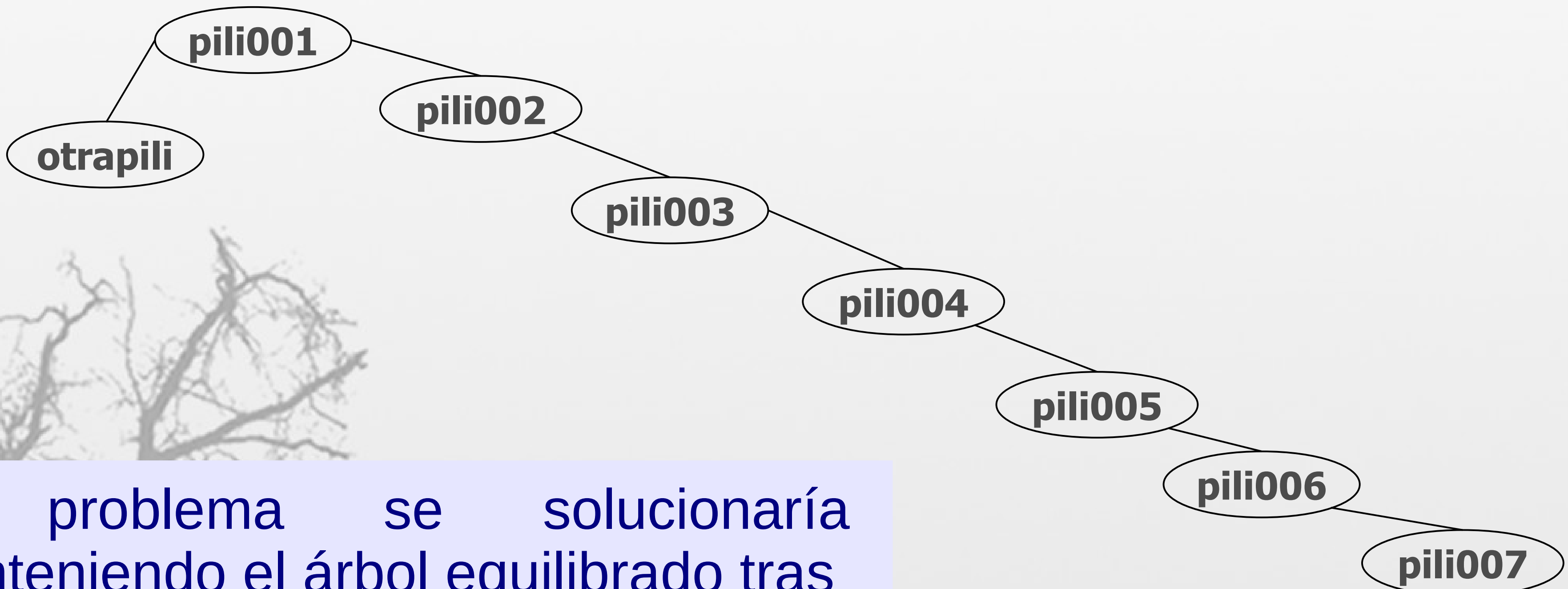
Lección 11: Árboles AVL

- Árboles equilibrados
- Árboles AVL
- Operaciones sobre árboles AVL
  - Inserción
  - Borrado
- Eficiencia de los AVL
- Consideraciones finales

Motivación

Mymail tiene finalmente más de un millón de usuarios de cuentas de correo, y el ABB ha dejado de ser eficiente, sobre todo al realizar inserciones de listados ordenados por alias



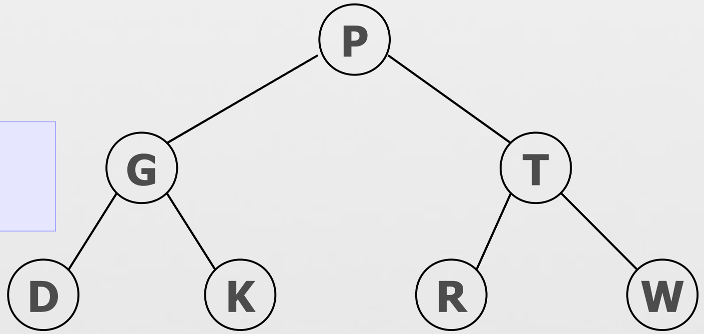
El problema se solucionaría manteniendo el árbol equilibrado tras realizar inserciones/borrados

Motivación

Mantener un árbol equilibrado puede suponer:

- Que cada nodo de los subárboles izquierdo y derecho tengan la misma altura
- Pero esta propiedad es muy rígida, sólo la cumplen los **árboles completos**
- Podemos relajar la condición a que para cada nodo, la diferencia de altura entre subárbol izquierdo y derecho sea de máximo 1

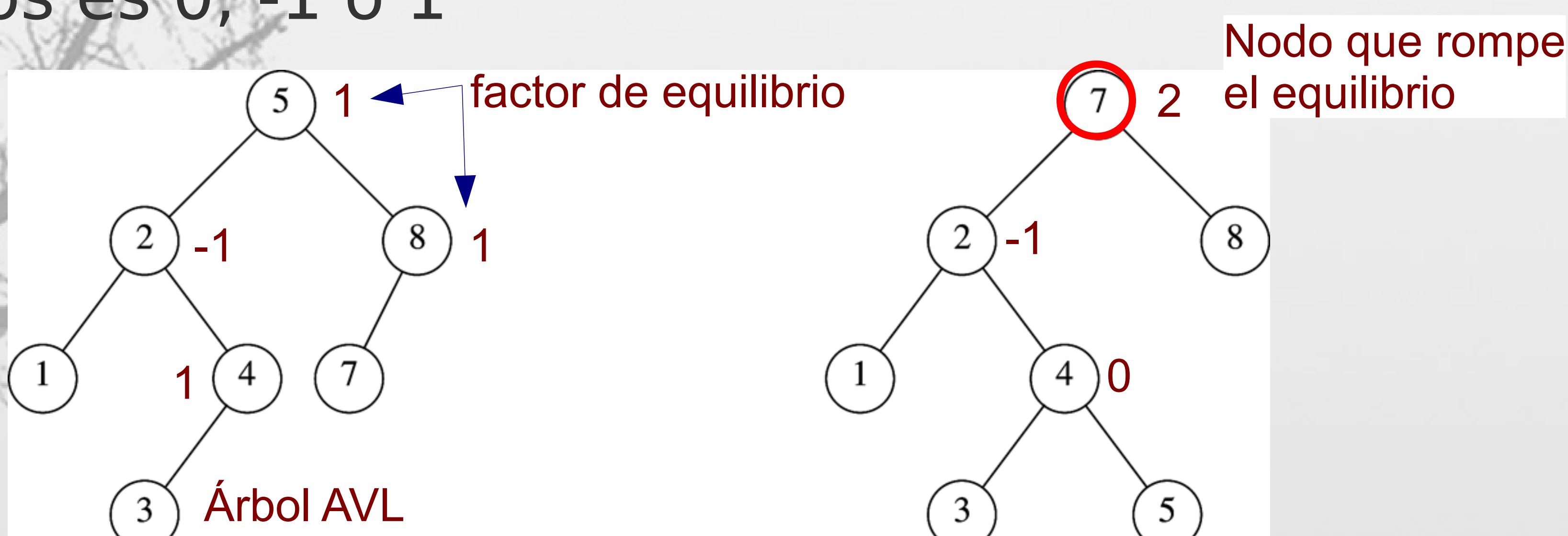
Ejemplo de árbol completo:



Definición de árbol AVL

Un **árbol AVL** es un ABB en el que las alturas de los subárboles izquierdo y derecho *de cada nodo* difieren como máximo en una unidad

- El **factor de equilibrio** de un nodo *a* (o **bal(a)**) es la altura de su subárbol izquierdo menos la altura del derecho: bal(a) = h(izq(a)) - h(der(a))
- En un árbol AVL, el factor de equilibrio de todos los nodos es 0, -1 ó 1



Definición de árbol AVL

Similar al ABB, pero guardando el factor de equilibrio

Factor de equilibrio en cada nodo (un byte)

