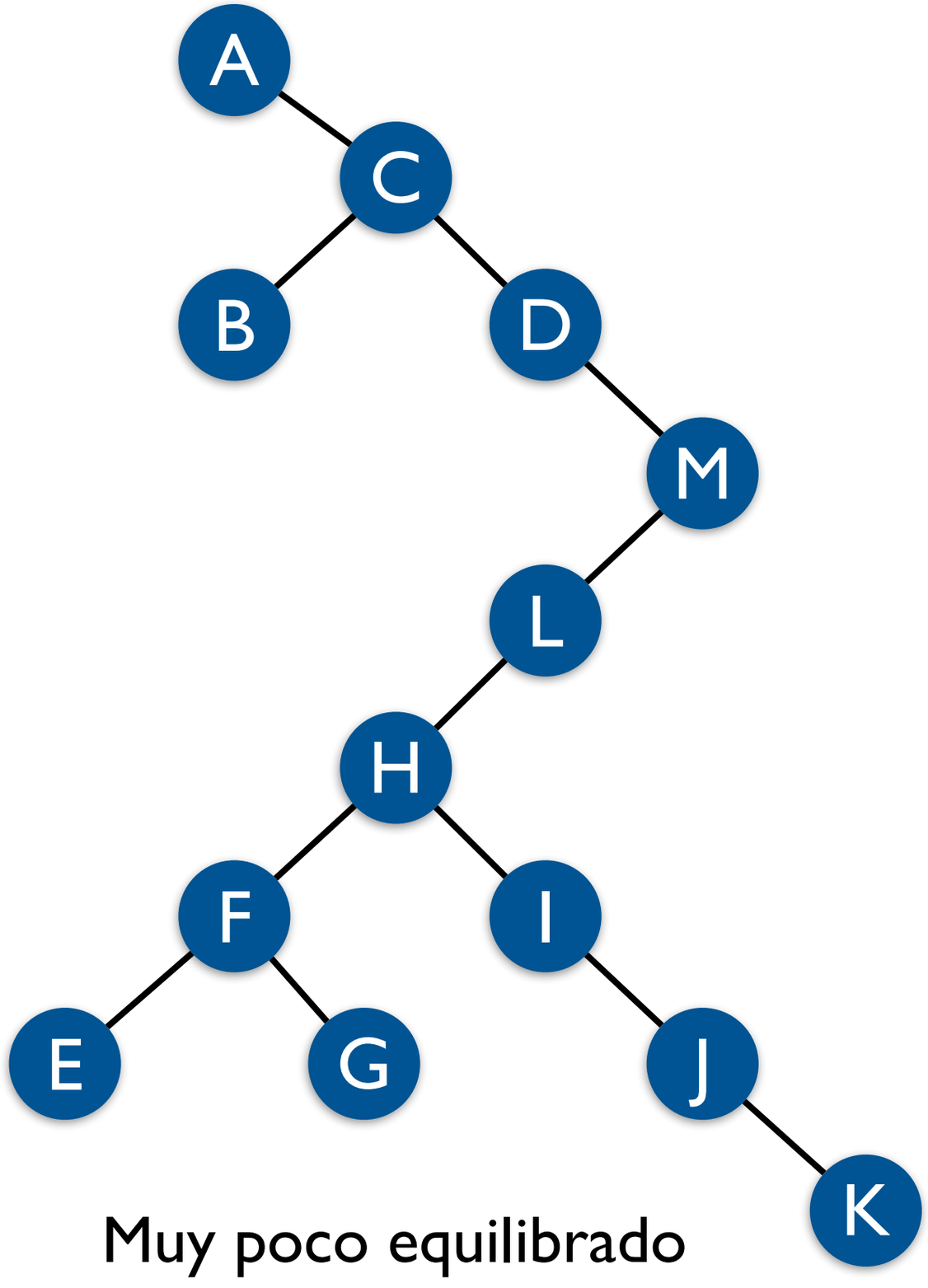
ÁRBOLES BINARIOS EQUILIBRADOS

Árboles AVL

Motivación

- En ocasiones, la construcción de los ABB conduce a árboles con características muy pobres para la búsqueda

- Ejemplo: Construir un ABB con {A, C, D M, L, H, I, B, F, G, J, K, E}



Muy poco equilibrado

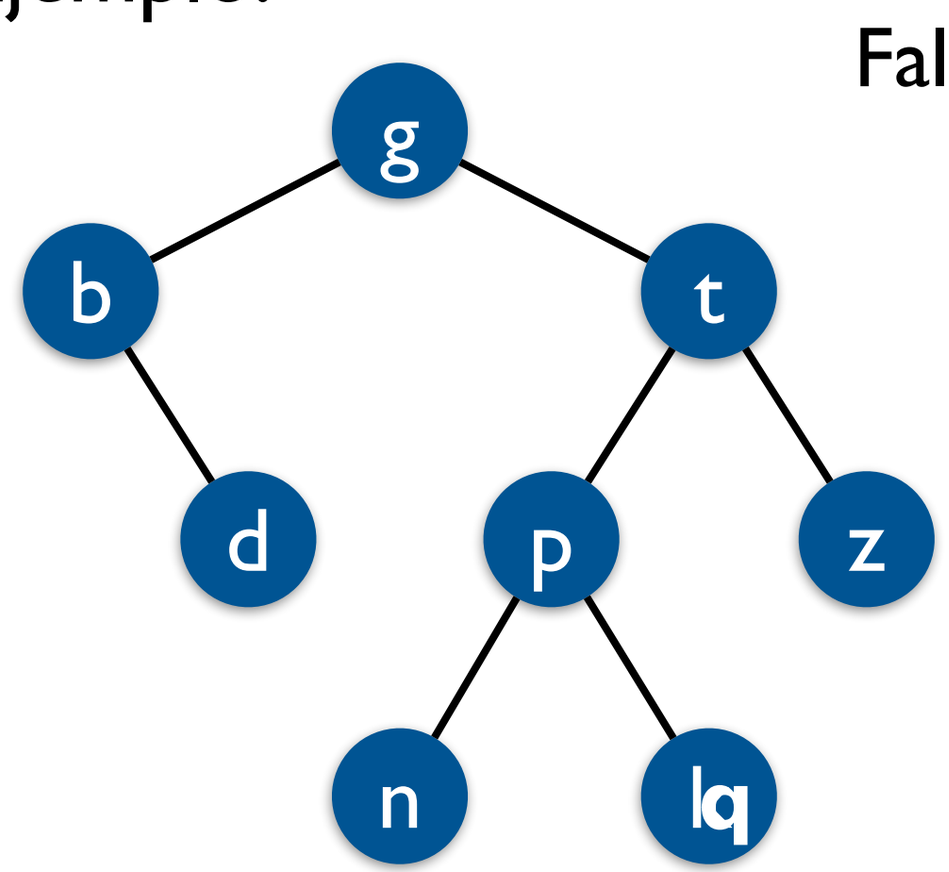
IDEA

Construir ABB equilibrados, impidiendo que en ningún nodo las alturas de los subárboles izquierdo y derecho difieran en más de una unidad

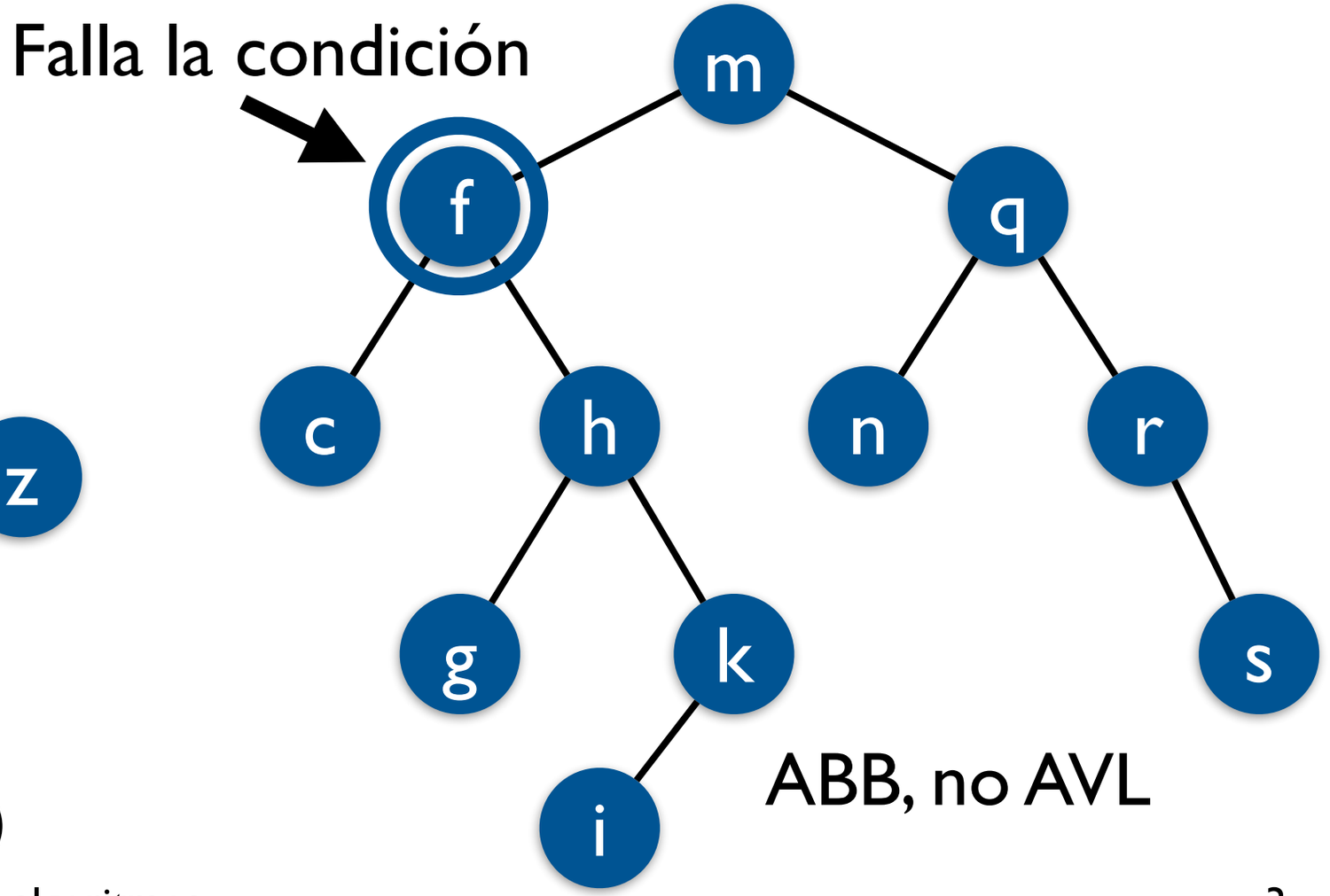
Árboles AVL

- Diremos que un árbol binario de búsqueda es un AVL (o que está equilibrado en el sentido de Addelson-Velski-Landis) si para cada uno de sus nodos se cumple que las alturas de sus dos subárboles difieren como máximo en 1

- Ejemplo:



AVL (ABB + Equilibrio)



ABB, no AVL

Eficiencia

- La altura de un árbol AVL está acotada por

$$\log_2(n+1) \leq h \leq 1.44 \log_2(n+2) - 0.33$$

- La altura de un AVL (esto es, la longitud de sus caminos de búsqueda) con n nodos nunca excede al 44% de la altura de un árbol completamente equilibrado con n nodos

- Consecuencia: en el peor de los casos, la búsqueda se puede realizar en $O(\log_2 n)$

Árboles AVL

- Nos interesan funciones para las operaciones de:
  - Pertenencia
  - Inserción
  - Borrado

- Debemos tener en cuenta que tendremos que diseñar funciones auxiliares que permitan realizar estas operaciones manteniendo el árbol equilibrado

Equilibrio en inserciones y borrados

- **Idea:** Usar un campo altura en el registro que represente cada uno de los nodos del AVL para determinar el factor de equilibrio (diferencia de altura entre los subárboles izquierdo y derecho), de forma que cuando esa diferencia sea >1 se hagan los reajustes necesarios en los punteros para que tenga una diferencia de alturas ≤ 1

