

# **ED-II-Apuntes-y-ejercicios.pdf**



**lauritavr**



**Estructuras de Datos II**



**2º Grado en Ingeniería Informática**



**Escuela Técnica Superior de Ingeniería  
Universidad de Huelva**

saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

X - X - X - X - X - X - X - X - X - X

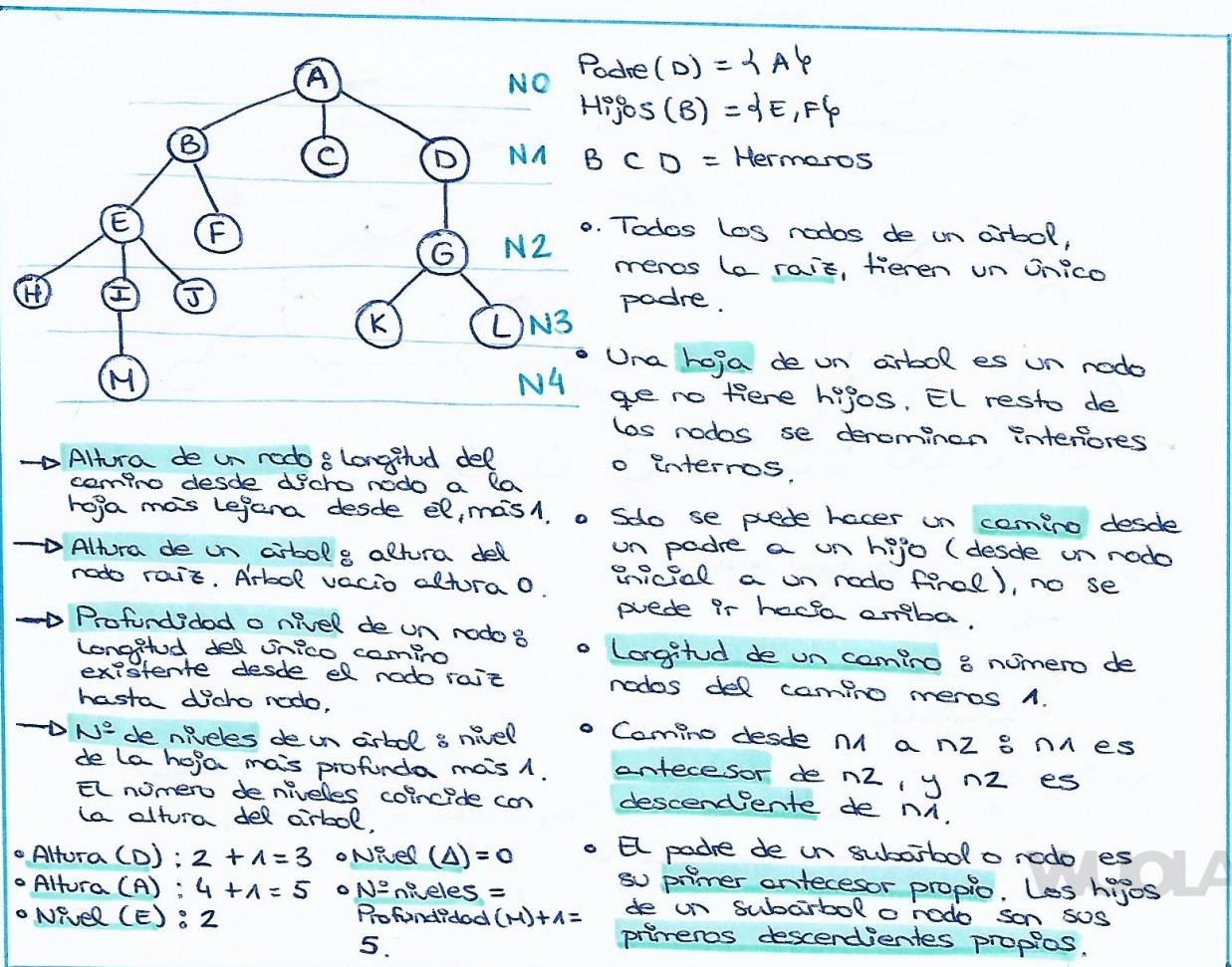
## TEMA 2 : ÁRBOLES

### ÁRBOLES

→ Estructuras que organizan sus elementos, denominados nodos, formando jerarquías. De entre estos nodos existe uno especial denominado raíz, situado en la cúspide de la jerarquía.

### CLASIFICACIÓN

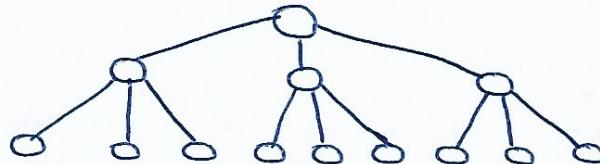
- Ordenado: cuando existe una relación de orden total en el conjunto de los subárboles de un árbol.
- General: si no existe limitación para el número de hijos que puede tener un nodo (no hay n)
- N-ario: existe un número fijo  $n$  que limita el número de hijos de un nodo. La máxima n de hijos por cada nodo.
- Etiquetado: cuando los nodos de un árbol contienen información.



- El grado de un árbol n-ario es el nº máximo de hijos que pueden tener sus subárboles.
- El nº máximo de nodos en el nivel  $i$ -ésimo de un árbol de grado  $n$  es  $n^i$ .

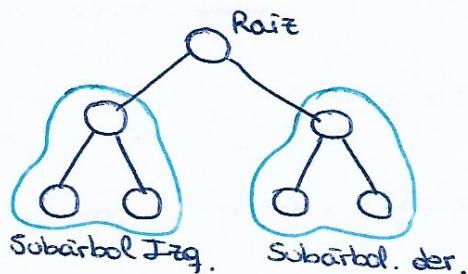
### EJEMPLO

Árbol de grado 3 ( $n=3$ ), en el nivel 2 puede haber como máximo  $n^i$  nodos,  $3^2 = 9$  nodos.



### ÁRBOLES BINARIOS

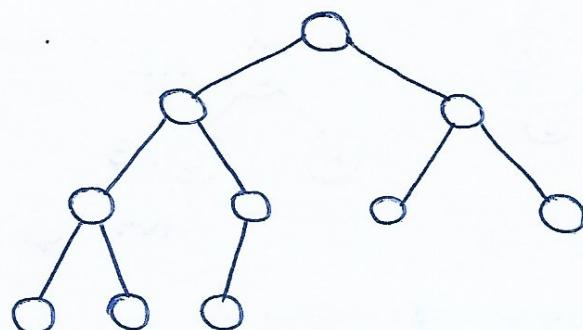
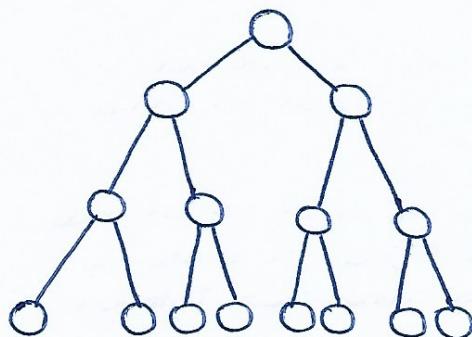
- 2 hijos como mucho
- Puede ser vacío.



### TÍPOS

Completo

Semicompleto



- Recorrido en profundidad

- Preorden
- Postorden
- Inorden

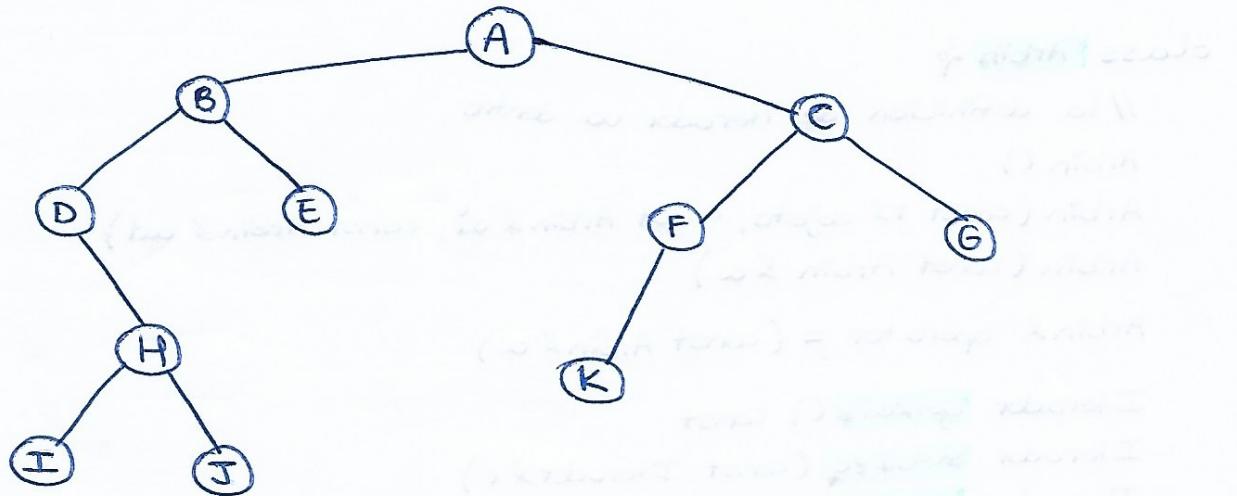
- Recorrido en anchura: los nodos se visitan por niveles, y dentro de cada nivel, de izquierda a derecha, comenzando por el nivel 0.

x  
-  
x  
-  
x  
-  
x  
-  
x  
-  
x  
-  
x  
-  
x  
-  
x  
-  
x  
-  
x  
-  
x  
-  
x

saboteas a tu propia persona?  
cómo?? escríbelo **aquí** y táchalo

**manual de instrucciones:** escribe sin filtros  
y una vez acabes, táchalo (si lo compartes en redes  
mencionándonos, te llevas 10 coins por tu cara bonita)

**DESFÓGATE CON WUOLAH**



→ PREORDEN : A, B, D, H, I, J, E, C, F, K, G

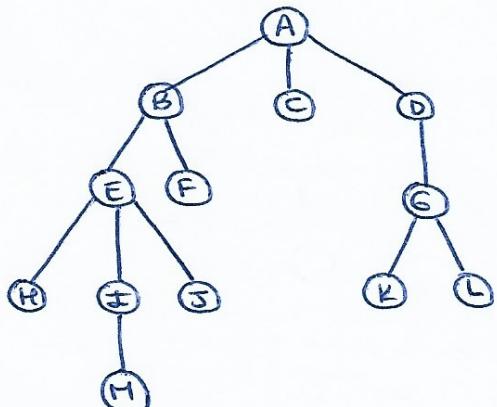
→ POSTORDEN : I, J, H, D, E, B, K, F, G, C, A

→ INORDEN : D, I, H, J, B, E, A, K, F, C, G

→ ANCHURA : A, B, C, D, E, F, G, H, K, I, J

### ARBOL GENERAL

- Cada nodo no tiene máximo, puede tener los hijos que quiera.
- No se permite que sea árbol vacío.
- operación básica "enraizar"
- bosque ordenado (lista) → hijos (subárboles)
- Recorrido en profundidad : (preorden, postorden)
- Recorrido en anchura.



• PREORDEN : A, B, E, H, I, M, J, F, C, D, G, K, L.

• POSTORDEN : H, I, M, J, E, F, B, C, K, L, G, D, A.

• ANCHURA : A, B, C, D, E, F, G, H, I, J, K, L, M.

```

class Arbin {
    // la definición de iterador va dentro.
    Arbin()
    Arbin(const T& objeto, const Arbin& ai, const Arbin& ad)
    Arbin(const Arbin &a)

    Arbin& operator = (const Arbin& a)

    Iterador getRaiz() const
    Iterador subIzq (const Iterador& r)
    Iterador subDer (const Iterador& r)

    bool esVacio() const
    int altura() const
    ~Arbin()
}

```

```
class Iterador { // dentro de Arbin
```

```

    const T& observar()

    bool arbolVacio() const
    bool operator != (const Iterador& i) const
    bool operator == (const Iterador &i) const
    int altura() const

```



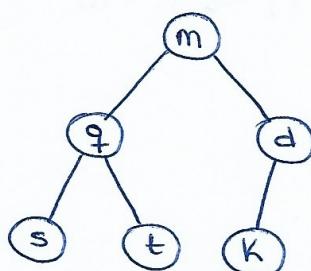
saboteas a tu propia persona? cómo??  
escríbelo **aquí** y  
táchalo

**manual de instrucciones:** escribe sin filtros  
y una vez acabes, táchalo (si lo compartes en redes  
mentionándonos, te llevas 10 coins por tu cara bonita)

## - BOLETÍN TEMA 2 -

### EJERCICIO 6

Frontera de un árbol binario es secuencia formada por los elementos almacenados en las hojas de un árbol binario, tomados de izquierda a derecha.



frontera: s t K

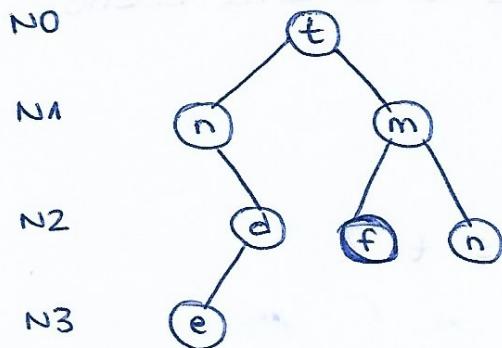
```

template <typename T>
void frontera (const Arbin <T> & a) {
    frontera (a, a.getRaiz())
}

template <typename T>
void frontera (const Arbin <T> & a, const typename Arbin <T> :: Iterador& r) {
    si ( no r.arbolVacio() ) entonces
        si ( a.SubIzq().arbolVacio() y a.SubDerecha().arbolVacio() )
            escribir (r.obtener())
        sino
            frontera ( a, a.SubIzq (r) )
            frontera ( a, a.SubDer (r) )
    fin
}
  
```

## EJERCICIO VER-NIVELES

Diseñar una función que visualice los nodos que están en el nivel  $n$  de un árbol binario.



ver-nivel ( $a, 1 \rightarrow n - m$ )



```

template < typename T >
void ver-nivel (const Arbin < T > & a, int n)
  inicio
    ver-nivel (a, a.getRaiz(), n)
  fin
  
```

```

template < typename T >
void ver-nivel (const Arbin < T > & a, const typename Arbin < T >::Iterador & r, int n)
  inicio
    si ( no r.arbolVacio() y n >= 0 )
      si ( n == 0 )
        r.observer();
      sino
        ver-nivel (a, a.subIzq(r), n - 1)
        ver-nivel (a, a.subDer(r), n - 1)
      fsi
    fsi
  fin
  
```

## EJERCICIO 5

Función que devuelva verdadero si el árbol es completo y falso en otro caso. Suponemos que el árbol vacío es completo.

```
template <typename T>
bool completo (const Arbin<T> &a) {
    return completo (a, a.getRaiz ());
}
```

```
template <typename T>
bool completo (const Arbin<T> &a, const typename Arbin<T> :: Iterator & r) {
    if (r.arbolVacio ())
        return true;
    else
        if (a.subIzq().altura == a.subDer().altura)
            return (completo(a, a.subIzq(r)) && completo(a, a.subDer(r)))
        else
            return false;
}
```

## EJERCICIO 8

Diseñar una función booleana que indique si dos árboles binarios A y B tienen la misma forma en cuanto a la distribución de sus nodos.

```
template <typename T>
bool igual (const Arbin<T>& a, const Arbin<T>& b)
return igual (a, a.getRaiz(), b, b.getRaiz())
}

template <typename T>
bool igual (const Arbin<T>& a, const typename Arbin<T>::Iterador &ra, const Arbin<T>& a, const typename Arbin<T>::Iterador &rb)
if (ra.arbolVacio() && rb.arbolVacio())
    return true;
else
    if(a.subIzq(ra).altura() == b.subIzq(rb).altura() &&
       a.subDer(ra).altura() == b.subDer(ra).altura())
        return (igual(a, a.subIzq(ra), b, b.subIzq(rb)) &&
                igual(a, a.subDer(ra), b, b.subDer(rb)))
    else
        return false
}
```

saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

### EJERCICIO 2

función que calcule la suma de los elementos de un árbol (entero).

```
template <typename>
int suma (const Arbin<T> & a) {
    return suma (a, a.getRaiz());
}

template <typename>
int suma (const Arbin<T> & a, const typename Arbin<T>::Iterador & r) {
    if (r.arbolVacio())
        return (r.obtener());
    else
        return ( suma (a, a.subDer(r)) + suma(a, a.subIzq(r)) +
            r.obtener());
}
```

### EJERCICIO 3

copia simétrica

```
template <typename>
void copia (const Arbin<T> & a) {
    Arbin<T> b;
    b.insertar(a.obtener());
    copia(a, a.getRaiz(), b);
    return b;
}

template <typename>
void copia (const Arbin<T> & a, const typename Arbin<T>::Iterador & r,
            Arbin<T> & b) {
```

# TEMA 3: ABB

ABB

→ tipo de elementos que almacena poseen una relación de orden total.

- todos los valores del subárbol izquierdo < valor raíz
- todos los valores del subárbol derecho > valor raíz

PROPIEDAD

→ el recorrido en inorden de un ABB obtiene los elementos ordenados de menor a mayor según la relación de orden existente entre los nodos.

OPERACIONES BÁSICAS

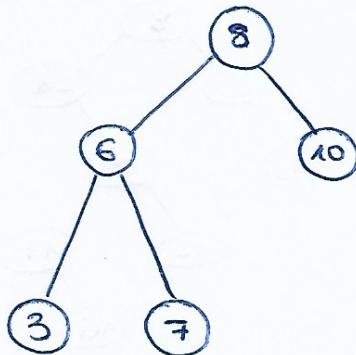
→ BÚSQUEDA, INSERCIÓN, BORRADO

se basan en el esquema de búsqueda de un elemento en el árbol.