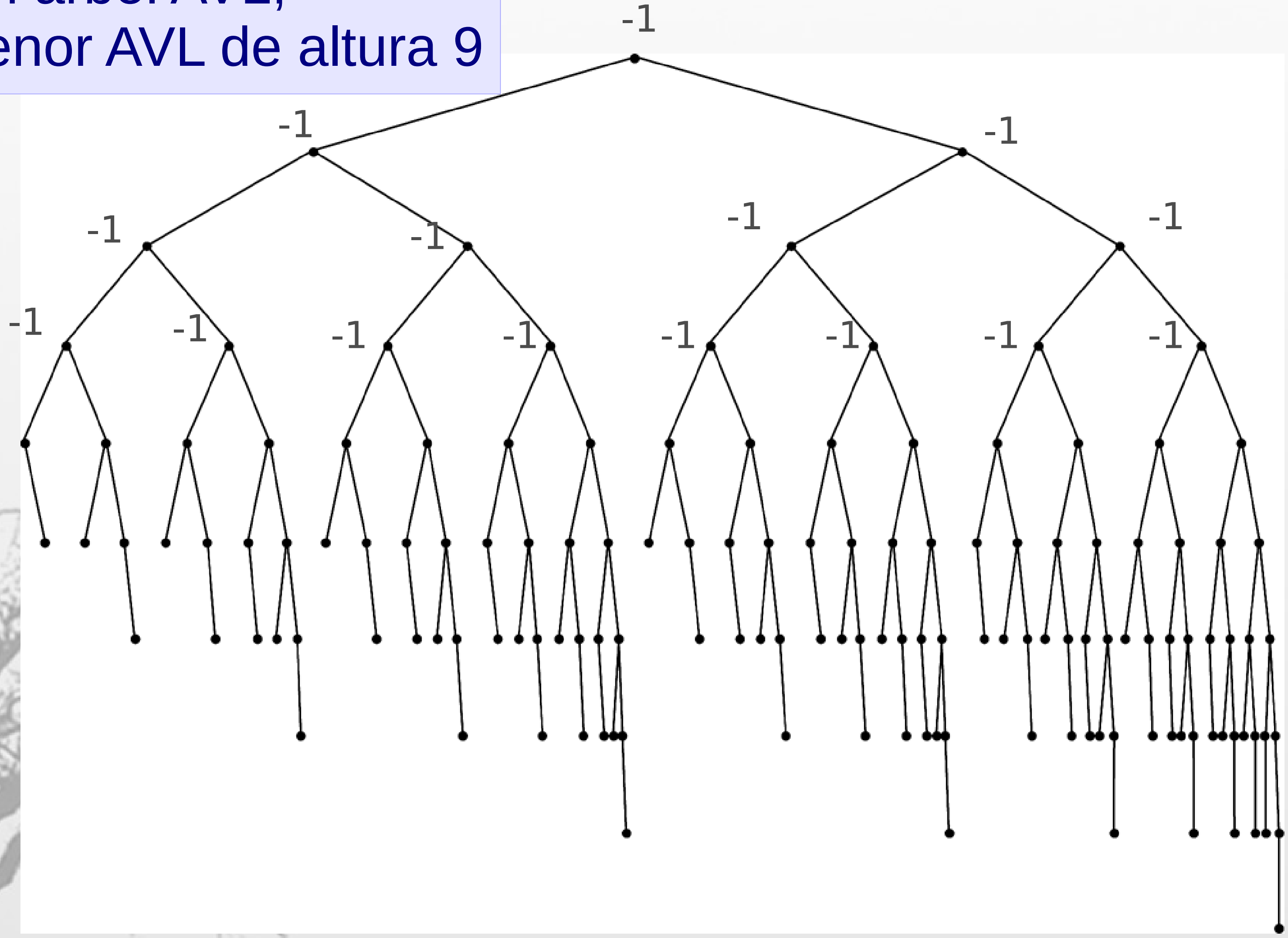
```
template <typename U>
class Nodo {
public:
   Nodo<U> *izq, *der;
   U dato;
   char bal; // -1, 0 , 1 para un árbol avl

   Nodo(U &ele): izq(0), der(0), bal(0), dato(ele) {}
};

template <typename T>
class Avl {
    Nodo<T> *raiz;
private:
    int inserta(Nodo<T>* &c, T &dato);
    void rotDecha(Nodo<T>* &p);
    void rotIzqda(Nodo<T>* &p);
public:
    Avl() { raiz=0; }
    bool inserta(T &dato){ return inserta(raiz,dato); }
…
};
```

Ejemplo:

Es un árbol AVL,
el menor AVL de altura 9



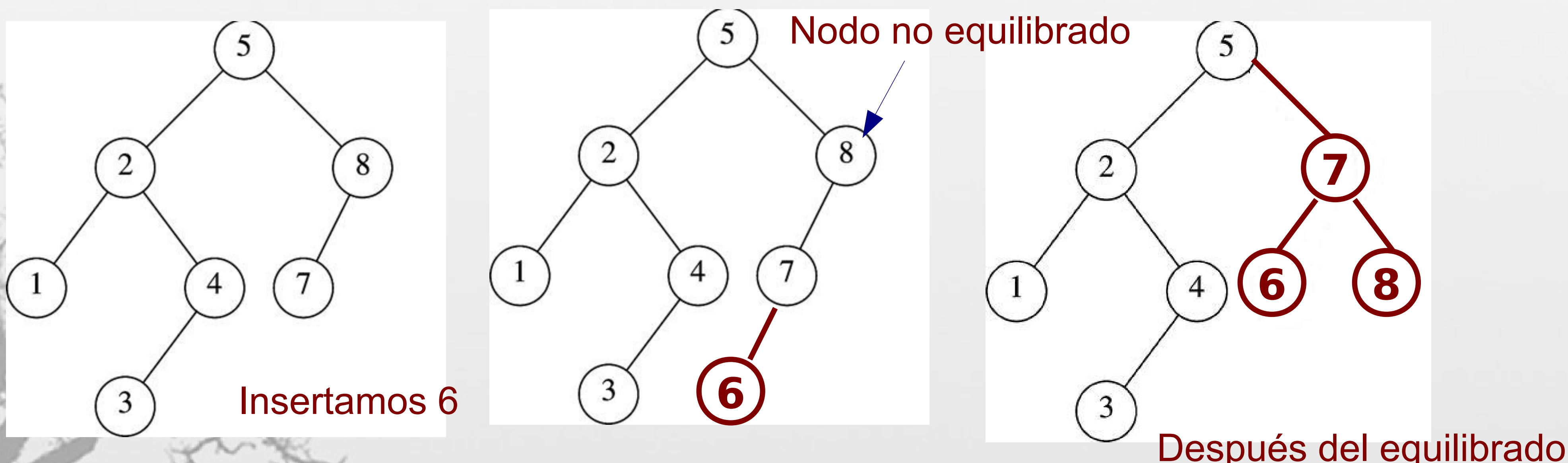
Operaciones en el AVL

Las operaciones de un árbol AVL son similares a las del ABB, con ciertas diferencias:

- **Buscar**: similar al ABB porque no modifica el árbol.
- **Recorridos**: Inorden, postorden y preorden son similares.
- **Insertar**: inserta con el ABB y equilibra el árbol en caso de que afecte el factor de equilibrio → lo vemos a continuación
- **Borrar**: implica reunificar y reestructurar el árbol, su implementación es más compleja

Insertar en un AVL

- La inserción en un AVL es igual que en un ABB, pero si el nuevo nodo genera un desequilibrio, hay que reestructurar el árbol mediante rotaciones
- Este desequilibrio sólo puede ocurrir en el camino desde el nodo insertado hasta la raíz

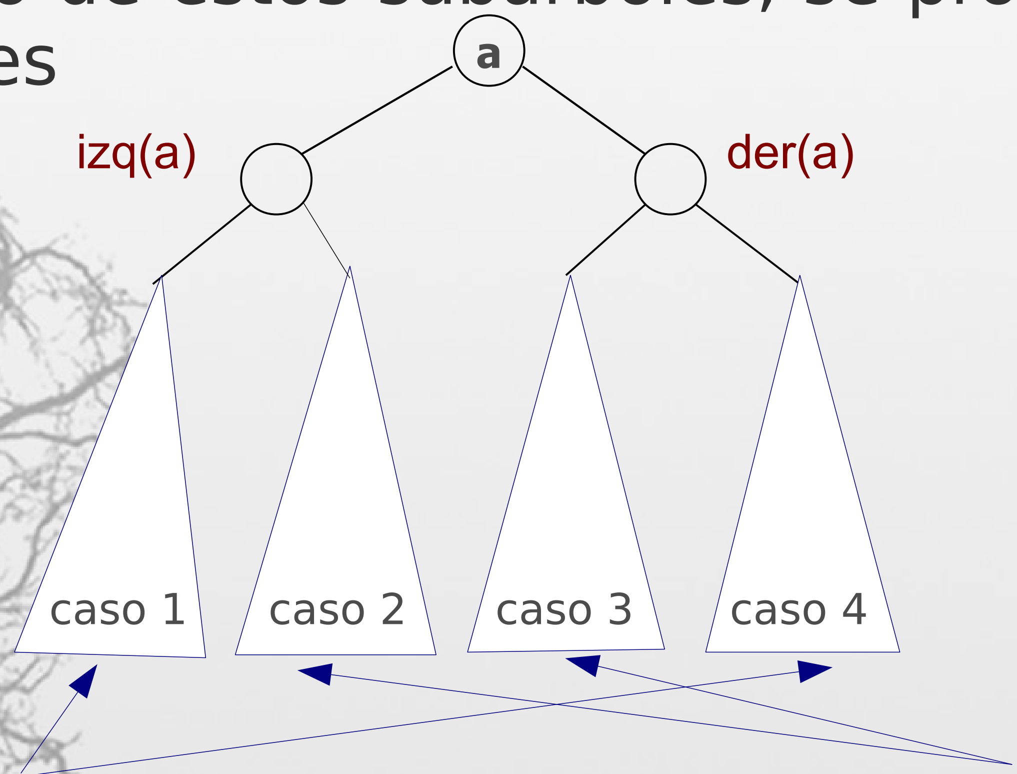


Rotaciones

Los desequilibrios se solucionan mediante **rotaciones**, que pueden ser de 4 tipos:

- Si *a* sufre un desequilibrio tras una inserción en cada uno de estos subárboles, se producen 4 casos diferentes