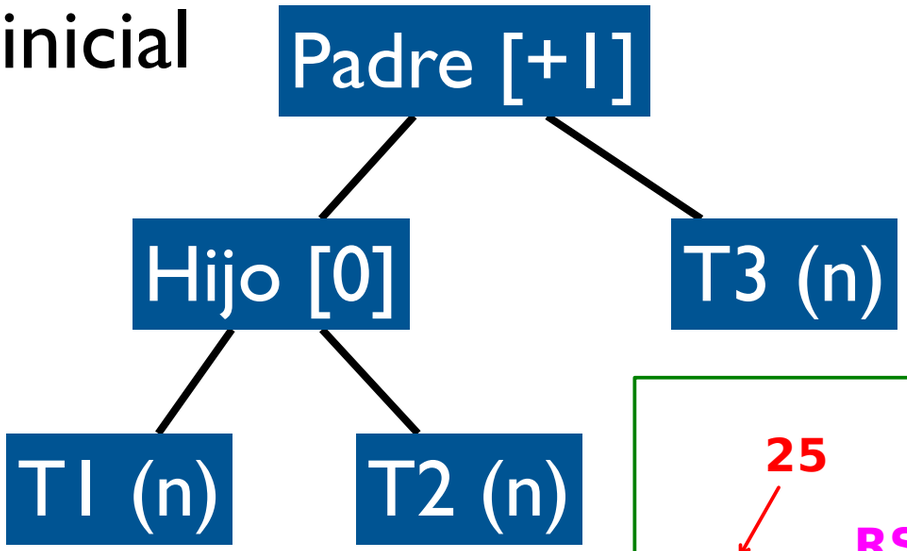
- Vamos a verlo en una serie de ejemplos en los que mostraremos todos los casos posibles

Equilibrio en inserciones y borrados

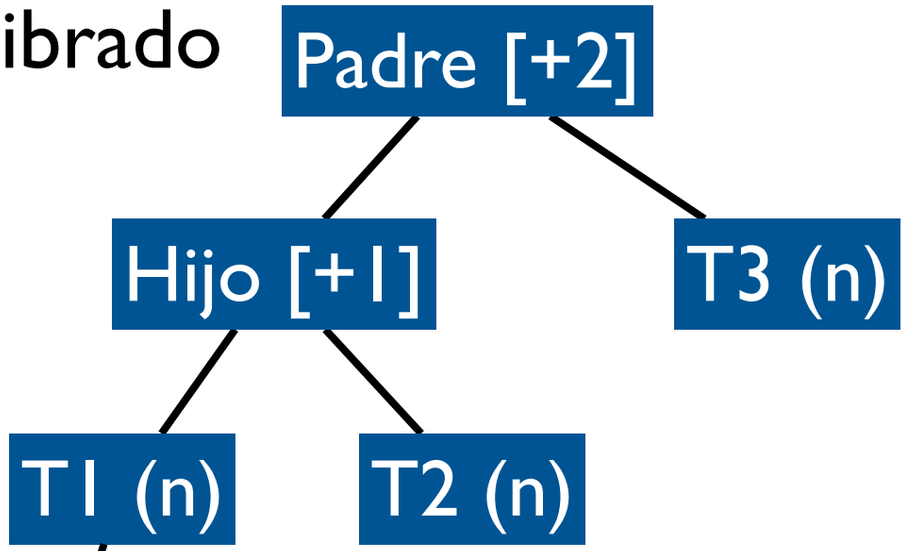
- Notaremos los subárboles como T_k, anotando entre paréntesis su altura (la altura de su raíz)
- Notaremos el factor de equilibrio como un valor con signo ubicado entre corchetes junto a cada padre o hijo
- Las dos situaciones posibles que pueden representarse son:
  - Rotaciones simples
  - Rotaciones dobles

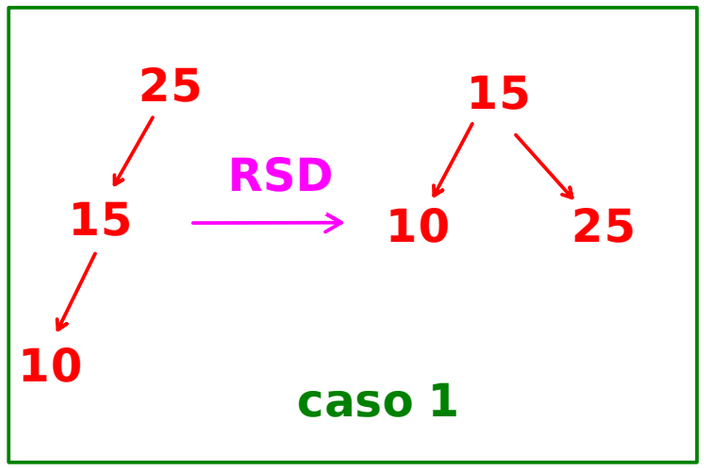
Rotación simple a la derecha

AVL inicial

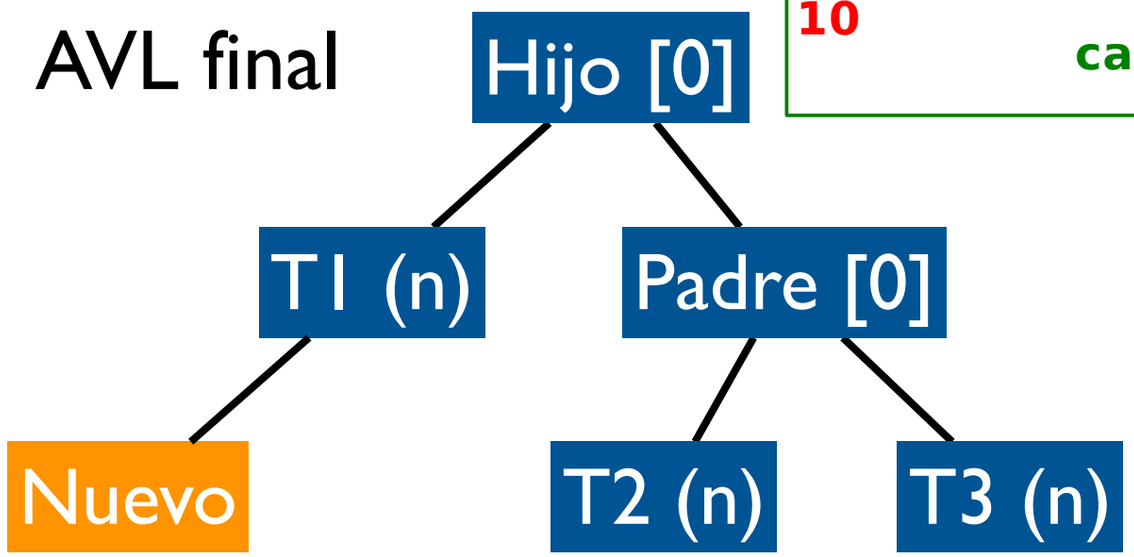


AVL desequilibrado tras insertar





AVL final



Reajustes necesarios para la rotación

padre→izqda = hijo→drcha

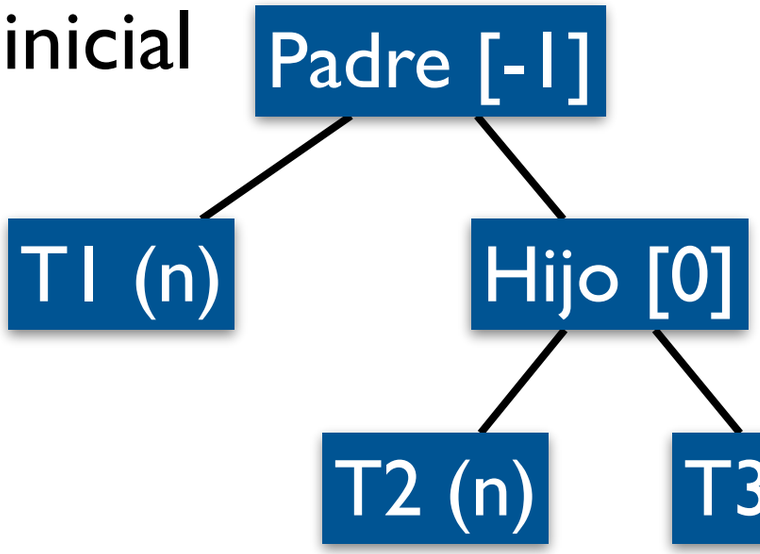
hijo→drcha = padre

a) Se preserva el inorden

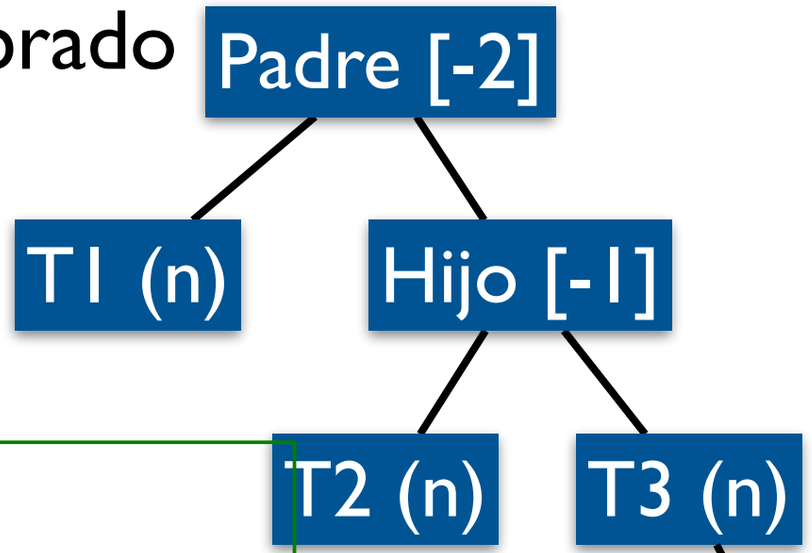
b) Altura del árbol final = altura arbol inicial

