Árboles B

AED II

2020

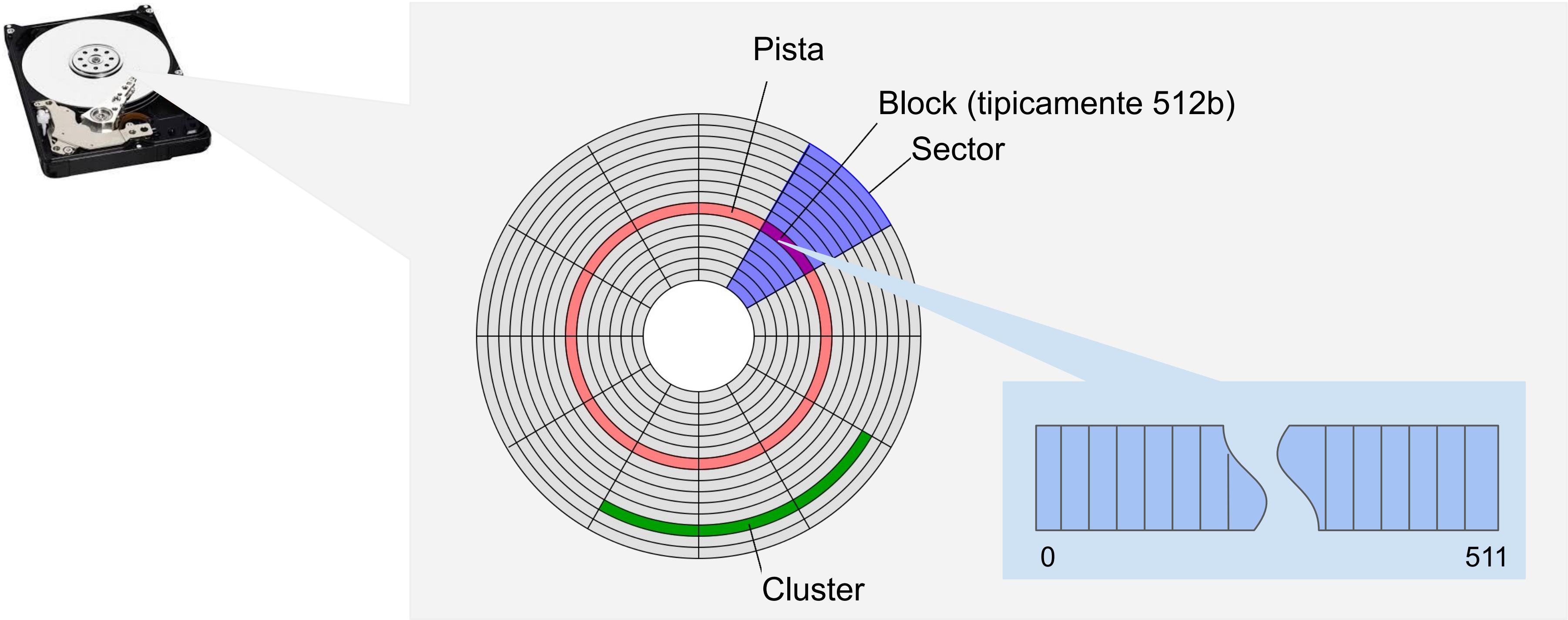
UBP UNIVERSIDAD BLAS PASCAL

árbol B

Creado por Rudolf Bayer y Ed McCreight

No han explicado el significado de la letra B de su nombre. Se cree que la B es de balanceado, dado que todos los nodos hoja se mantienen al mismo nivel en el árbol. La B también puede referirse a Bayer, o a Boeing, porque sus creadores trabajaban en los Boeing Scientific Research Labs por ese entonces.

árbol B: estructura lógica de un disco duro



Se puede direccionar cada byte del disco duro mediante el nro de pista, de sector y el offset

UBP UNIVERSIDAD BLAS PASCAL

base de dato en disco

Estructura

ID	10
Nombre	50
Domicilio	50
Curso	8
Carrera	10
Total	**128**

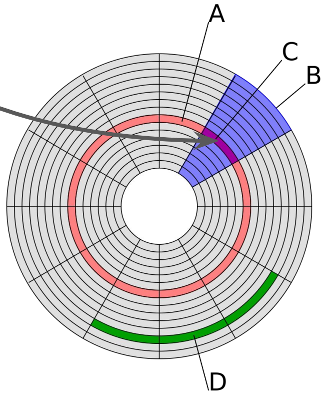
Alumnos

1	Juan Perez	...
2	Pedro Garcia	...
3	Maria Juarez	...
4	Daniela Arce	...
5	Alicia Beri	...
6	Damian Capa	...
7	Hecto Biere	...

