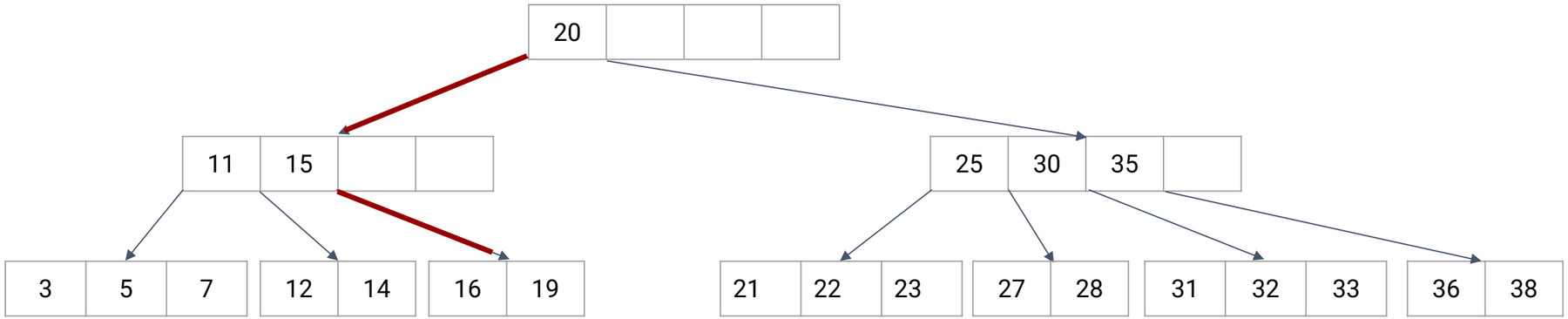
Borrar el 18:



Se debe procurar hacer el borrado siempre en las hojas ya que es más simple
Por tanto, si el key a remover no está en una hoja, se debería hacer un cambio con el anterior (elemento más a la derecha del hijo izquierdo) o siguiente (elemento más a la izquierda del hijo derecho) elemento.
Similar a como se trabaja con BST

Borrado en árboles B (caso 0)

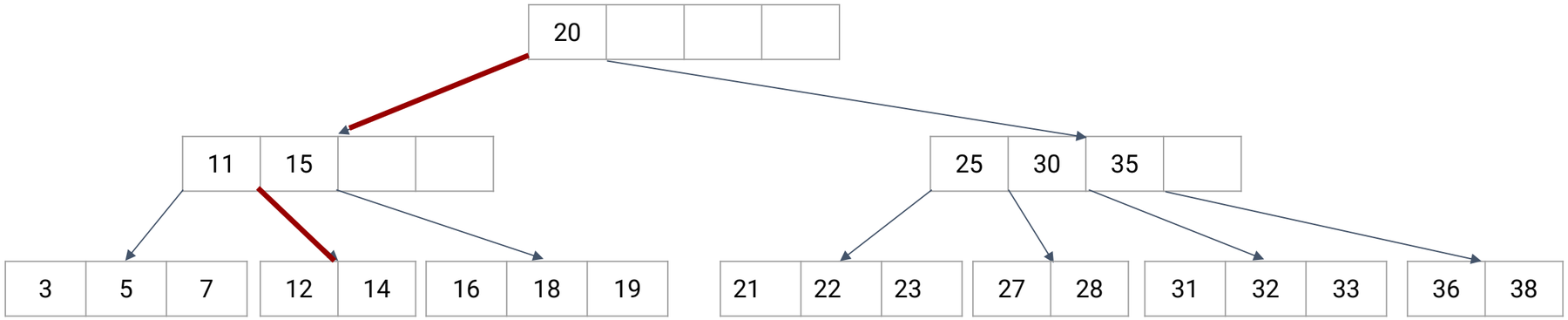
Borrar el 18:



Se debe procurar hacer el borrado siempre en las hojas ya que es más simple
Por tanto, si el key a remover no está en una hoja, se debería hacer un cambio con el anterior (elemento más a la derecha del hijo izquierdo) o siguiente (elemento más a la izquierda del hijo derecho) elemento.
Similar a como se trabaja con BST

Borrado en árboles B (caso 1)

Borrar el 14:



Si al borrar una key no tenemos suficientes keys para mantener las propiedades, entonces aplicamos una rotación

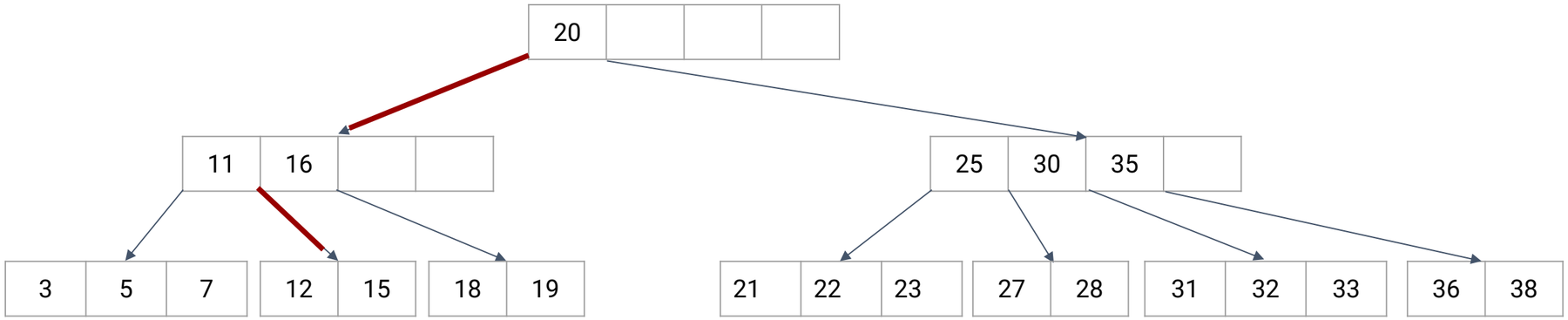
Lo primero es verificar a los nodos hermanos y ver si están sobre el mínimo de keys.