Si A es un ABB y e el elemento a buscar:

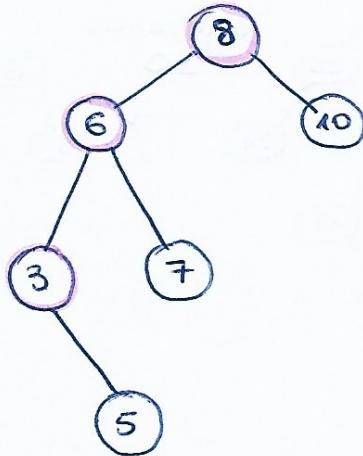
- ① Si A es un árbol vacío, e no está en el árbol.
- ② En caso contrario, se compara e con la raíz de A y se pueden dar 3 casos:
  1.  $e = \text{raíz}(A)$  → elemento encontrado
  2.  $e < \text{raíz}(A)$  → se repite el proceso de búsqueda dentro del subárbol izquierdo de A.
  3.  $e > \text{raíz}(A)$  → se repite el proceso de búsqueda dentro del subárbol derecho de A

INSERCIÓN



Insertar los  
el 5

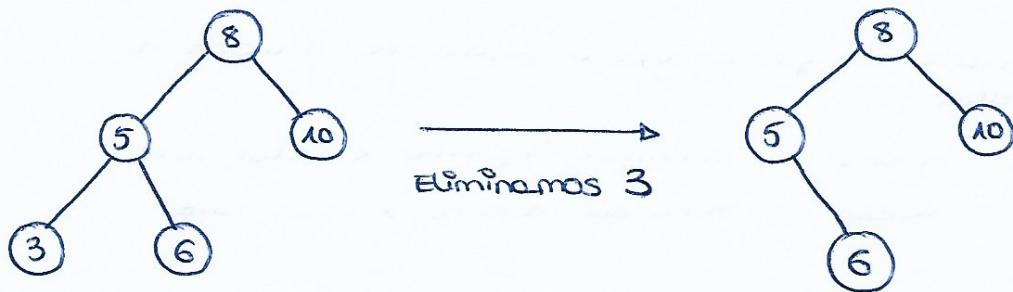
3 comparaciones  
↓  
(3, 6, 3)



## BORRADO

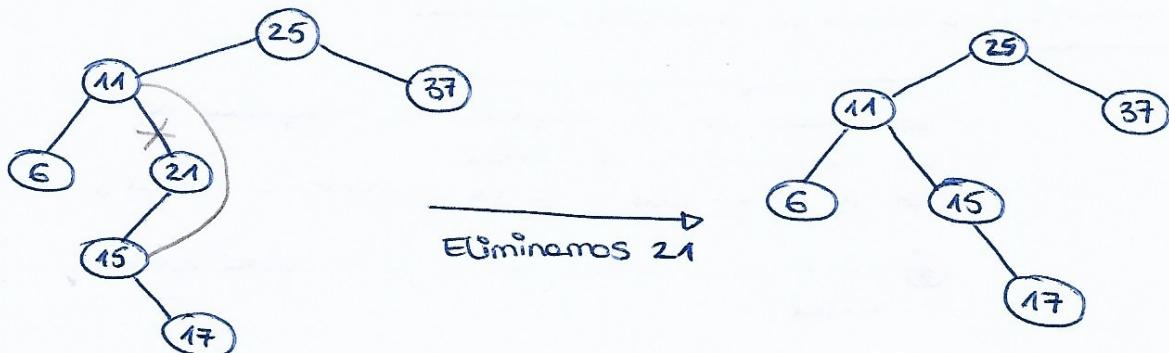
### 1 ELIMINAR UN NODO SIN HIJOS (HOJA)

- Simplemente se elimina el nodo



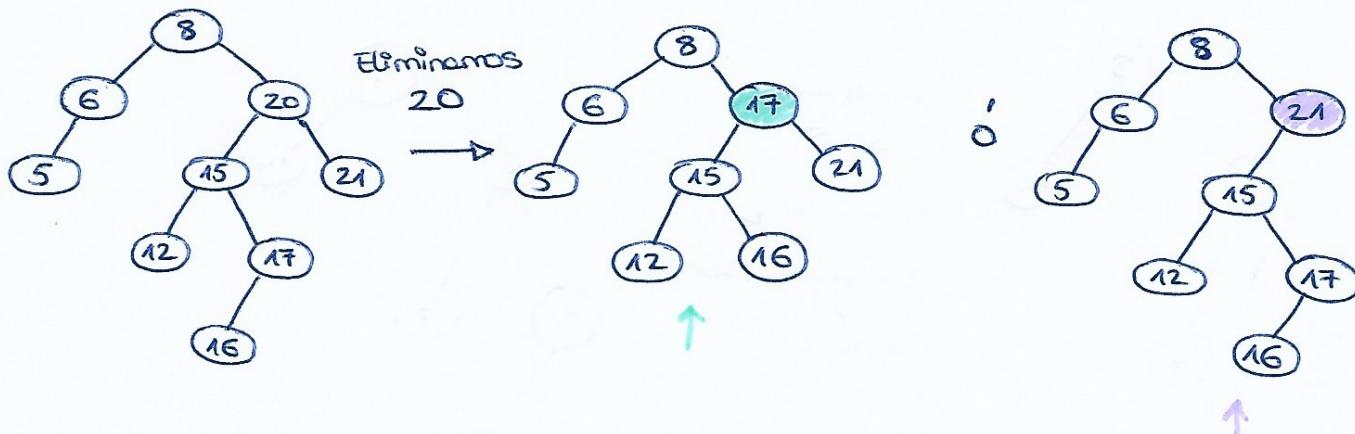
### 2 ELIMINAR UN NODO CON UN SOLO HIJO

- El padre del nodo que queremos eliminar pasa a apuntar al hijo del nodo que está siendo eliminado.



### 3 ELIMINAR UN NODO CON DOS HIJOS

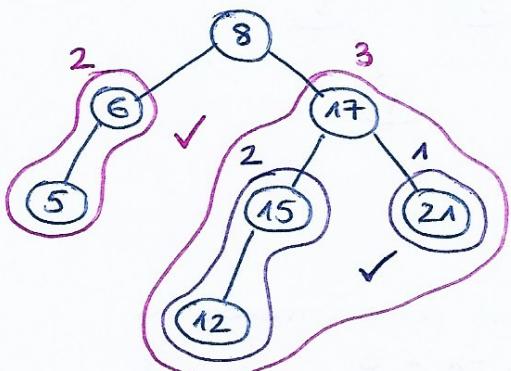
1. Se reemplaza el nodo que queremos eliminar con el elemento máximo de su subárbol izquierdo o el elemento mínimo de su subárbol derecho.
2. Se elimina el nodo máximo o mínimo (según el caso)
3. Dicho nodo será una hoja o tendrá un solo hijo, por lo que la operación se resolverá fácilmente mediante uno de los 2 primeros casos.



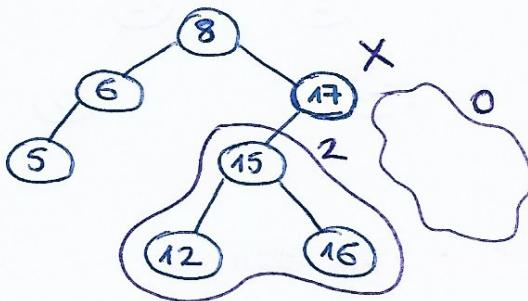
AVL

→ ÁRBOLES DE BÚSQUEDA EQUILIBRADOS → la altura del árbol es siempre la menor posible

Un árbol está equilibrado si la diferencia entre las alturas de sus subárboles siempre es menor o igual que 1.

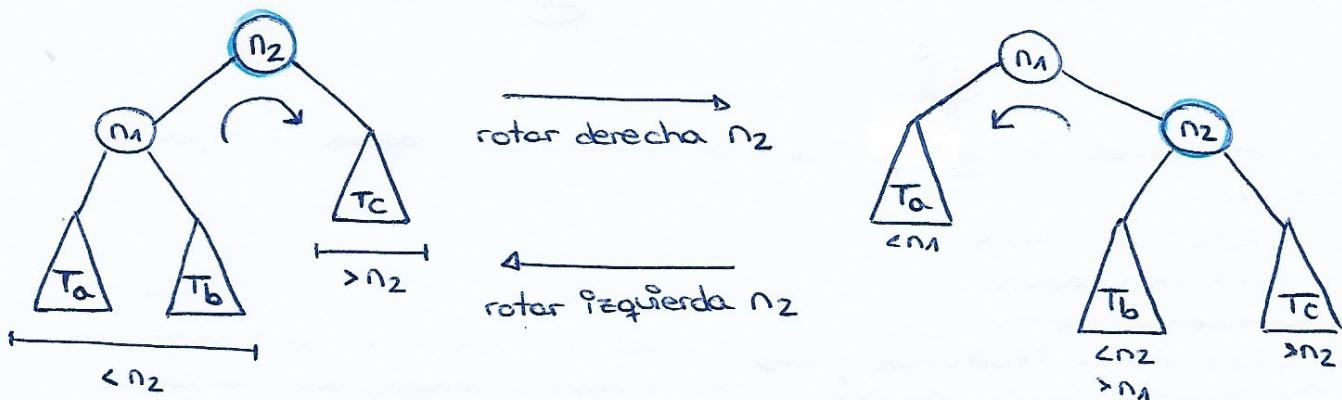


ÁRBOL EQUILIBRADO

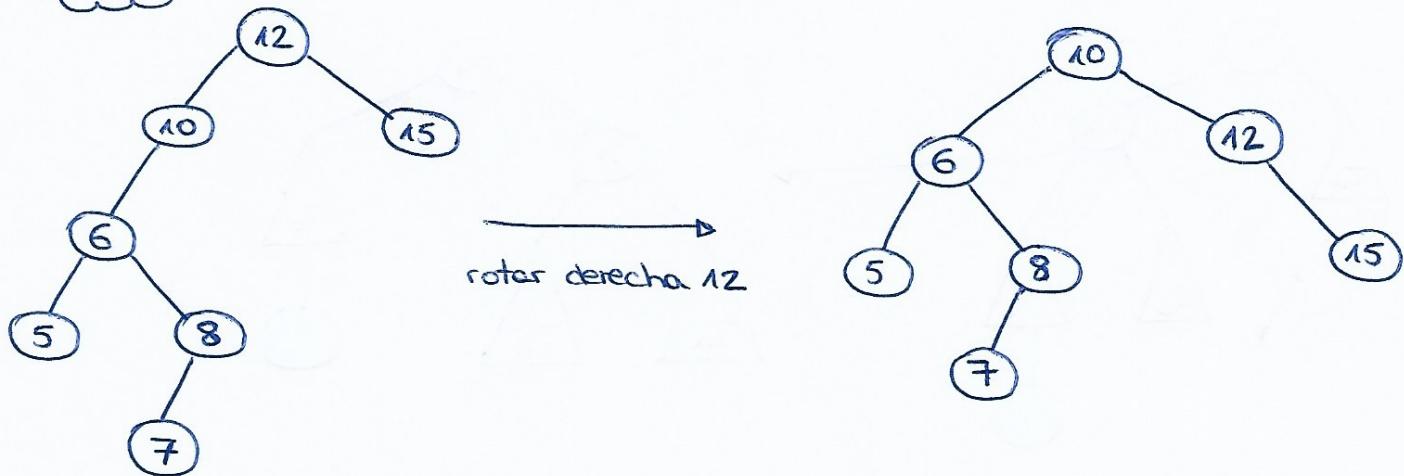


ÁRBOL NO EQUILIBRADO

- Cuando se realiza una operación de inserción o borrado, se puede perder la propiedad de equilibrio. Para mantener dicha propiedad, se realiza una rotación.



EJ Ø



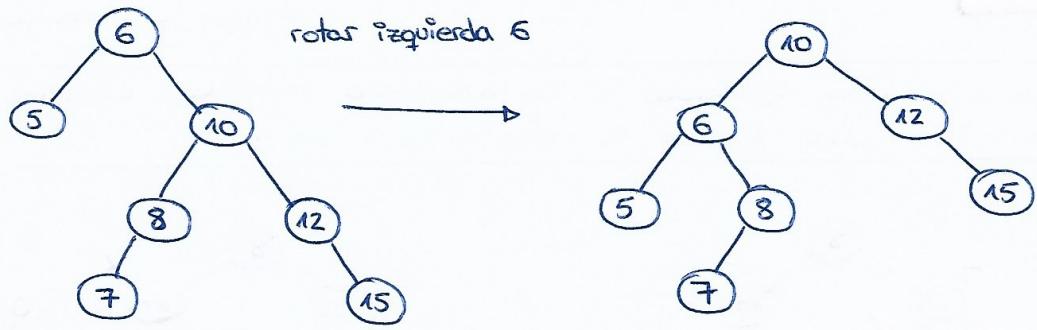


saboteas a tu propia persona? cómo??  
escríbelo **aquí** y táchalo

**manual de instrucciones:** escribe sin filtros

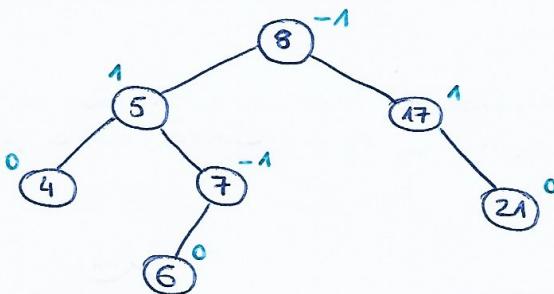
y una vez acabes, táchalo (si lo comparten en redes mencionándonos, te llevas 10 coins por tu cara bonita)

EJ :



#### FACTOR DE EQUILIBRIO ...

... de un nodo es la altura de su subárbol derecho menos la altura de su subárbol izquierdo. VALORES : -1, 0, 1

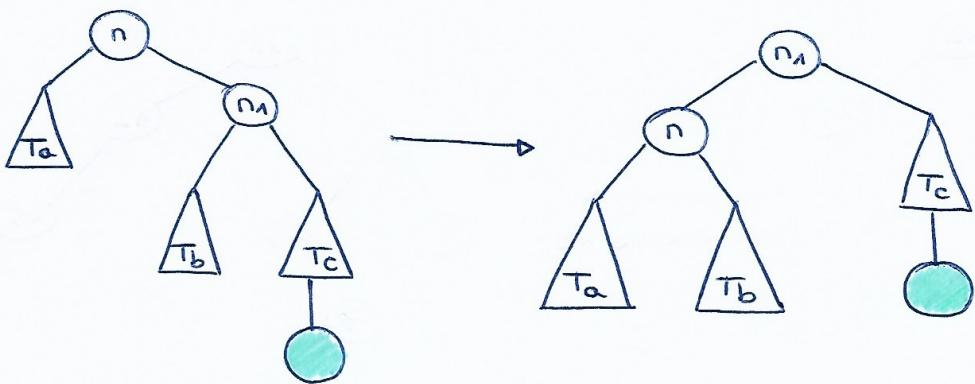


Las situaciones de desequilibrio se solucionan mediante 2 tipos de rotaciones:

- rotación simple
- rotación doble

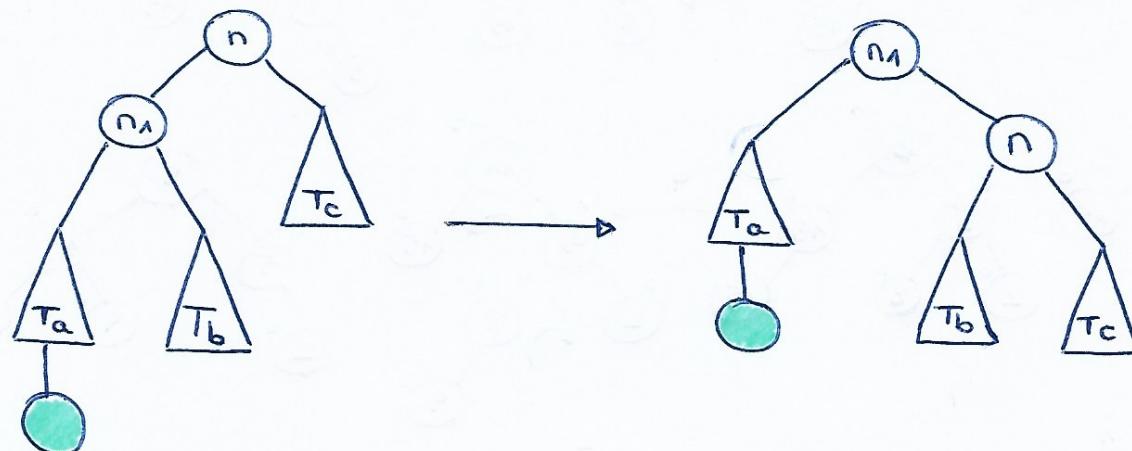
#### ROTACIÓN SIMPLE IZQUIERDA

→ cuando el nuevo nodo se inserta en el subárbol derecho del subárbol derecho de n.



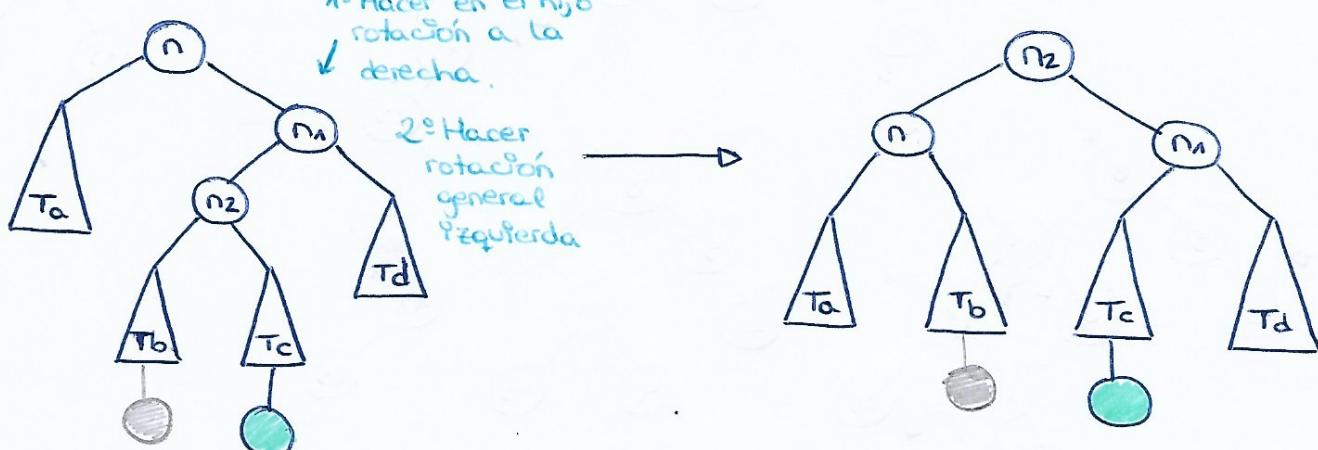
### ROTACIÓN SIMPLE DERECHA

→ cuando el nuevo nodo se inserta en el subárbol izquierdo del subárbol izquierdo de  $n$ .



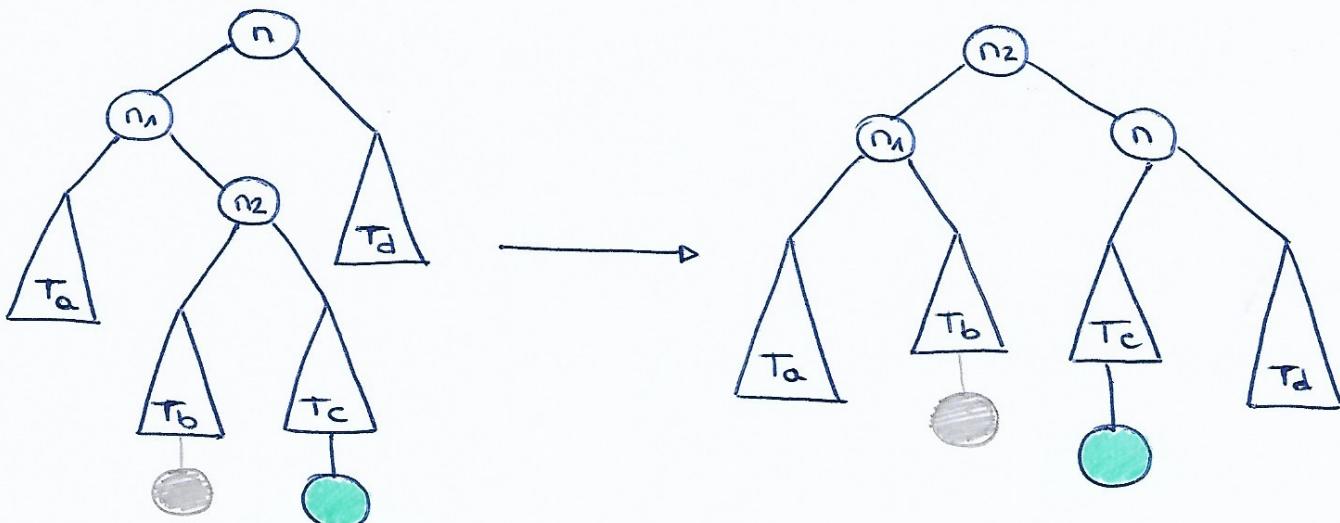
### ROTACIÓN DOBLE DERECHA - IZQUIERDA

→ cuando el nuevo nodo se inserta en el subárbol izquierdo del subárbol derecho de  $n$ .