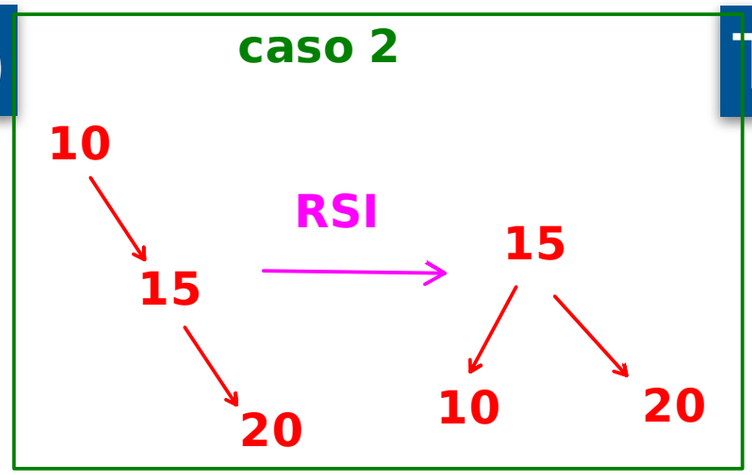
Rotación simple a la izquierda

AVL inicial

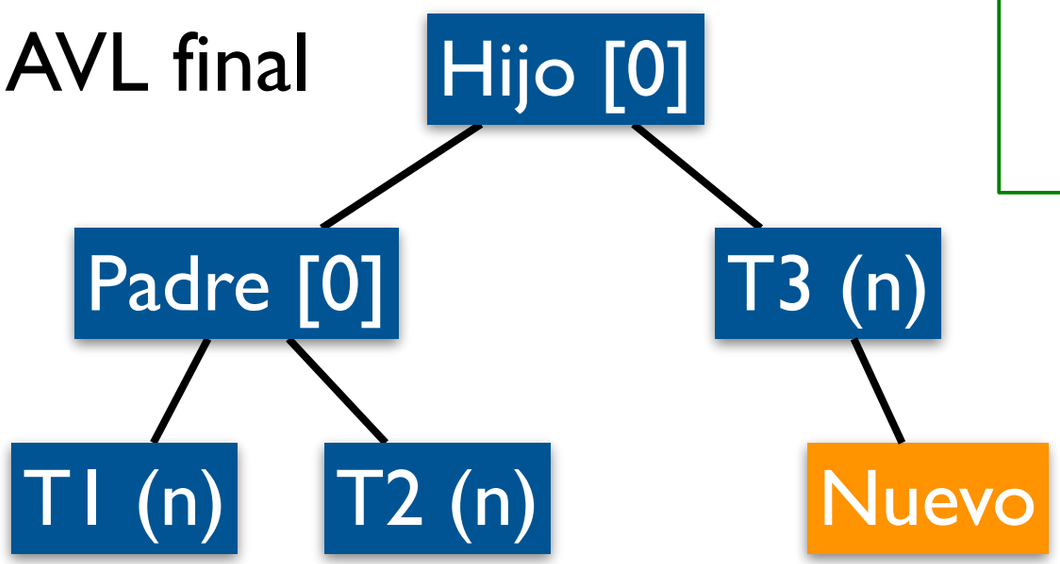


AVL desequilibrado tras insertar





AVL final



Reajustes necesarios para la rotación

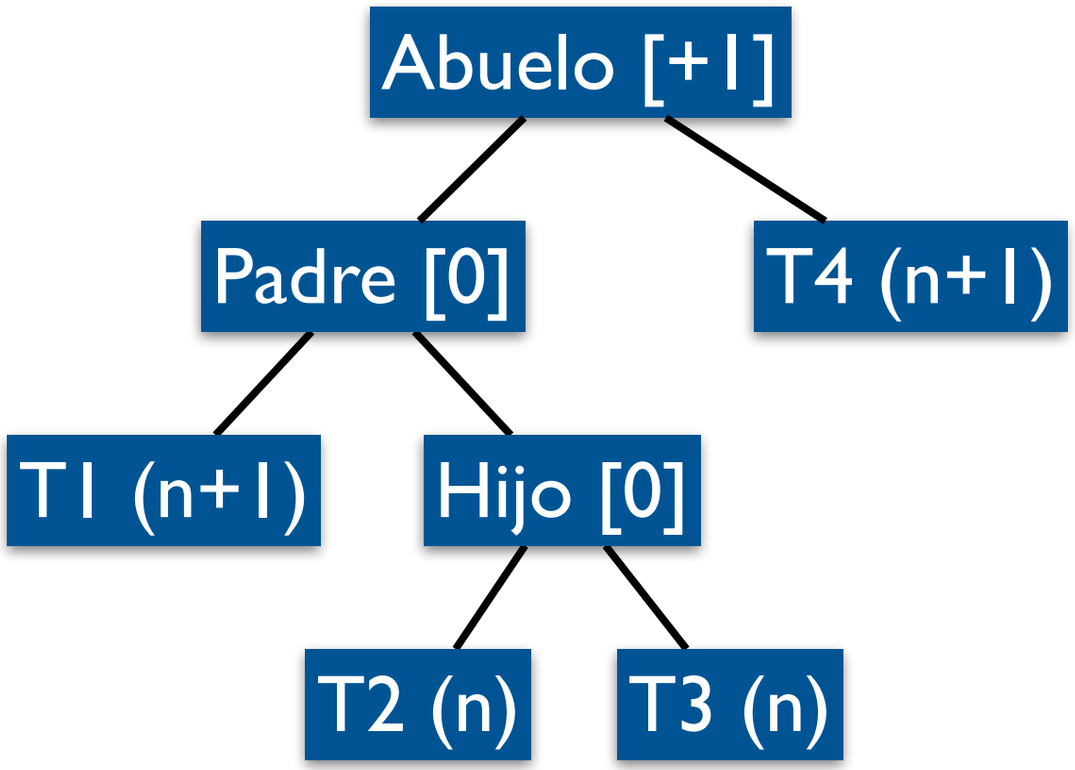
padre→drcha = hijo→izqda

hijo→izqda = padre

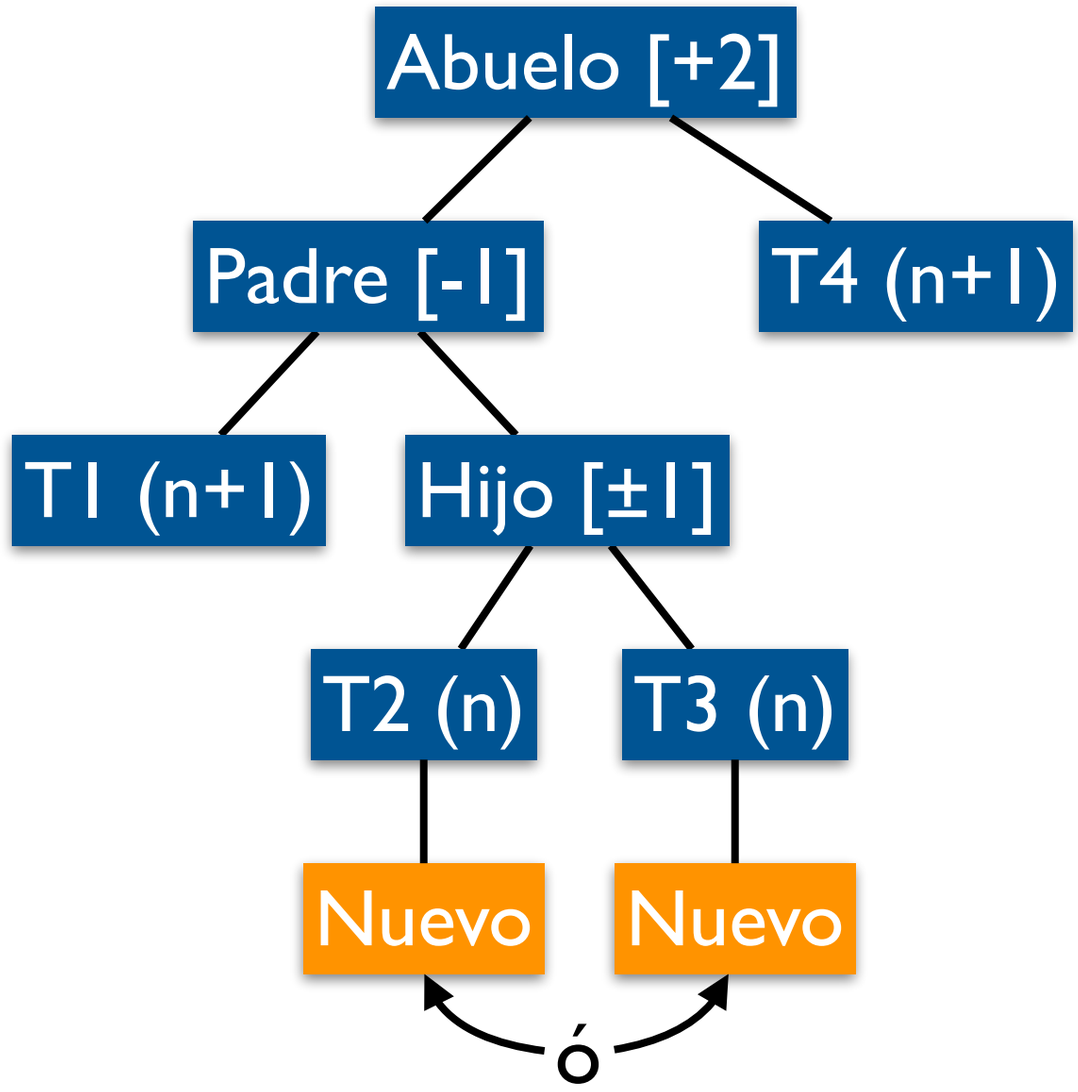
a) Se preserva el inorden

b) Altura del árbol final = altura arbol inicial

Rotación doble a la derecha

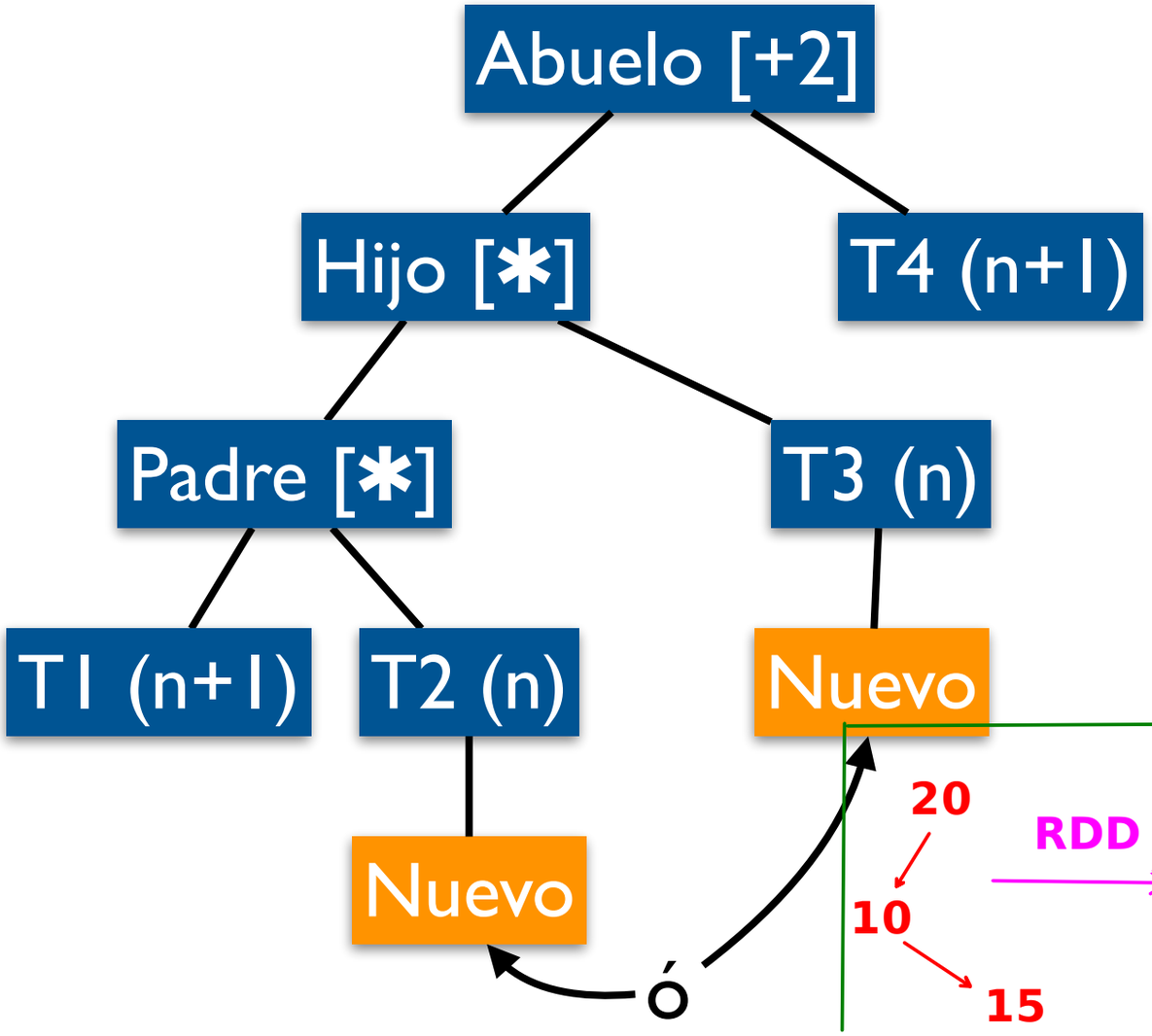


AVL inicial