...

399	Enzo Zárate	...
400	Ana Fernandez	...

4 registros en un bloque de 512b



Necesitamos 100 bloques para alamacenar los 400 registros

base de dato en disco

```
select * from Alumnos where id = 294
```

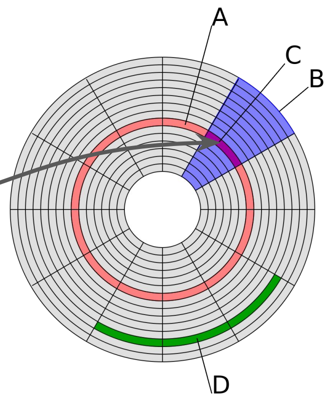
Hasta 100 accesos a disco

Alumnos

1	Juan Perez	...
2	Pedro Garcia	...
3	Maria Juarez	...
4	Daniela Arce	...
5	Alicia Beri	...
6	Damian Capa	...
7	Hecto Biere	...

...

	Enzo Zárate	...
	Ana Fernandez	...



índices

Contiene el número de registro y un puntero a la dirección del disco donde está almacenado

Alumnos

1	Juan Perez	...
2	Pedro Garcia	...
3	Maria Juarez	...
4	Daniela Arce	...
5	Alicia Beri	...
6	Damian Capa	...
7	Hecto Biere	...

...

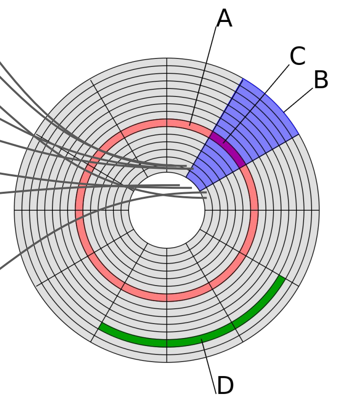
	Enzo Zárate	...
	Ana Fernandez	...

Estructura Idx

ID	10
Puntero	6
Total	**16**

Idx

1	346
2	347
3	348
4	346
5	346
6	346
7	346
400	947



El índice también se va a almacenar en el disco

512 / 16 = 32
400 / 32 = 12.875 ≅ **13**

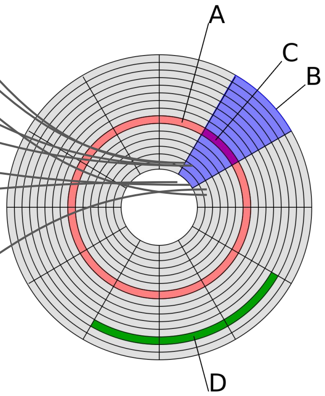
índices

```
select * from Alumnos where id = 294
```

13 + 1 = 14 accesos a disco

Idx

1	346
2	347
3	348
4	346
5	346
6	346
7	346
400	947



El índice también se va a almacenar en el disco

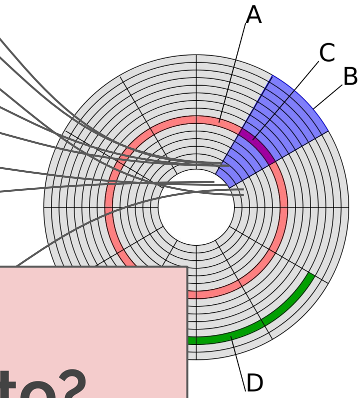
$512 / 16 = 32$
$400 / 32 = 12.875 \cong \mathbf{13}$