Casos simétricos:
1 y 4
2 y 3



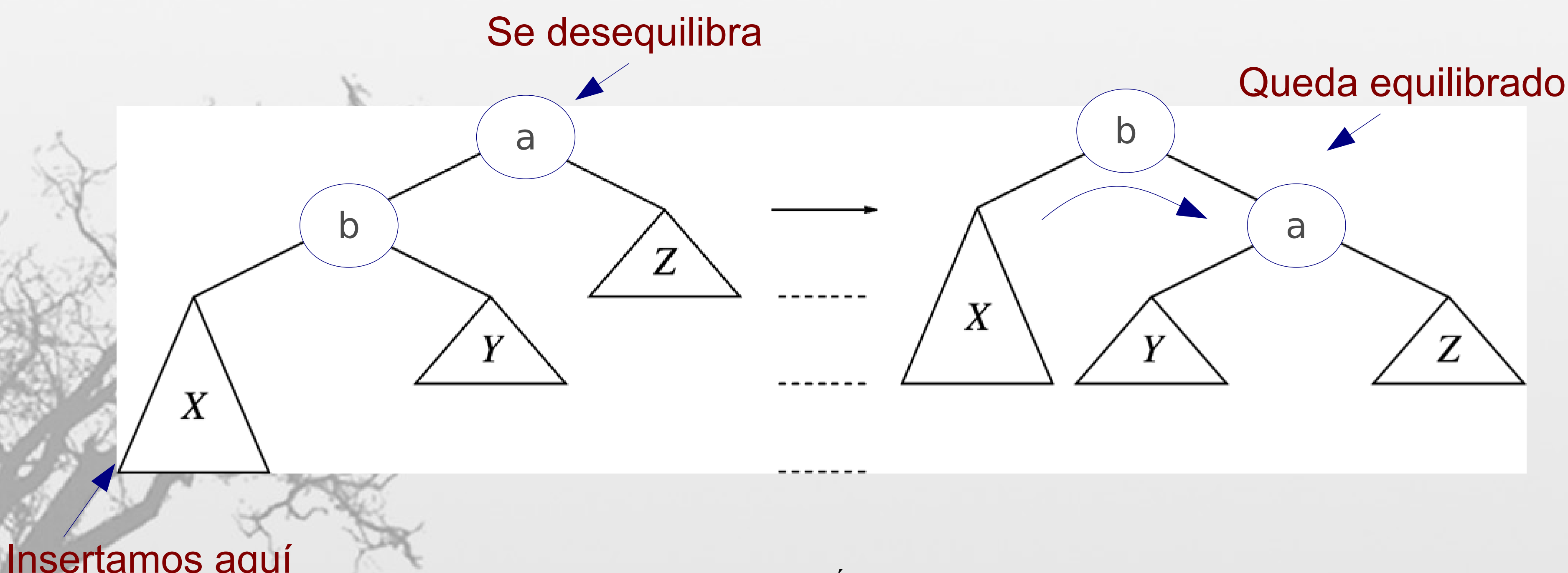
Rotaciones simples

Rotaciones dobles

Rotaciones simples:

El **caso 1** se produce cuando hay un desequilibrio en *a* tras insertar en el hijo izquierdo de su hijo izquierdo: bal(a)= 2, bal(izq(a)) = 1

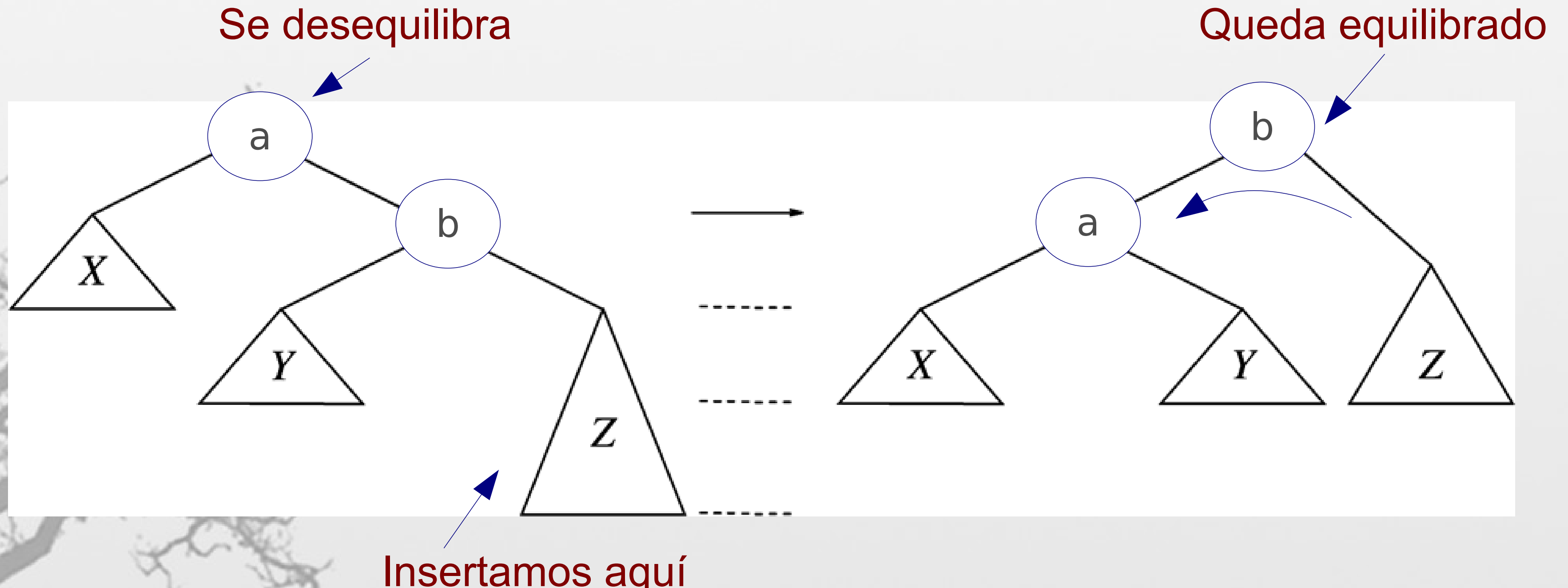
- En estos casos de hace un **giro o rotación simple a la derecha**



Rotaciones simples:

El **caso 4** se produce cuando hay un desequilibrio en *a* tras insertar en el hijo derecho de su hijo derecho: bal(a) = -2, bal(der(a)) = -1

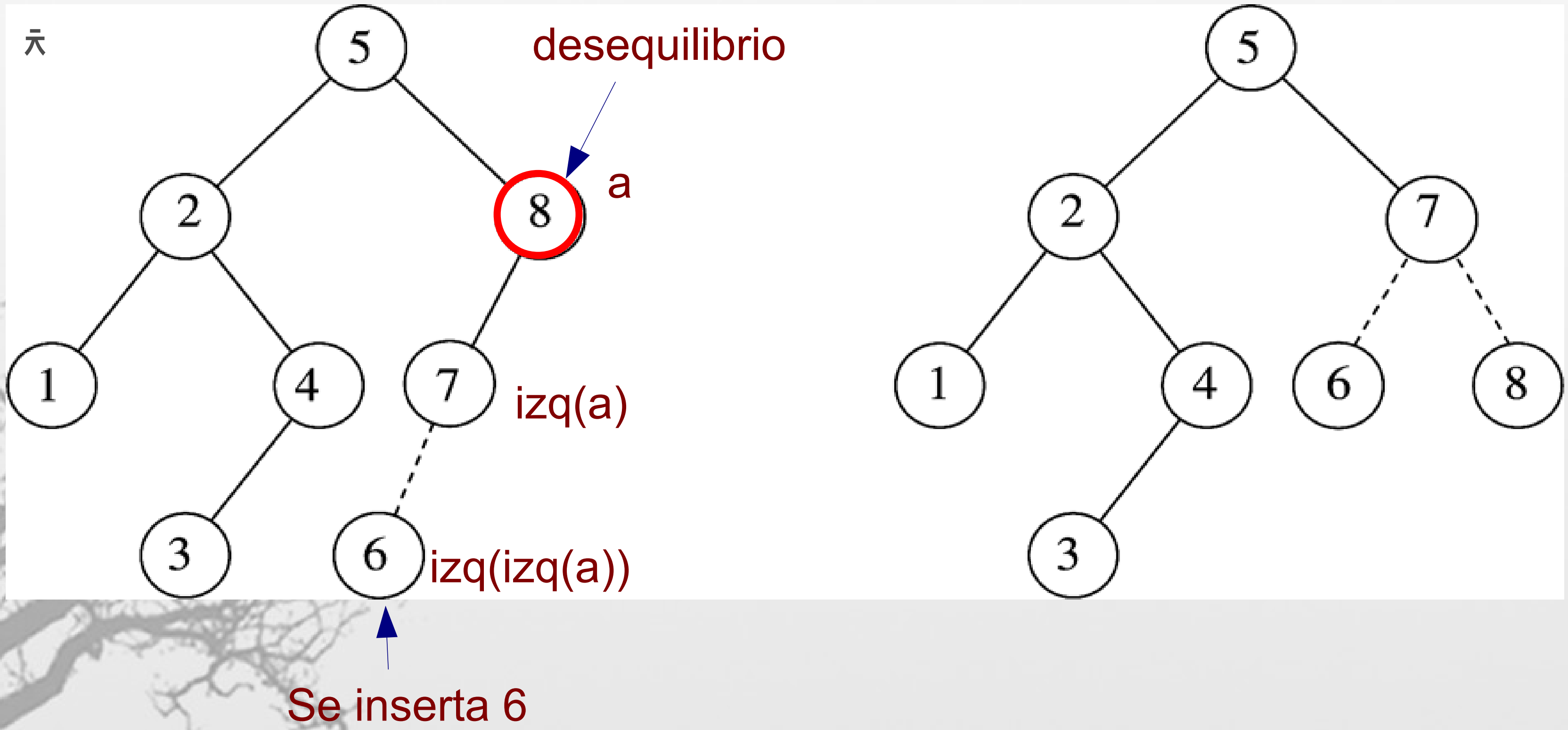
- En estos casos de hace un giro simple a la izquierda



Rotaciones simples:

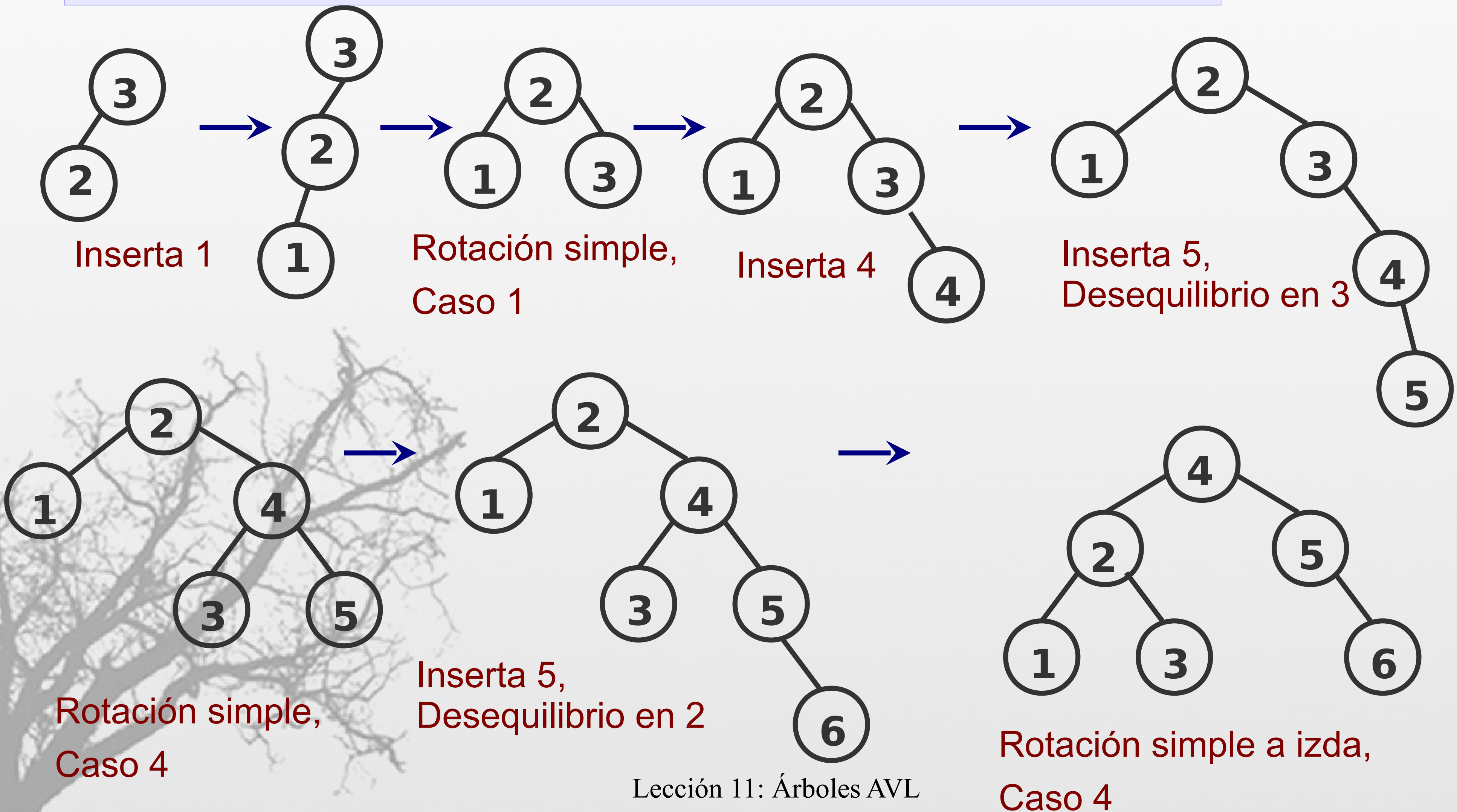
Ejemplo de caso 1:
El nodo 8 queda desequilibrado tras insertar el nodo 6: caso 1

Después la rotación simple a derechas



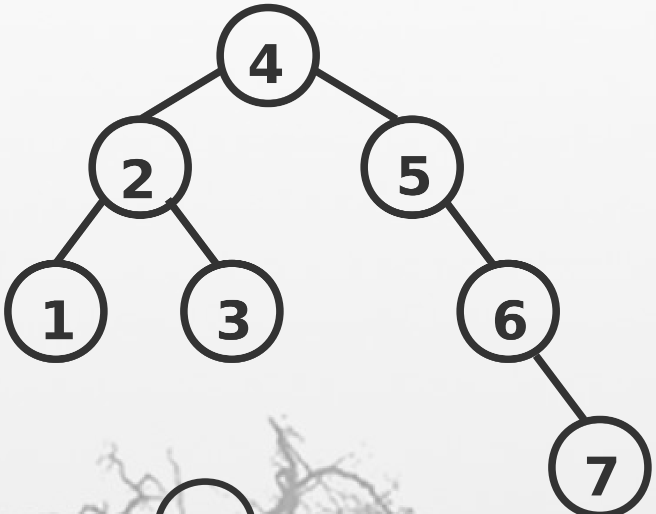
Rotaciones simples:

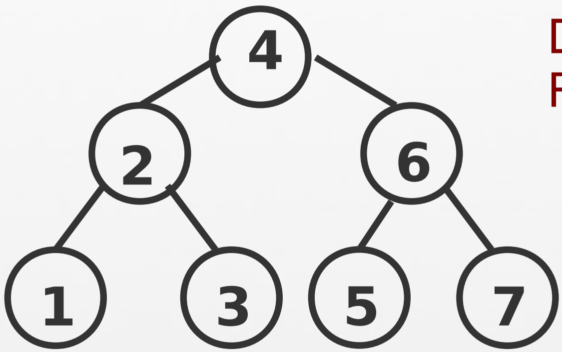
Ejemplo de inserción consecutiva de: 3, 2, 1, 4, 5 y 6



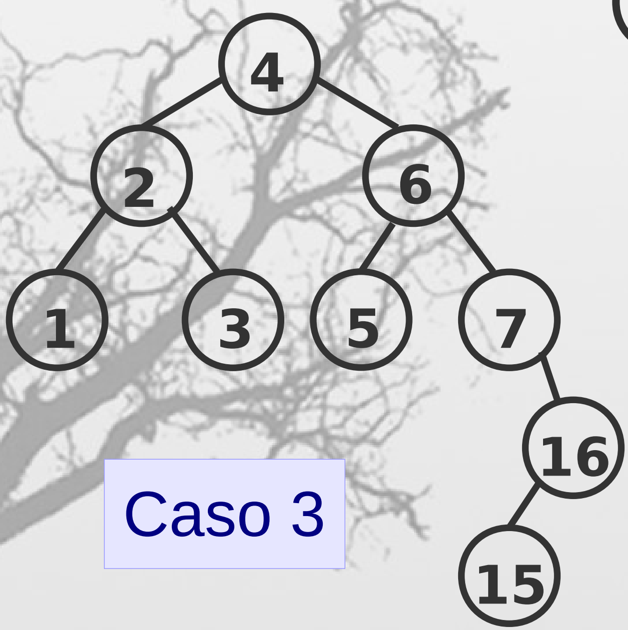
Rotaciones simples:

Seguimos insertando: 7, 16 y 15





Inserta 7
Desequilibrio en 5
Rotación a izdas.



Caso 3

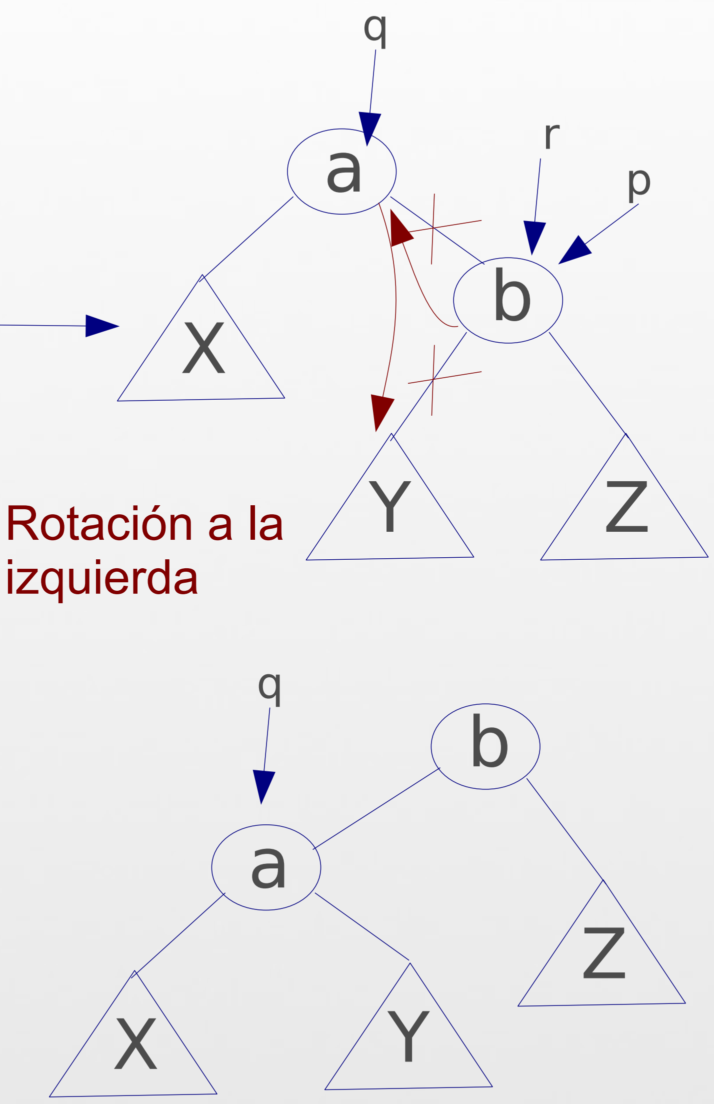
Inserta 16, OK
Inserta 15, desequilibrio en 7

Inserción en el hijo izquierdo del hijo derecho de 7 ,
¡Es un caso 3: no se soluciona con un giro simple!