Se toma el nodo anterior o siguiente, y su padre baja al nodo de donde estamos borrando el elemento

La key del nodo hermano sube como padre

Borrado en árboles B (caso 1)

Borrar el 14:



Si al borrar una key no tenemos suficientes keys para mantener las propiedades, entonces aplicamos una rotación

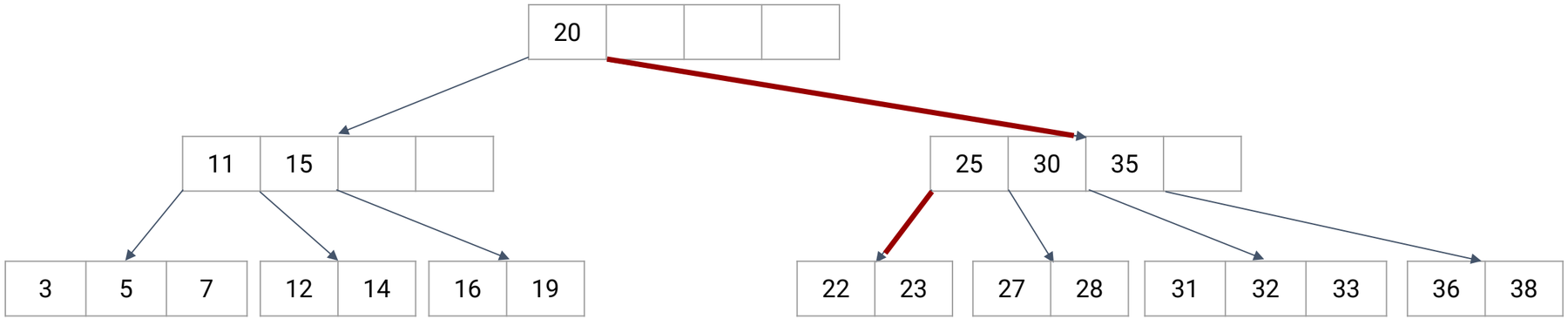
Lo primero es verificar a los nodos hermanos y ver si están sobre el mínimo de keys.

Se toma la key anterior o siguiente, y su padre baja al nodo de donde estamos borrando el elemento

La key del nodo hermano sube como padre

Borrado en árboles B (caso 2)

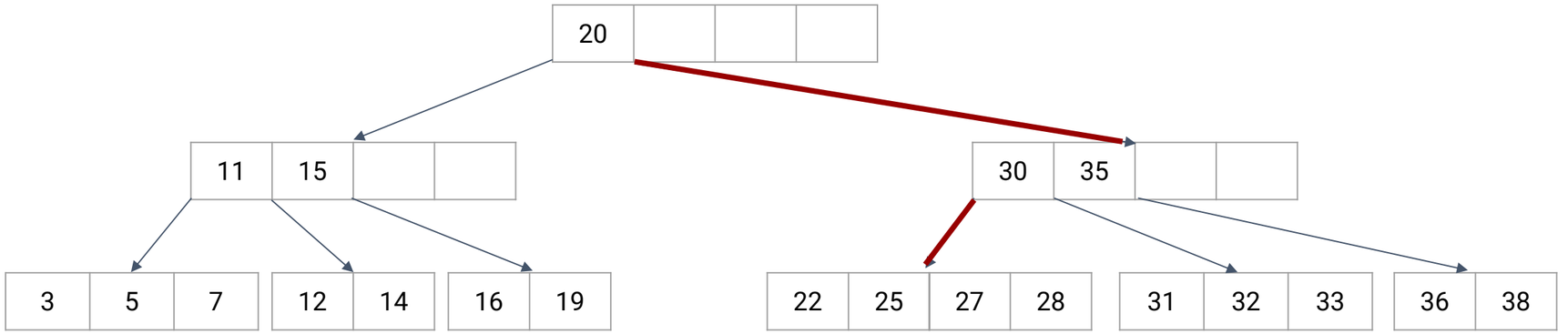
Borrar el 23:



Si los nodos hermanos no tienen suficientes elementos para remover una key, entonces se hace un merge
Se mueve la key padre hacia abajo, y se combinan los dos hijos

Borrado en árboles B (caso 2)

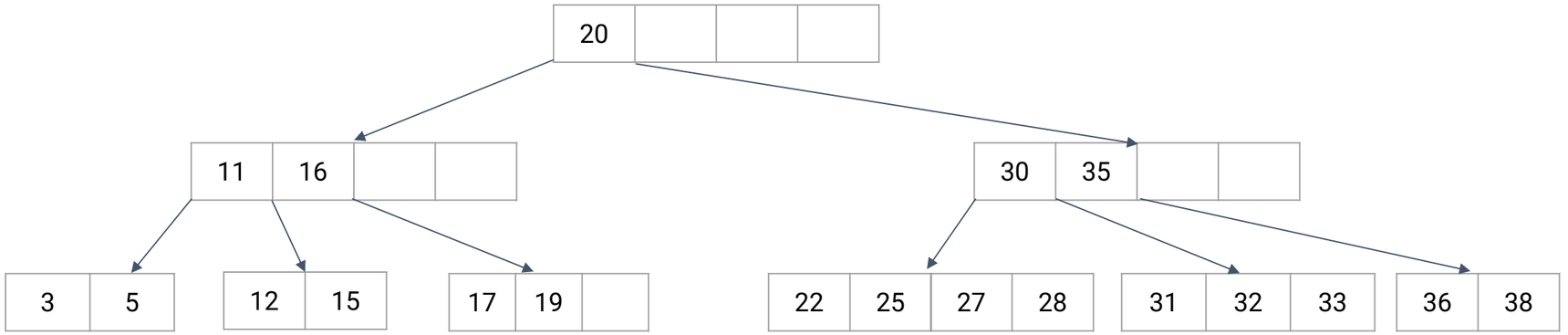
Borrar el 23:



Si los nodos hermanos no tienen suficientes elementos para remover una key, entonces se hace un merge
Se mueve la key padre hacia abajo, y se combinan los dos hijos

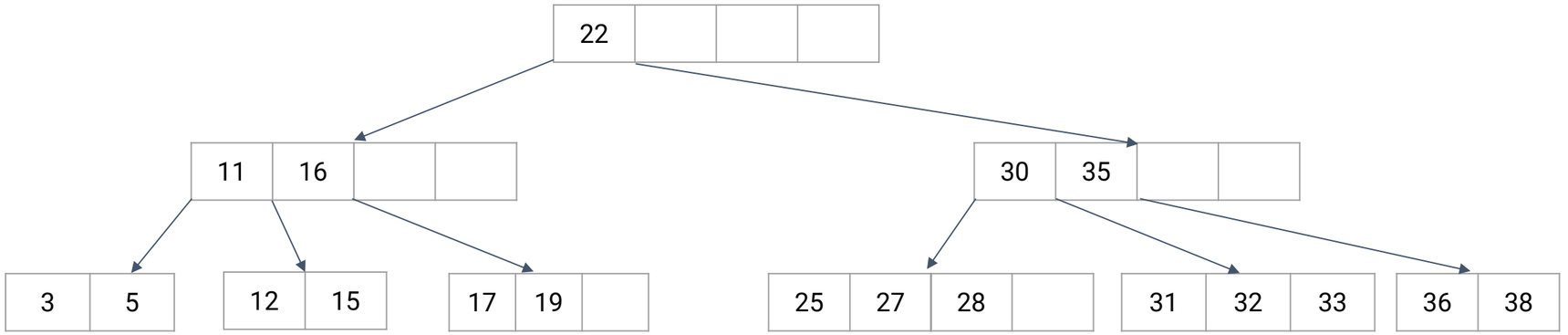
Borrado en árboles B (caso ??)

Borrar el 20:



Borrado en árboles B (caso 3)

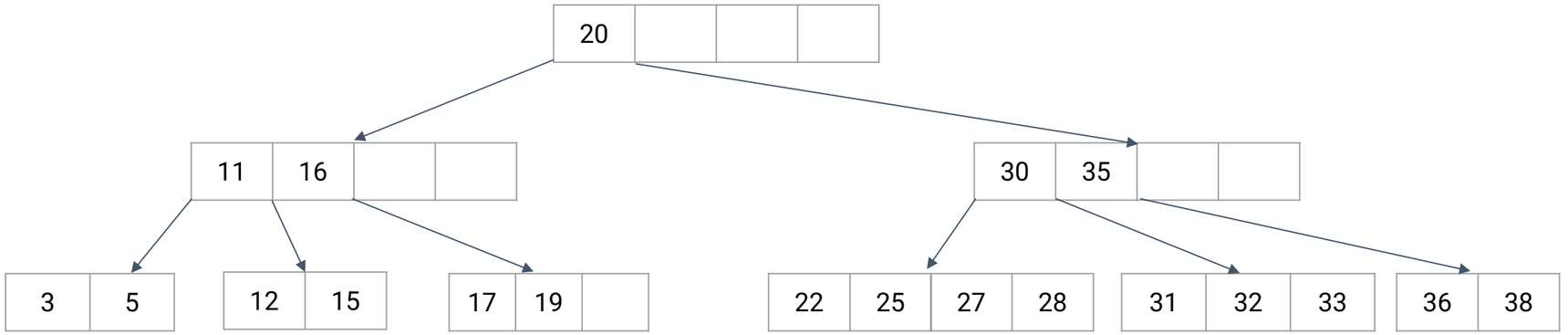
Borrar el 20:



- Reemplazar por el sucesor
- Eliminar el sucesor

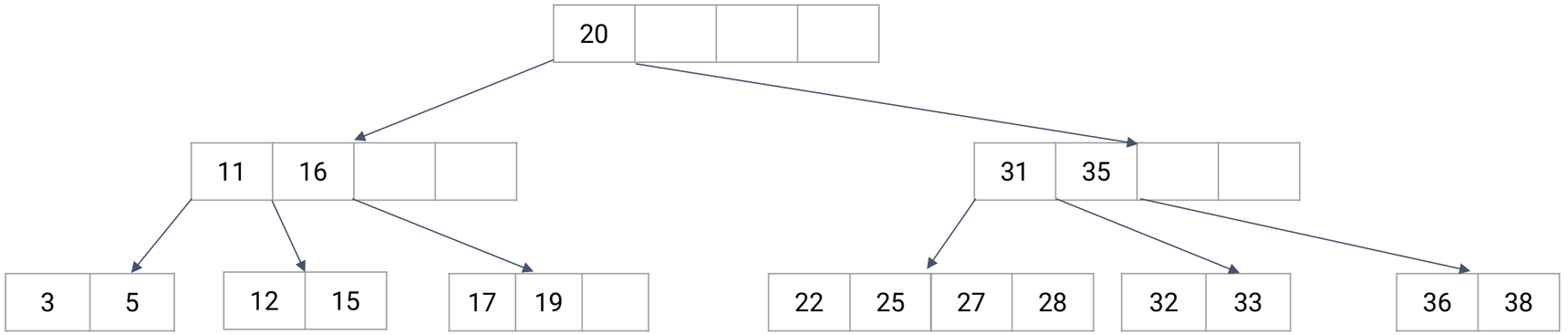
Borrado en árboles B (caso 3)