índices

```
select * from Alumnos where id = 294
```

Idx

1	346
2	347
3	348
4	346
5	346
6	346
7	346



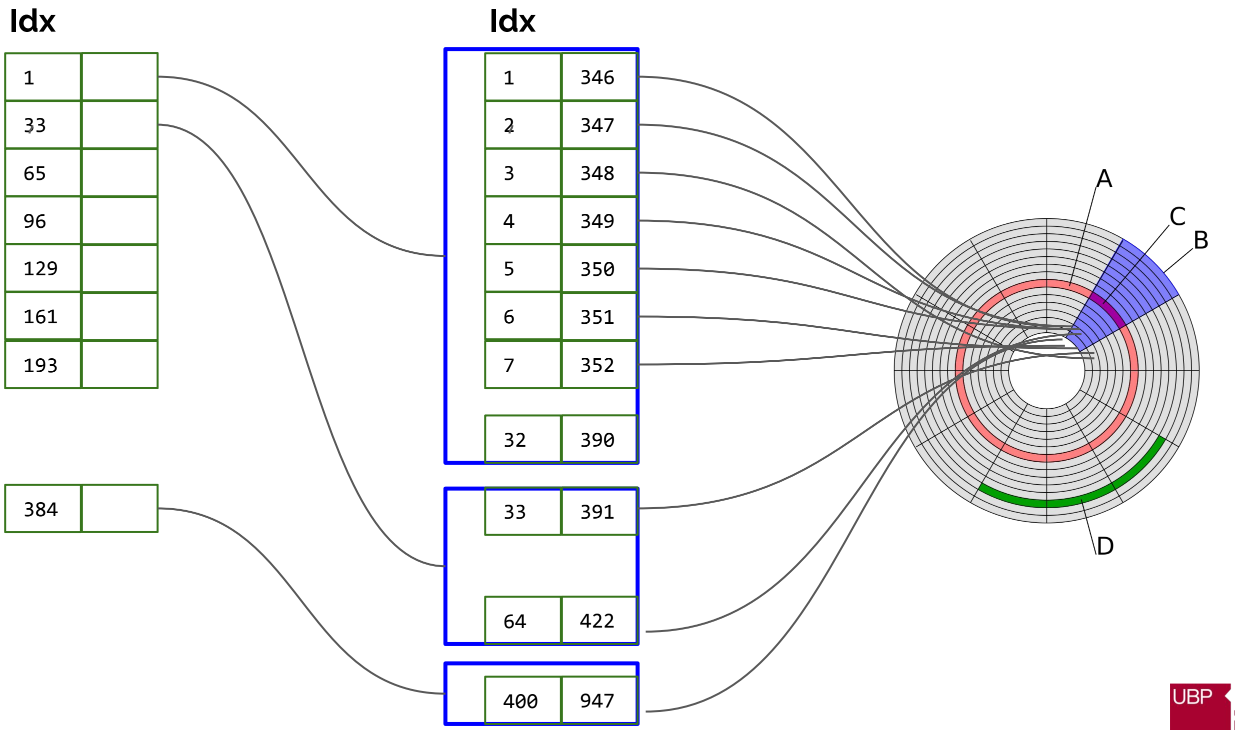
13 + 1 = 14 accesos a disco

¿Podemos mejorar esto?