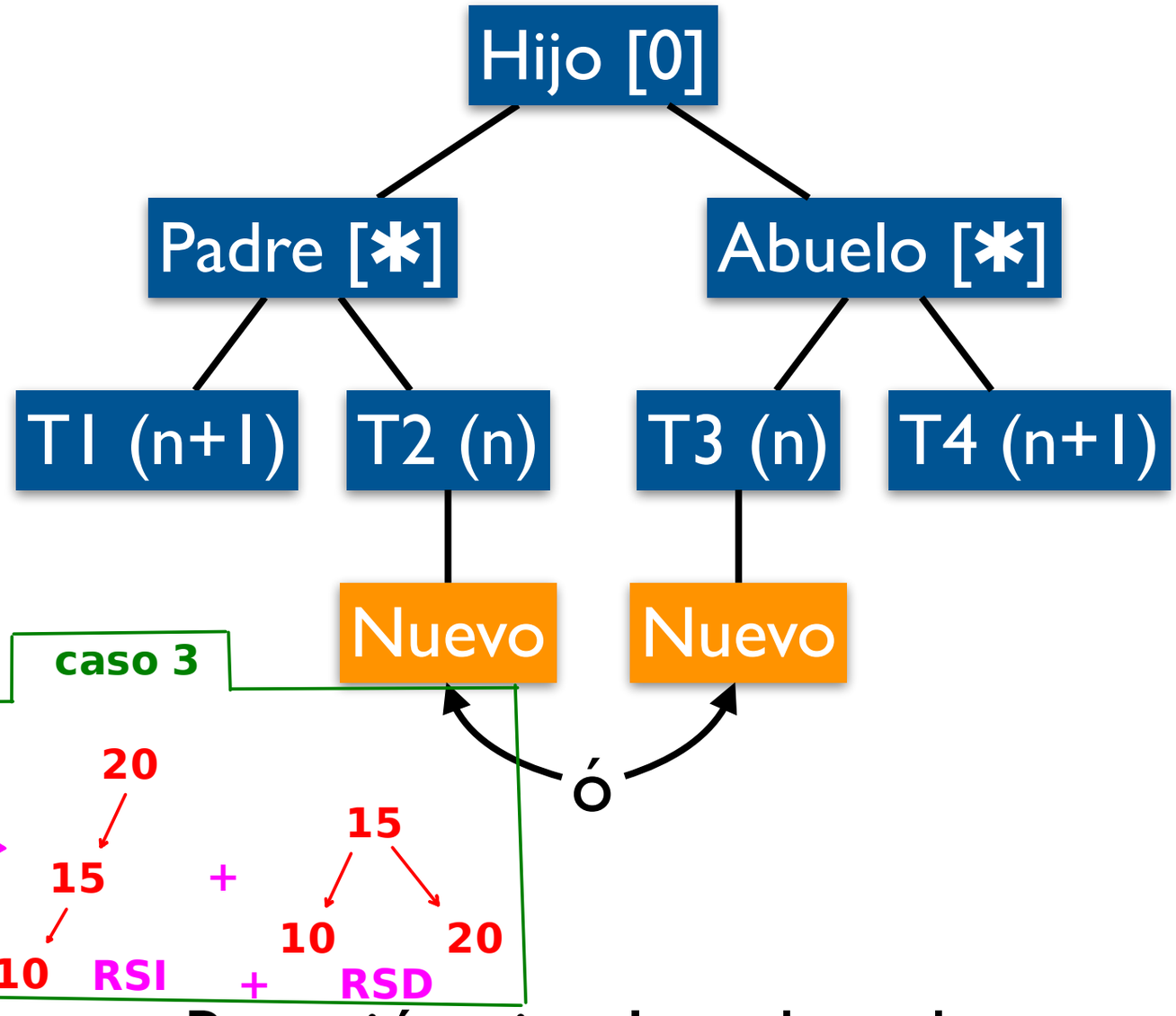


AVL desequilibrado tras insertar

Rotación doble a la derecha

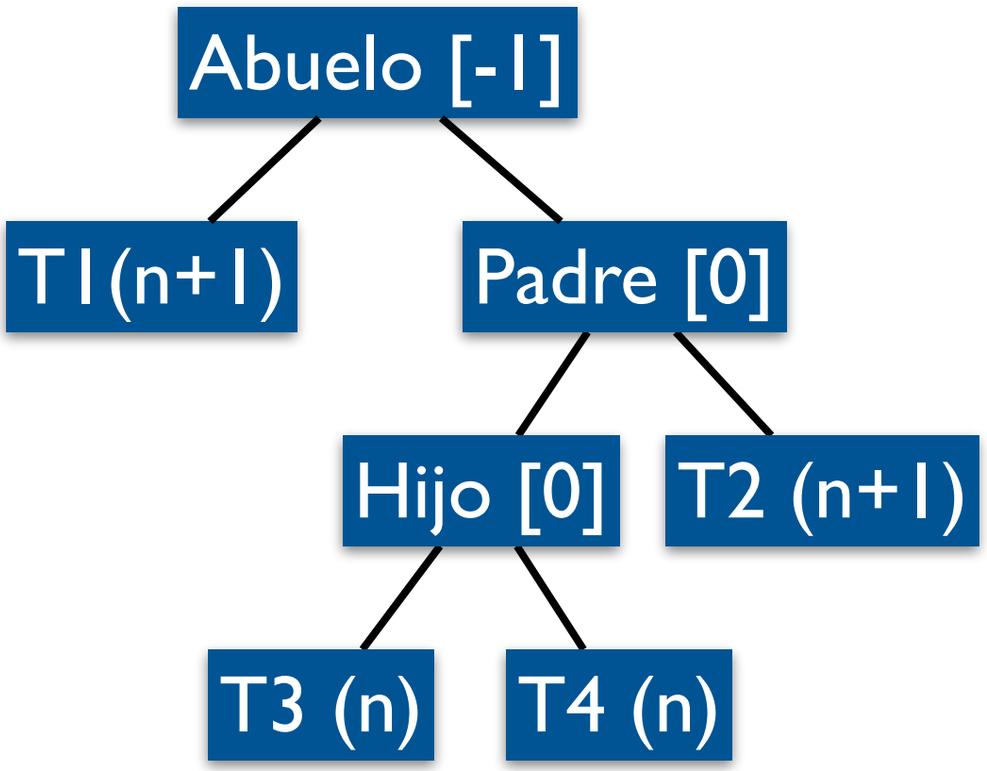


Rotación simple a izquierda en padre

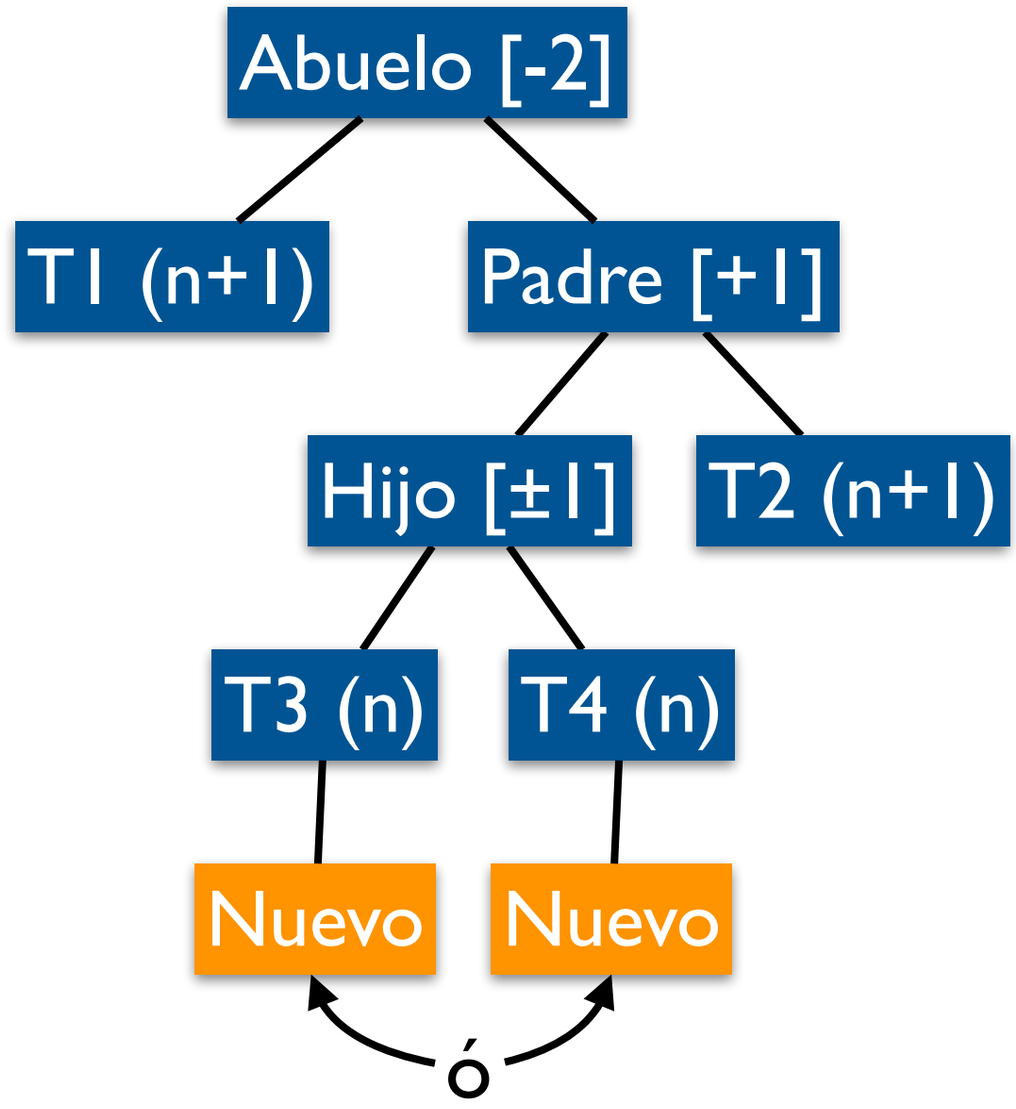


Rotación simple a derecha en abuelo

Rotación doble a la izquierda

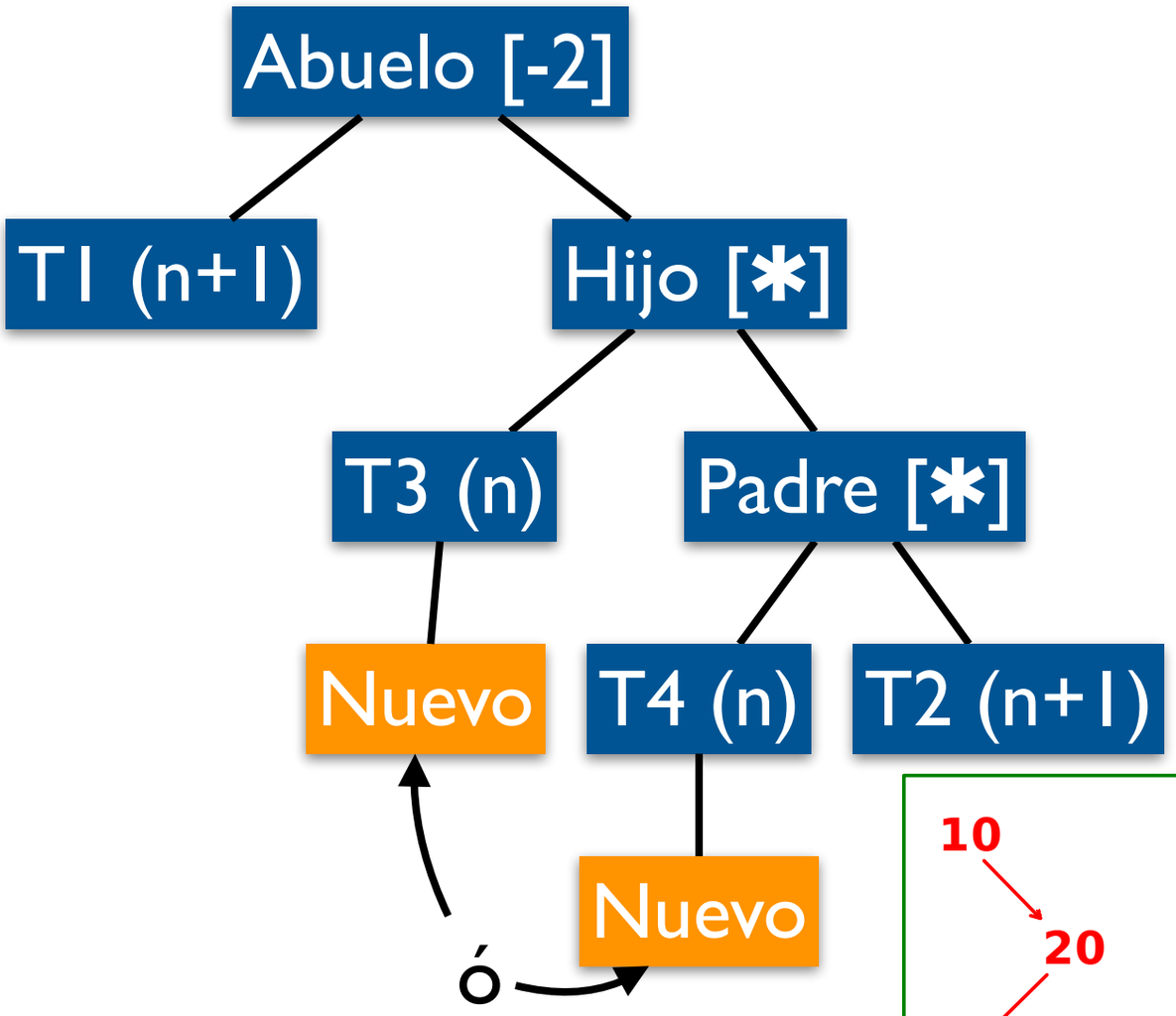


AVL inicial

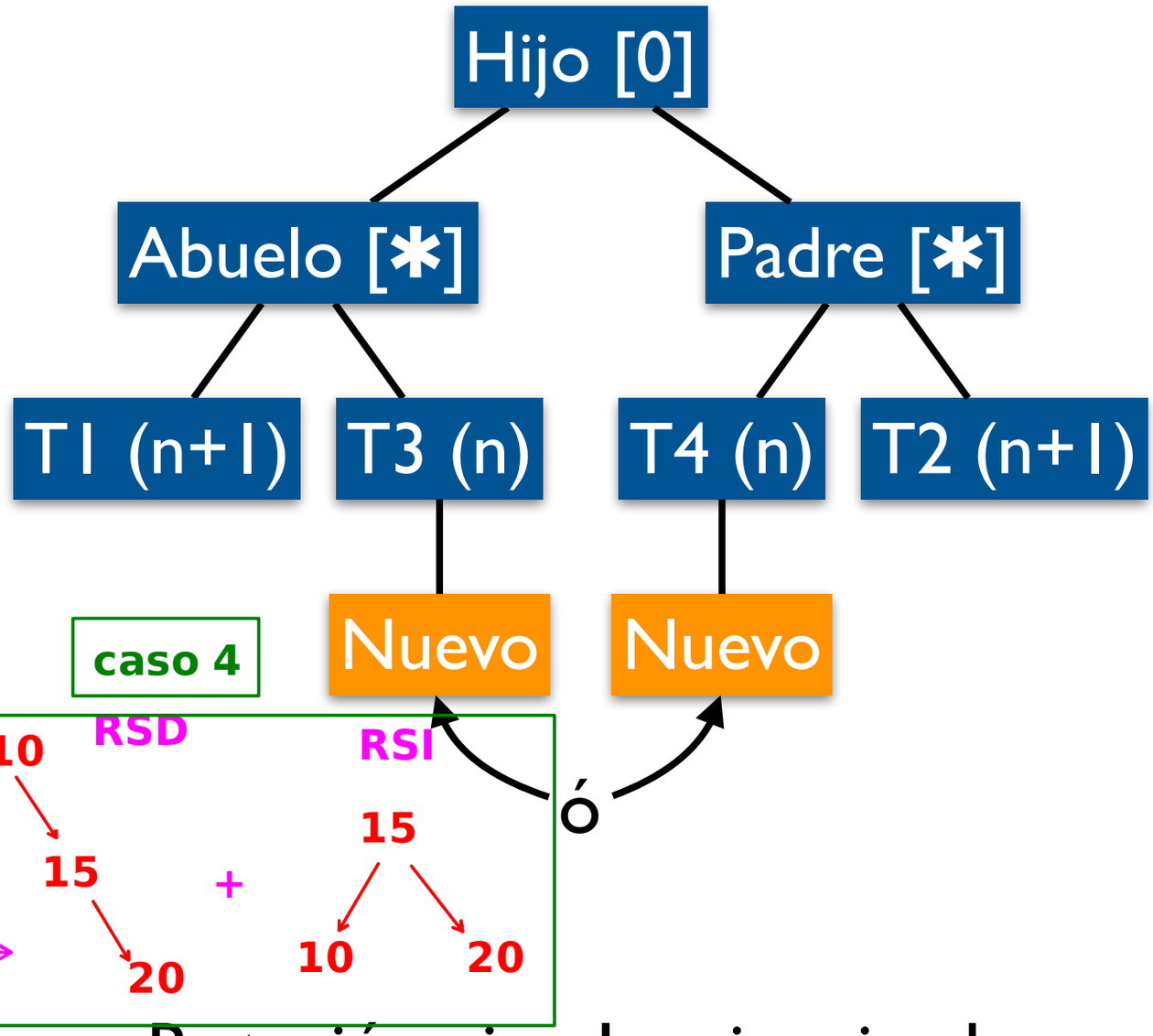


AVL desequilibrado tras insertar

Rotación doble a la izquierda



Rotación simple a derecha en padre



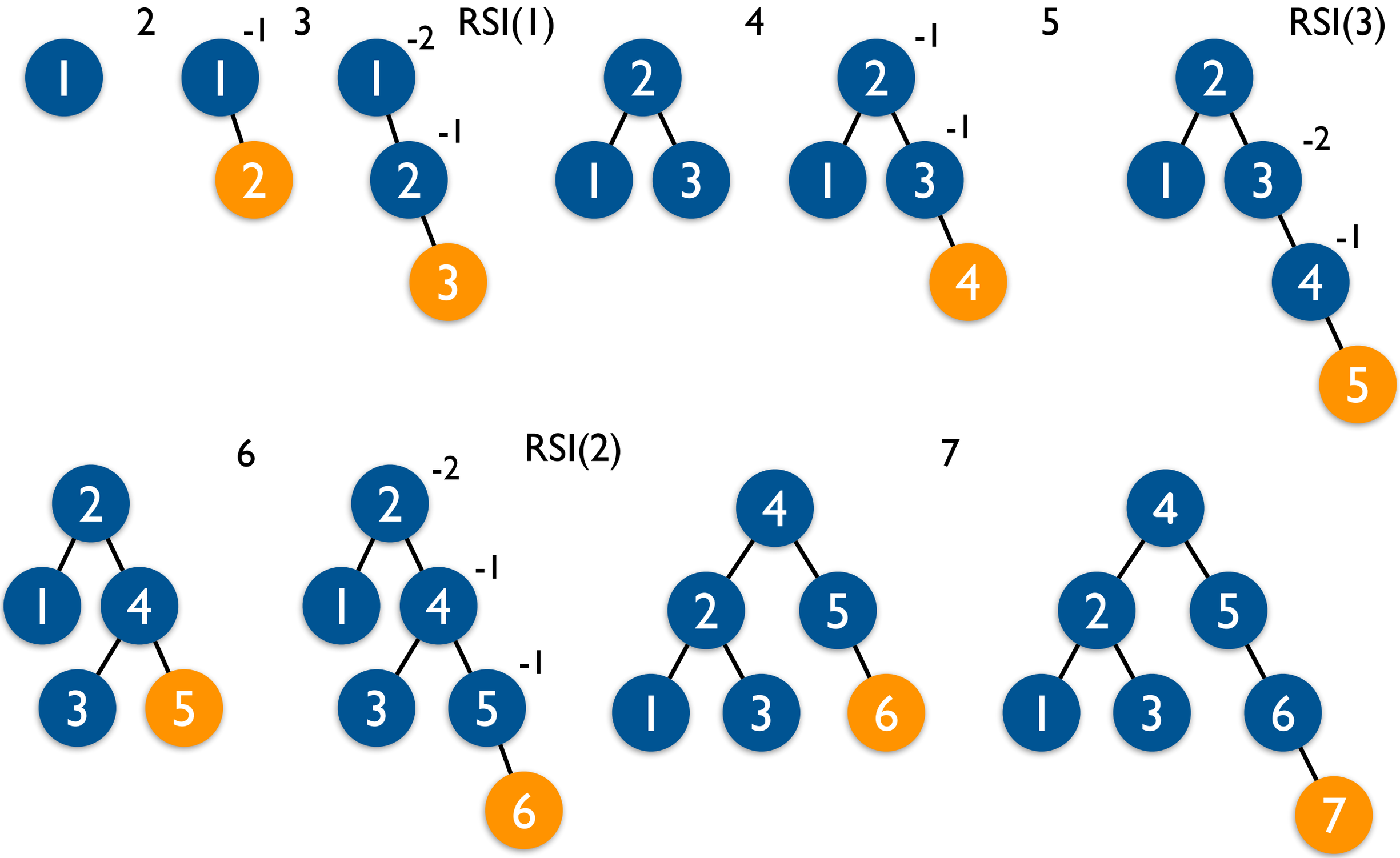