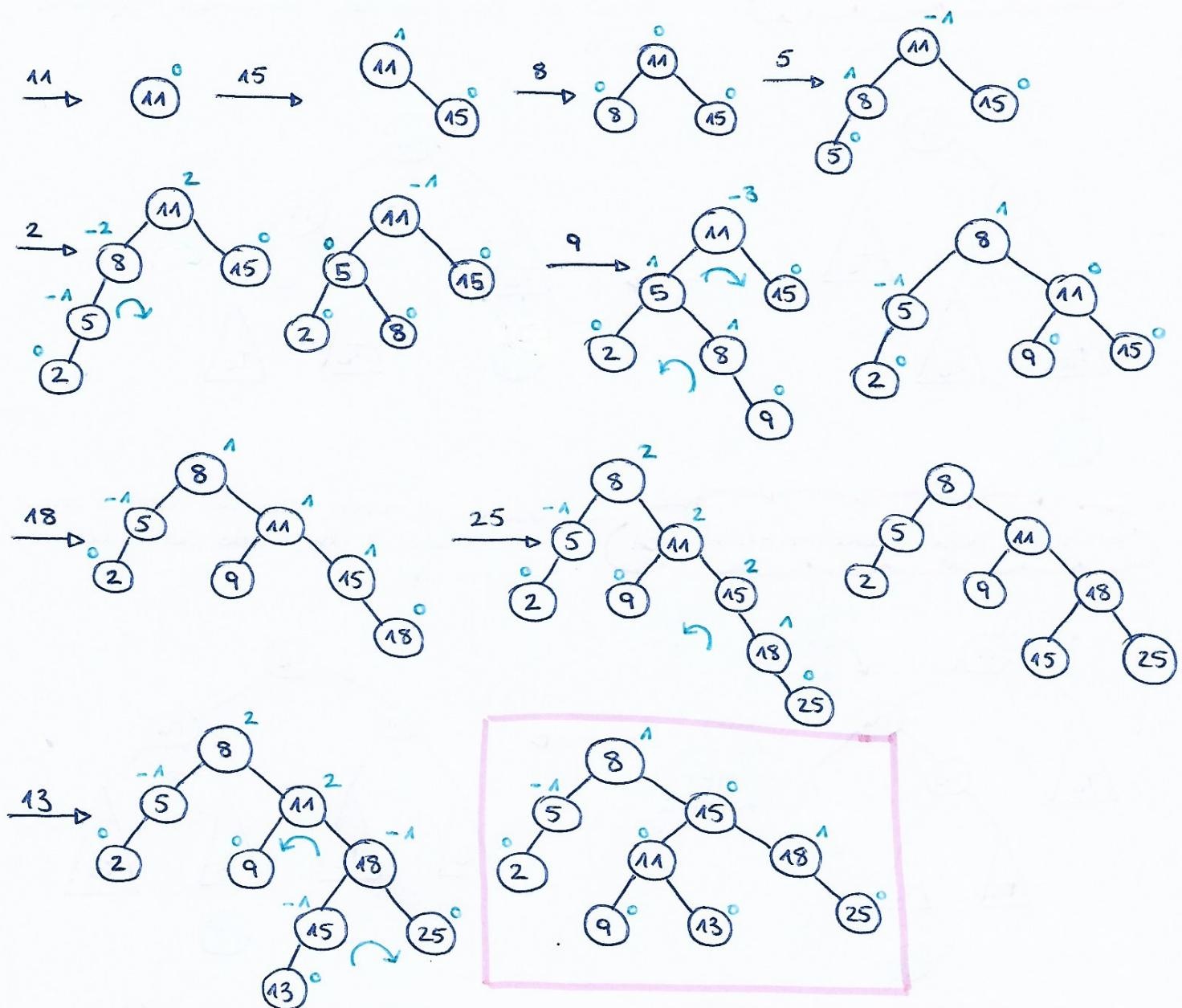
### ROTACIÓN DOBLE IZQUIERDA - DERECHA

→ cuando el nuevo nodo se inserta en el subárbol derecho del subárbol izquierdo de  $n$ .



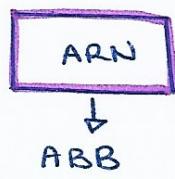
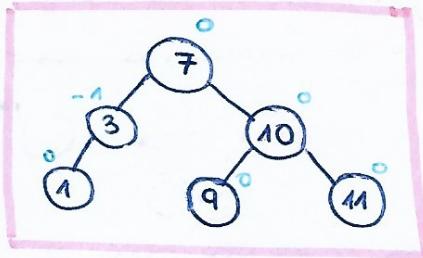
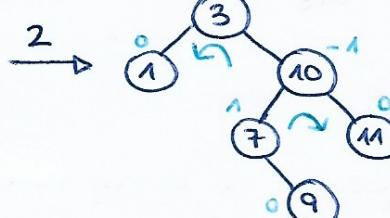
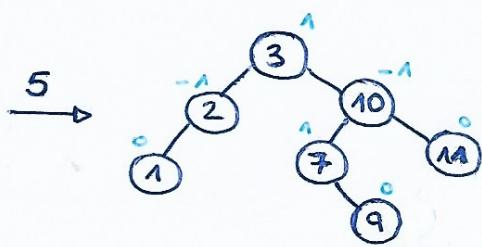
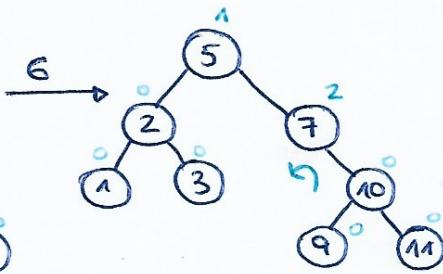
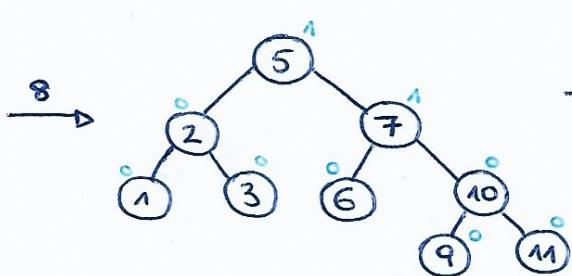
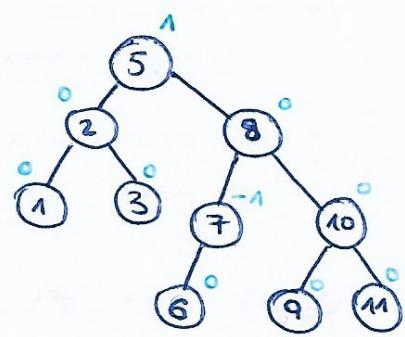
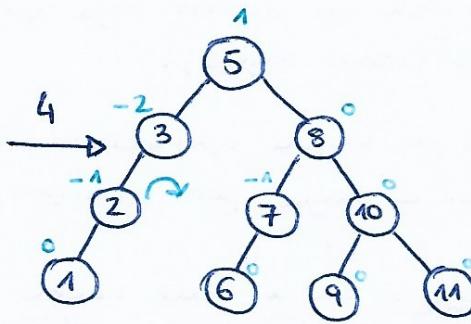
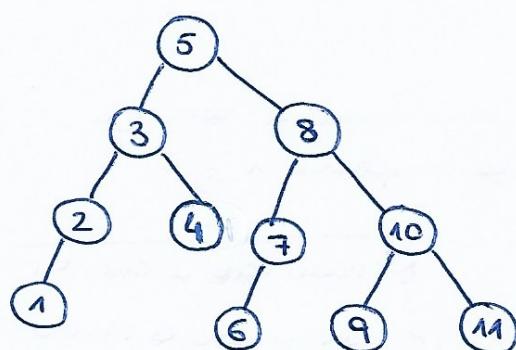
## EJEMPLO INSERCIÓN AVL

→ 11 - 15 - 8 - 5 - 2 - 9 - 18 - 25 - 13



EJEMPLO BORRADO AVL

$\rightarrow 4 - 8 - 6 - 5 - 2$



→ Cada nodo puede ser rojo o negro, equilibrado.

- ①. La raíz es negra.
- ②. Todo nodo es rojo o negro.
- ③. Todo nodo externo (enlaces a nulo) es negro.
- ④. Condición roja: si un nodo es rojo, sus hijos son negros. (al revés no).
- ⑤. Condición negra: Todo camino desde un nodo hasta cada uno de los nodos externos descendientes de él, contiene el mismo número de nodos negros.

- La propiedad 3 y 5 permiten asegurar que el número de nodos rojos en un camino no es mayor que el número de nodos negros.
- La longitud del camino más largo es como mucho el doble del camino más corto:

saboteas a tu propia persona? cómo??  
escríbelo **aquí** y  
táchalo

**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

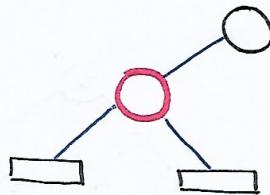


### INserción ARN

- El nuevo nodo consiste en un nodo rojo que contiene el nuevo dato y dos hijos externos (nodos negros).
- Si como resultado de alguna operación, la raíz pasa a ser roja, se puede cambiar a negra para cumplir la propiedad 1.

#### 1 EL PADRE DEL NODO N ES DE COLOR NEGRO

- No se necesita reajustar el árbol



N: nuevo nodo a insertar

P: Padre U: Tío G: Abuelo

#### 2 EL NODO N NO TIENE PADRE Y PASA A SER LA RAÍZ DEL ÁRBOL

- Se cambia de color la raíz

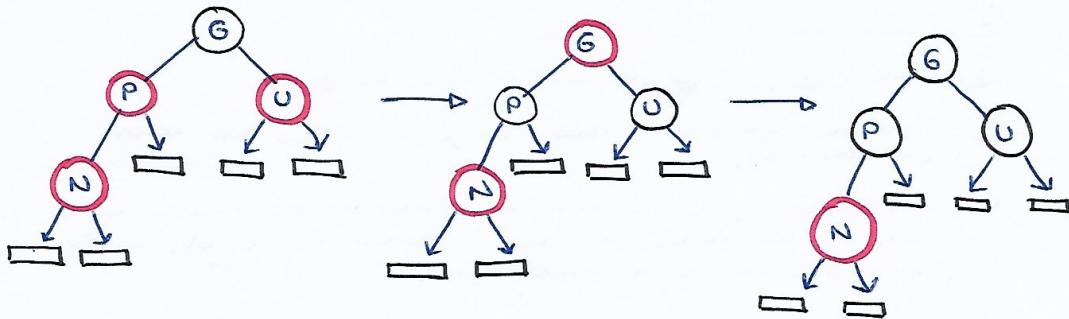


#### 3 EL PADRE DEL NODO N ES ROJO Y NO ES LA RAÍZ

(Los casos simétricos se abordan de forma análoga)

##### 3.1 EL TÍO DE N ES ROJO

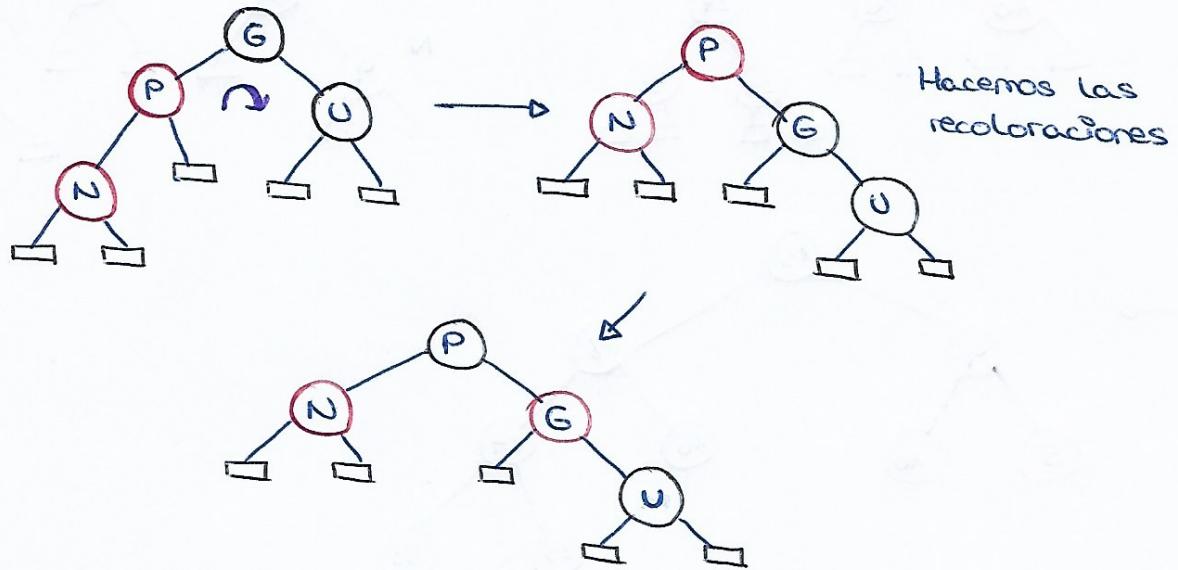
- Se cambia los colores del padre, tío y abuelo de N.



### 3.2 EL TÍO DE N ES NEGRO

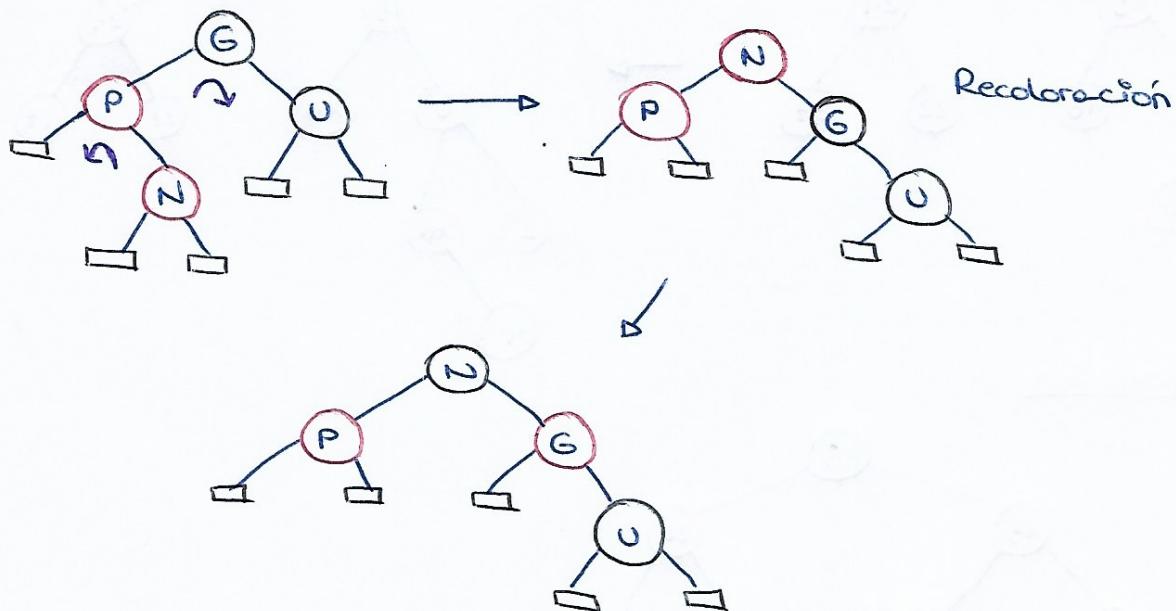
#### 3.2.A Y ADEMÁS N ES HIJO IZQUIERDO

Se resuelve con una rotación simple a la derecha sobre el abuelo de N, y cambiando de color el padre y al abuelo.