El índice también se va a almacenar en el disco

índices

El nuevo índice tendrá una entrada por cada **bloque** del segundo índice

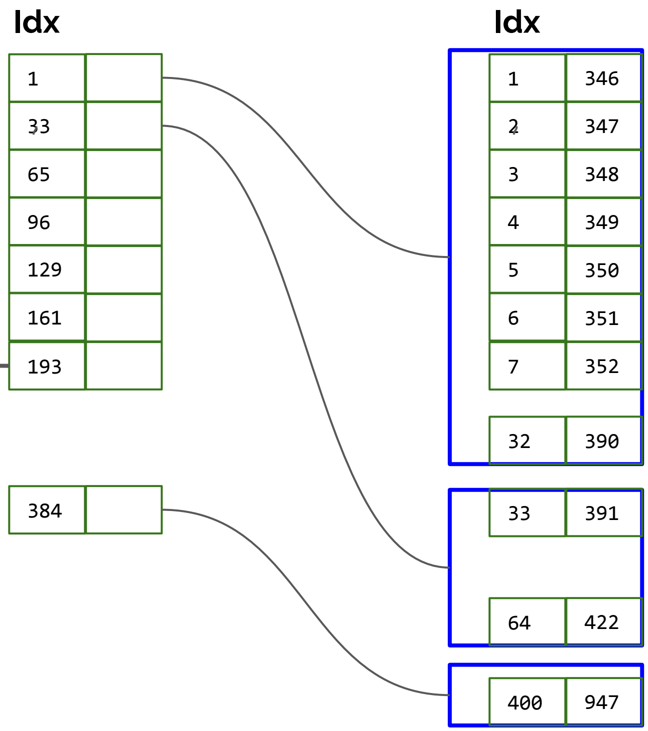


Tiene 400 / 32 ≅ 13 entradas.
Cabe en 1 bloque

índices

```
select * from Alumnos where id = 294
```

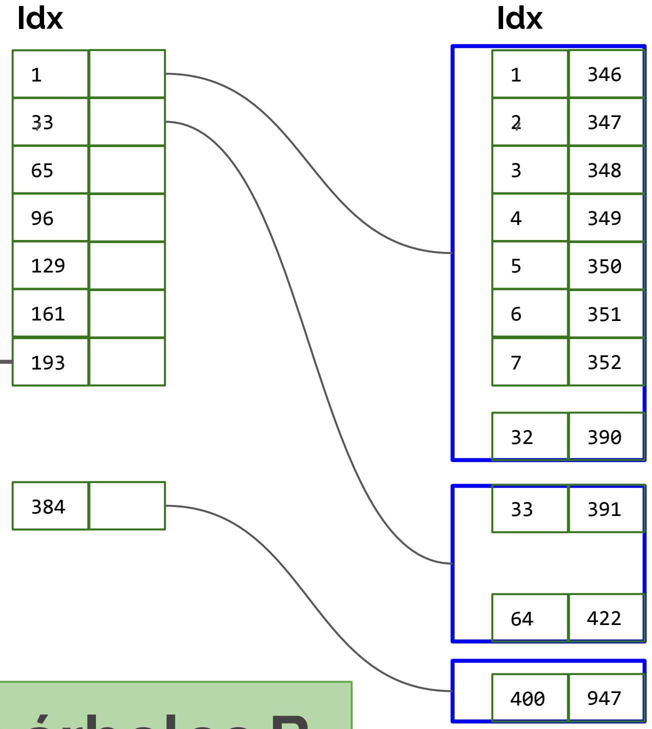
1 acceso para leer el primer índice completo
1 acceso para leer el bloque correspondiente del segundo índice
1 acceso para traer el bloque que contiene el registro
----
3 accesos a disco



índices

```
select * from Alumnos where id = 294
```

1 acceso para leer el primer índice completo
1 acceso para leer el bloque correspondiente del segundo índice
1 acceso para traer el bloque que contiene el registro
----
3 accesos a disco



Esta es la idea que origina los árboles B

árbol B

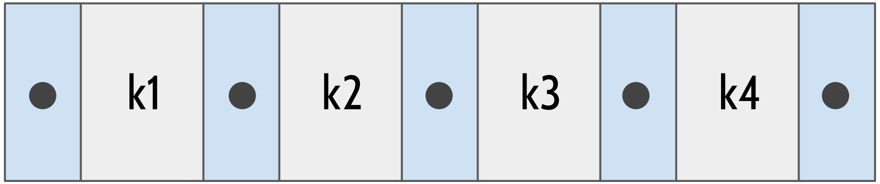
Los árboles B son árboles autobalanceados que **tienen en cuenta el aspecto físico** del almacenamiento mediante la agrupación de varios elementos en cada nodo del árbol.

Eso reduce la cantidad de lecturas en disco que es una operación muy lenta

Teniendo en cuenta que la mayoría de las operaciones tiene costo O(h), tratamos de mantener la altura (h) pequeña colocando la mayor cantidad posible de claves en cada nodo

Si la estructura se almacena en disco es usual dimensionar los nodos con el tamaño del los bloques de disco