Rotación simple a izquierda en abuelo

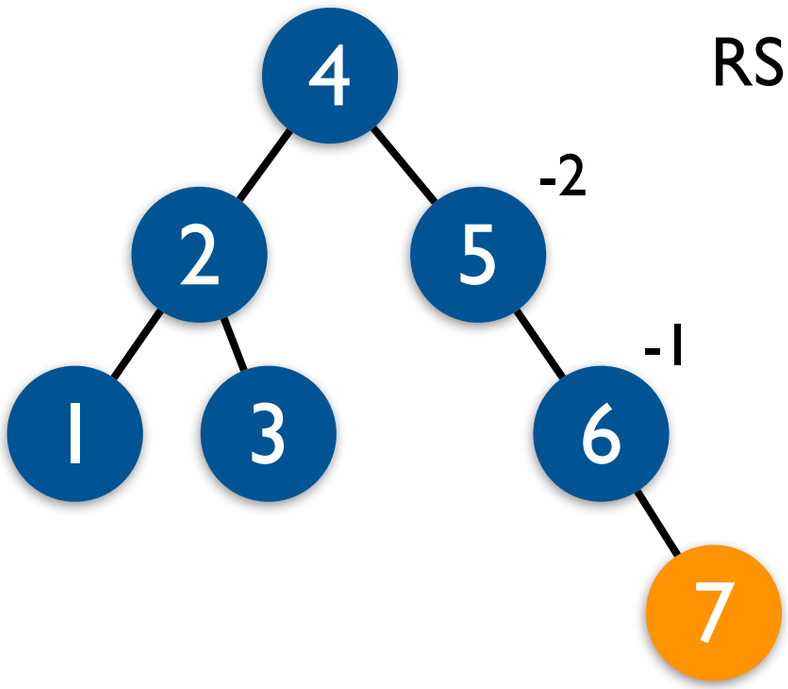
¿Qué rotación utilizar?

Si la inserción se realiza en:

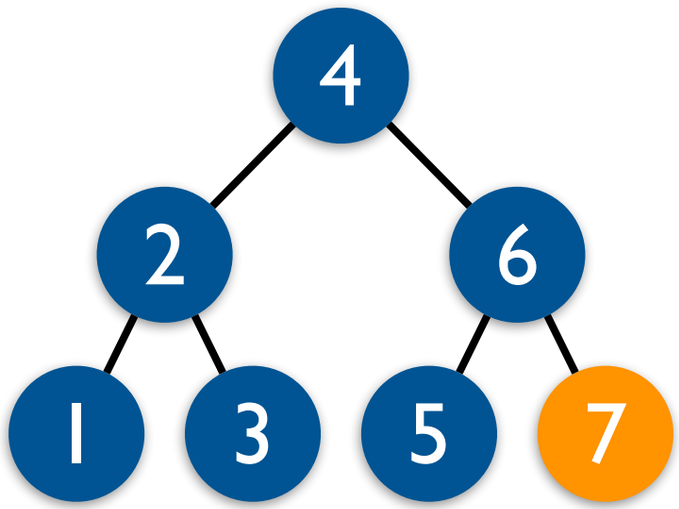
- el hijo izquierdo del hijo izquierdo del nodo desequilibrado → RSD
- el hijo derecho del hijo derecho del nodo desequilibrado → RSI
- el hijo derecho del hijo izquierdo del nodo desequilibrado → RDD
- el hijo izquierdo del hijo derecho del nodo desequilibrado → RDI

Ejemplo

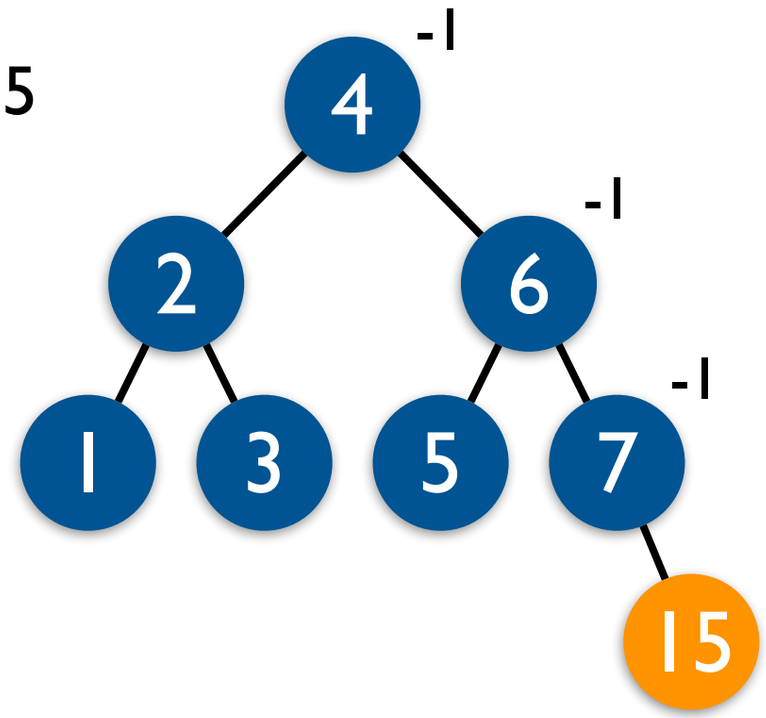




RSI(5)