Rotaciones simples:

```
template<typename T>
void Avl<T>::rotIzqda(Nodo<T>* &p){
    Nodo<T> *q = p, *r;
    p = r = q->der;
    q->der = r->izq;
    r->izq = q;
    q->bal++;
    if (r->bal < 0) q->bal += -r->bal;
    r->bal++;
    if(q->bal > 0) r->bal += q->bal;
}

template<typename T>
void Avl<T>::rotDecha(Nodo<T>* &p){
    Nodo<T> *q = p, *l;
    p = l = q->izq;
    q->izq = l->der;
    l->der = q;
    q->bal--;
    if (l->bal > 0) q->bal -= l->bal;
    l->bal--;
    if(q->bal < 0) l->bal -= -q->bal;
}
```



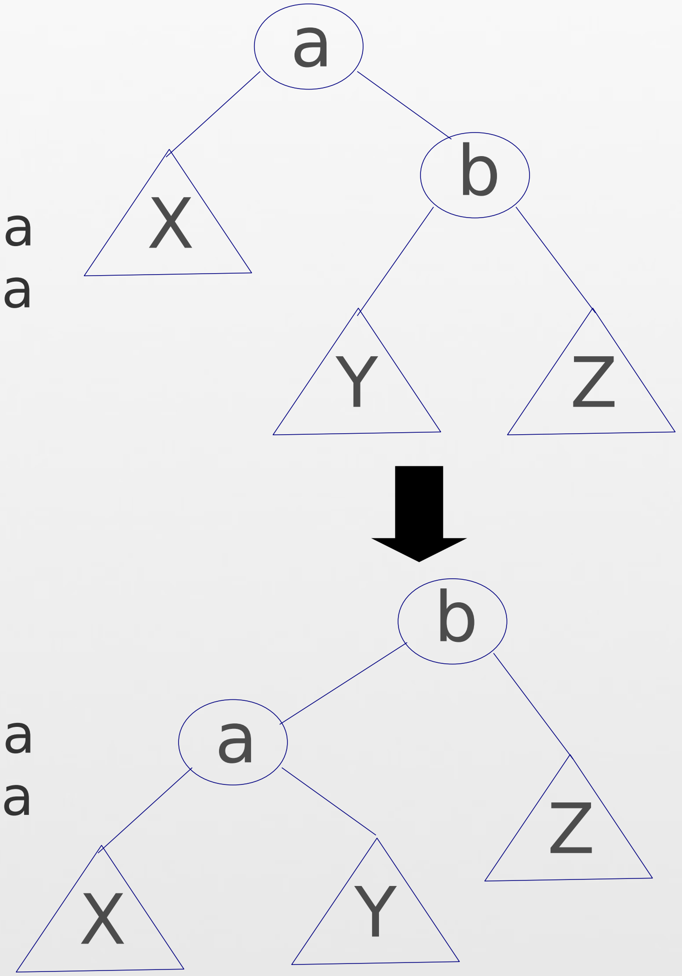
Reajuste de factor de equilibrio

1. *a* pierde una altura por la izq → ++bal(a)
2. *a* pierde el subárbol Z por la izq

- Si h(Z) < h(Y) (bal(b) > 0) no afecta
- Si h(Z) > h(Y) (bal(b) < 0) entonces *a* ha perdido altura por la izq. Hay que sumar la diferencia h(Z) – h(Y) → bal(a) += -bal(b)

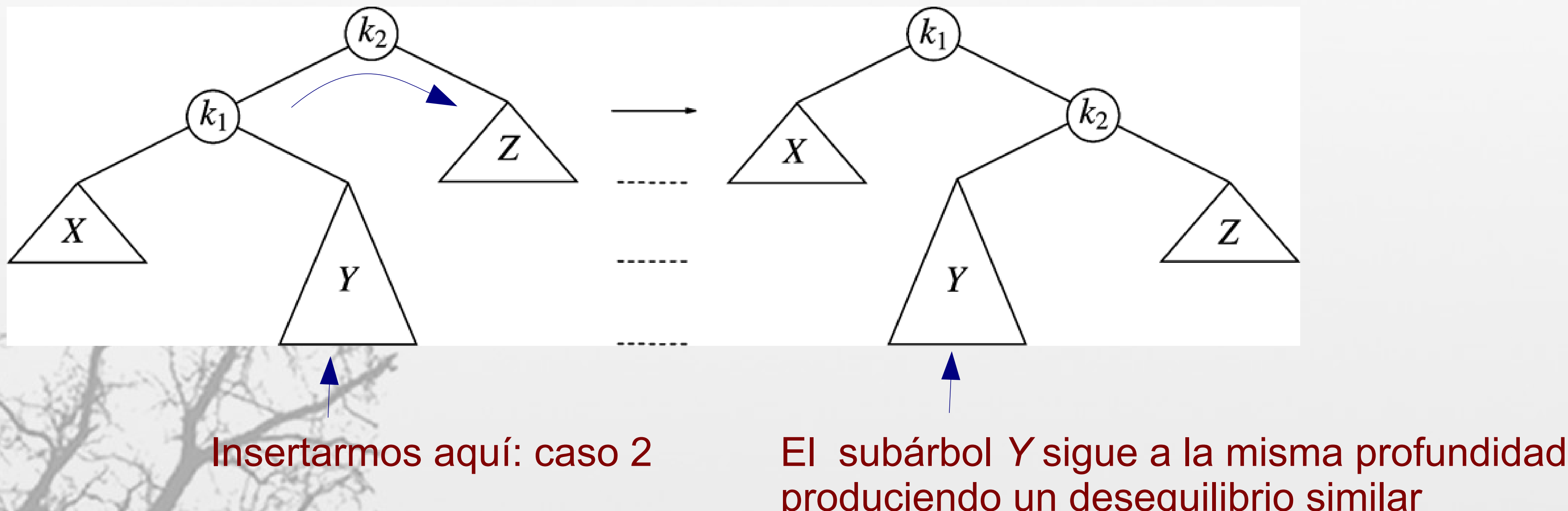
3. *b* gana una altura por la izq → ++bal(b)
4. *b* tiene ahora el subárbol X por la izq

- Si h(X) < h(Y) (bal(a) < 0) no afecta
- Si h(X) > h(Y) (bal(a) > 0) entonces *b* ha ganado altura por la izq. Hay que sumar la diferencia h(X) – h(Y) → bal(b) += bal(a)



Rotaciones dobles

Los **casos 2 y 3** no se solucionan con rotaciones simples porque tras la rotación los desequlibrios permanecen:

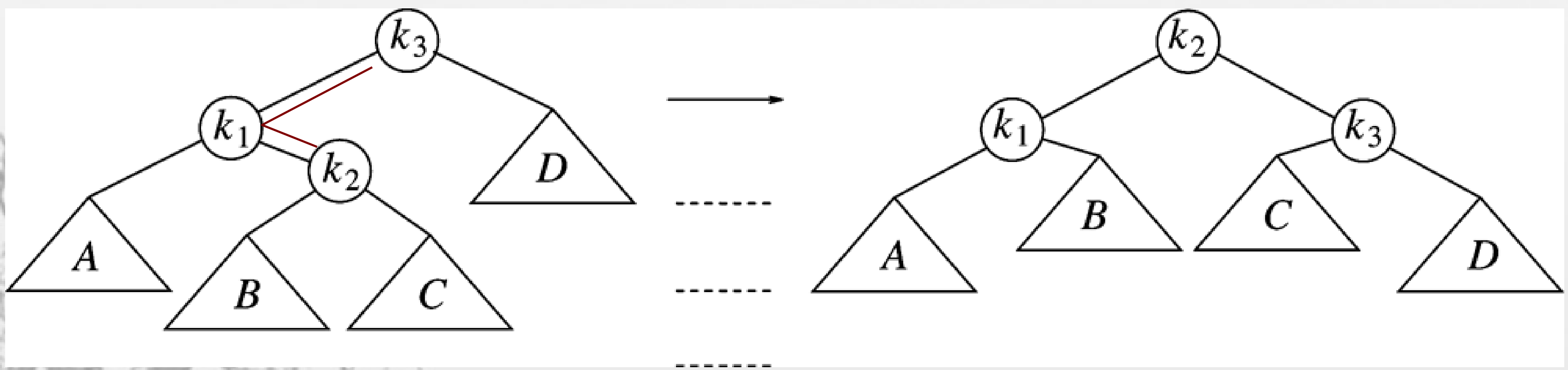


La solución es realizar dos rotaciones simples

Rotaciones dobles: caso 2

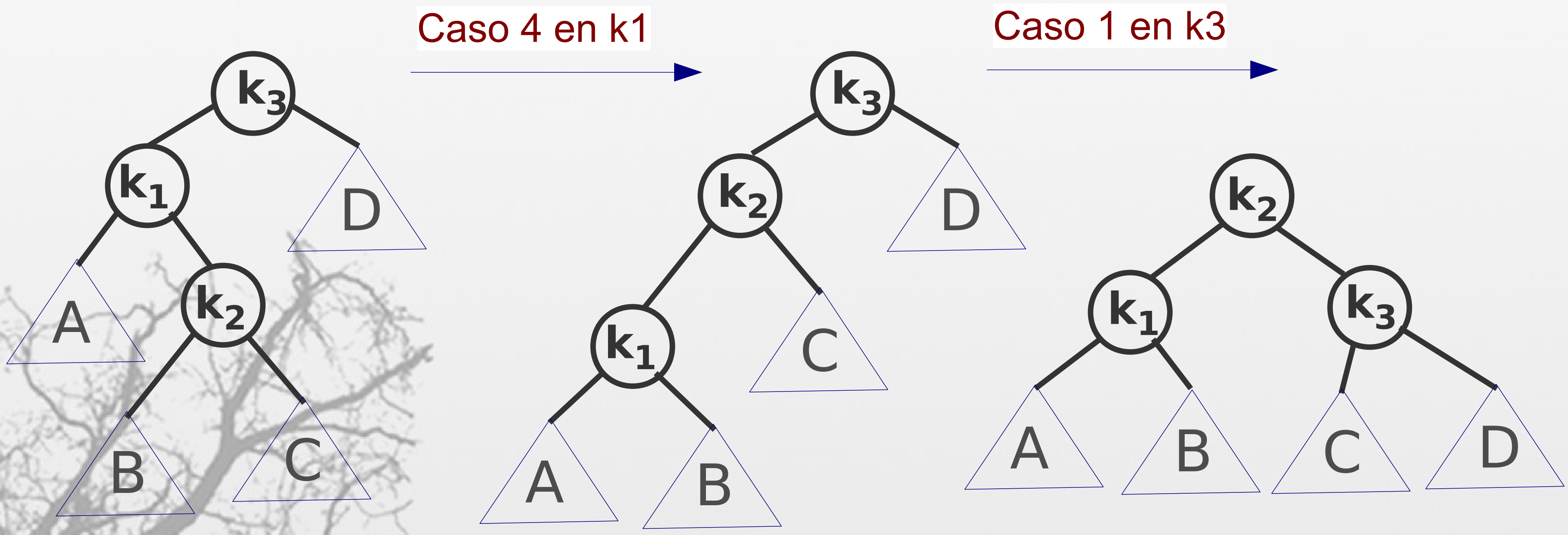
El caso 2 se produce tras insertar en los subárboles B o C (o el propio k2), generándose desequilibrios en el nodo k3: bal(k3) = 2, bal(k1) = -1