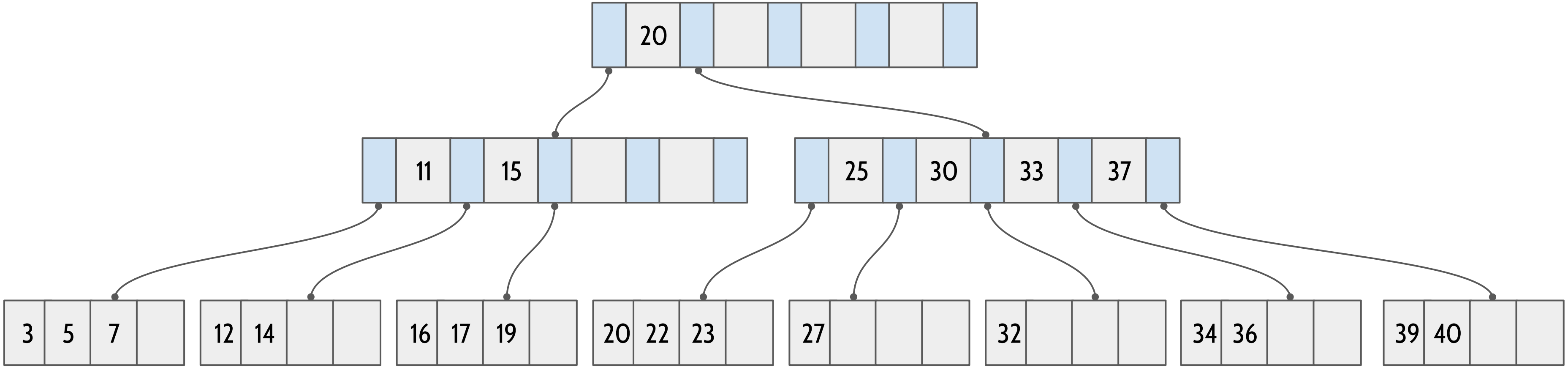
árbol B



$$k_1 < k_2 < k_3 < k_4$$

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

árbol B



árbol B: propiedades

Todas las hojas están en el mismo nivel

Un árbol B se define en términos de grado **m** que es la máxima cantidad de hijos que puede tener cada nodo

Cada nodo tiene como máximo **m** hijos y **m-1** claves

Cada nodo excepto el raíz tiene como mínimo **m/2** claves. La raíz debe contener al menos una clave

Las claves en un nodo están ordenadas en orden ascendente

árbol B: operaciones

- Búsqueda
- Recorrido
- Inserción
- Eliminación

árbol B: búsqueda

Llamamos k a la clave que buscamos

Comenzamos desde la raíz hacia abajo

Por cada nodo que visitamos, si ese nodo tiene la clave retornamos el nodo

Si no, visitamos el nodo hijo que está antes que la primera clave que es mayor a la que estamos buscando

Si llegamos a una hoja y no está la clave k devolvemos NULL

árbol B: recorrido

Es similar al recorrido in-order de un árbol binario

Comenzamos por el hijo de más a la izquierda, repetimos el proceso para los demás hijos y claves y finalmente visitamos recursivamente el hijo de la derecha