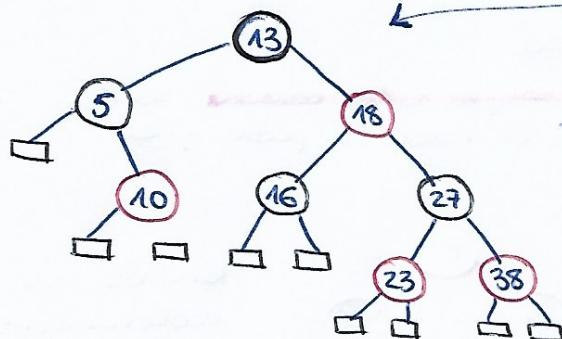
#### 3.2.B Y ADEMÁS N ES HIJO DERECHO

Se resuelve con una rotación doble izquierda-derecha (P-G), seguida de la recoloración, y cambio color a N y al abuelo



### EJEMPLO INSERCIÓN ARN

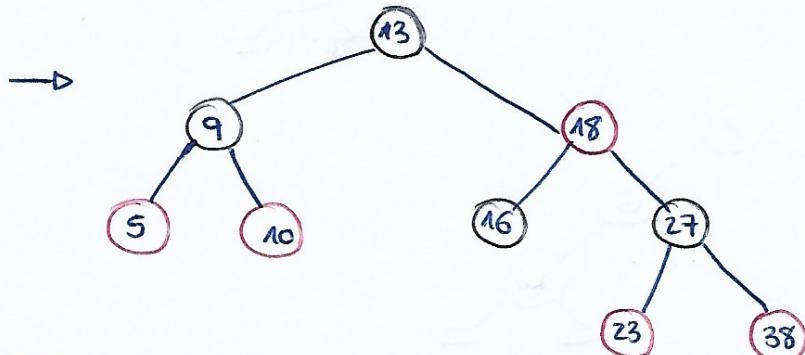
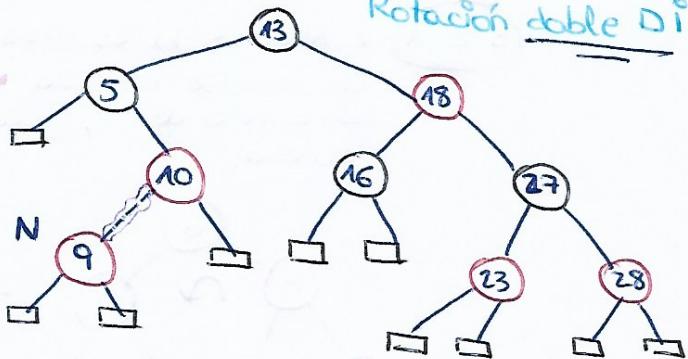
→ Insertar 9



Caso simétrico

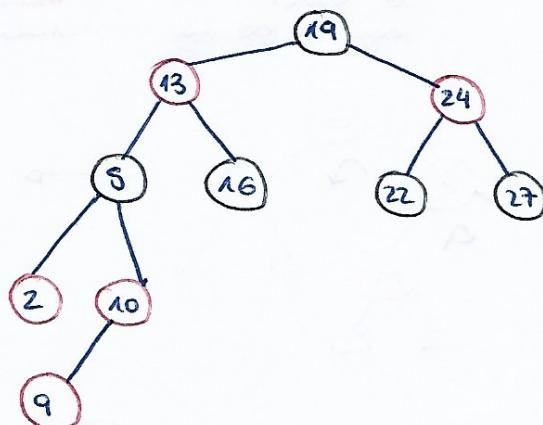
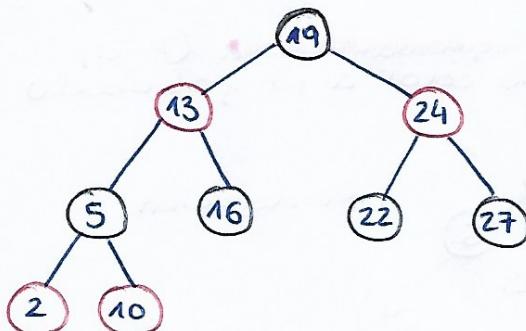
3.2.B

Rotación doble D<sub>i</sub>

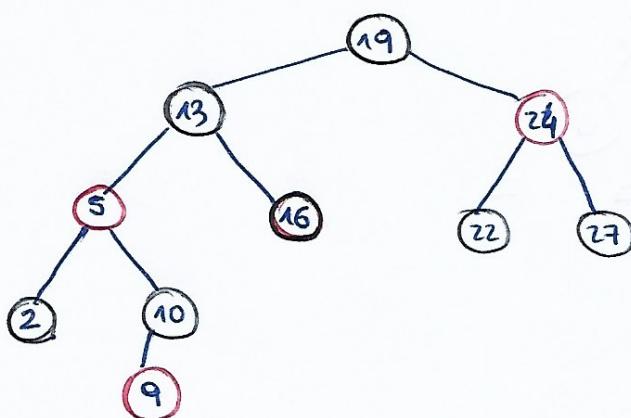


### EJEMPLO INSERCIÓN ARN

→ Insertar 9



Recoloración

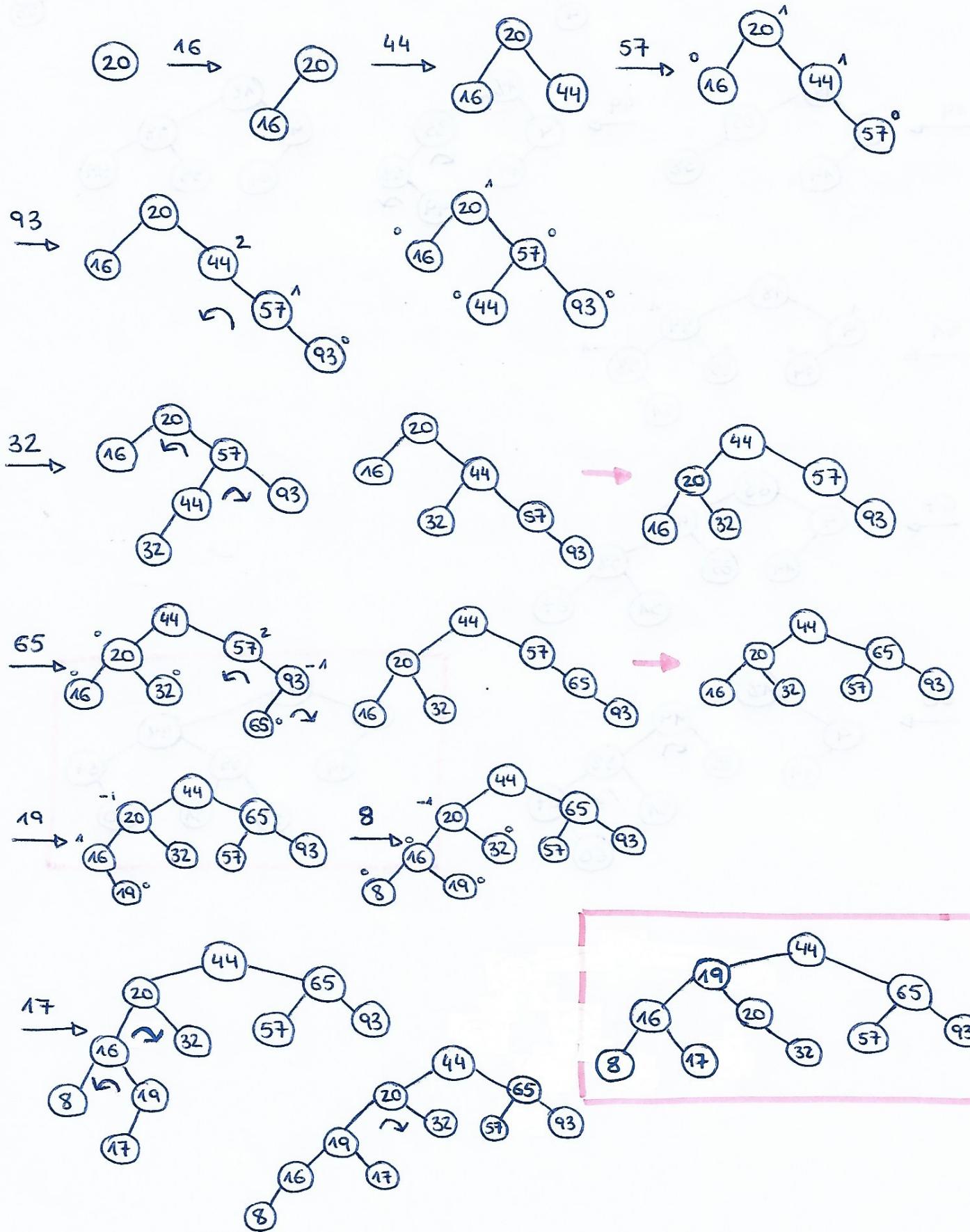


# - BOLETÍN TEMA 3 -

PARTE 2 AVL

EJERCICIO 6

20 - 16 - 44 - 57 - 93 - 32 - 65 - 19 - 8 - 17



saboteas a tu propia persona? cómo??  
escríbelo **aquí** y  
táchalo

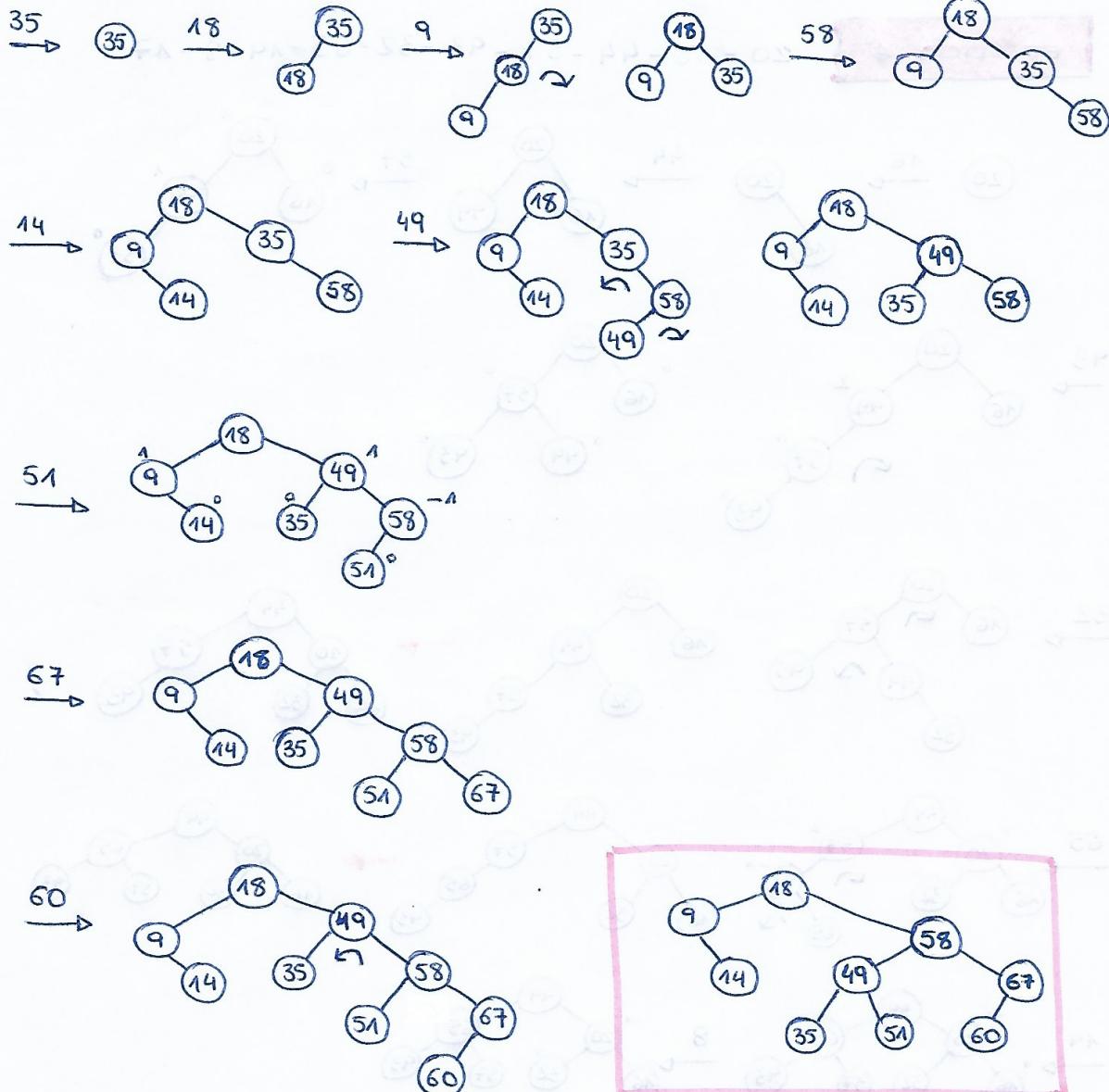
**manual de instrucciones:** escribe sin filtros  
y una vez acabes, táchalo (si lo compartes en redes  
mencionándonos, te llevas 10 coins por tu cara bonita)

X — X — X — X — X — X — X — X — X

DESFÓGATE CON WUOLAH

EJERCICIO 7

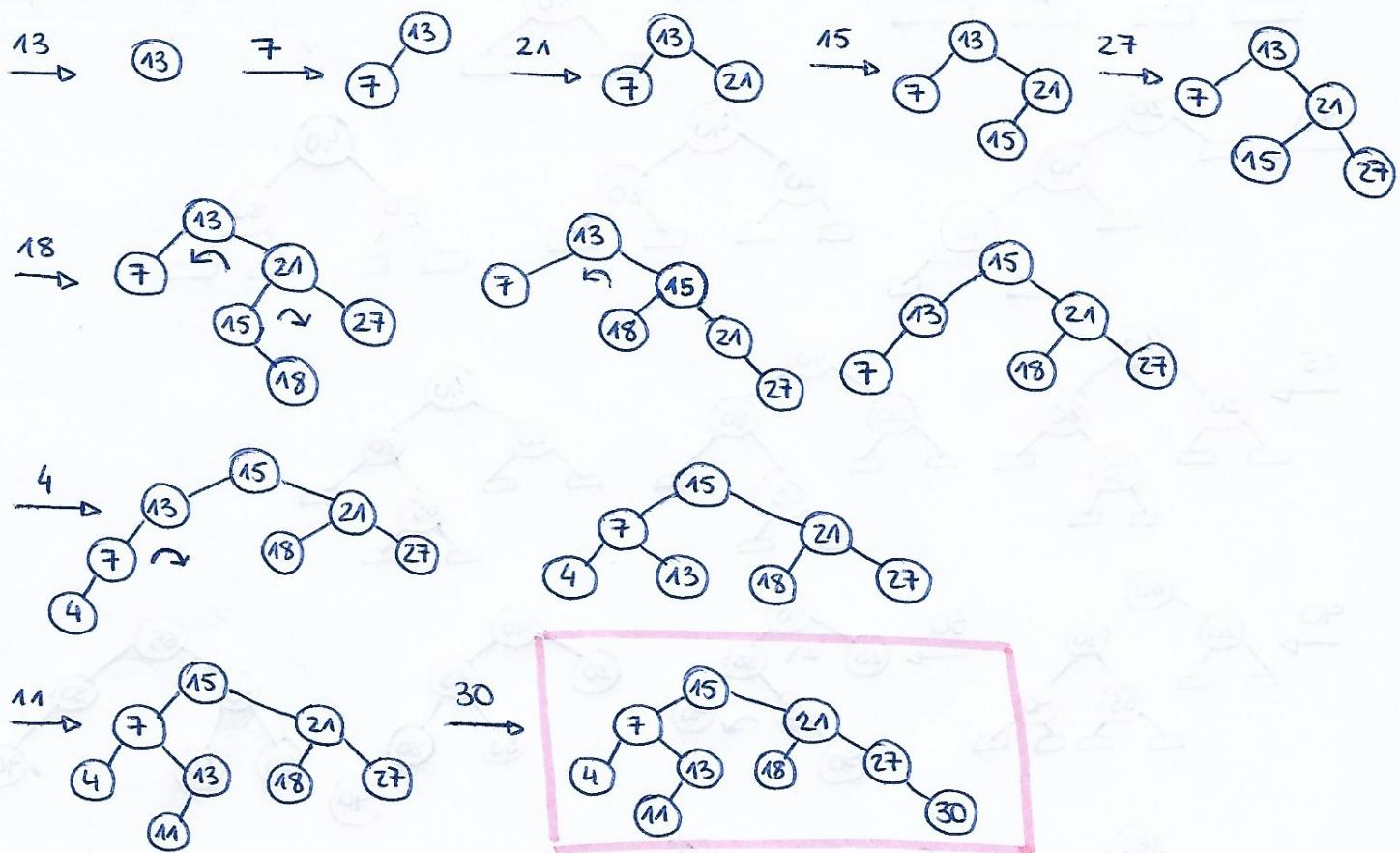
35 - 18 - 9 - 58 - 14 - 49 - 51 - 67 - 60



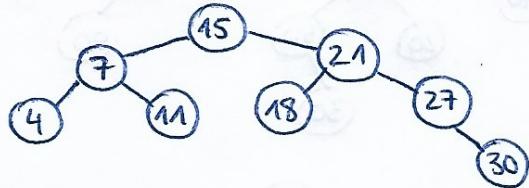
EJERCICIO 9

13 - 7 - 21 - 15 - 27 - 18 - 4 - 11 - 30

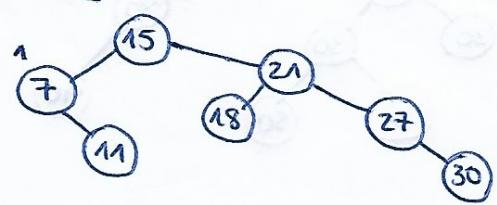
13 - 4 - 15 (eliminar)



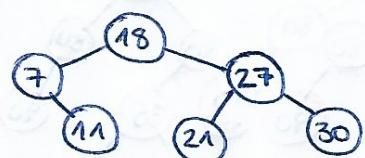
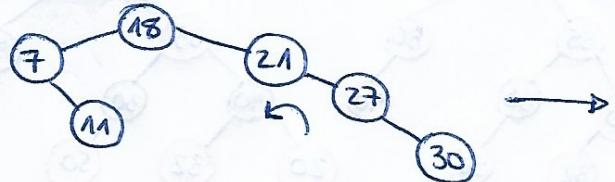
BORRAR 13