15



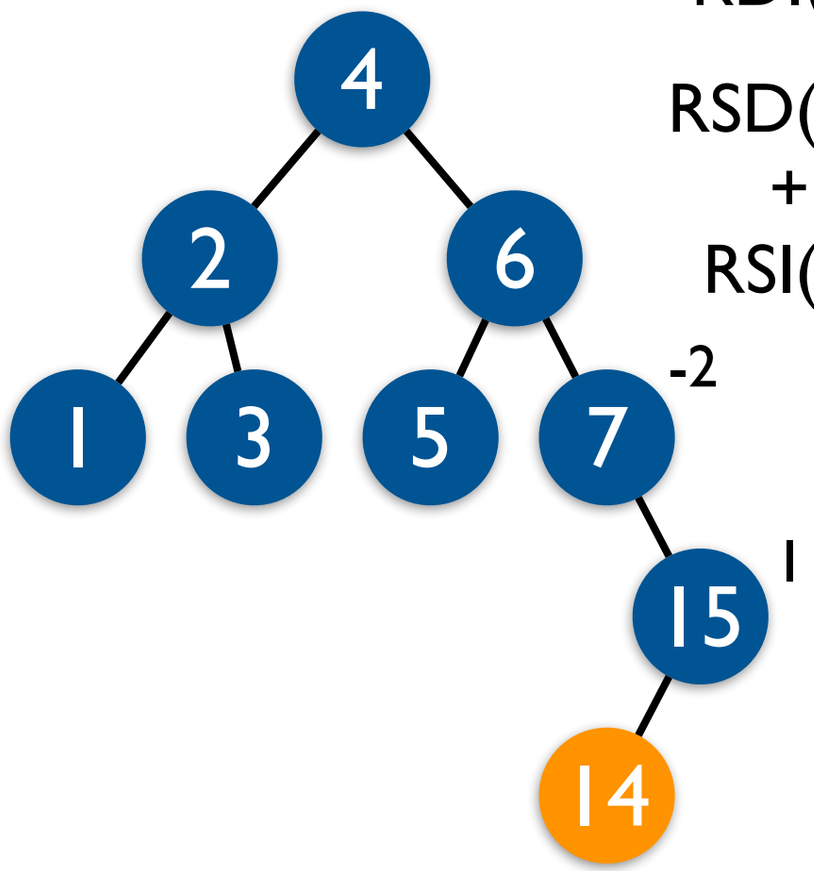
14

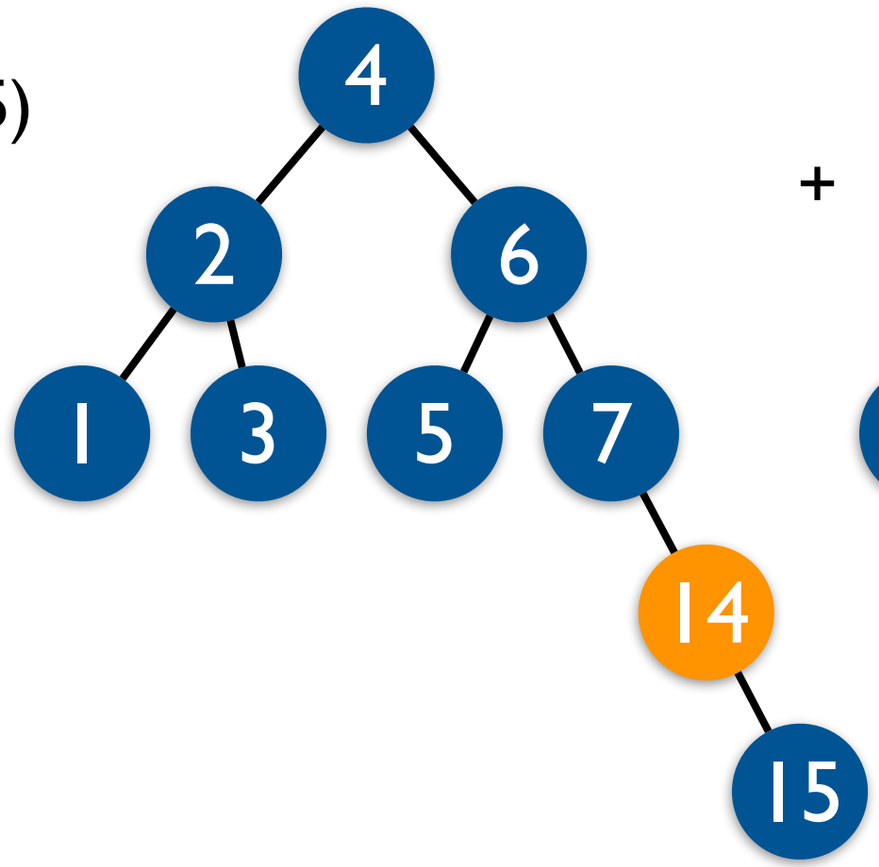
RDI(7)

RSD(15)

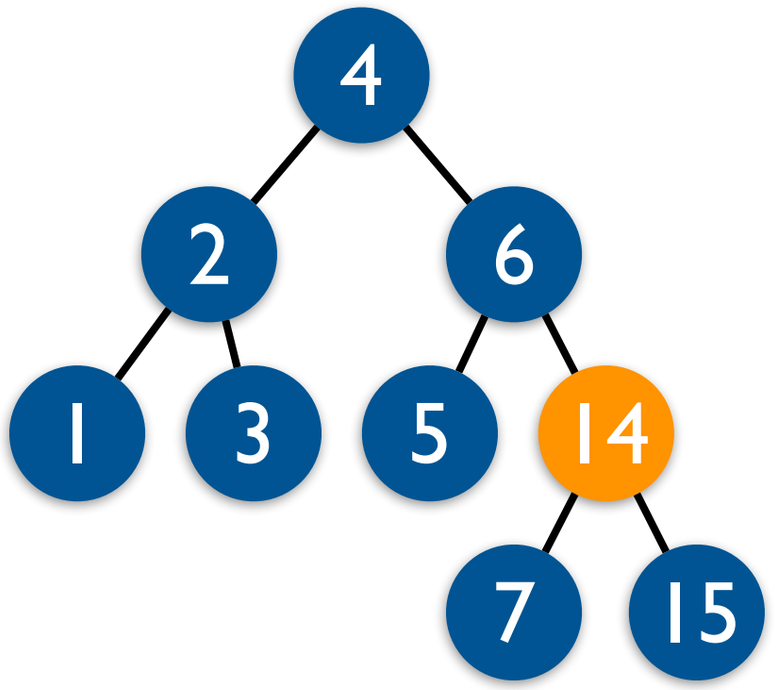
\+

RSI(7)

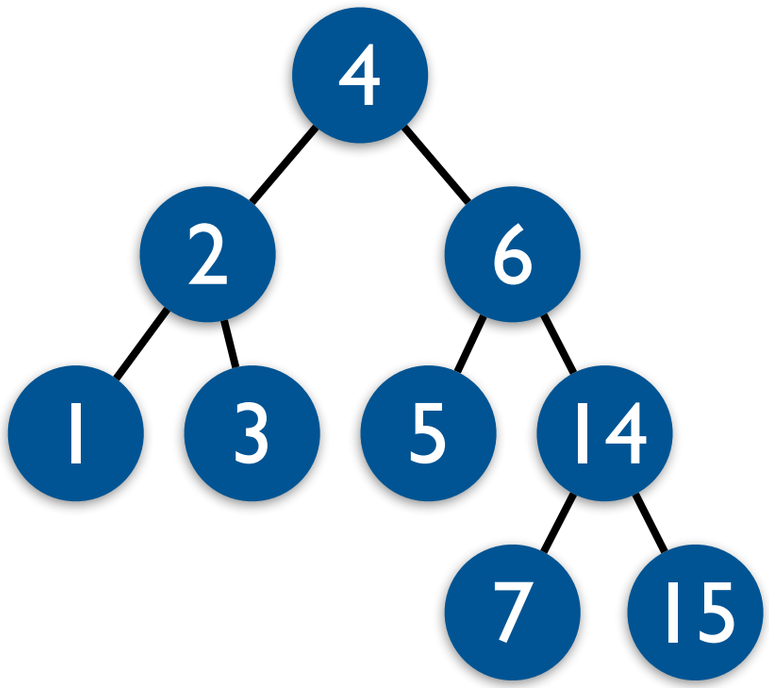




\+

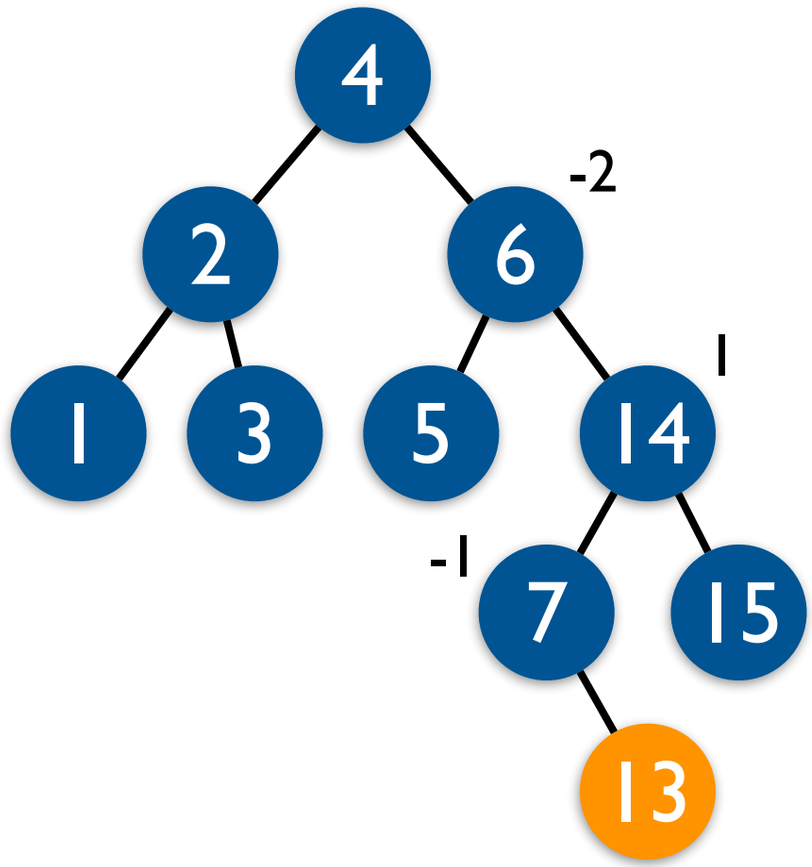


13

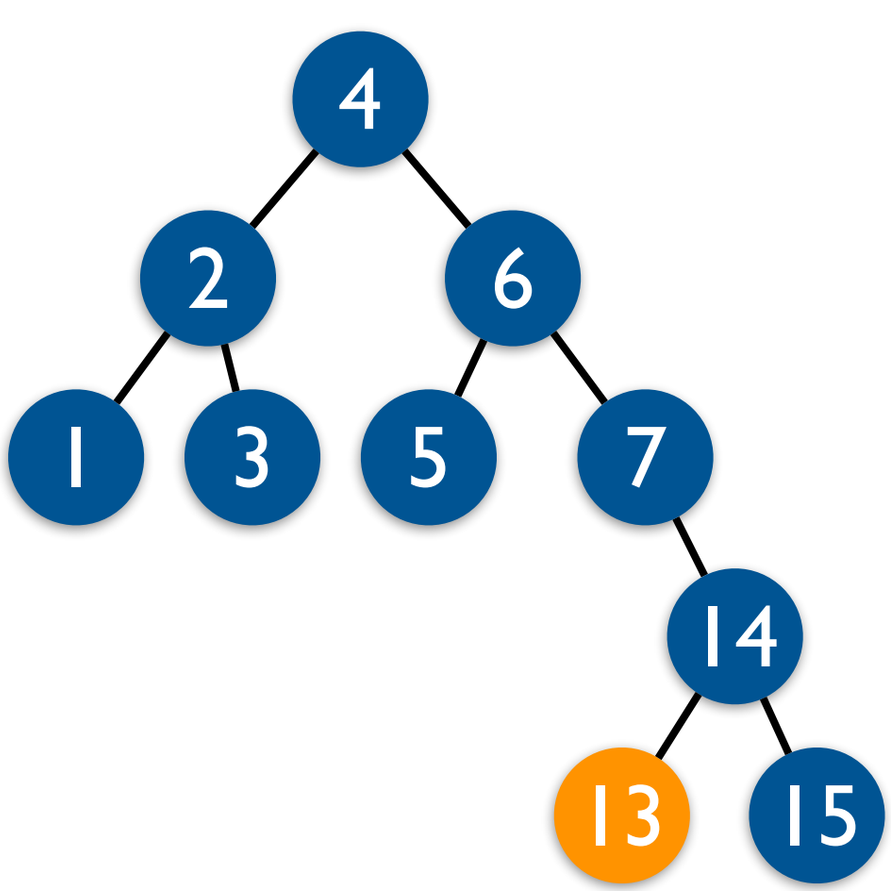


RDI(6)

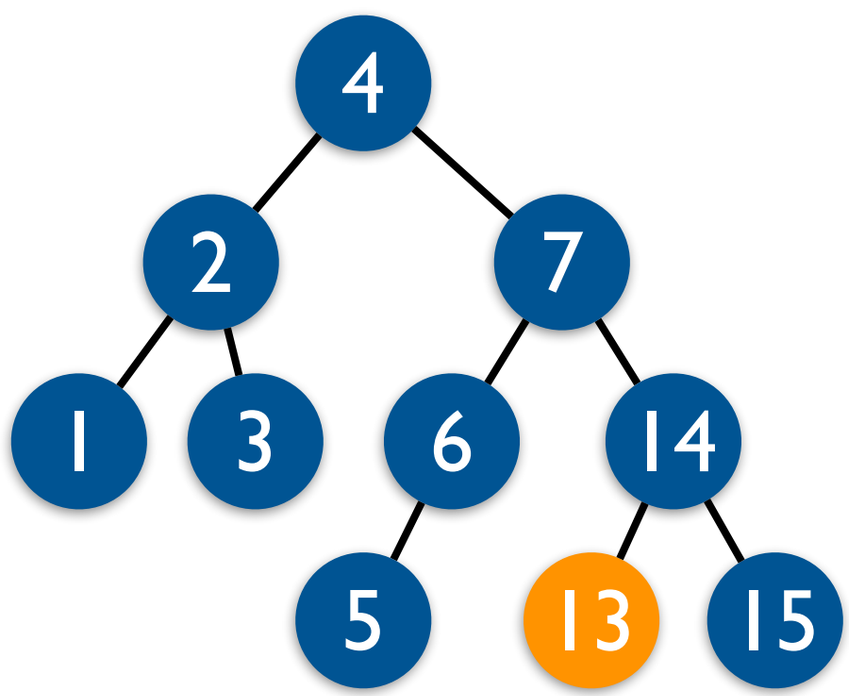
RSD(14)
+
RSI(6)