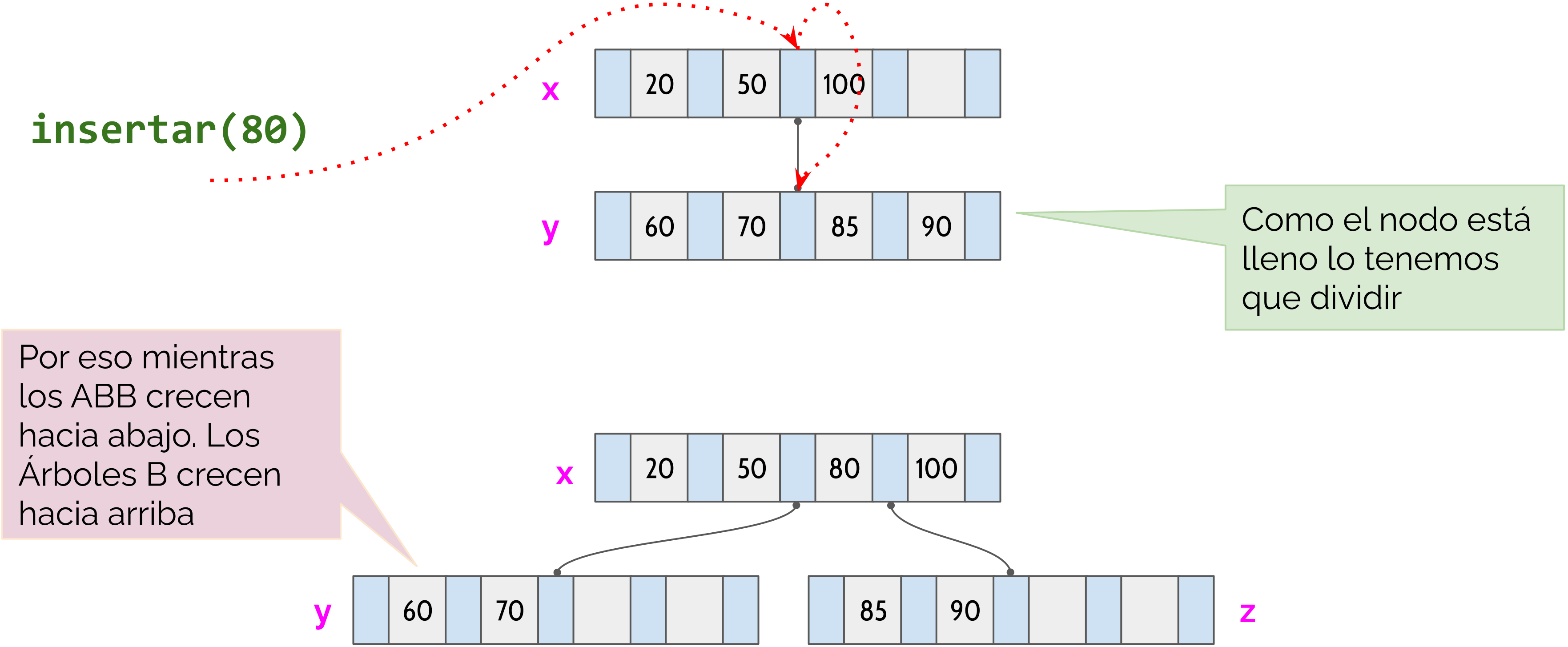
árbol B: inserción

Una nueva clave se inserta siempre en un nodo hoja

Como en un ABB comenzamos desde la raíz y vamos bajando hasta que encontramos una hoja

Cuando llegamos a la hoja insertamos la clave en la hoja si esta tiene espacio, de lo contrario debemos dividir este nodo

árbol B: inserción - dividir nodo



árbol B: inserción (algoritmo proactivo)

Inicializar **x** como root

Mientras **x** no sea hoja hacer:

- Encontrar el siguiente hijo a ser visitado y asignarlo a **y**
- Si y no está lleno hacer que **x** apunte a **y**
- Si **y** está lleno: dividirlo y hacer que **x** apunte a una de las partes de **y**
  - Si **k** es menor que la clave del medio de **y** entonces **x** apunta a la primera mitad de **y**
  - sino a la segunda parte. Cuando dividimos, movemos una clave de **y** a su padre **x**

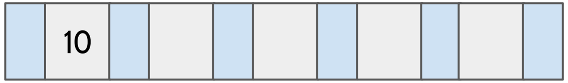
El loop termina cuando encontramos una hoja, la hoja contendrá espacio, ya que vamos dividiendo los nodos llenos en el camino. De modo que simplemente insertamos la clave **k**

UBP UNIVERSIDAD BLAS PASCAL

árbol B: inserción

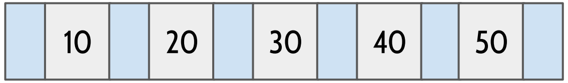
m = 6

```
insertar(10)
insertar(20)
insertar(30)
insertar(40)
insertar(50)
insertar(60)
insertar(70)
insertar(80)
insertar(90)
```



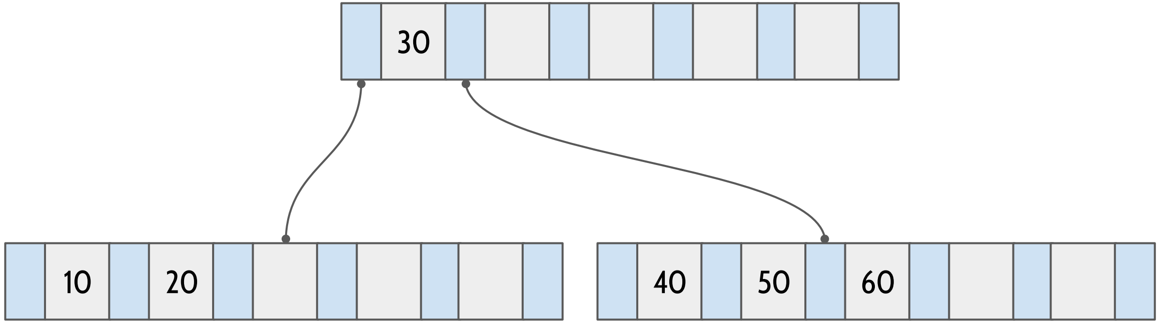
árbol B: inserción

```
m = 6

insertar(10)
insertar(20)
insertar(30)
insertar(40)
insertar(50)
insertar(60)
insertar(70)
insertar(80)
insertar(90)
```