BORRAR 4

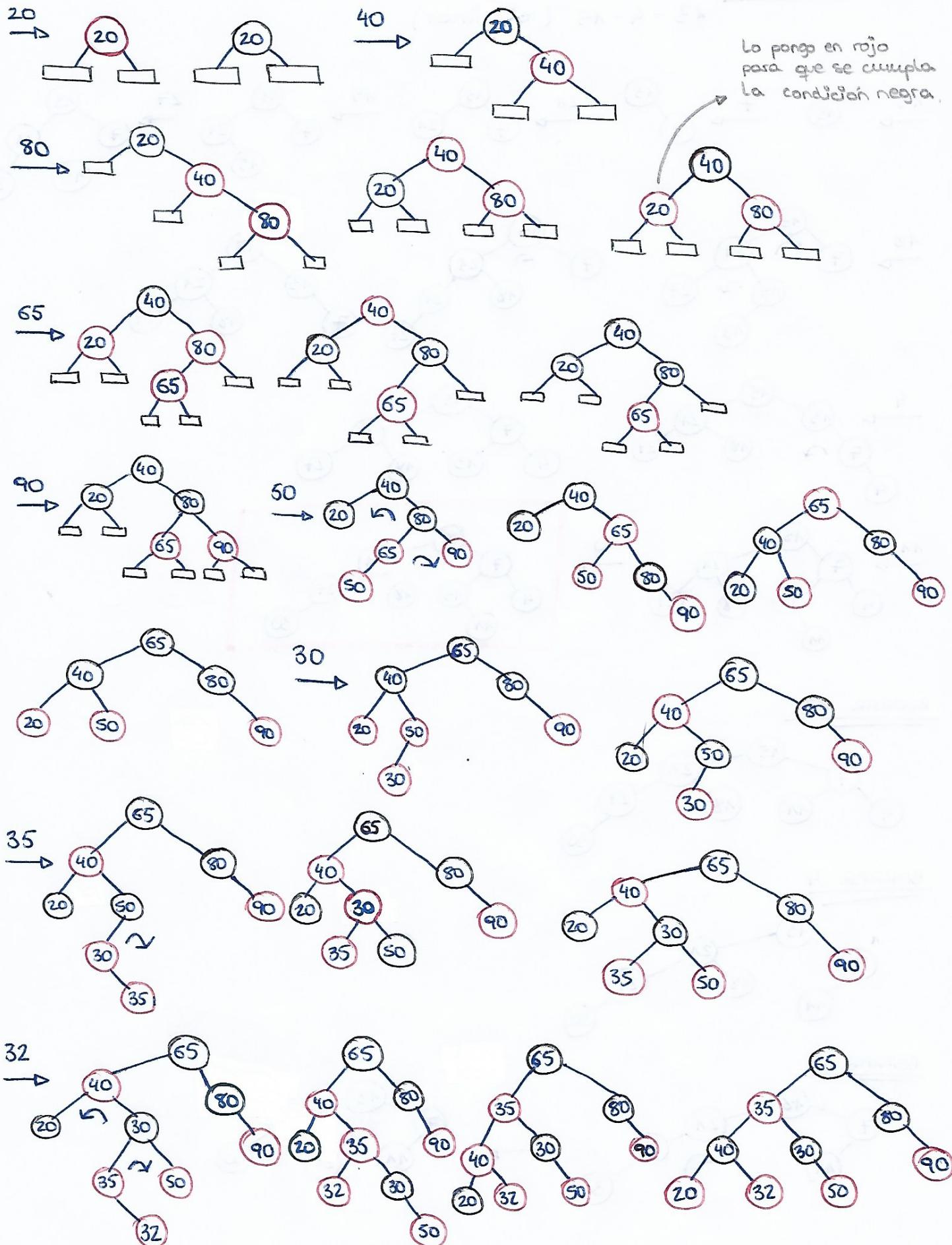


BORRAR 15



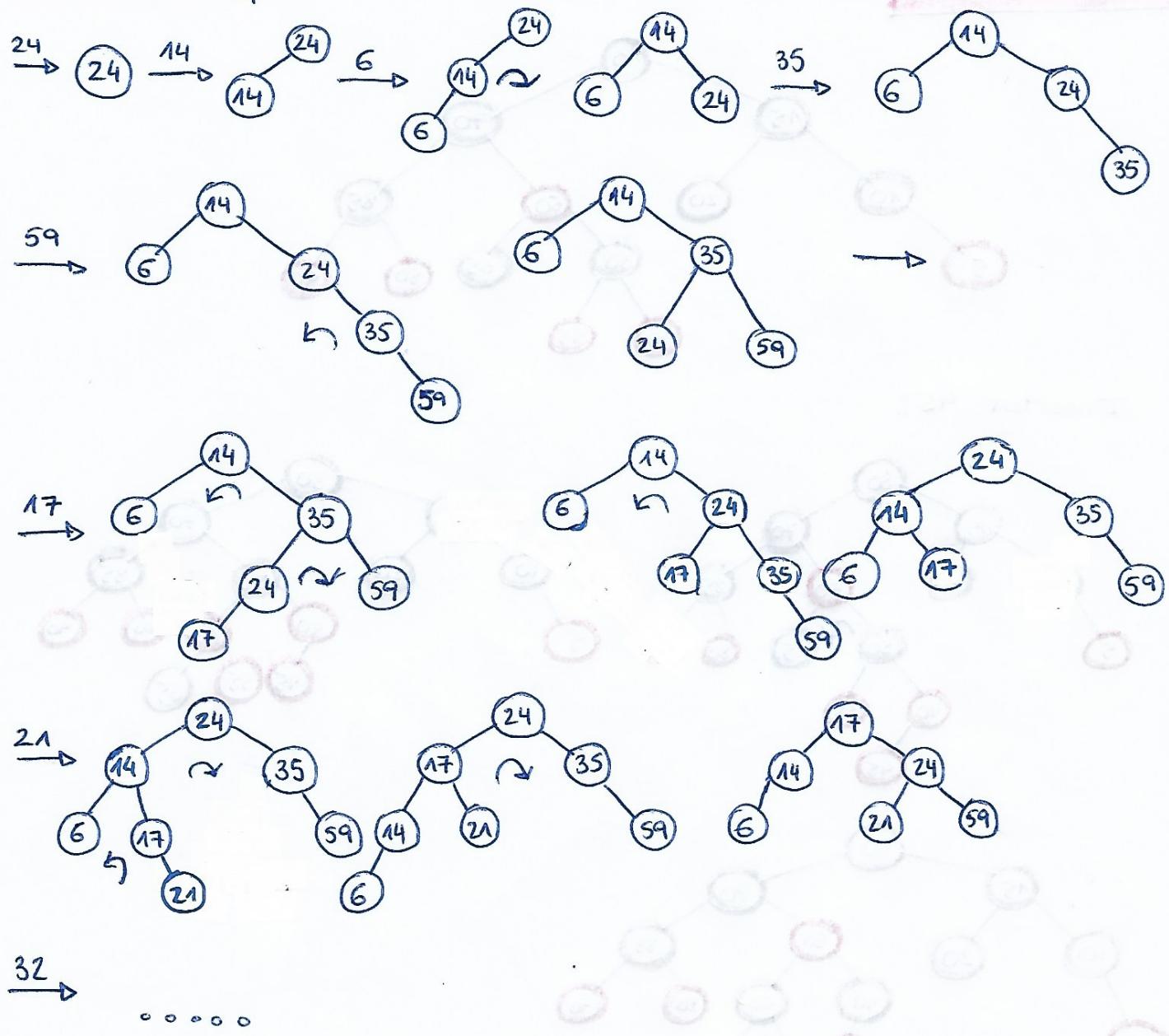
## EJERCICIO 10

20 - 40 - 80 - 65 - 90 - 50 - 30 - 35 - 32 - 70 - 60



EJERCICIO 8

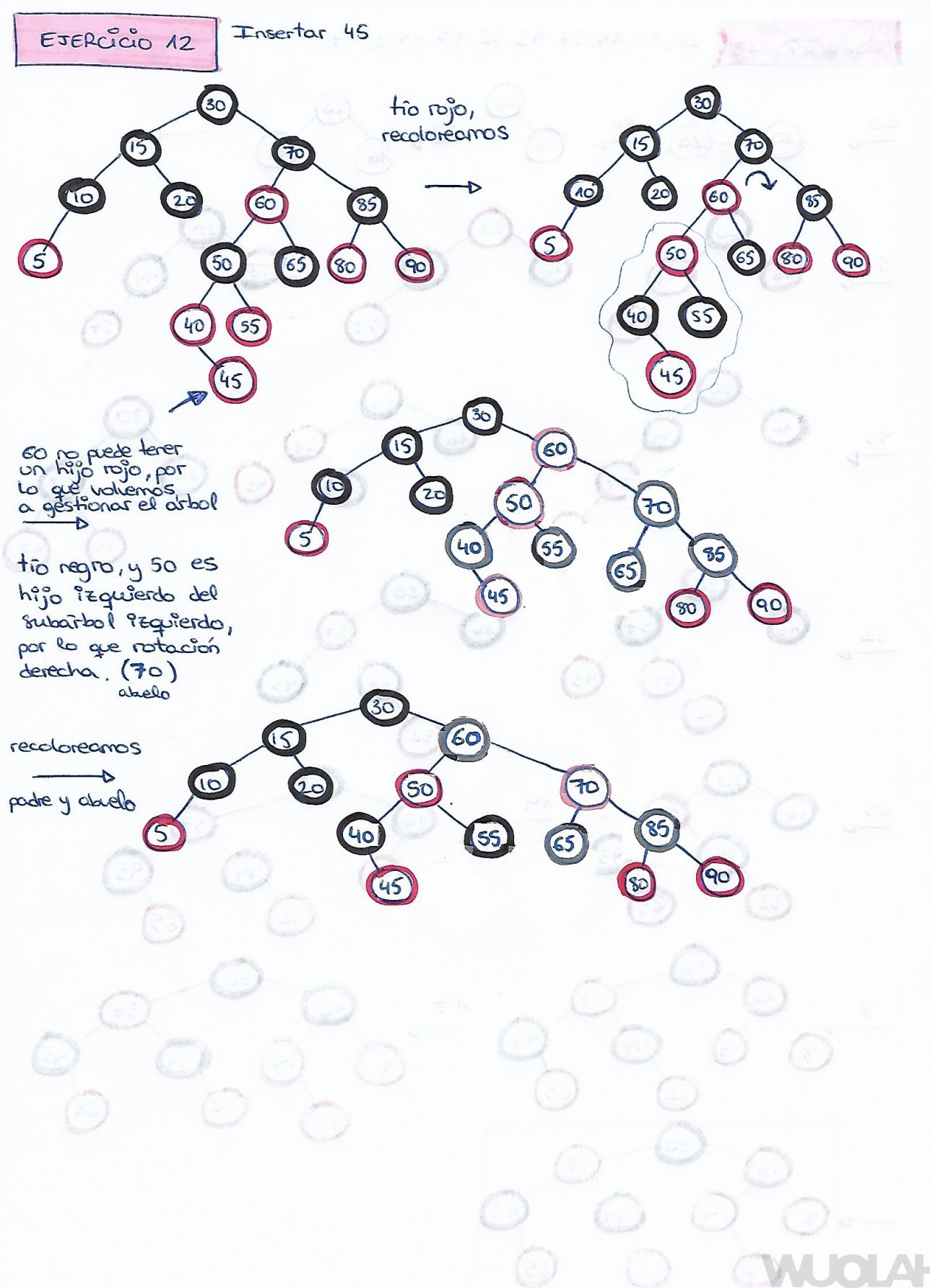
24, 14, 6, 35, 59, 17, 21, 32, 4, 7, 15, 22.



saboteas a tu propia persona? cómo??  
escríbelo **aquí** y  
táchalo

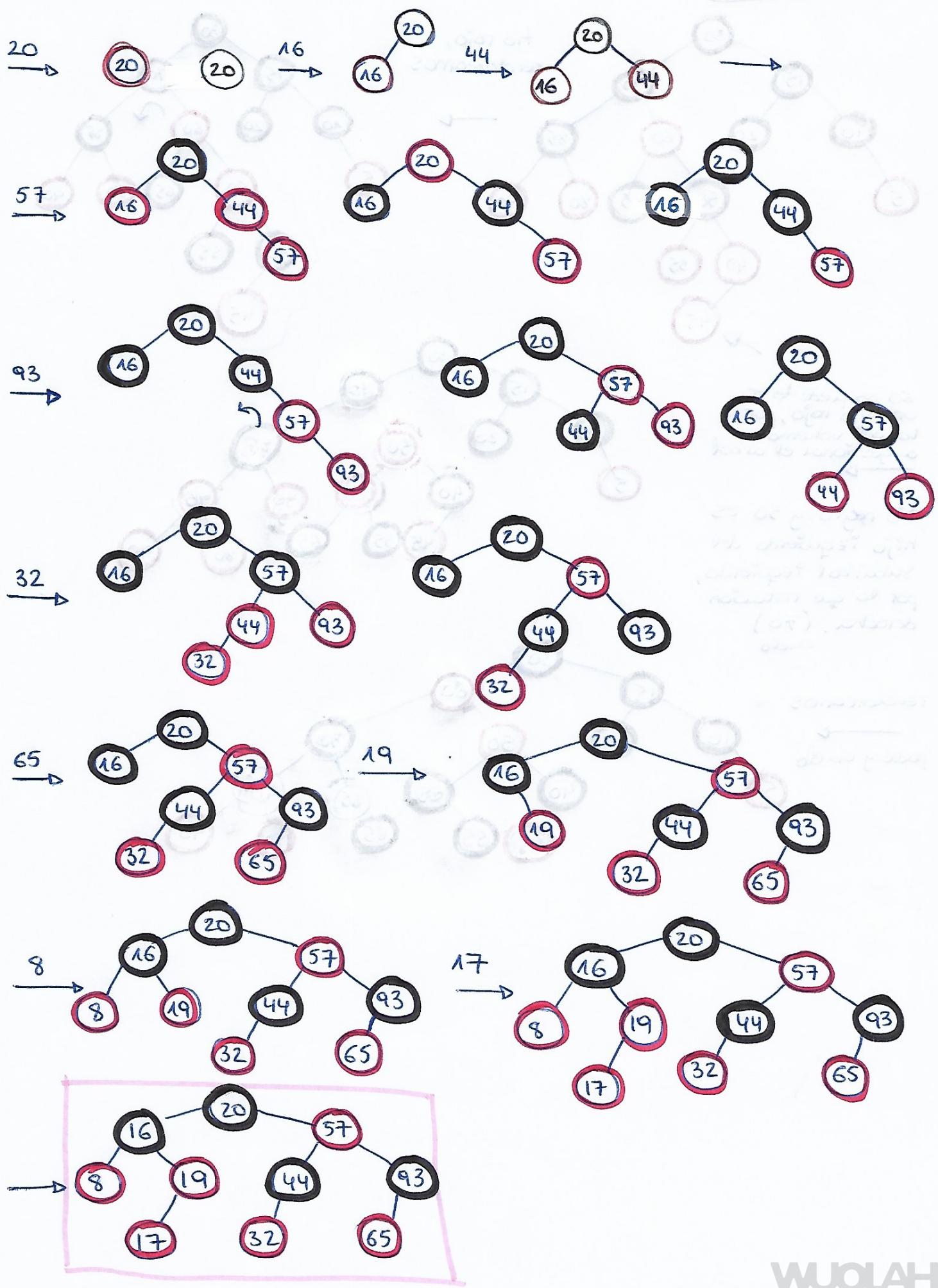
**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)



### EJERCICIO 13

20, 16, 44, 57, 93, 32, 65, 19, 8, 17.



# TEMA 4 : DICCIONARIOS

## Diccionario

→ Permite representar una aplicación entre dos conjuntos cuando no es posible describirla mediante un algoritmo.

Los valores del conjunto origen o dominio de la aplicación se denominan claves.



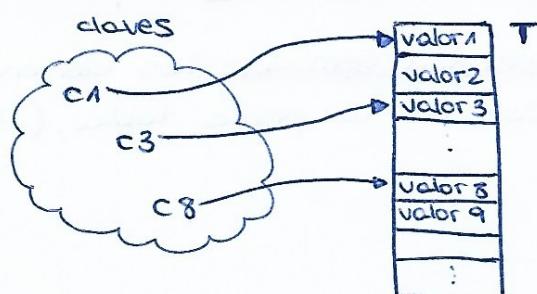
→ Un conjunto de pares  $(c, v)$ , donde  $c$  se denomina clave y  $v$  representa el valor asociado a la clave.  
Las claves son únicas.

## PRINCIPALES OPERACIONES

- Crear un diccionario vacío
- Insertar un nuevo par
- Comprobar la existencia de una clave
- Consultar el valor  $v$  asociado a una clave  $c$ .
- Modificar el valor  $v$  asociado a una clave  $c$ .
- Borrar un par  $(c, v)$
- Comprobar si el diccionario está vacío.

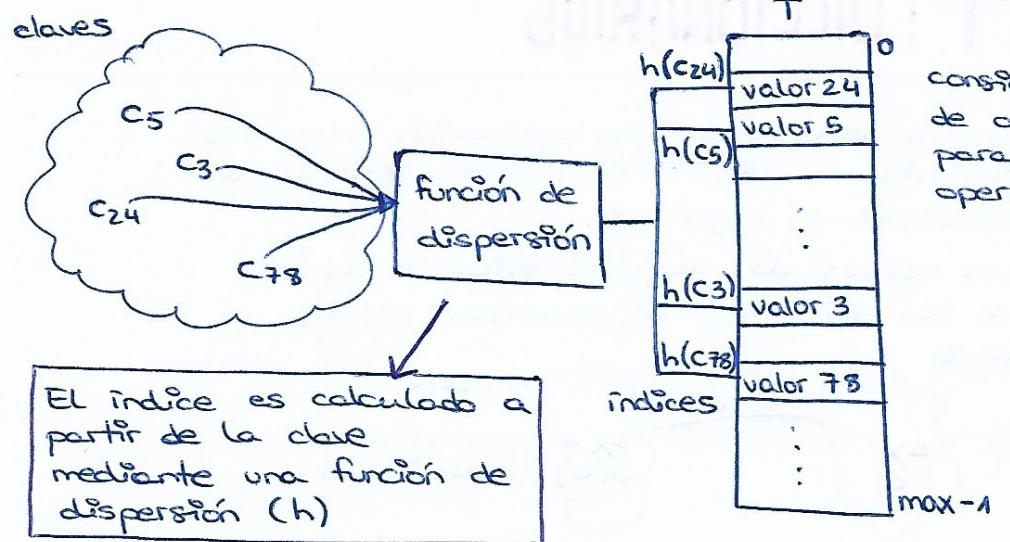
## TABLAS DISPERSAS

- Si el conjunto de las claves está formado por un número pequeño de números naturales (cercaos entre ellos) y no hay dos elementos con la misma clave, se puede utilizar una representación mediante una tabla (tabla de acceso directo)
- Si las claves no son numéricas o no se cumple lo anterior, debería existir una función inyectiva que transforme cada clave en un número (índice) comprendido en el rango de la tabla.



Alternativa a las tablas de acceso directo:

TABLAS DISPERSIAS  
O TABLAS HASH



- No es necesario que la función  $h$  sea inyectiva. Se dividen las claves en  $n$  clases de equivalencia. En cada clase se encuentran las claves, que al aplicarles la función  $h$ , devuelven el mismo valor (posición en la tabla).
- $h(c_i)$  es el índice o valor de dispersión calculado por  $h$  al suministrarle la clave  $c_i$ .
- Se dice que  $c_i$  se dispersa en la posición  $h(c_i)$  de la tabla.
- De esta forma se soluciona el problema de las claves no numéricas o números muy grandes.
- Aspecto fundamental  $\rightarrow$  realizar una buenas elección de la función  $h$ .
- La función debe distribuir las claves de la manera más uniforme posible, de forma que la probabilidad de que 2 claves distintas obtengan el mismo valor de  $h$  sea lo más baja posible.

### Decisiones a tomar

- ① Elegir función  $h$ . Como la función no es inyectiva es posible que una posición calculada para una clave ya esté ocupada  $\rightarrow$  **colisión**
- ② Seleccionar un método para resolver las colisiones:

2.1. Dispersion abierta: Los elementos que colisionan se sitúan en una lista a partir de la posición calculada.

2.2. Dispersion cerrada: Los elementos que colisionan se sitúan en la propia tabla. (Algoritmo de recolocación de elementos)

saboteas a tu propia persona? cómo??  
escríbelo **aquí** y táchalo

**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

X  
— X — X — X — X — X — X — X —

### CARACTERÍSTICAS DE LAS FUNCIÓNES DE DISPERSIÓN

1. Rapidez de cálculo
2. Exhastividad: todo índice o valor de dispersión  $h(c_i)$  debe tener asociado como mínimo una clave.
3. Distribución uniforme: Todos los valores de dispersión deben tener, aprox., el mismo número de claves asociadas.

### → Traducción de cadenas a enteros

- Suma de los caracteres: a cada carácter se le calcula su valor decimal y se suman los números obtenidos.
- Suma ponderada de los caracteres: la cadena se trata como un entero en base  $n$  ( $n = \text{nº de caracteres del código}$ )

### → Funciones que obtienen valores en un intervalo

- División: se calcula el resto de dividir el número entre  $\max$  ( $\max = \text{tamaño de la tabla}$ ) → suele ser  $n^{\text{º primo}}$ .
- Método del cuadrado: consiste en calcular el cuadrado de la clave  $c$ . El índice se obtiene de los dígitos de  $c^2$  que ocupan una determinada posición.
- Plegamiento: consiste en dividir la clave  $c$  en partes de igual número de dígitos y operar con ellas, tomando como índice los dígitos menos significativos del resultado.