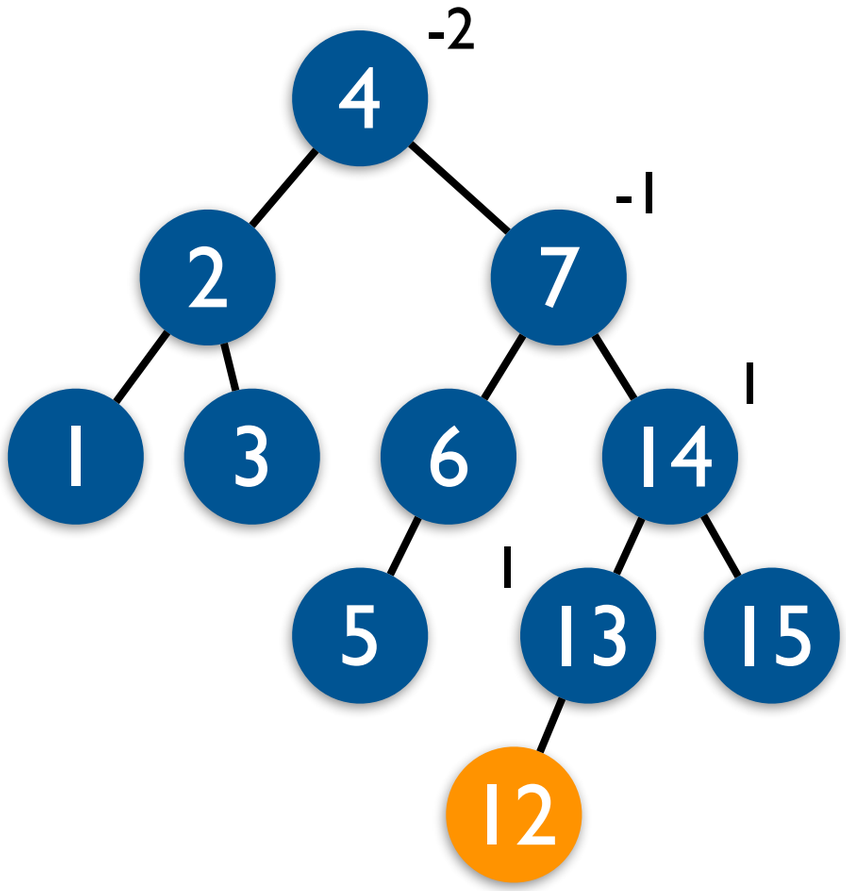
12



+

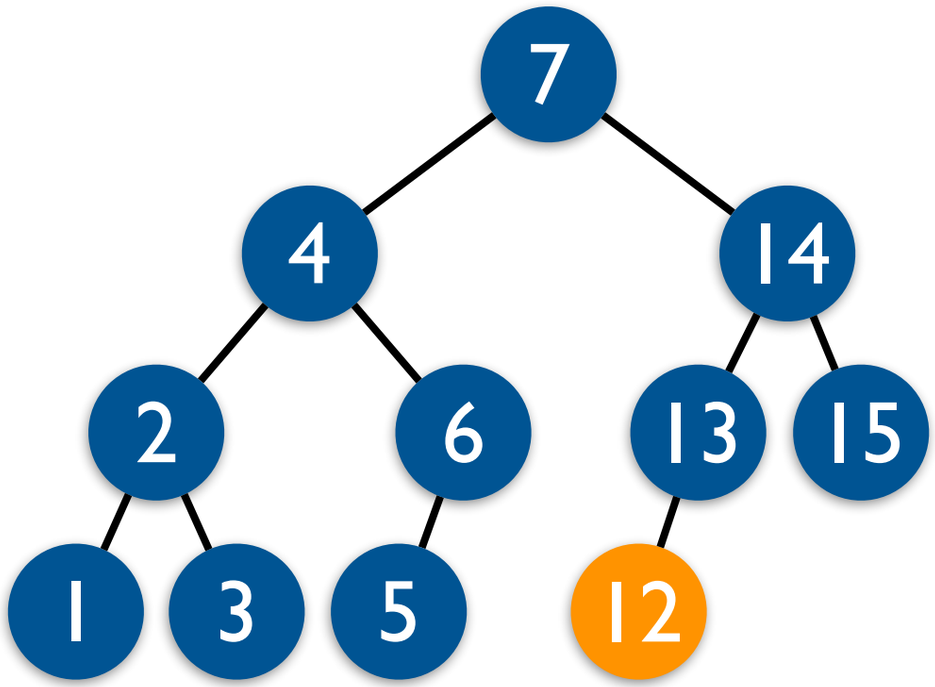


12



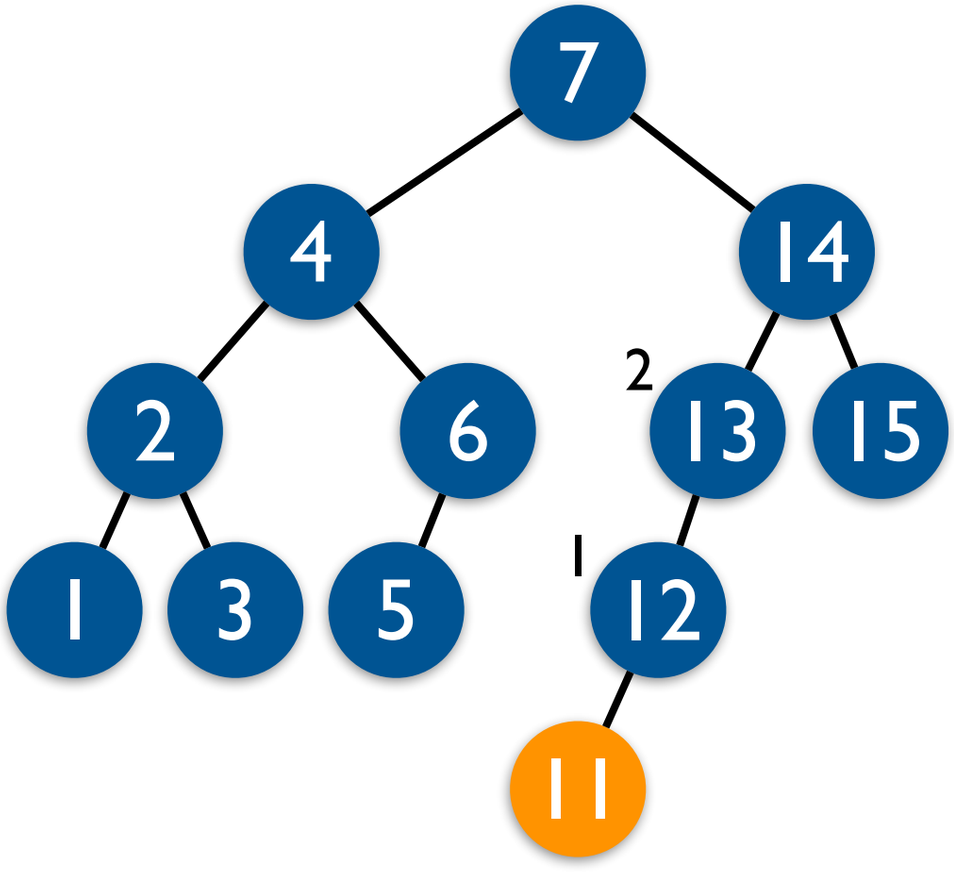
RSI(4)

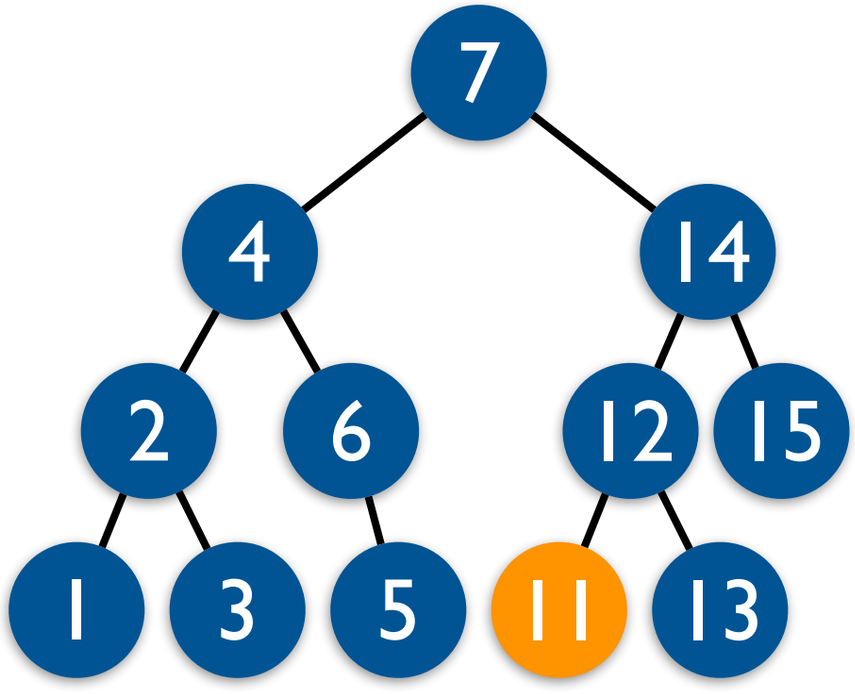
11



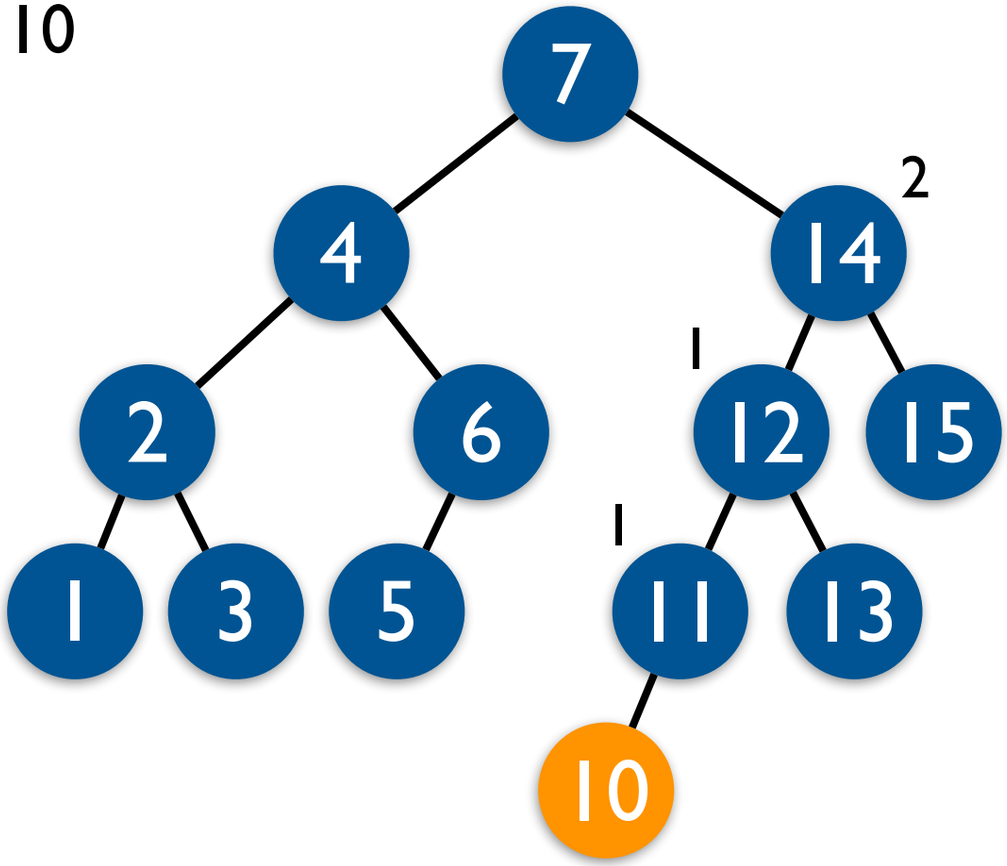
RSD(13)