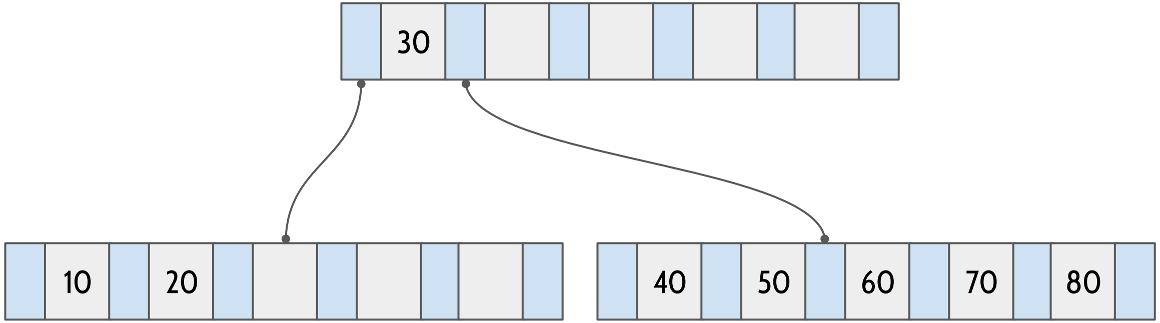
árbol B: inserción

```
m = 6

insertar(10)
insertar(20)
insertar(30)
insertar(40)
insertar(50)
insertar(60)
insertar(70)
insertar(80)
insertar(90)
```



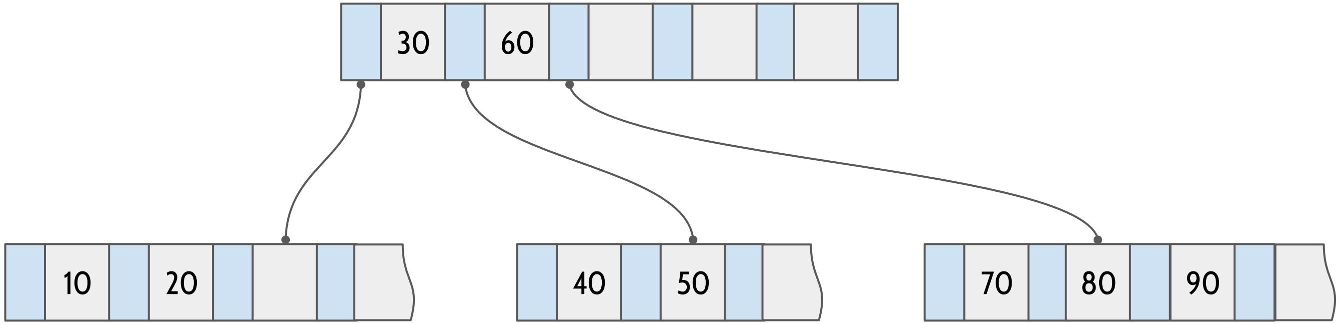
árbol B: inserción

```
m = 6

insertar(10)
insertar(20)
insertar(30)
insertar(40)
insertar(50)
insertar(60)
insertar(70)
insertar(80)
insertar(90)
```



árbol B: inserción

```
m = 6

insertar(10)
insertar(20)
insertar(30)
insertar(40)
insertar(50)
insertar(60)
insertar(70)
insertar(80)
insertar(90)
```



Ejercicio 2.1.

Analizá la implementación provista de Arbol B e implementá las operaciones:

Recorrer()

Buscar(int)

árbol B: eliminación

1. Si la clave k está en el nodo x y el nodo x es una hoja. Eliminamos k de x.

2. Si la clave k está en el nodo x y el nodo x es un nodo interno:

2.a) Si el hijo y que precede a k en el nodo x tiene al menos t claves, entonces encontrar el predecesor k0 de k en el subarbol con raiz en y. Eliminar recursivamente k0 y remplazar k por k0

2.b) Si y tiene menos de t claves, entonces examinar el hijo z que sigue a k en el nodo x. Si z tiene al menos t claves, entonces encontrar el sucesor k0 de k en el subarbol con raiz en z. Recursivamente eliminar k0 y remplazar k por k0 en x

árbol B: eliminación

2.c) Si los dos y y z tienen t-1 claves, combinar k y los de z en y, de modo que x pierde k y el puntero a z e y ahora contiene 2t-1 claves. Después liberar z y recursivamente eliminar k de y.

3. Si la clave k no está presente en el nodo interno x, determinar la raíz x.c(i) del subárbol correspondiente que debe contener k, si k está efectivamente en el árbol. Si x.c(i) tiene solo t-1 claves ejecutar 3a o 3b para garantizar que tenemos un nodo que contiene por lo menos t claves. Finalizar aplicando este proceso recursivamente en x

3.a) Si x.c(i) tiene solo t-1 claves pero tiene un hermano inmediato con al menos t claves, le damos x.c(i) una clave extra pasando una clave desde x hacia abajo hacia x.c(i), moviendo una clave desde el hermano inmediato izquierdo o derecho de x.c(i) hacia arriba a x y moviendo el puntero al hijo correspondiente del hermano a x.c(i)