Borrar el 30:



Borrado en árboles B (caso 3)

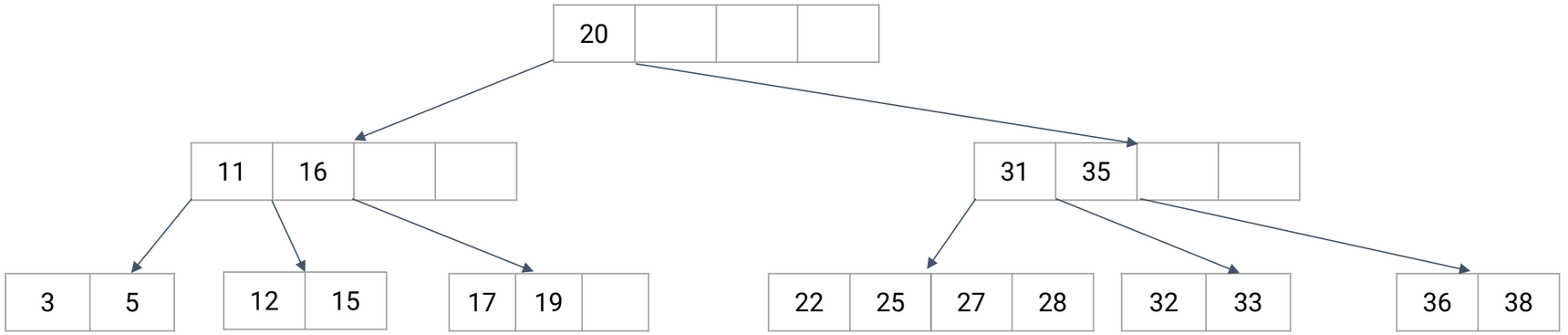
Borrar el 30:



- Reemplazar por el sucesor
- Eliminar el sucesor

Borrado en árboles B (caso 3)

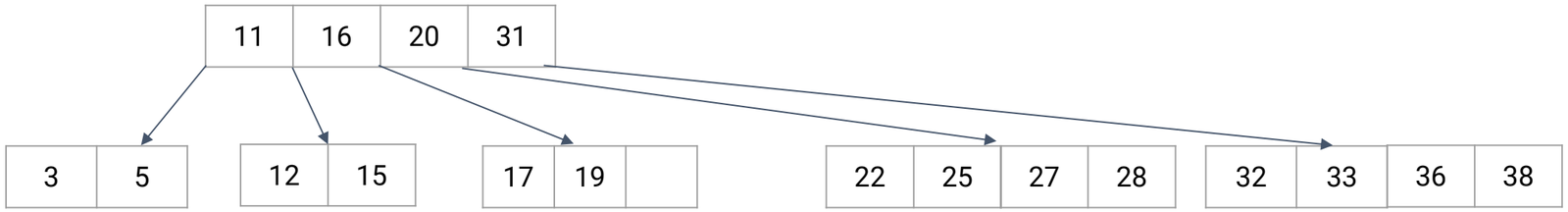
Borrar el 35:



- Reemplazar por el sucesor
- Eliminar el sucesor

Borrado en árboles B (caso 3)

Borrar el 35:



- Reemplazar por el sucesor
- Eliminar el sucesor

Árboles B

Considerando un árbol de orden M=3.
Insertar:

45
75
100
36
120
70
11
111
47
114

Eliminar:

100
111
45

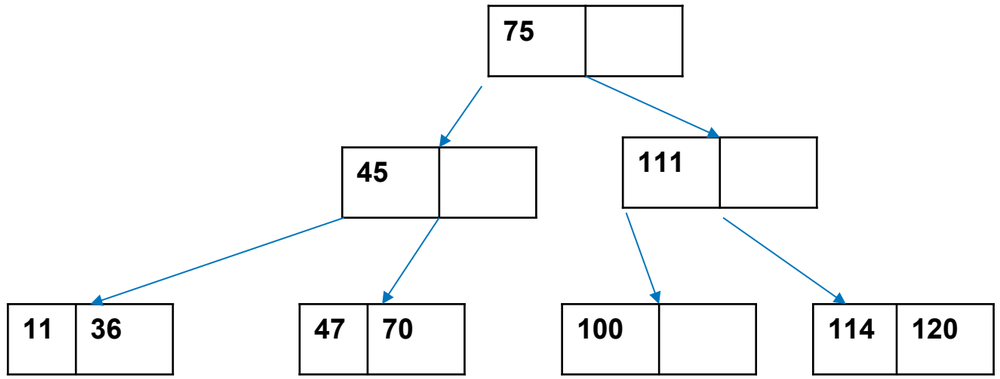
Árboles B

Considerando un árbol de orden 3.
Insertar:

45
75
100
36
120
70
11
111
47
114

Eliminar:

100
111
45



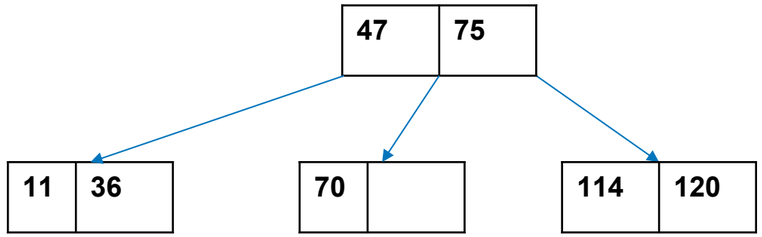
Árboles B

Considerando un árbol de orden 3.
Insertar:

45
75
100
36
120
70
11
111
47
114

Eliminar:

100
111
45



Para eliminar un elemento en nodo interno, usar el suscesor

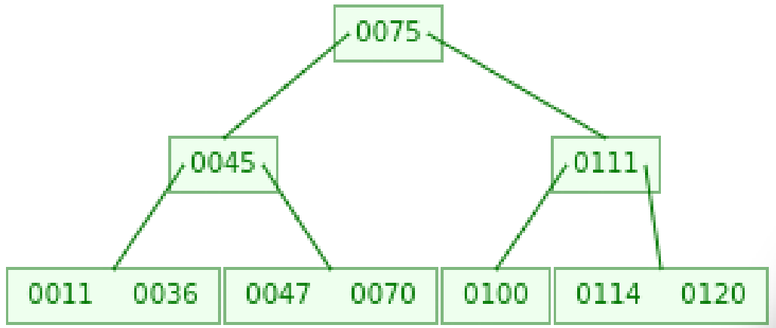
Árboles B

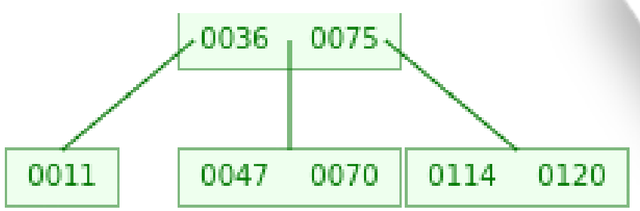
Considerando un árbol de orden 3.
Insertar:

45
75
100
36
120
70
11
111
47
114

Eliminar:

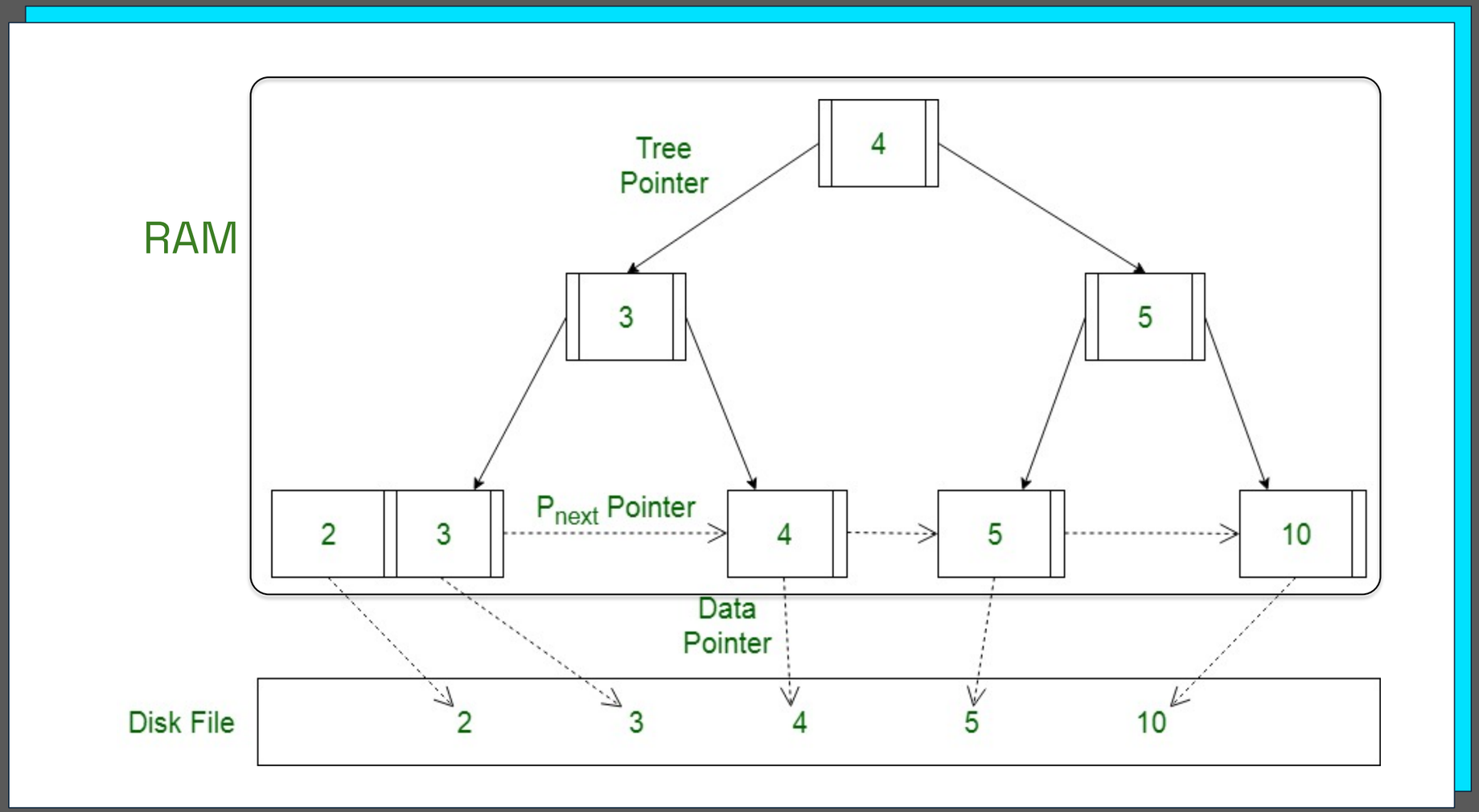
100
111
45





Para eliminar un elemento en nodo interno, usar el predecesor

B+ Tree


RAM
Tree
Pointer
4
3
5
2
3
P_{next} Pointer
4
5
10
Data
Pointer
Disk File
2
3
4
5
10