- La solución se produce poniendo a k2 de raíz



Rotaciones dobles: caso 2

Para ello se hace una rotación simple a la izquierda en k1, seguida de una rotación a la derecha en k3

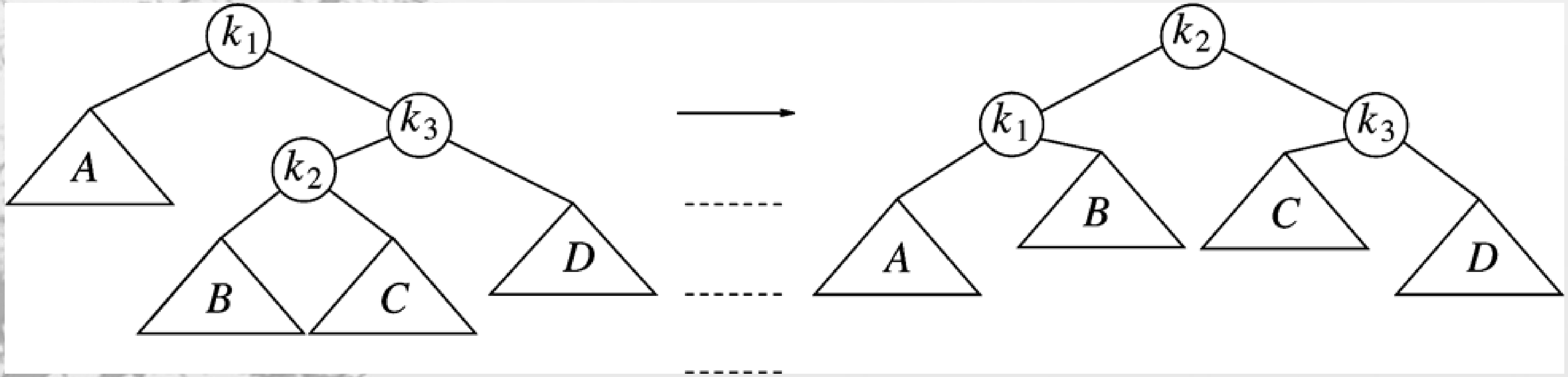


Caso 2 = caso 4 + caso 1

Rotaciones dobles: caso 3

El caso 3 es simétrico al 2, se produce tras insertar en los subárboles B o C (o el propio k2), generándose desequilibrios en el nodo k1: bal(k1)=-2, bal(k3)=1

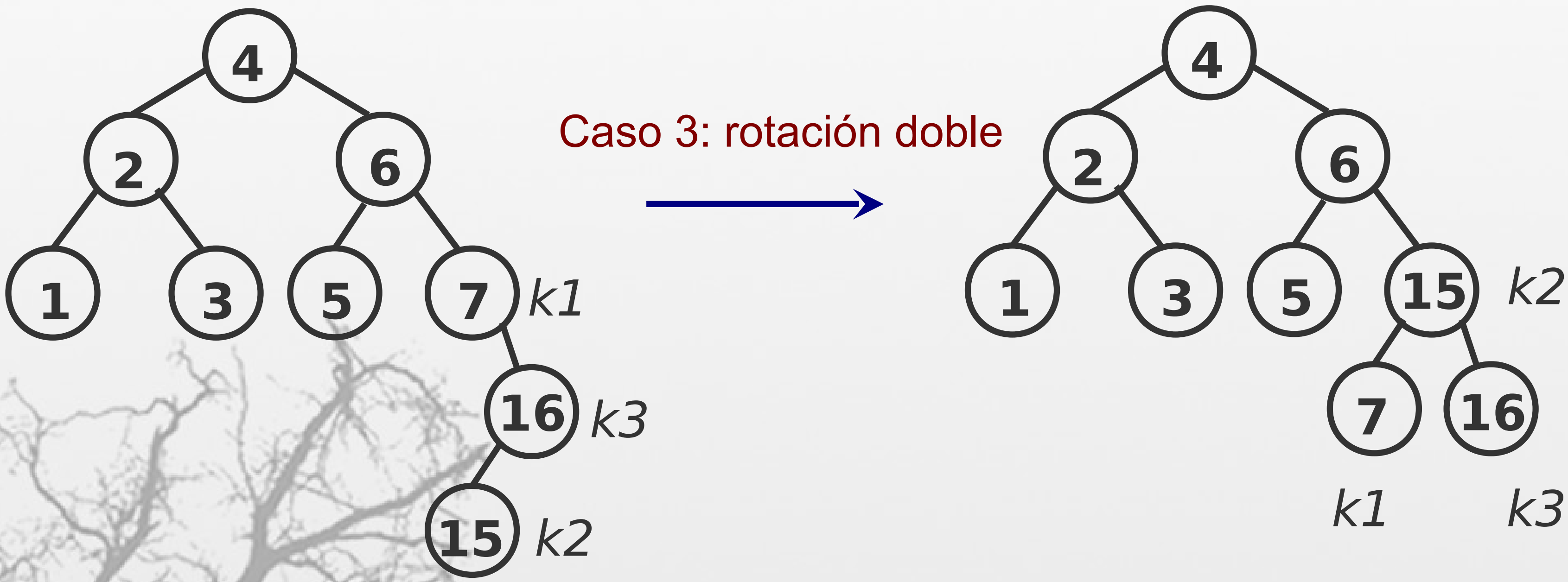
- La solución se produce poniendo a k2 de raíz, para ello se hace una rotación simple a la derecha en el nodo k3, seguido por una rotación simple a la izquierda en el nodo k2