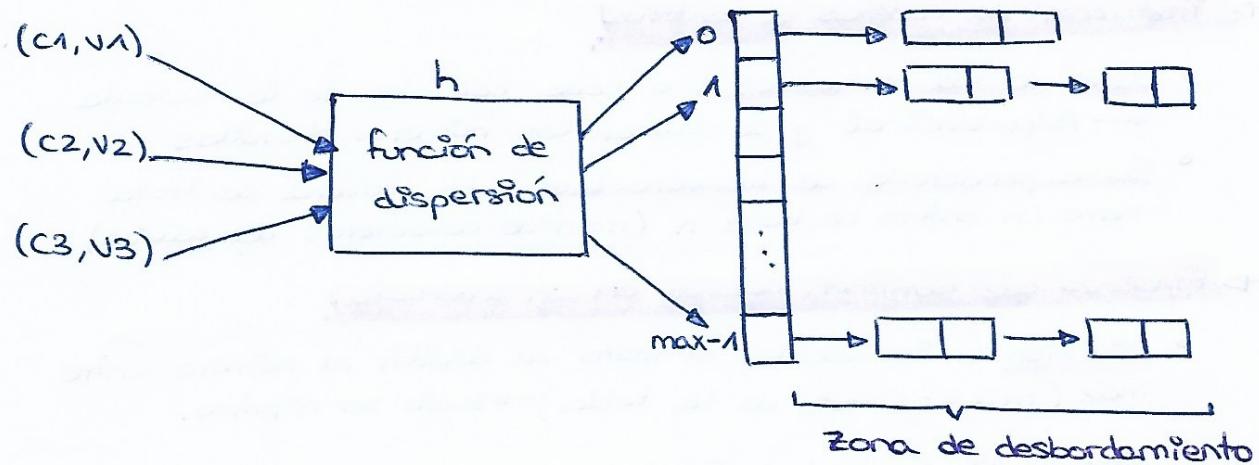
### → Claves universales de funciones de dispersión.

La dispersión universal es una técnica que permite elegir aleatoriamente una función de dispersión de forma que se obtenga un rendimiento promedio.

## RESOLUCIÓN DE COLISIONES

DISPERSIÓN ABIERTA → POR ENCADENAMIENTO.

- Todos los pares cuya clave es asignada por "h" a una misma posición, se almacenan en una lista asociada a dicha posición.
- La zona de memoria dinámica donde se almacenan las listas asociadas a cada posición se denomina zona de desbordamiento.



DISPERSIÓN CERRADA → RECOLOCACIÓN EN LA TABLA

- Consiste en almacenar las claves cuyo índice ya esté ocupado (colisión), en otra posición libre de la tabla.
- Las colisiones se resuelven calculando una secuencia de huecos de dispersión.
- Los pares reubicados ocupan posiciones de futuras claves. El funcionamiento de este método será bueno si el factor de carga  $\alpha$  es menor de 0'8.
  - El factor de carga indica el nivel de ocupación de la tabla con pares almacenados.

$$\alpha = \frac{n}{\text{max}} \quad \begin{cases} \alpha = 1 & \rightarrow \text{igual n.º de pares que de posiciones} \\ \alpha < 1 & \rightarrow \text{menos pares que posiciones} \\ \alpha > 1 & \rightarrow \text{más pares que posiciones.} \end{cases}$$

- - Prueba lineal
  - Prueba cuadrática

## → Características de la dispersión cerrada.

- Los tiempos de las operaciones con la tabla tienden a degenerar cuando se trabaja con un factor de carga alto. Mejor mantenerlo por debajo de 0'8.
- En la secuencia de búsqueda se entremezclan claves procedentes de diferentes valores de la función de dispersión.
- Se requiere algún mecanismo que permita distinguir una posición libre de la que no lo está → asociar un valor lógico a cada posición de la tabla que indique si está o no ocupada.

## BÚSQUEDA

1. Acceder a la posición  $h(c)$
  2. Si la posición esta ocupada por  $c$ , fin de la búsqueda.
  3. Si la posición no está ocupada por  $c$ , seguir la misma secuencia de  $P_k$  que se empleó al insertar la clave, hasta encontrar la clave, una posición vacía o finalizar la secuencia de índices  $P_k$ .
- Este algoritmo de búsqueda plantea problemas si se permiten eliminaciones de clave → el algoritmo podría finalizar con "no encontrado" al llegar a una posición liberada por otra clave almacenada en una de las posiciones de la secuencia  $P_k$ .
  - Solución:  marcar las posiciones de la tabla con uno de los valores q "Libre", "ocupado", "liberado" p. La búsqueda continuará al encontrar un valor "liberado".

# - BOLETÍN TEMA 4 -

## Ejercicio tabla Hash moodle

Tabla dispersa de tamaño 9, utilizando dispersión cerrada.

$$h(x) = (7x + 1) \bmod \text{tamaño}$$

Ins(13), Ins(20), Ins(4), Ins(8), Del(4), Buscar(8), Ins(16)

- a) Mostrar contenido de la tabla usando "prueba lineal" como método de recolocación.

H	
I	
N	
S	
E	
R	
C	
O	
N	
E	
S	

$$\left\{ \begin{array}{l} h(13) = (7 \cdot 13 + 1) \bmod 9 = 2 \\ h(20) = (7 \cdot 20 + 1) \bmod 9 = 6 \\ h(4) = (7 \cdot 4 + 1) \bmod 9 = 2 \\ \quad \downarrow \\ \quad \text{colisión} \\ \quad \text{Prueba lineal} \rightarrow 3 \\ h(8) = (8 \cdot 7 + 1) \bmod 9 = 3 \\ \quad \downarrow \\ \quad \text{colisión} \\ \quad \text{Prueba lineal} \rightarrow 4 \end{array} \right.$$

Índice	Clave	Estado
0		
1		
2	13	Ocupado
3	4	Ocupado
4	8	Ocupado
5		
6	20	Ocupado
7		
8		

### Del(4)

$$h(4) = (4 \cdot 7 + 1) \bmod 9 = 2 \rightarrow \text{Ocupada}(13)$$

$$p_1(4) = (h(4) + 1) \bmod 9 = 3 \rightarrow \text{ENCONTRADA}$$

$\downarrow$   
Borrar clave y poner  
estado a "Libre"

### Buscar(8)

$$h(8) = (8 \cdot 7 + 1) \bmod 9 = 3 \rightarrow \text{LIBERADA}$$

$\downarrow$   
Seguir buscando

$$p_1(8) = (h(8) + 1) \bmod 9 = 4 \rightarrow \text{ENCONTRADA}$$

### Ins(16)

$$h(16) = (7 \cdot 16 + 1) \bmod 9 = 5$$

Índice	Clave	Estado
0		
1		
2	13	Ocupado
3	X	Ocupado Liberado
4	8	Ocupado
5	16	Ocupado
6	20	Ocupado
7		
8		

saboteas a tu propia persona? cómo??  
escríbelo **aquí** y táchalo

**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

X — X — X — X — X — X — X — X — X

DESFÓGATE CON WUOLAH

- b) Indicar qué casos de inserciones y búsquedas tienen el mayor coste en cuanto al número de operaciones necesarias.
- b.1) La inserción de una clave  $x$  tal que  $h(x) = 4$ , es la que tiene mayor coste  $\rightarrow$  requiere 4 operaciones para poder insertarla.
- b.2) La búsqueda de una clave  $y$  tal que  $h(y) = 2$ , es la que tiene mayor coste  $\rightarrow$  puede requerir hasta 6 comparaciones para encontrarla.

# TEMA 5: GRAFOS

## CONCEPTOS

$$G = (V, A)$$

$V \rightarrow$  conjunto de vértices o nodos  
 $A \rightarrow$  conjunto de aristas o arcos

- Las aristas representan relaciones entre los vértices, de forma que una arista es un par  $(v, w)$  de vértices de  $V$ .

## TIPOS

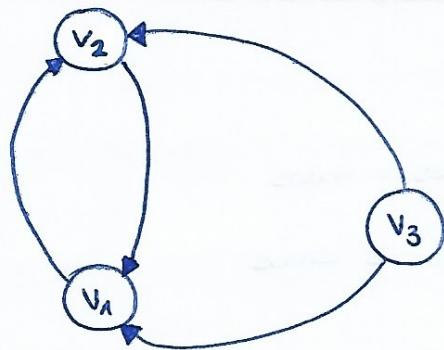
- Grafo dirigido: Aquel cuyas aristas forman pares ordenados  $(v \rightarrow w)$
- Grafo no dirigido: Aquel cuyas aristas son pares no ordenados
- Grafo etiquetado o valorado: Se asocia información a cada arista.
- Grafo no etiquetado: No se asocia ninguna información a las aristas.

## DEFINICIONES

- Camino: Secuencia de vértices
- Longitud de un camino: número de vértices menos 1
- Camino simple: si todos sus vértices son distintos.
- Ciclo: camino de longitud no nula que empieza y termina en el mismo vértice.
- Ciclo simple: si cada vértice y cada arista del ciclo son distintos (excepto primer y último vértices)
- En un grafo no dirigido, el grado de un vértice v es el número de aristas que contiene a  $v$ .
- Vértice sucesor o adyacente: del vértice  $x$ , si existe una arista que tenga por origen a  $x$  y por destino a  $y$ .
- Vértice x antecesor del vértice y: si existe una arista que tenga por origen a  $x$  y por destino a  $y$ .
- Grafo dirigido
  - Grado de entrada: nº de aristas que llegan.
  - Grado de salida: nº de aristas que salen.

## EJEMPLO 2

## GRAFO DIRIGIDO



Número de nodos : 3

Número de aristas : 4

$V = \{V_1, V_2, V_3\}$

$A = \{(V_1, V_2), (V_2, V_1), (V_2, V_3), (V_3, V_1)\}$

Camino :  $V_3, V_2, V_1, V_2$

Ciclo :  $V_1, V_2, V_1$

Grado Entr(V<sub>2</sub>) = 2

Grado Sal(V<sub>2</sub>) = 1

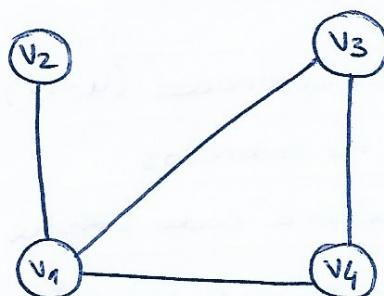
Adyacentes(V<sub>3</sub>) = {V<sub>1</sub>, V<sub>2</sub>}

Antecesores(V<sub>3</sub>) = {} Ø

Antecesores(V<sub>2</sub>) = {V<sub>1</sub>, V<sub>3</sub>}

## EJEMPLO 1

## GRAFO NO DIRIGIDO



Número de nodos : 4

Número de aristas : 4

$V = \{V_1, V_2, V_3, V_4\}$

$A = \{(V_2, V_1), (V_1, V_3), (V_3, V_4), (V_4, V_1)\}$

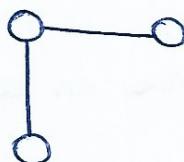
Camino :  $V_1, V_3, V_4, V_1, V_2$

Ciclo :  $V_1, V_3, V_4, V_1$

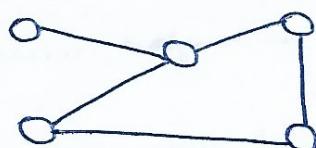
Grado(V<sub>3</sub>) : 2

Adyacentes(V<sub>3</sub>) : {V<sub>1</sub>, V<sub>4</sub>}

- Grafo no dirigido CONEXO : existe un camino entre cualquier par de nodos que forman el grafo.

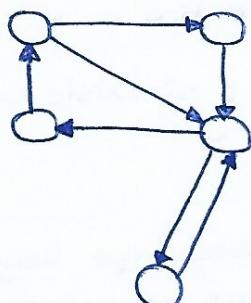


NO CONEXO



CONEXO

- Grafo dirigido FUERTEMENTE CONEXO : existe un camino entre cualquier par de nodos que forman el grafo.

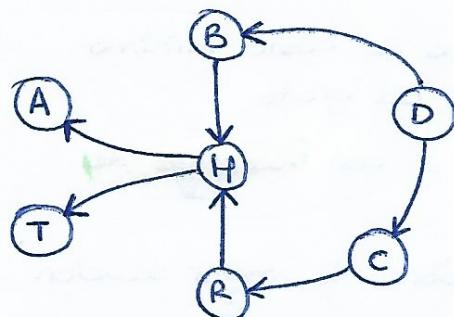


## RECORRIDOS SOBRE GRAFOS

EN ANCHURA

→ cola →

para mantener los vértices que se vayan a procesar posteriormente.

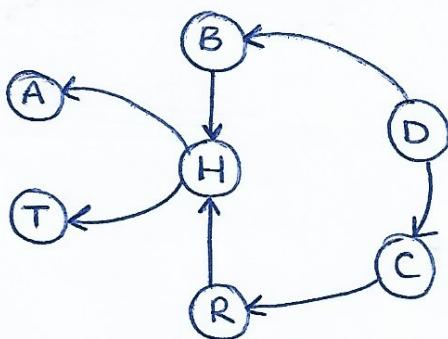


Vért. recorridos desde D	Estado cola
	Frente D Final
D	B C
DB	C H
DBC	H R
DBCH	R A T
DBCHR	A T
DBCHRA	T
DBCHRAT	Cola vacía

EN PROFUNDIDAD

→ pila →

para mantener los vértices que se vayan a procesar posteriormente.



Vért. recorridos desde D	Estado pila
	Fondo D Cima
D	B C
DC	B R
DCR	B H
DCRH	B A T
DCRHT	B A
DCRHTA	B
DCRHTAB	pila vacía

## ORDENACIÓN TOPOLOGICA

Es la ordenación lineal de todos los nodos de un grafo según la distribución y sentido de sus aristas.

- Aplicable solo a Grafos Dirigidos Acíclicos.
- Un vértice solo se visita si han sido visitados todos sus predecesores. Al comenzar, solo se pueden visitar los nodos que no tienen ningún predecesor.

(mismo grafo de arriba)

L	D	C	B	R	H	A	T
numPred	A	B	C	D	H R	T	
	1	1	1	1	1 1	1	1

saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

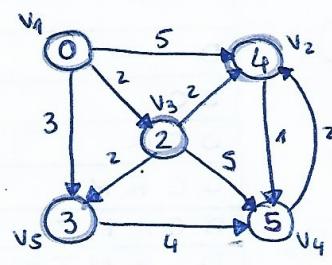
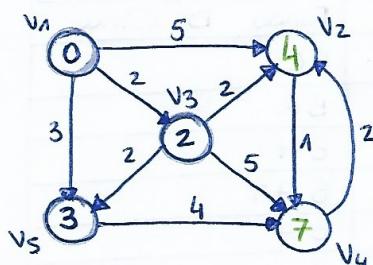
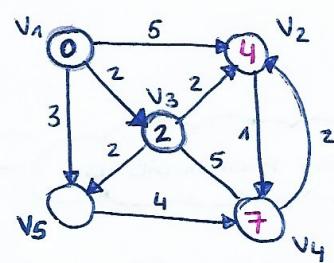
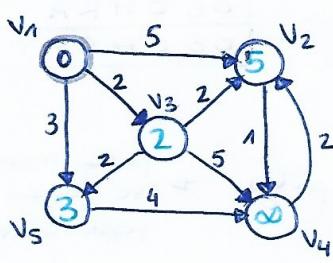
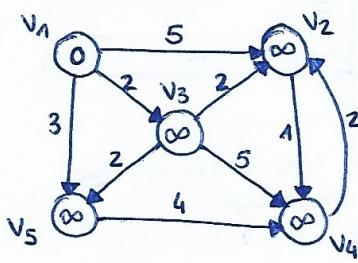


### CAMINOS MÍNIMOS SOBRE GRAFOS

- Coste de un camino  $\rightarrow$  suma de los costes de las aristas

### DIJKSTRA

- Resuelve el problema de encontrar el camino de coste mínimo desde un vértice  $v$  al resto de los vértices del grafo.
- El grafo tiene que ser dirigido, valorado y con factores de peso positivos.
- Algoritmo voraz: en cada etapa se selecciona la mejor solución entre las disponibles.



Iteración	Tratados	T	D[2]	D[3]	D[4]	D[5]
Inicial		$v_1 v_2 v_3 v_4 v_5$	$\infty$	$\infty$	$\infty$	$\infty$
1	$v_1$	$v_2 v_3 v_4 v_5$	5	2	$\infty$	3
2	$v_1 v_3$	$v_2 v_4 v_5$	4	2	7	3
3	$v_1 v_3 v_5$	$v_2 v_4$	4	2	7	3
4	$v_1 v_3 v_5 v_2$	$v_4$	4	2	5	3

Porque es el que tiene mínimo coste

$$A[1] = 0 \quad A[2] = 3 \quad A[3] = 1$$

$$A[4] = 2 \quad A[5] = 1$$

$A =$ vector que guarda el antecesor óptimo.

## FLOYD

- Se basa en el esquema de programación dinámica
- Utiliza una tabla para ir almacenando los resultados correspondientes a instancias más sencillas del problema a resolver.

EJEMPLO EN HOJA IMPRESA

### ÁRBOL DE EXPANSIÓN DE COSTE MÍNIMO

- Se aplica en grafos no dirigidos, conexos y etiquetados (etiquetas no negativas).

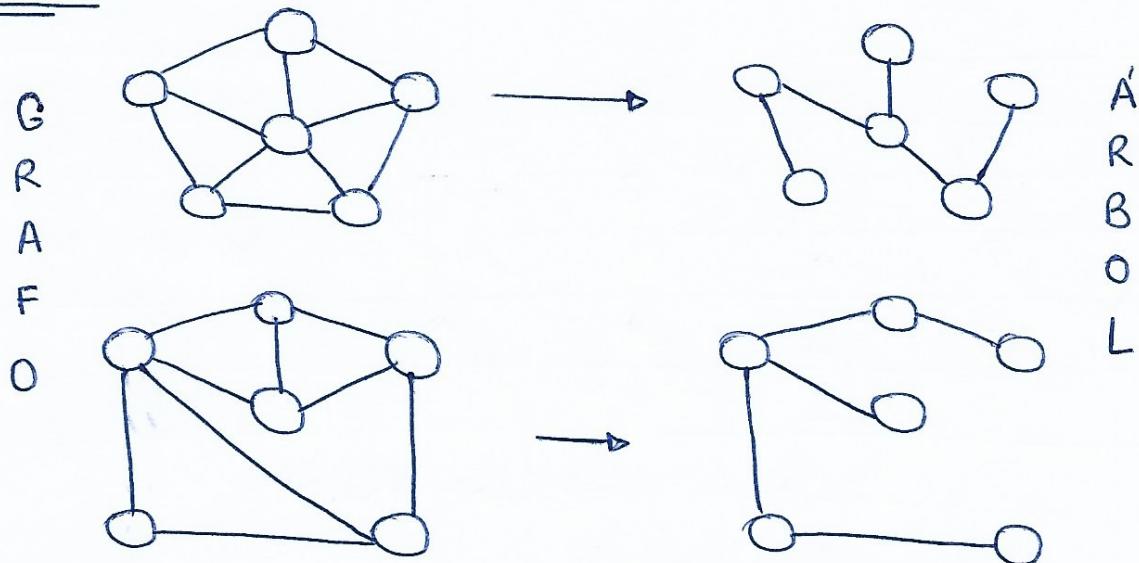
#### ■ Árbol libre

- Grafo no dirigido, conexo y acíclico
- Diferencia con los árboles generales: no tienen raíz y los hijos no están ordenados.
- Todo árbol libre con  $n$  vértices tiene  $n-1$  aristas
- Si se borra una arista deja de ser conexo, y si se añade una nueva arista, entonces aparecen ciclos.