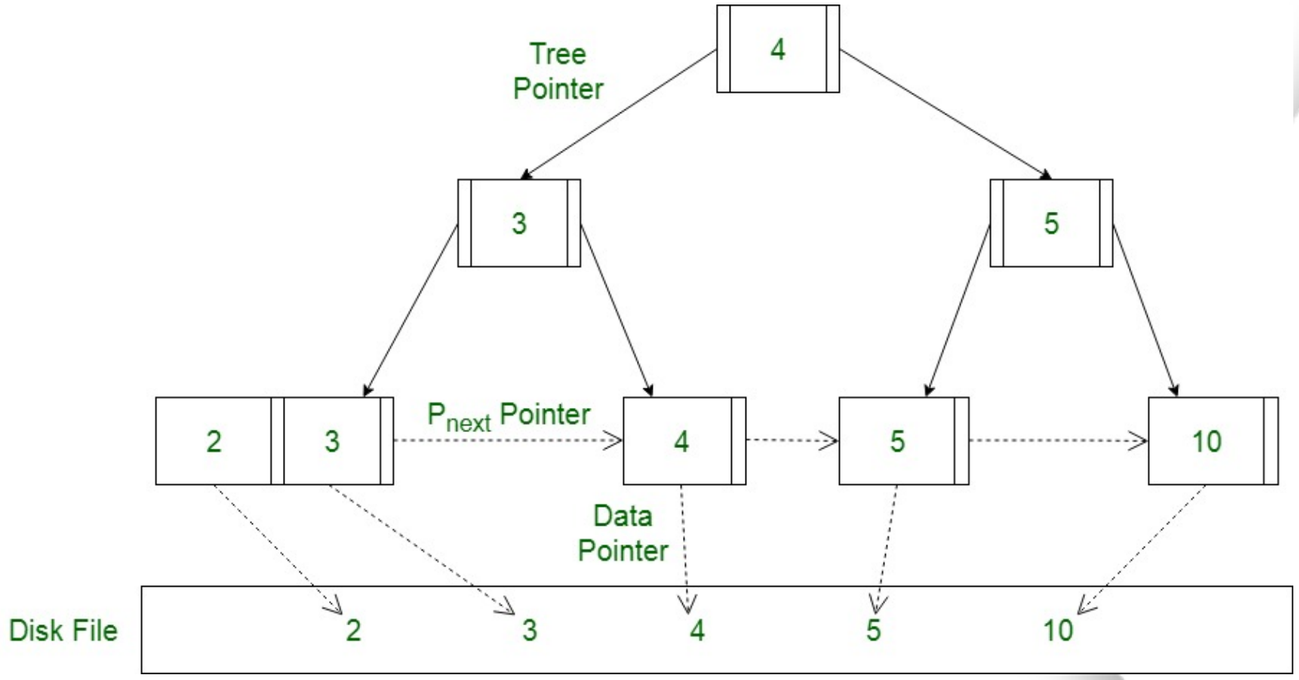
Árboles B+

Son una extensión de los árboles B, por tanto su estructura es bastante similar.

Una diferencia importante es que **toda la data se almacena en las hojas y están conectadas con la siguiente hoja**.

Igual que en los árboles B, todas las hojas están en el mismo nivel.

Los **nodos internos son solo marcadores** que nos indican a donde ir.

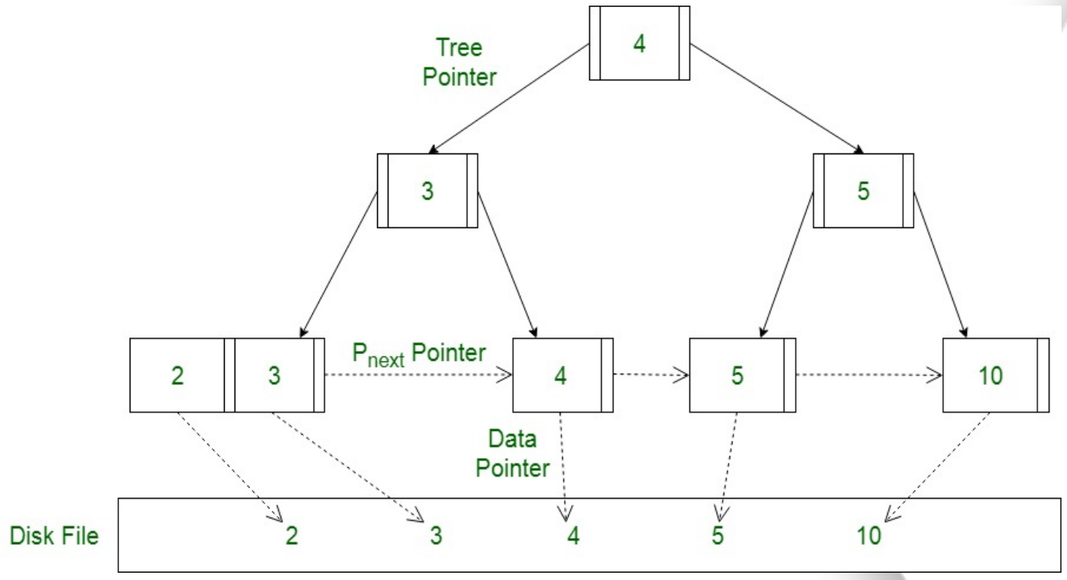


Árboles B+

Cuál creen que es la optimización que nos brindan los árboles B+?
Un ejemplo son range queries en bases de datos. Por ejemplo podríamos obtener los datos con índice 3 hasta 8

Como no hay datos asociados en los nodos y sólo keys, los nodos pueden tener más valores que encajan en un bloque de memoria

Obtener todos los datos almacenados solo requiere una pasada linear sobre las hojas, mientras que en un árbol B se necesitaría recorrer todos los niveles