árbol B: eliminación

3.a) Si x.c(i) tiene solo t-1 claves pero tiene un hermano inmediato con al menos t claves, le damos x.c(i) una clave extra pasando una clave desde x hacia abajo hacia x.c(i), moviendo una clave desde el hermano inmediato izquierdo o derecho de x.c(i) hacia arriba a x y moviendo el puntero al hijo correspondiente del hermano a x.c(i)

3.b) Si x.c(i) y ambos hermanos inmediatos de x.c(i) tienen t-1 claves, combinar x.c(i) con un hermano, esto incluye mover una clave de x hacia abajo al nuevo nodo combinado, en donde estará al medio.

Ejercicio 2.2.

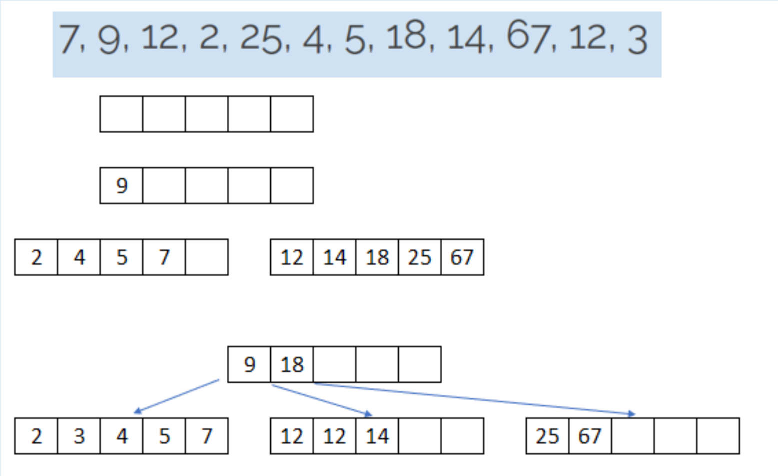
Implementá la operación `eliminar(int clave)` en un árbol B

Ejercicio 2.3.

En un árbol B con m=6 dibujá los pasos intermedios y el árbol B resultante de insertar la siguiente secuencia:

7, 9, 12, 2, 25, 4, 5, 18, 14, 67, 12, 3

(solamente redibujá el árbol cuando este cambie de forma)

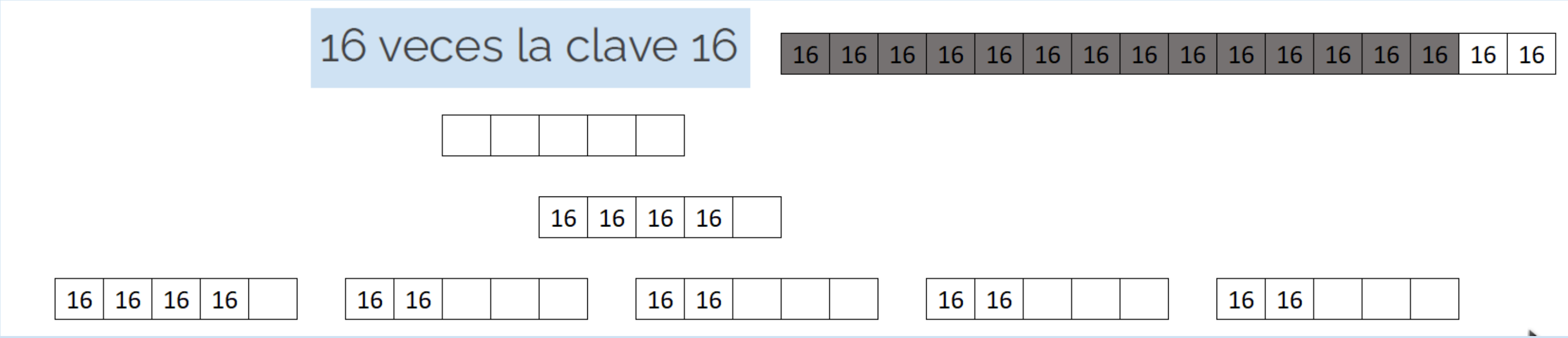


Ejercicio 2.4.

En un árbol B con m=6 dibujá los pasos intermedios y el árbol B resultante de insertar la siguiente secuencia:

16 veces la clave 16

(solamente redibujá el árbol cuando este cambie de forma)



Ejercicio 2.5.

Implementá la operación `buscar(int clave)` en un árbol B