#### ■ Árbol de extensión o de recubrimiento de coste mínimo:

- Subconjunto  $G'$  del grafo  $G$
- Contiene todos los vértices de  $G$ .
- Es conexo y acíclico
- No existe otro  $G''$  tal que la suma de los costes de las aristas de  $G''$  sea menor que la suma de los costes de las aristas de  $G'$ .

#### EJEMPLOS:

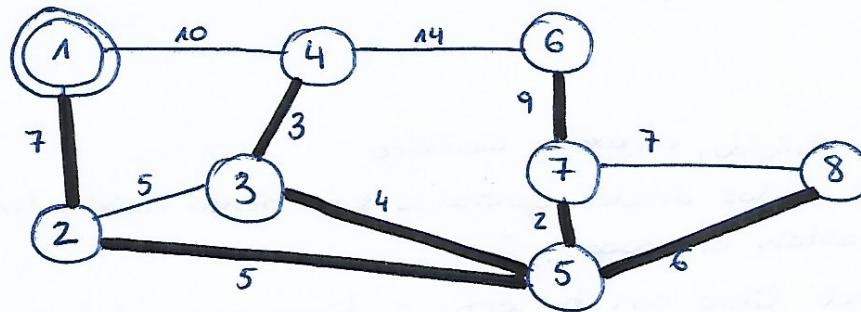


PRIM



Parte de un vértice cualquiera y va extendiendo el árbol de recubrimiento, incorporando un nuevo vértice en cada iteración, hasta cubrir todos los vértices del grafo.

- Dado un grafo, se mantienen 2 particiones:
  - U - Vértices que se encuentran enlazados por las aristas del árbol que se está construyendo,
  - V - Resto de vértices.
- En cada iteración se incrementa el subconjunto U con un nuevo vértice, hasta que  $U=V$ .
- Inicialmente, U contiene un único vértice que puede ser cualquiera.



KRUSCAL

- Durante el algoritmo se mantendrá siempre un bosque de árboles. Inicialmente, el bosque contiene tantos árboles como vértices tiene el grafo.

» EJEMPLO EN HOJA IMPRESA



saboteas a tu propia persona? cómo??  
escríbelo **aquí** y  
táchalo

**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

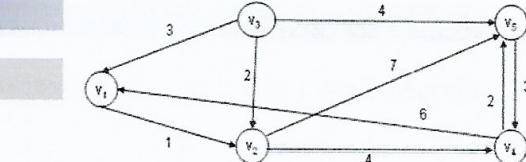


### Solución

$D_4 =$

0	1	$\infty$	5	7
10	0	$\infty$	4	6
3	2	0	6	4
6	7	$\infty$	0	2
9	10	$\infty$	3	0

$D_3 [1,4] + D_3 [4,5]$



$A =$

0	0	0	2	4
4	0	0	0	4
0	0	0	2	0
0	1	0	0	0
4	4	0	0	0

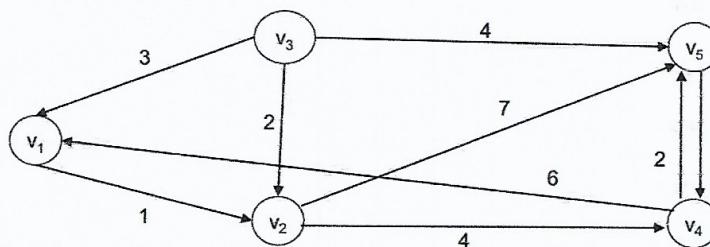
$D_5 = D_4 =$

0	1	$\infty$	5	7
10	0	$\infty$	4	6
3	2	0	6	4
6	7	$\infty$	0	2
9	10	$\infty$	3	0

$A =$

0	0	0	2	4
4	0	0	0	4
0	0	0	2	0
0	1	0	0	0
4	4	0	0	0

### Ejemplo Floyd



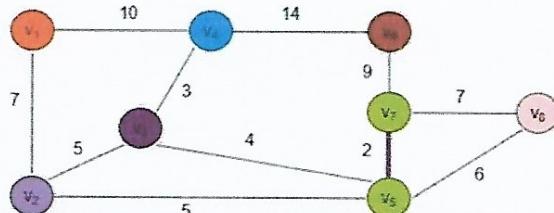
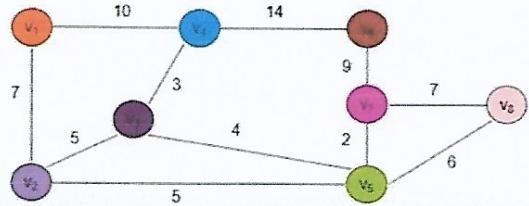
V1	V2	V3	V4	V5
0	0	0	2	4
4	0	0	0	4
0	0	0	2	0
0	1	0	0	0
4	4	0	0	0

Camino de  $v_5$  a  $v_2$ :

$5 \rightarrow 4 \rightarrow 1 \rightarrow 2$

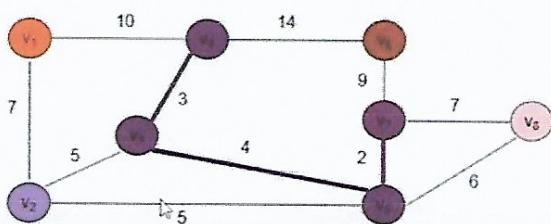
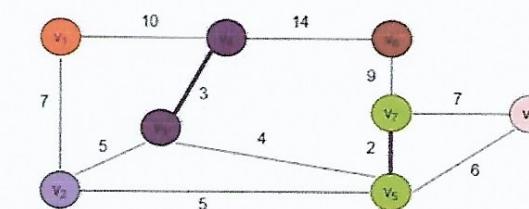
$v_5 - v_4$   
(directo)       $v_4 - v_2$   
(pasa por  $v_1$ )

# KRUSCAL



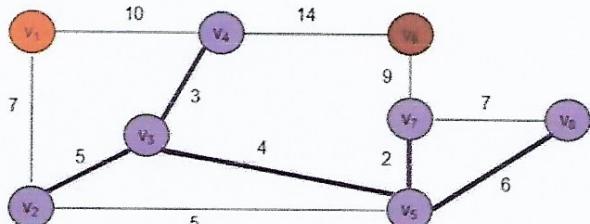
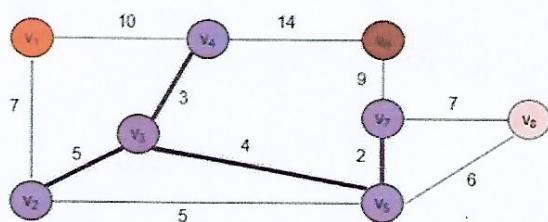
B	v1	v2	v3	v4	v5	v6	v7	v8
0	1	2	3	4	5	6	7	

B	v1	v2	v3	v4	v5	v6	v7	v8
0	1	2	3	4	5	4	7	



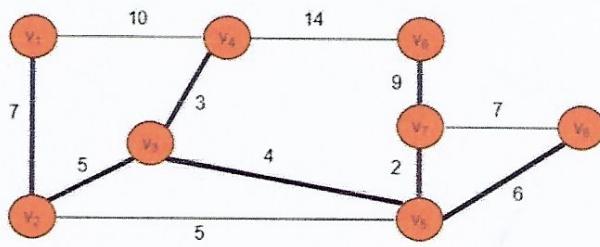
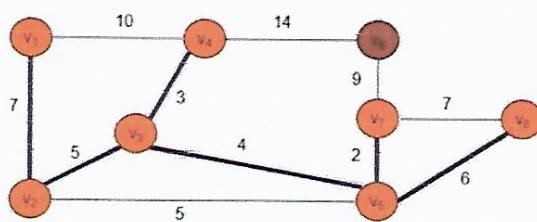
B	v1	v2	v3	v4	v5	v6	v7	v8
0	1	2	2	4	5	4	7	

B	v1	v2	v3	v4	v5	v6	v7	v8
0	1	2	2	2	5	2	7	



B	v1	v2	v3	v4	v5	v6	v7	v8
0	1	1	1	1	5	1	7	

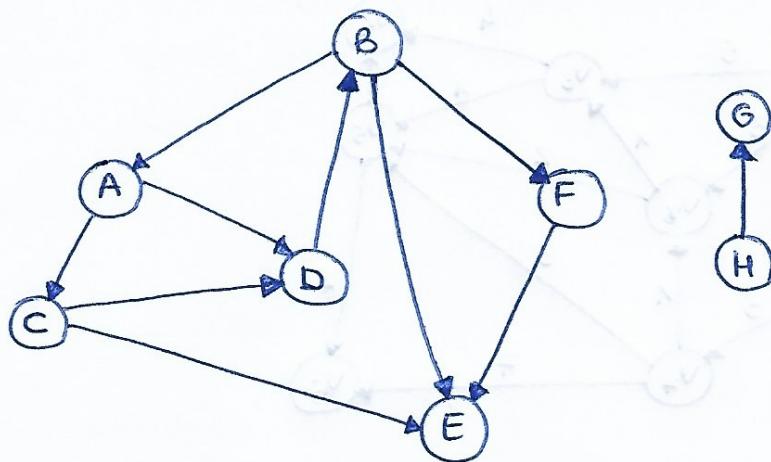
B	v1	v2	v3	v4	v5	v6	v7	v8
0	1	1	1	1	1	5	1	1



B	v1	v2	v3	v4	v5	v6	v7	v8
0	0	0	0	0	0	5	0	0

B	v1	v2	v3	v4	v5	v6	v7	v8
0	0	0	0	0	0	0	0	0

EJERCICIO 6



RECORRIDO EN ANCHURA → COLA.

Frente D Final

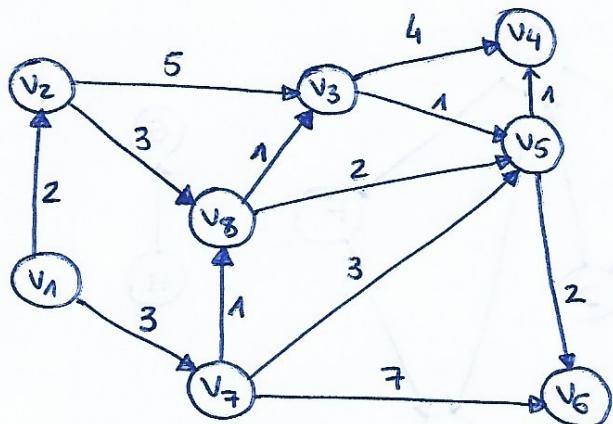
Vértices recor. desde D	Estado cola
D	B
DB	A E F
DBA	E F C
DBAE	F C
DBAEF	C
DBAEFC	cola vacía

RECORRIDO EN PROFUNDIDAD → PILA

Fondo D Cima

Vértices recor. desde D	Estado pila
D	B
DB	A E F
DBF	A E
DBFE	A
DBFEA	C
DBFEAC	pila vacía

## EJERCICIO 7



Iteración	Tratados	T	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]	D[8]
Inicial		v1, v2, v3, v4, v5, v6, v7, v8	∞	∞	∞	∞	∞	∞	∞
1	v1	v2, v3, v4, v5, v6, v7, v8	(2)	∞	∞	∞	∞	3	∞
2	v1, v2	v3, v4, v5, v6, v7, v8	2	7	∞	∞	∞	3	5
3	v1, v2, v7	v3, v4, v5, v6, v8	2	7	∞	6	10	3	4
4	v1, v2, v7, v8	v3, v4, v5, v6	2	(5)	∞	6	10	3	4
5	v1, v2, v7, v8, v3	v5, v4, v6	2	5	9	(6)	10	3	4
6	v1, v2, v7, v8, v3, v5	v4, v6	2	5	(7)	6	8	3	4
7	v1, v2, v7, v8, v3, v5, v4	v6	2	5	7	6	8	3	4

$$A[1] = 0$$

$$A[2] = 1$$

$$A[3] = 8$$

$$A[4] = 5$$

$$A[5] = 7$$

$$A[6] = 5$$

$$A[7] = 1$$

$$A[8] = 7$$

X - X - X - X - X - X - X

saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

**manual de instrucciones:** escribe sin filtros y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

## TEMA 6: COLAS DE PRIORIDAD

### CONCEPTOS

- Los elementos que entran en la cola de prioridad van saliendo de ella por un orden de prioridad establecido, en lugar de por orden de llegada.
- Cada elemento debe contener un valor numérico que represente su prioridad.

### Operaciones más usuales :

- "atender" (eliminar) al primer elemento → mayor prioridad
- insertar un nuevo elemento (atendiendo a su prioridad)
- "saber" (obtener) quien es el primer elemento.

### IMPLEMENTACIÓN CON SECUENCIAS

- Los elementos, con sus prioridades, se almacenan en una secuencia.
- 2 implementaciones, en función de si la secuencia está o no ordenada según la prioridad de los elementos.

#### SECUENCIA NO ORDENADA

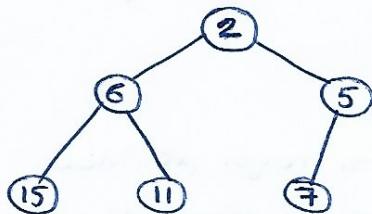
- insertar (e) : añadir e al final de la secuencia
- primero(), suprimir() : inspeccionar todos los elementos de la secuencia buscando el elemento de mayor prioridad.

#### SECUENCIA ORDENADA

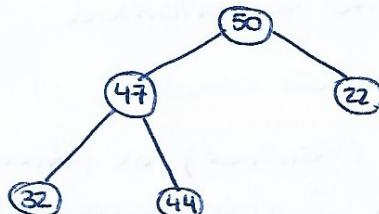
- primero(), suprimir() : acceso o eliminación del primer elemento de la secuencia.
- insertar (e) : recorrer la secuencia hasta encontrar la posición adecuada donde insertar e. La secuencia debe mantenerse ordenada.

## IMPLEMENTACIÓN CON MONTÍCULOS

- Consigue complejidad  $O(\log(n))$  para la inserción y la eliminación.
- Un montículo binario de mínimos es un árbol semicompleto en el que el valor de la clave (prioridad) almacenada en cualquier nodo es menor que los valores de clave de sus hijos. De forma análoga se define el montículo binario de máximos.



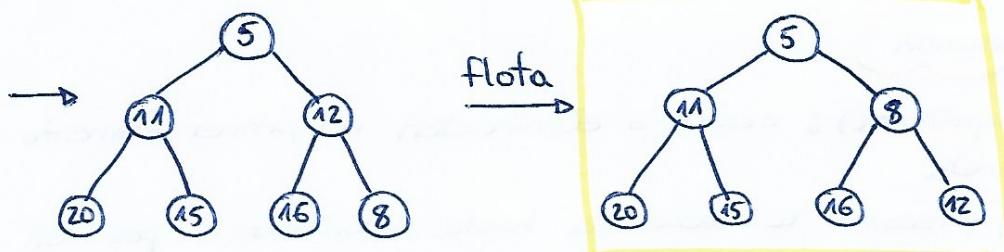
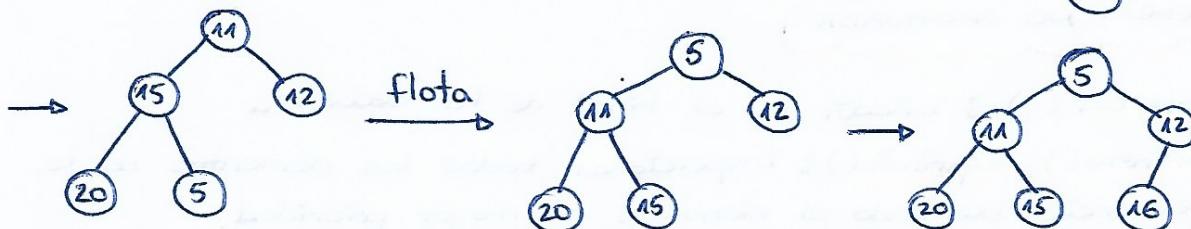
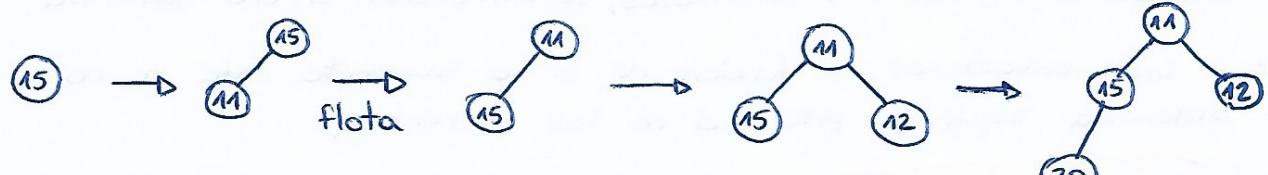
MONTÍCULO BINARIO DE MÍNIMOS



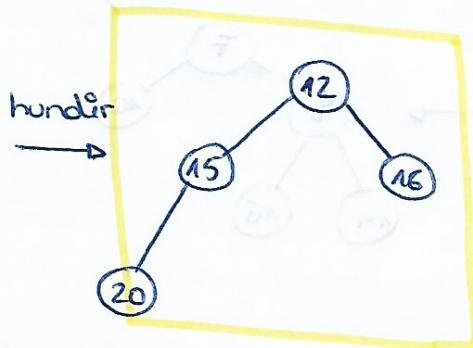
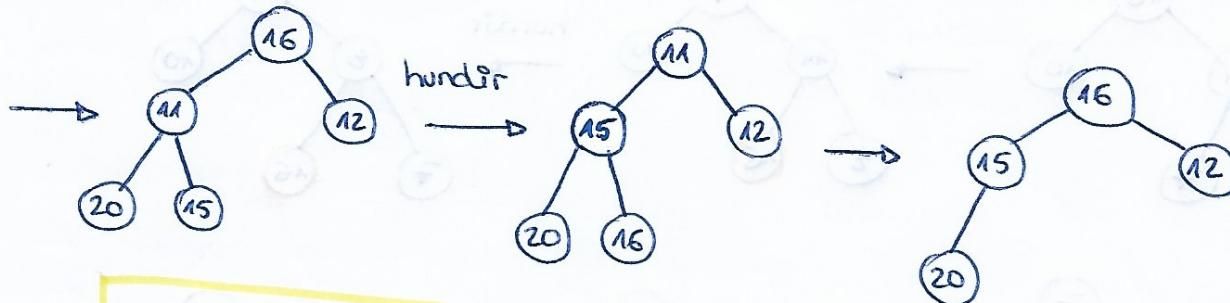
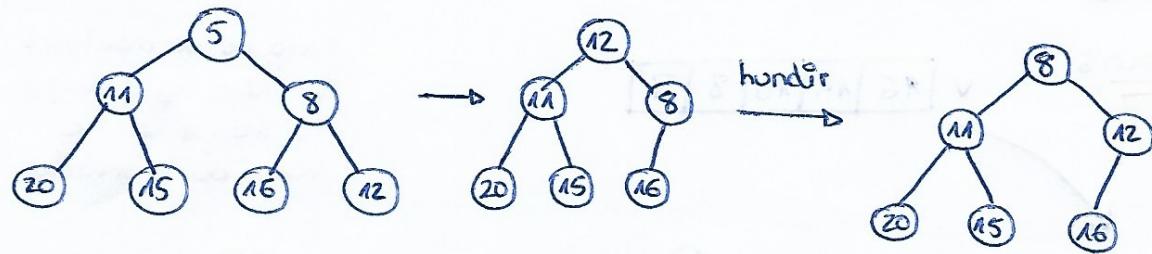
MONTÍCULO BINARIO DE MÁXIMOS