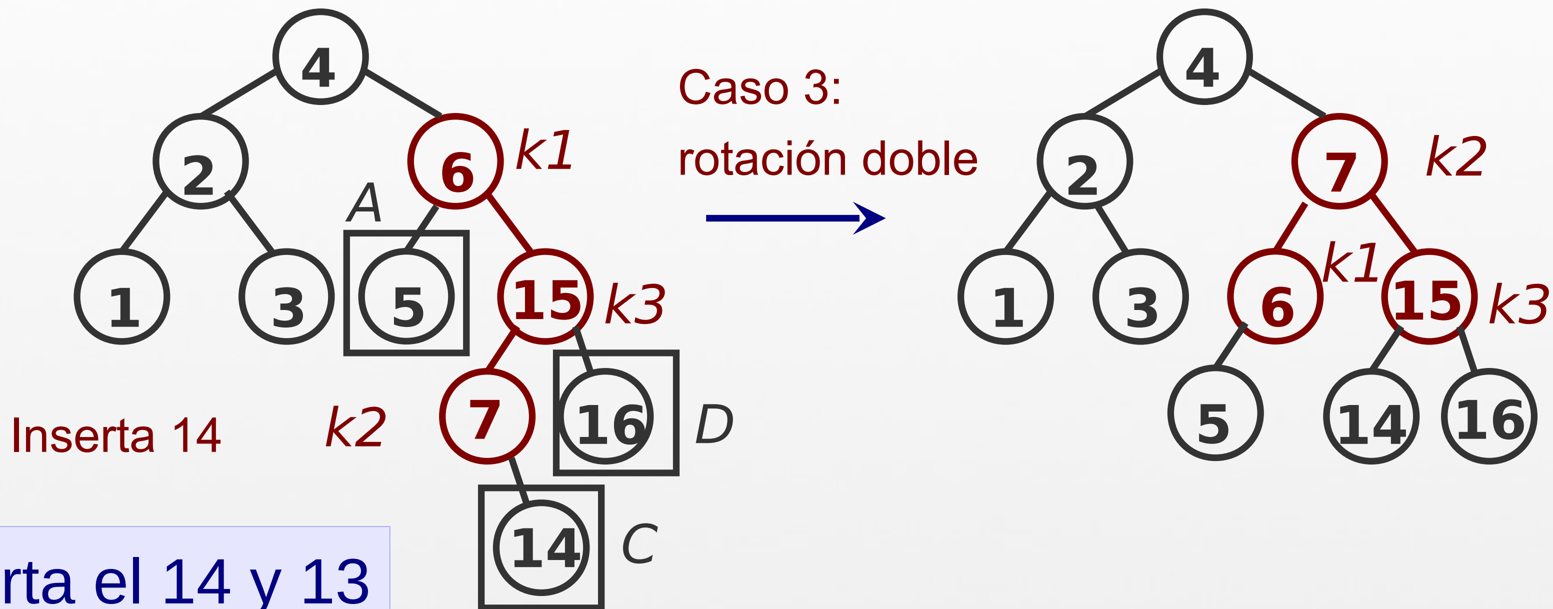
Caso 3 = caso 1 + caso 4

Ejemplo de rotaciones

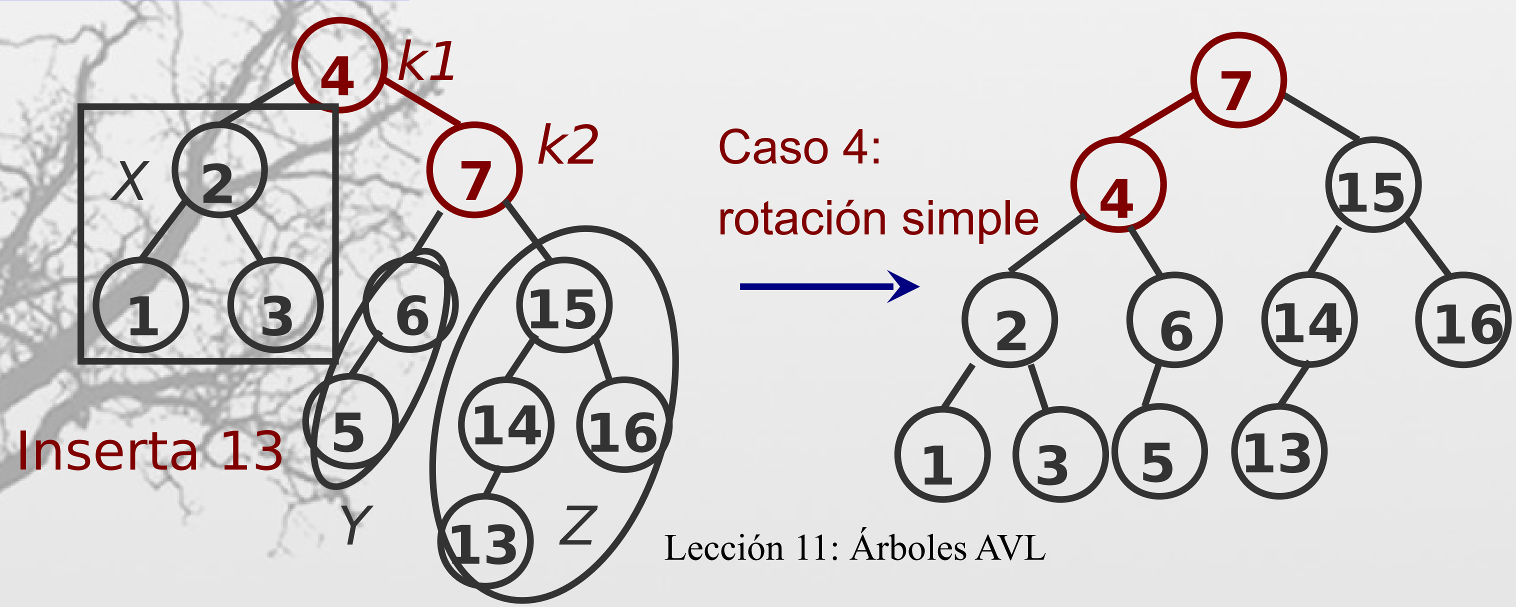
Retomamos el ejemplo que inserta el 15 después del 16



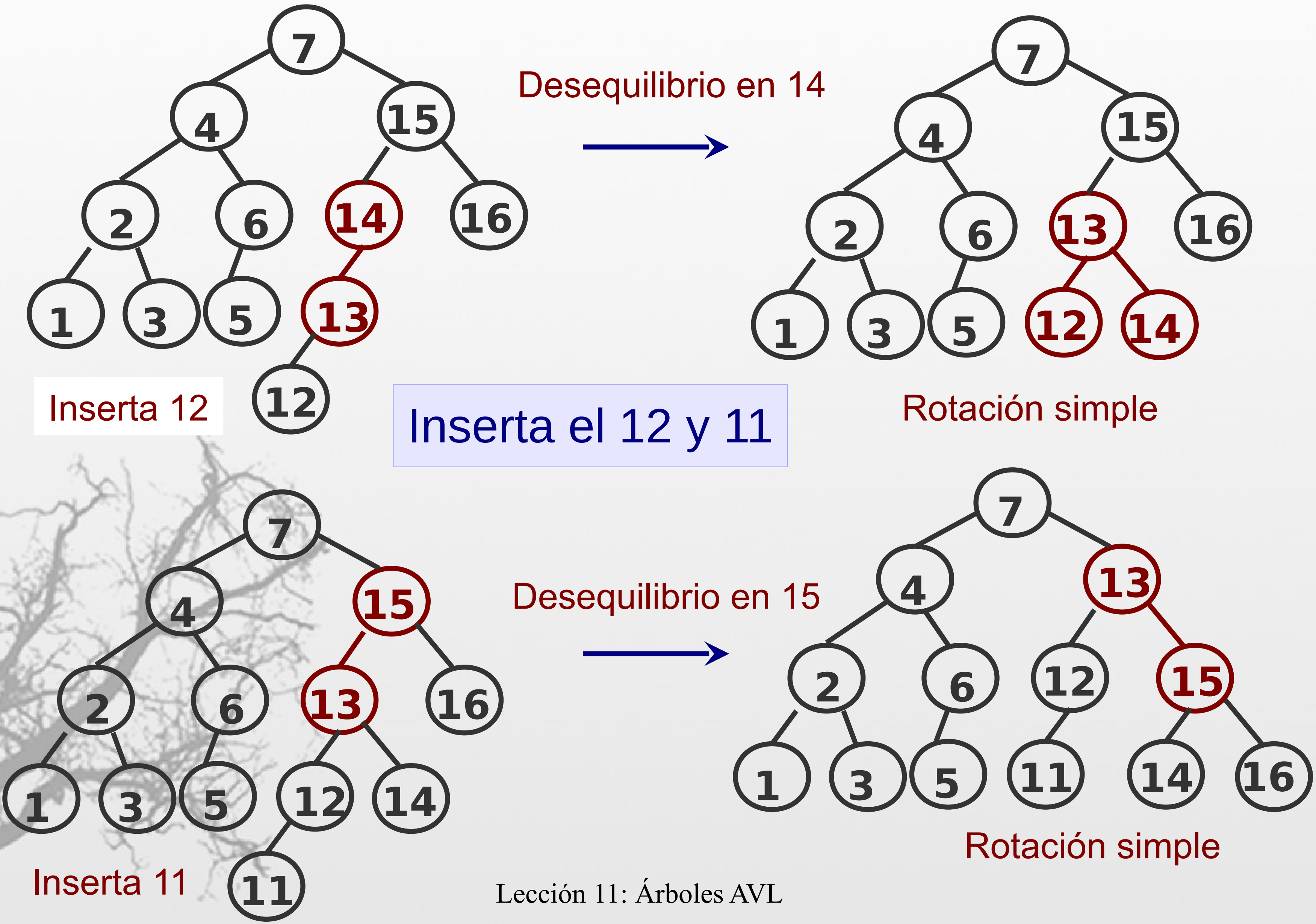
Ejemplo de rotaciones



Inserta el 14 y 13



Ejemplo de rotaciones



Código de inserción

```
template<typename T>
int Avl<T>::inserta(Nodo<T>* &c, T &dato){
    Nodo<T> *p = c;
    int deltaH = 0;
    if (!p){
        p = new Nodo<T>(dato);
        c = p; deltaH=1;
    }
    else if (dato > p->dato){
        if (inserta(p->der, dato)){
            p->bal--;
            if (p->bal == -1) deltaH=1;
            else if (p->bal == -2) {
                if (p->der->bal == 1) rotDecha(p->der);
                rotIzqda(c);
    }  }   }
    else if (dato < p->dato){
        if (inserta(p->izq, dato)){
            p->bal++;
            if (p->bal==1) deltaH = 1;
            else if (p->bal == 2){
                if (p->izq->bal == -1) rotIzqda(p->izq);
                rotDecha(c);
    }   }   }
    return deltaH;
}
```

La recursión llega a una hoja,
Inserción similar al ABB

Rotación doble:caso 3

Rotación simple: caso 4

Rotación doble:caso 2

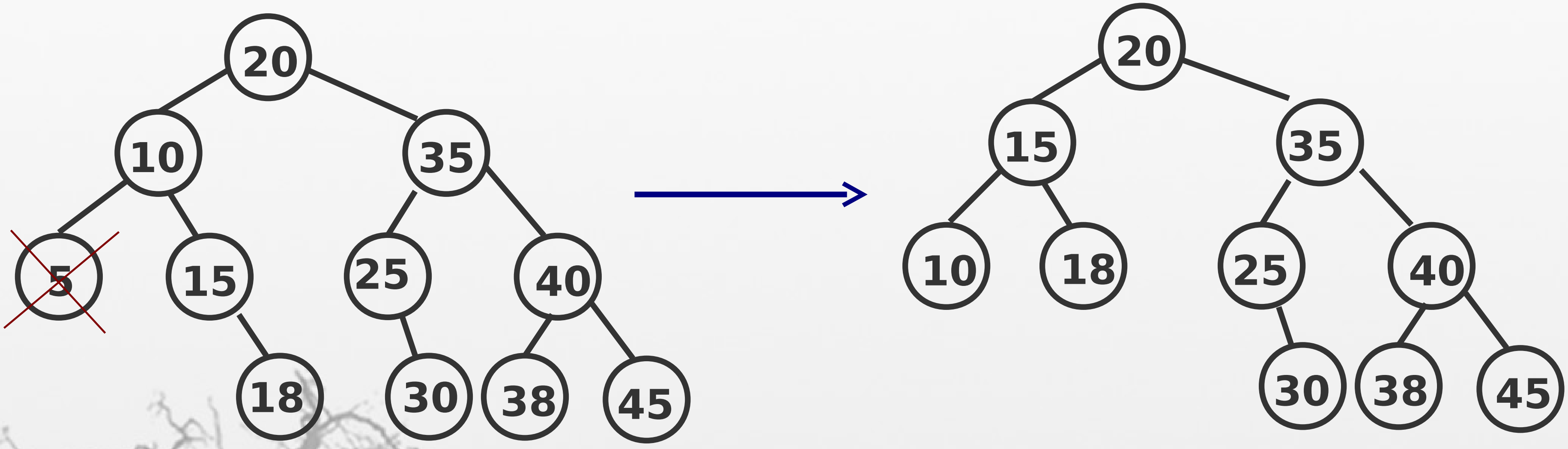
Rotación simple: caso 1

Borrar un nodo en los AVL

- Borrar un nodo *x* como en un ABB, recordar que el nodo eliminado finalmente es una hoja
- Trazar el camino desde la hoja a eliminar hasta la raíz
- Para cada nodo *x* encontrado en el camino, comprobar si las altura de sus hijos izquierdo y derecho difieren en máximo uno
  - Si es así, ir hacia el padre
  - Si no, realizar las pertinentes rotaciones