Árboles B+

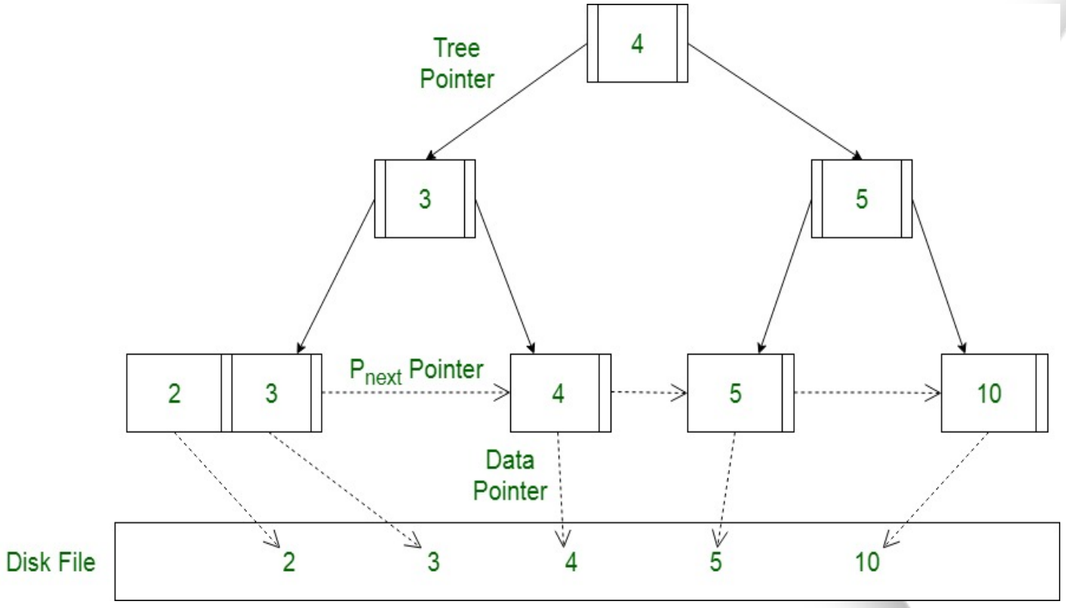
Qué problema creen que traen los árboles B+?

La implementación del insertar y borrado es más compleja

En los árboles B los nodos internos también contiene data, se podría ubicar los nodos más requeridos cerca a la raíz.

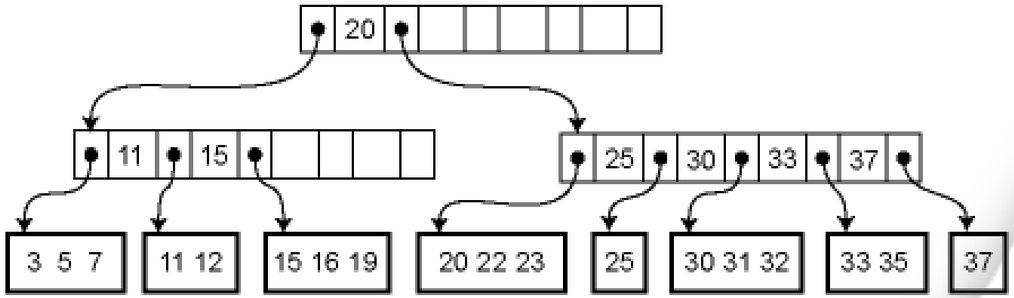
En el B+, cuando se encuentra la key en un nodo interno, se debe continuar hasta llegar a las hojas.

Las keys se repiten en los nodos internos y hojas



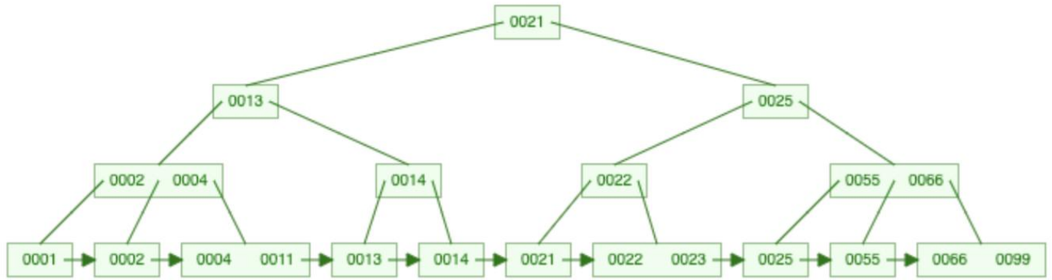
Árboles B+

Insertar:
Recuerden que en un B+, los valores internos siempre estarán en las hojas. Por ejemplo, root estará a la derecha más a la izquierda o izquierda más a la derecha



Al insertar siempre recuerden mantener el valor en último nivel y solo subir un puntero a los nodos internos/root

Se debe elegir el lado del cual se subirá el elemento, puede ser un elemento de la izquierda o derecha

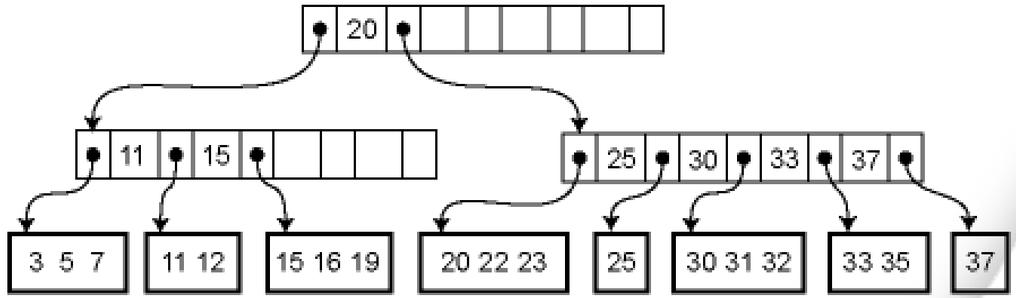


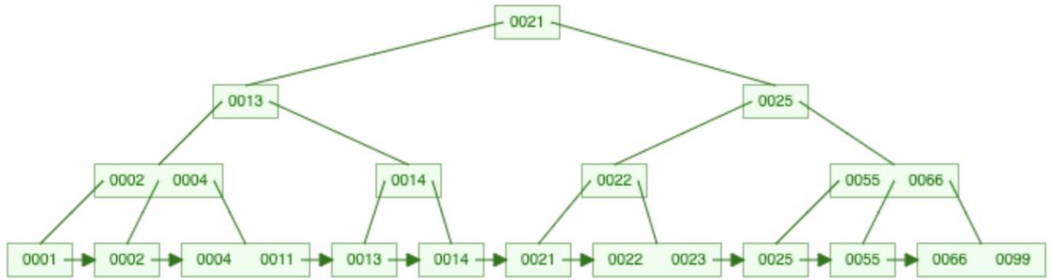
Árboles B+

Remover:
Al eliminar un elemento, primero deben encontrarlo en la hojas, marcando los nodos internos con el mismo valor.