10

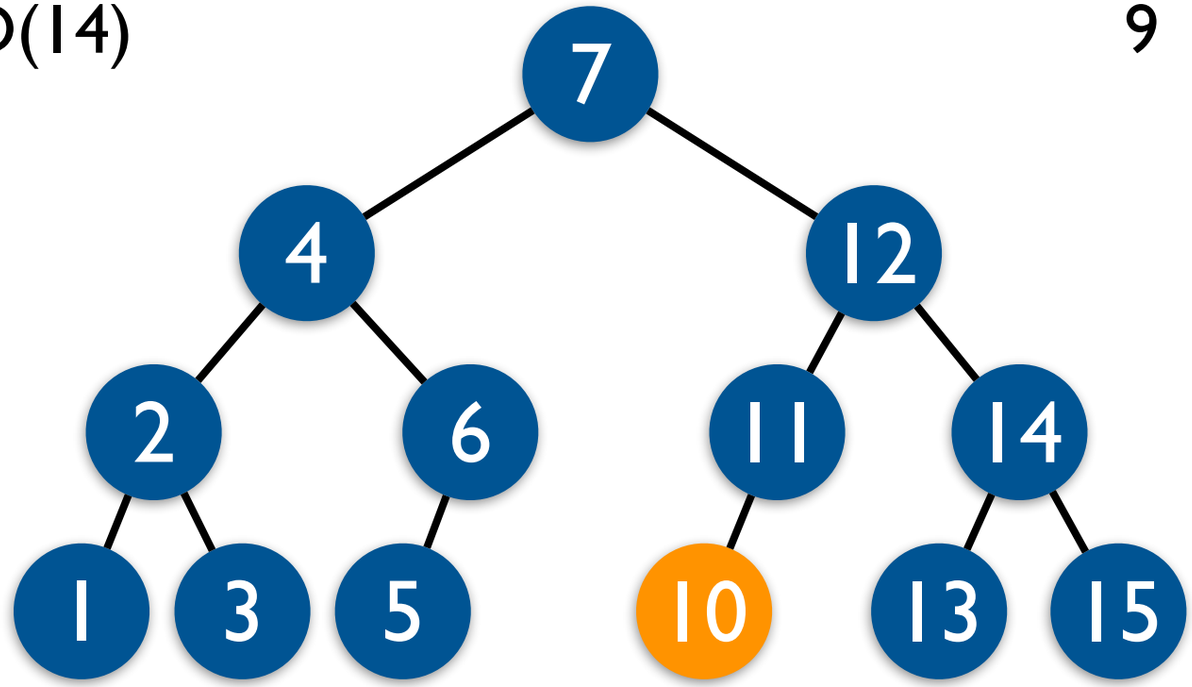




10

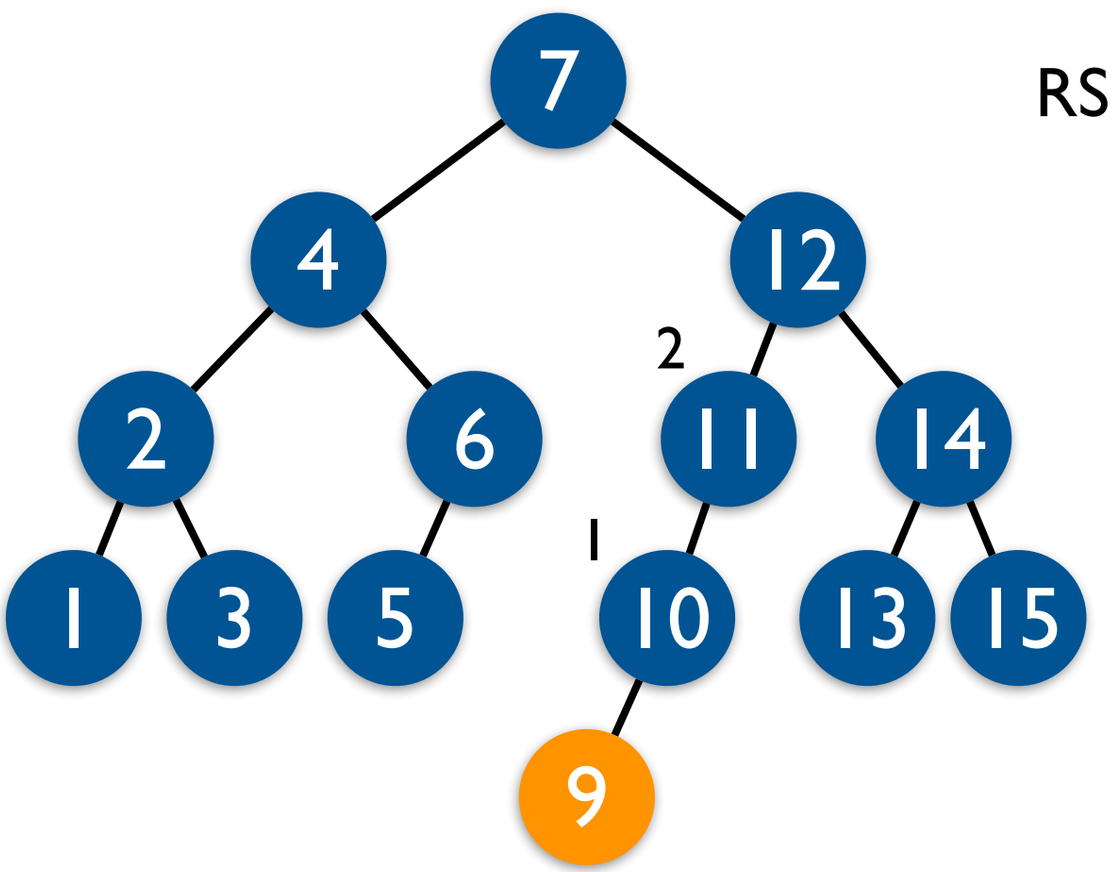


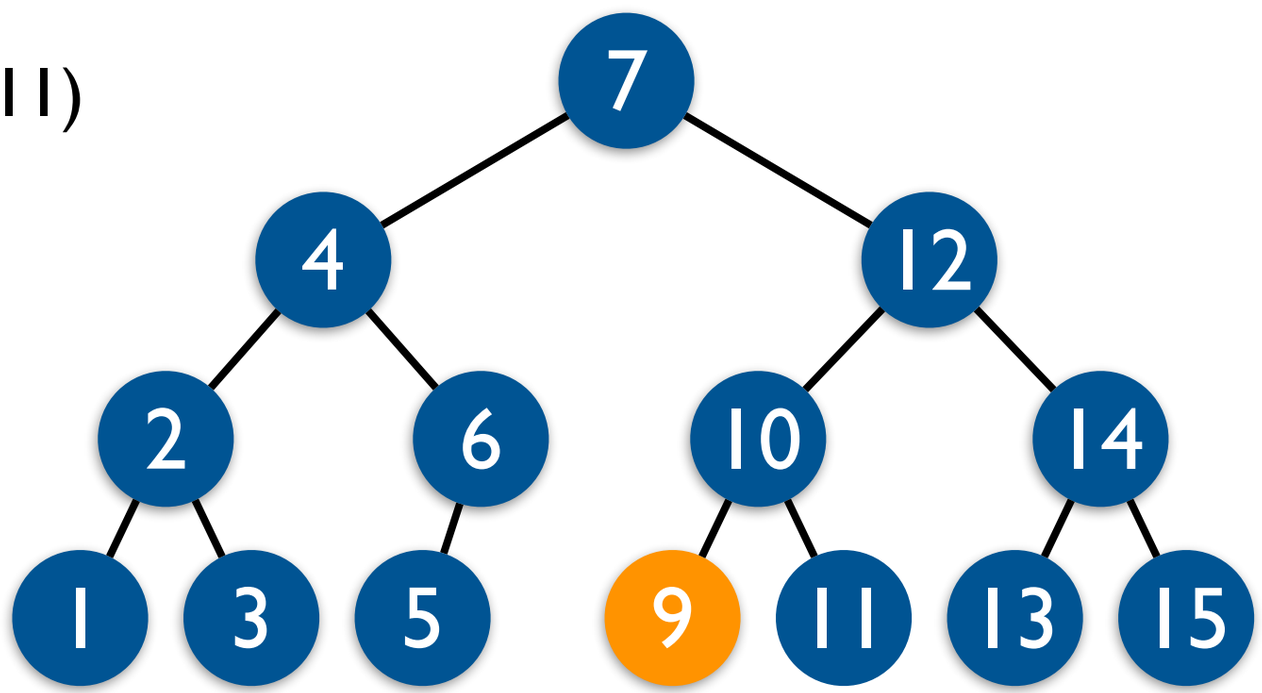
RSD(14)



9

RSD(11)





AVL Final

Árboles equilibrados AVL

- Son árboles binarios de búsqueda equilibrados. Las operaciones de inserción y borrado tienen un orden de eficiencia logarítmico.
- Se caracterizan porque para cada nodo se cumple que la diferencia de la altura de sus dos hijos es como mucho de una unidad.
- La especificación coincide con la del Árbol binario de búsqueda.
- La implementación varía en las operaciones que modifican la altura de un nodo: insertar y borrar.

Implementación

```
template <class Tbase>
void AVL<Tbase>::ajustarArbol
  (ArbolBinario<Tbase>::Nodo &n)
{
   int aIzda;
   int aDcha;
   ArbolBinario<Tbase>::Nodo hIzda, hDcha;

   // Ajustamos desde n hasta la raíz del árbol
   while (n!=ArbolBinario<Tbase>::NODO_NULO) {
     aIzda = altura(arbolb.HijoIzqda(n));
     aDcha = altura(arbolb.HijoDrcha(n));

     if (abs(aIzda-aDcha)>1) // Hay que ajustar
       if (aIzda>aDcha) {
         hIzda = arbolb.HijoIzqda(n);
         if (altura(arbolb.HijoIzqda(hIzda)) >
       altura(arbolb.HijoDrcha(hIzda)))
           rotarHijoIzqda(n);
         else {
           rotarHijoDrcha(hIzda);
```

```
            rotarHijoIzqda(n);
        }
    }
    else { // Exceso de altura por la dcha
        hDcha = arbolb.HijoDrcha(n);
        if (altura(arbolb.HijoIzqda(hDcha)) >
    altura(arbolb.HijoDrcha(hDcha))) {
            rotarHijoIzqda(hDcha);
            rotarHijoDrcha(n);
        }
        else
            rotarHijoDrcha(n);
    }
    n = arbolb.Padre(n);
  }
}
```

```
template <class Tbase>
void AVL<Tbase>::rotarHijoIzqda
  (ArbolBinario<Tbase>::Nodo &n)
{
  assert(n!=ArbolBinario<Tbase>::NODO_NULO);

  char que_hijo;
```

simétrico rotarHijoDrcha

```
ArbolBinario<Tbase>::Nodo elPadre =
  arbolb.Padre(n);
```

```
ArbolBinario<Tbase> A;
arbolb.PodarHijoIzqda(n, A);
```

```
ArbolBinario<Tbase> Aux;
if (elPadre!=ArbolBinario<Tbase>::NODO_NULO)
  if (arbolb.HijoIzqda(elPadre)==n) {
    arbolb.PodarHijoIzqda(elPadre, Aux);
    que_hijo = IZDA;
  }
  else {
    arbolb.PodarHijoDrcha(elPadre, Aux);
    que_hijo = DCHA;
  }
else
  Aux = arbolb;
```

doble o simple ?

```
ArbolBinario<Tbase> B;
A.PodarHijoDrcha(A.Raiz(), B);
```

```
Aux.InsertarHijoIzqda(Aux.Raiz(), B);
A.InsertarHijoDrcha(A.Raiz(), Aux);
```

```
    if (elPadre!=ArbolBinario<Tbase>::NODO_NULO) {
      if (que_hijo==IZDA) {
        arbolb.InsertarHijoIzqda(elPadre, A);
        n = arbolb.HijoIzqda(elPadre);
      }
      else {
        arbolb.InsertarHijoDrcha(elPadre, A);
        n = arbolb.HijoDrcha(elPadre);
      }
    }
    else {
      arbolb = A;
      n = arbolb.Raiz();
    }
}
```

```
template <class Tbase>
void AVL<Tbase>::rotarHijoDrcha
  (ArbolBinario<Tbase>::Nodo &n)
{
  assert(n!=ArbolBinario<Tbase>::NODO_NULO);

  char que_hijo;
```

```
ArbolBinario<Tbase>::Nodo elPadre =
  arbolb.Padre(n);

ArbolBinario<Tbase> A;
arbolb.PodarHijoDrcha(n, A);

ArbolBinario<Tbase> Aux;
if (elPadre!=ArbolBinario<Tbase>::NODO_NULO)
  if (arbolb.HijoIzqda(elPadre)==n) {
    que_hijo = IZDA;
    arbolb.PodarHijoIzqda(elPadre, Aux);
  }
  else {
    que_hijo = DCHA;
    arbolb.PodarHijoDrcha(elPadre, Aux);
  }
else
  Aux = arbolb;

ArbolBinario<Tbase> B;
A.PodarHijoIzqda(A.Raiz(), B);

Aux.InsertarHijoDrcha(Aux.Raiz(), B);
A.InsertarHijoIzqda(A.Raiz(), Aux);
```

```
    if (elPadre!=ArbolBinario<Tbase>::NODO_NULO) {
      if (que_hijo==IZDA) {
        arbolb.InsertarHijoIzqda(elPadre, A);
        n = arbolb.HijoIzqda(elPadre);
      }
      else {
        arbolb.InsertarHijoDrcha(elPadre, A);
        n = arbolb.HijoDrcha(elPadre);
      }
    }
    else {
      arbolb = A;
      n = arbolb.Raiz();
    }
}
```