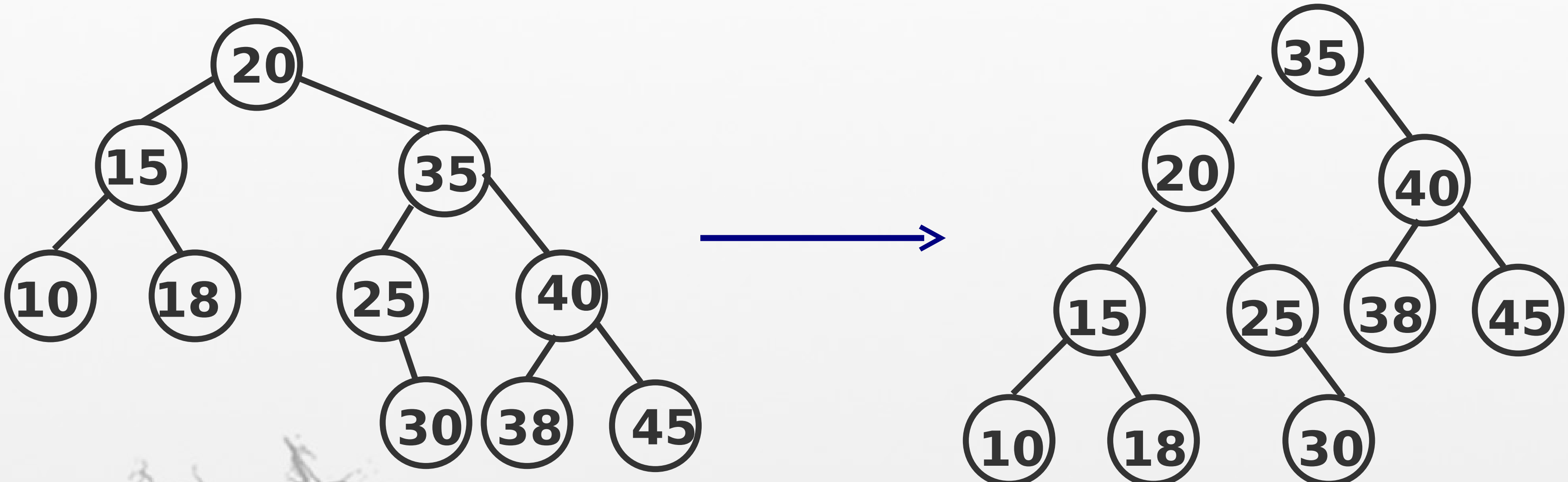
Borrar un nodo en los AVL

Eliminar 5



Al borrar 5, el nodo 10 queda desbalanceado,
Se hace un giro simple a la izquierda

Borrar un nodo en los AVL

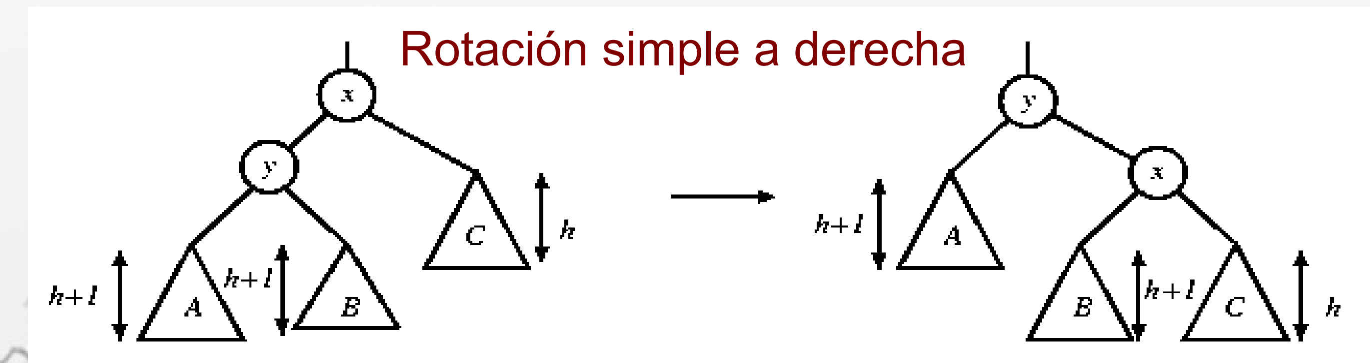


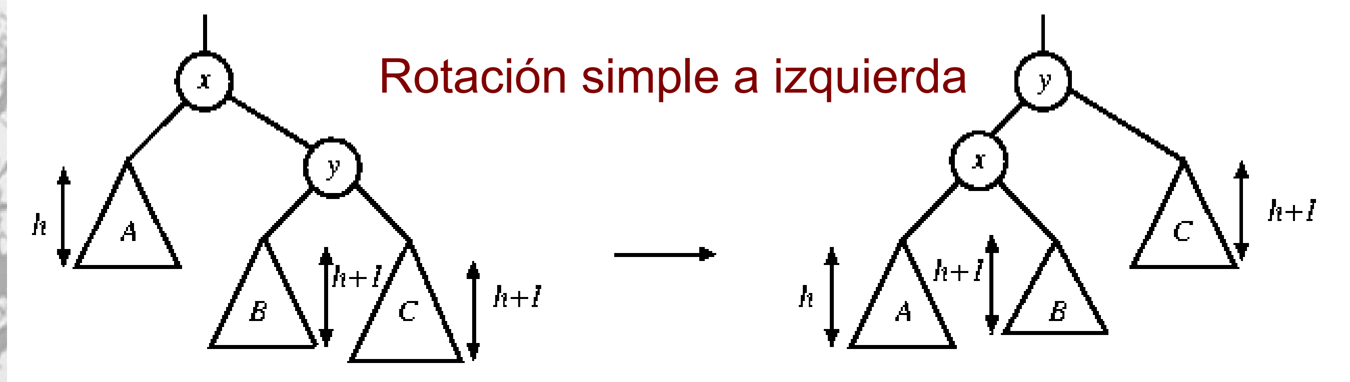
Se comprueba el nodo padre: 20 está desequilibrado,

Se hace una rotación simple a la izquierda

Borrar un nodo en los AVL

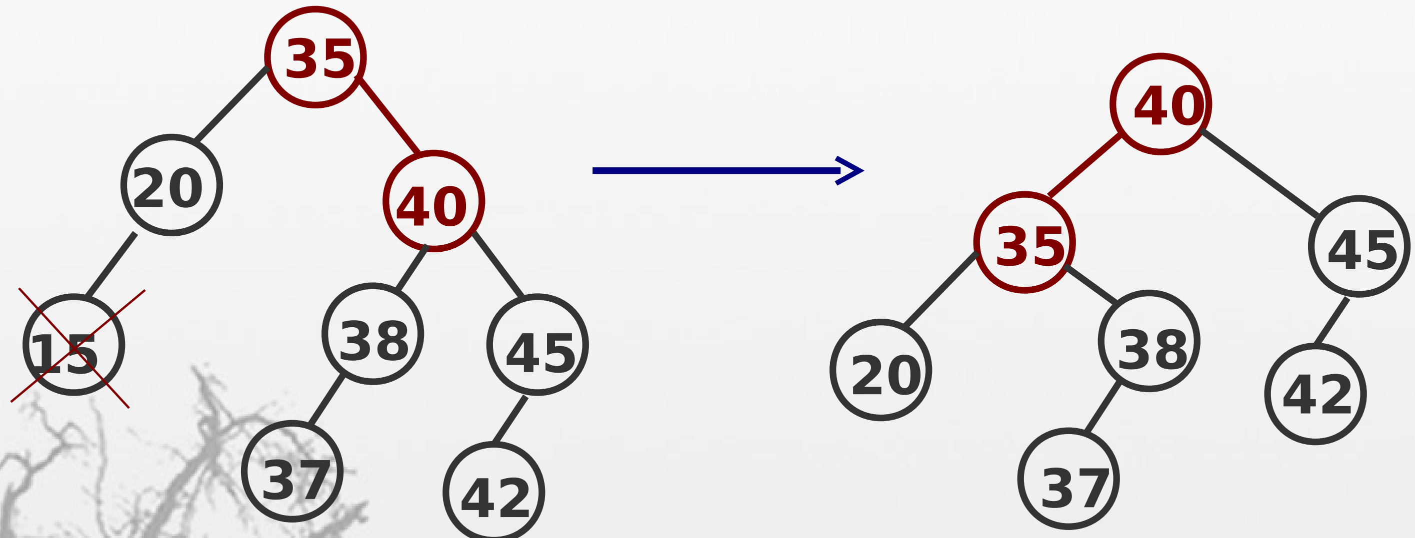
Las estrategias de rotación, simples o dobles son válidas para el borrado, sólo hay un caso nuevo: dos hijos del nodo *y* son de la misma altura:





Borrar un nodo en los AVL

Borra 15



Rotación simple a izquierda

Eficiencia de los AVL

El tiempo de búsqueda en un AVL es O(*log*n)

El tiempo de inserción en un AVL es O(*log*n):

- Insertar en un ABB es O(*log*n)
- Trazar el camino desde el nuevo nodo hasta la raíz por cada nodo encontrado es O(*log*n)
    - Ver la diferencia de alturas: O(1)
    - Si se satisface la propieda del AVL, proceder con el siguiente nodo: O(1)
    - Sino realizar una rotación: O(1)

El mismo razonamiento se aplica al borrado: O(*log*n)

Consideraciones finales

Los árboles AVL resuelven el problema de la eficiencia de la búsqueda.

- Sin embargo plantea un problema para iterar hacia delante o hacia atrás en el árbol
- Los recorridos recursivos no permiten este tipo de iteracion.
- STL resuelve este problema mediante árboles rojo-negro, que mantienen punteros al nodo padre.
- También es interesante trabajar diferenciando la clave del dato y permitiendo trabajar con claves repetidas, tal y como lo hace STL