### INserción

15, 11, 12, 20, 5, 16, 8 (DE MÍNIMOS)



BORRADO



### Posición

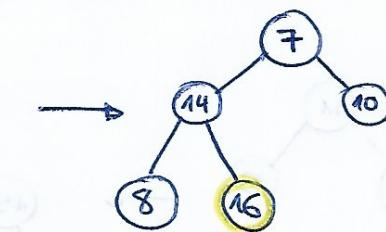
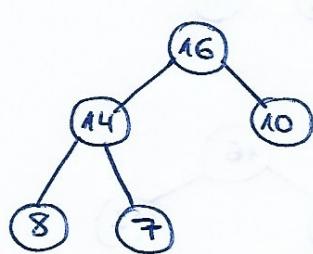
- Para los hijos : Si un nodo ocupa la posición  $i$ 
  - Su hijo izquierdo está en  $2i$
  - Su hijo derecho está en  $2i+1$
- Para el padre : Si un nodo ocupa la posición  $i$ 
  - El padre ocupa la posición  $\lfloor i/2 \rfloor$

## HEAPSORT

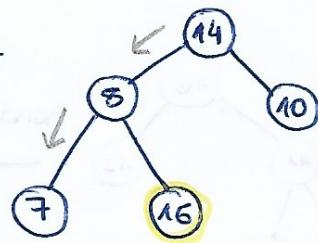
→ Ordenación mediante montículos

### EJEMPLO 8

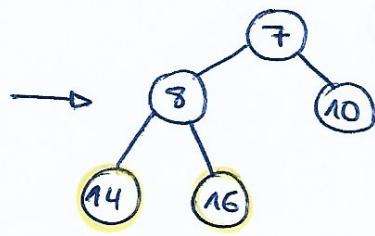
V	16	14	10	8	7
---	----	----	----	---	---



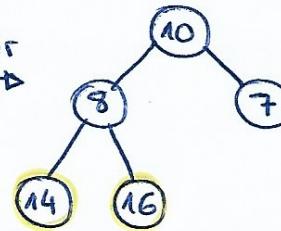
hundir



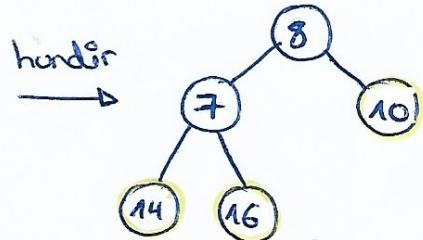
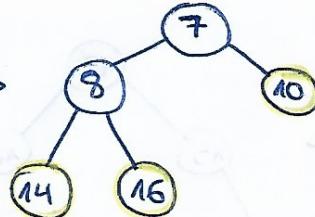
Corro es de máximos  
se mira qué hijo  
es Mayor a la  
hora de hundirlo



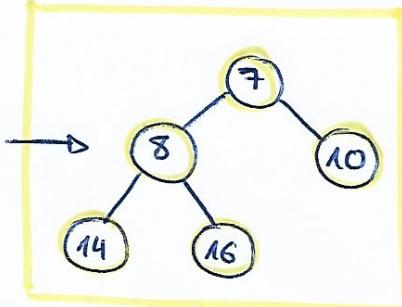
hundir



→



hundir





saboteas a tu propia persona? cómo??  
escríbelo **aquí** y  
táchalo

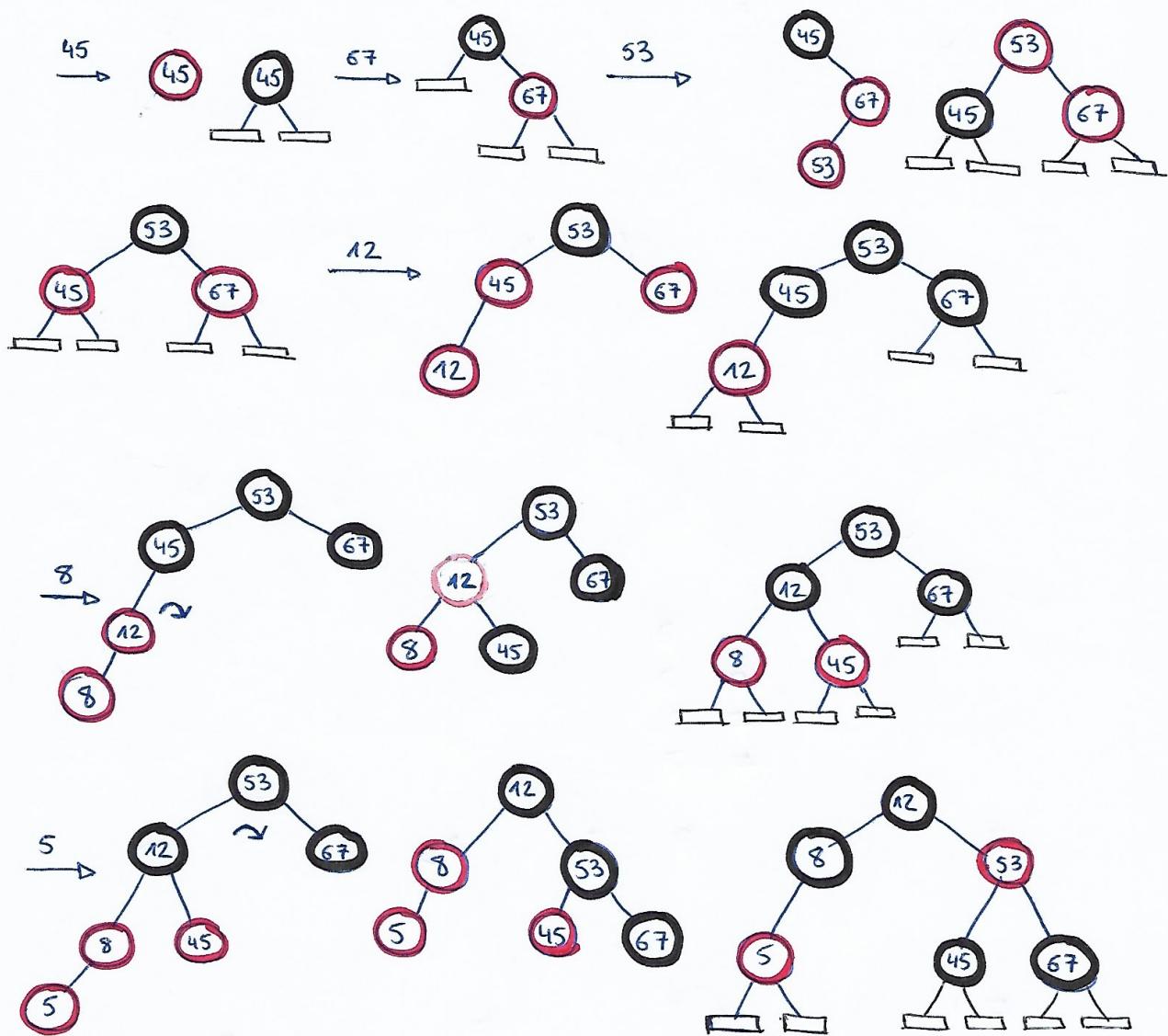
**manual de instrucciones:** escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

EXAMEN 2012

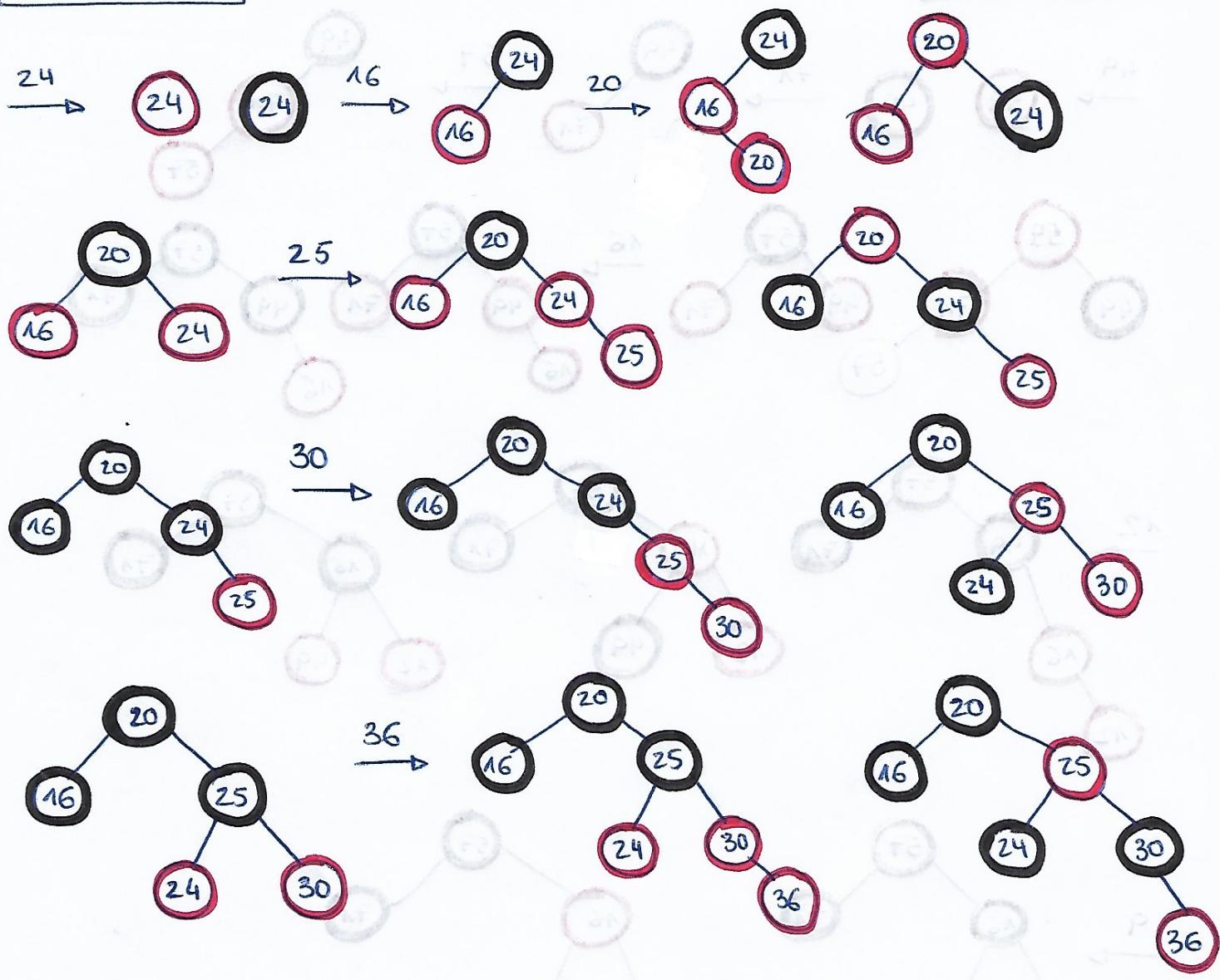
EJERCICIO 3

45, 67, 53, 12, 8, 5



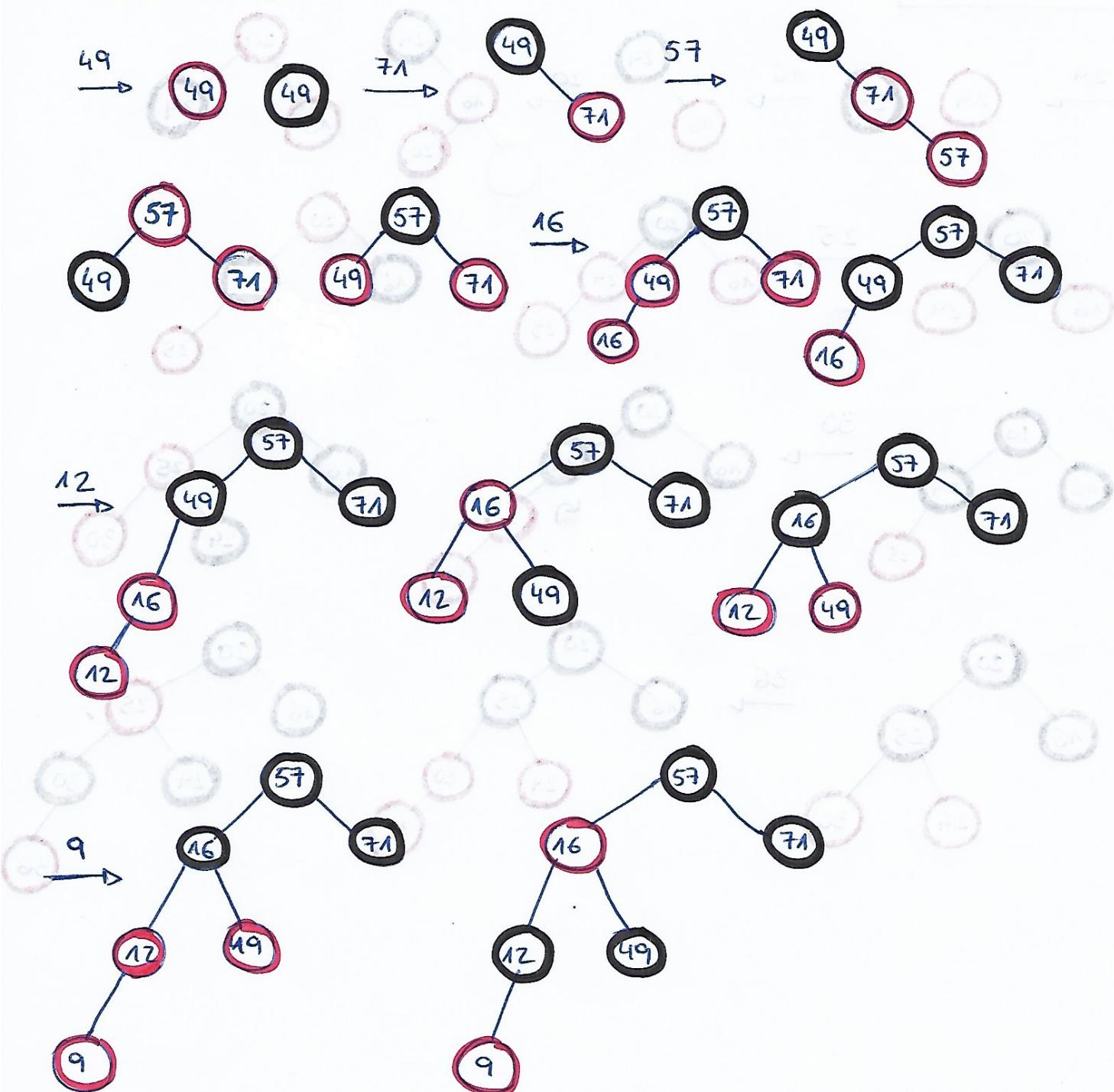
EJERCICIO 3

24, 16, 20, 25, 30, 36



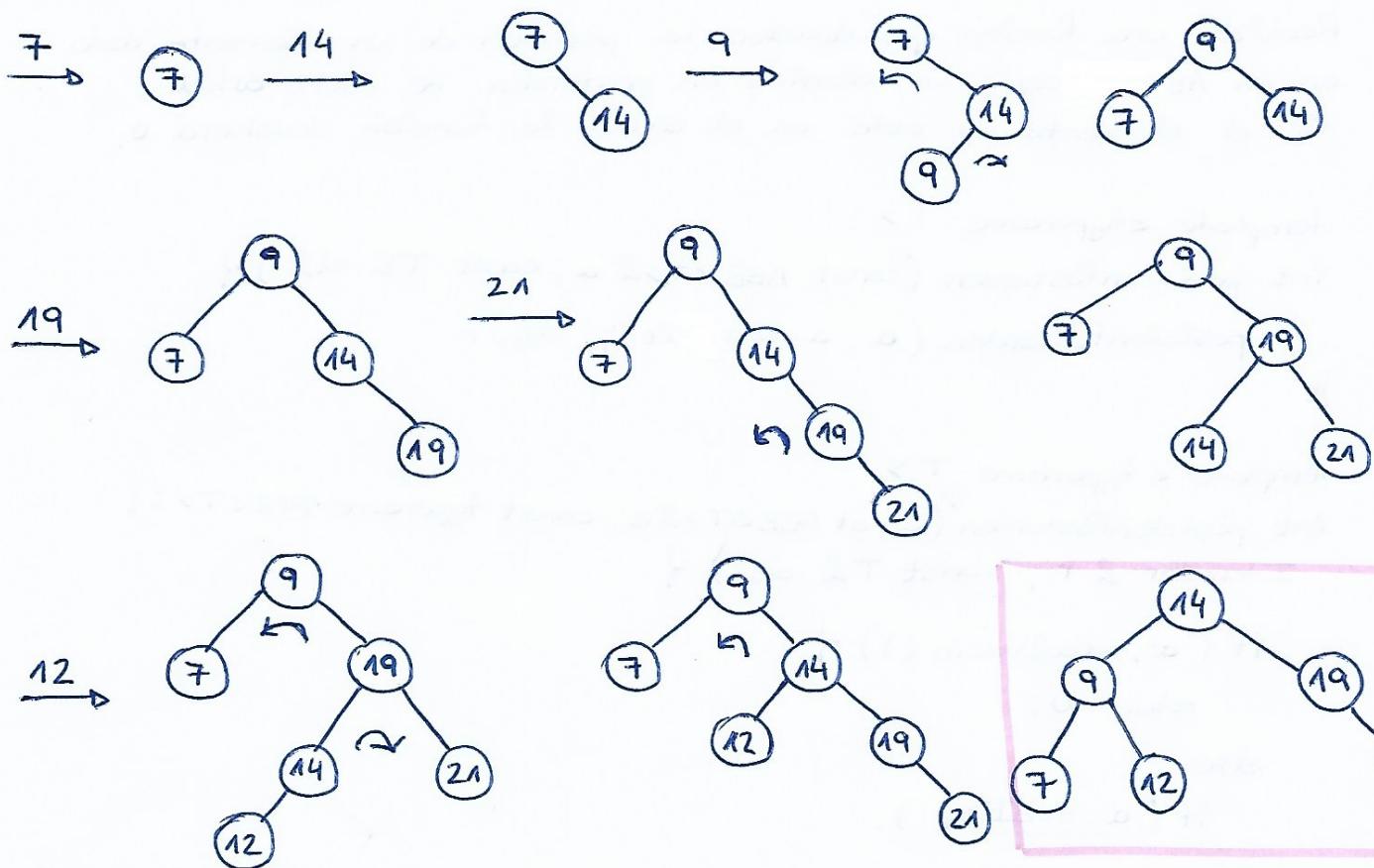
EJERCICIO 3

49, 71, 57, 16, 12, 9



EJERCICIO 2

7, 14, 9, 19, 21, 12 (borrar después 19, 21)



Borrar 19



Borrar 21