También se deben actualizar todos los punteros de los nodos internos/root que apuntaban a ese valor con su sucesor.

El sucesor normalmente es el siguiente (derecha más a la izquierda) o anterior (izquierda más a la derecha).





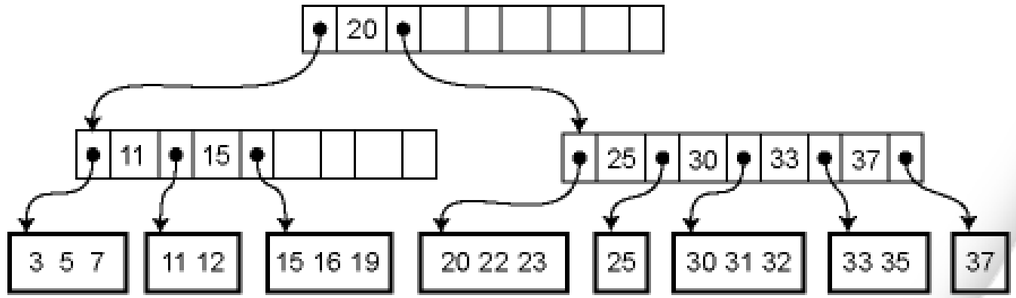
Árboles B+

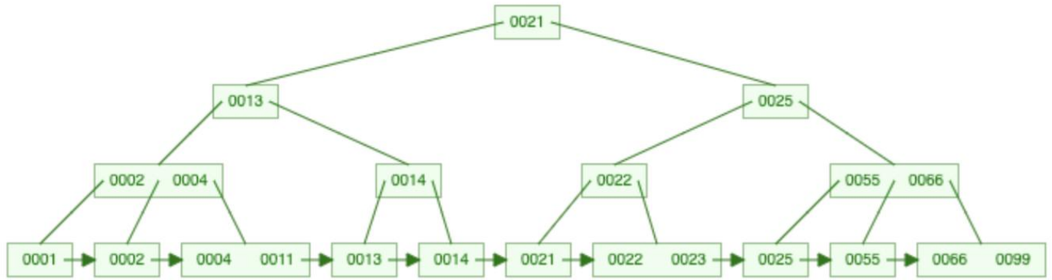
Remover casos:

1. Si hay suficientes keys, solo se elimina.

2. Si no hay suficientes keys, se debe prestar de uno de los hermanos inmediatos. No se olviden de actualizar el padre.

3. En caso no se puedan prestar, se deberá hacer un merge. Posterior se deberán actualizar los nodos internos con el sucesor de ser necesario.





Árboles B+

Considerando un árbol de orden M=3.

Insertar:

45
75
100
36
120
70
11
111
47
114

Eliminar:

75
100
70

- **El criterio de distribucion en el split para una cantidad impar, seria asignar la mitad + 1 al nodo de la izquierda**
- **Una llave en un nodo interno, es el máximo del subárbol izquierdo.**

Árboles B+

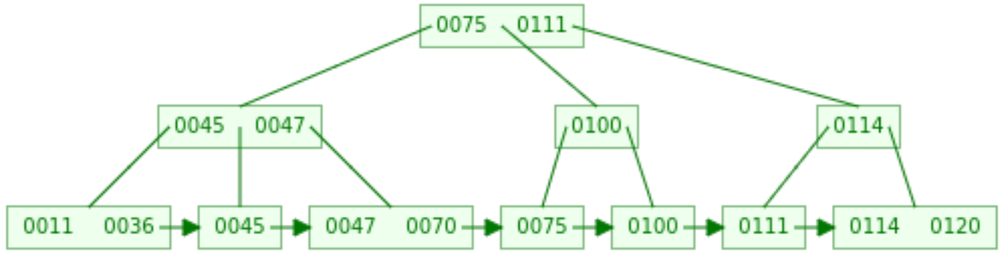
Considerando un árbol de orden M=3.

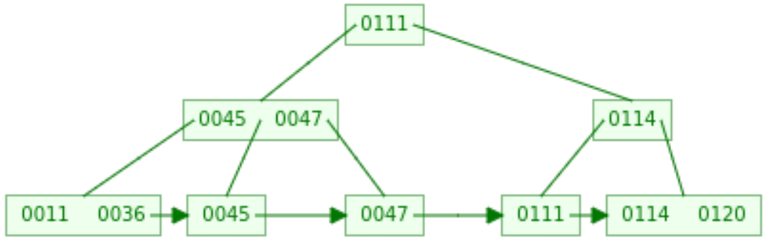
Insertar:

45
75
100
36
120
70
11
111
47
114

Eliminar:

75
100
70





Árboles B+

Considerando un árbol de orden M=4.

Insertar:

45
75
100
36
120
70
11
111
47
114
74

Eliminar:

75
100
114

- **El criterio de distribucion en el split para una cantidad impar, seria asignar la mitad + 1 al nodo de la izquierda**
- **Una llave en un nodo interno, es el máximo del subárbol izquierdo.**