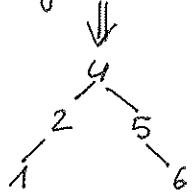
LECCIÓN 12: Arboles Binarios de Búsqueda Equilibrados AVL

Los ABB (árboles binarios de búsqueda) en el mejor de los casos nos permiten hacer búsquedas de $O(\log_2(n))$. Pero puede darse que se construya A1BB cuyo tiempo de ejecución del algoritmo de búsqueda tarde $O(n)$.

Ej: 1 — 2 — 3 — 4 — 5 — 6 $\Rightarrow$ Busqueda en $O(n)$

Nada Balanceado

OBJETIVO $\Rightarrow$ Equilibrar el A1BB para poder hacer búsquedas en $O(\log_2(n))$.



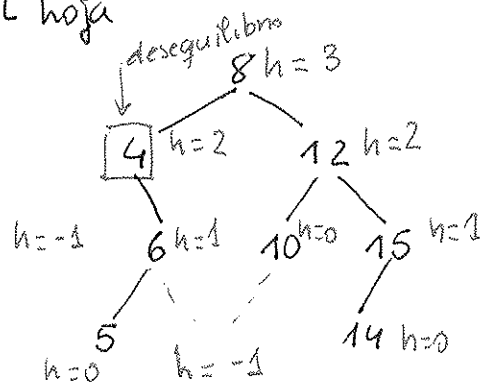
El equilibrio según los AVL no es perfecto (el mismo nº de nodos en el subarbol izq como en el subarbol derecho). El AVL pretende que la diferencia en altura en un nodo n de su hizq y su hder no sea mayor que 1

ARBOLES AVL (Adelson, Velskii, Landis)

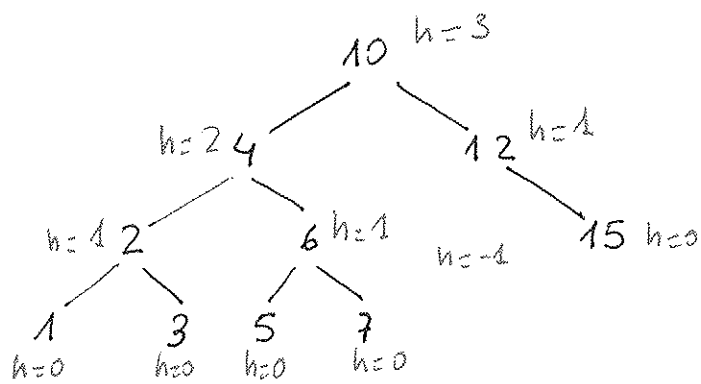
Un AVL es un ABB con la condición en que las diferencias en alturas de los subarboles que cuelgan de cualquier nodo como mucho es 1.

La altura de un arbol vacio es -1.

[ALTURA: Longitud del camino más largo desde un nodo a un nodo hoja]



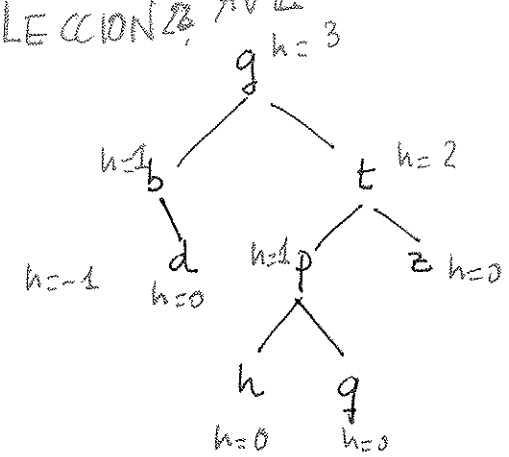
NO AVL



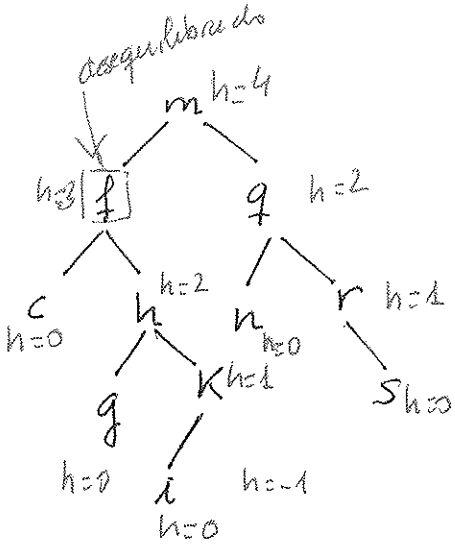
SI AVL

2 AVL

LECCION 28, AVL



SI AVL



Funciones que nos interesa de un AVL

- Buscar un elemento
- Insertar
- Borrar

Representación del AVL

```
template <class T>
struct info-nodo_AVL {
    T et;
    info-nodo-AVL * hizq, *hder, *padre
    int altura;
};
```

Busqueda en un AVL — es identica a la busqueda en un ABB

```
template <class T>
info-nodo_AVL<T> * Busqueda (info-nodo_AVL<T>*n, T x) {
        if (n==0) return 0;
        else {
            if (n->et==x) return n;
            else if (x<n->et)
                    return Busqueda (n->hizq, x);
                else return Busqueda (n->hder, x);
        }
}
```

LECCION 22: AVL

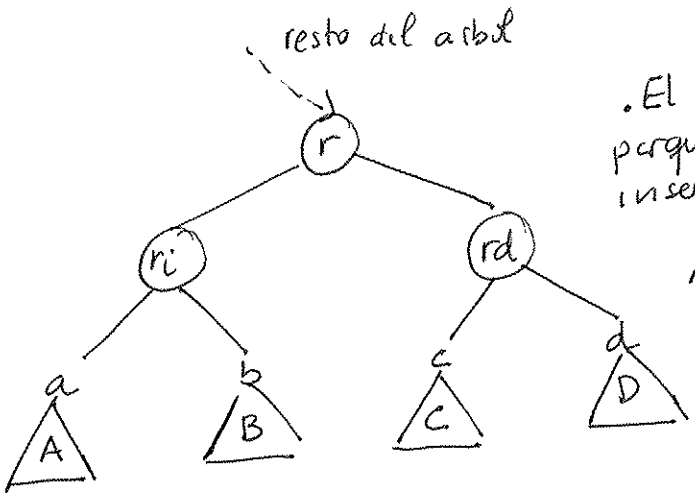
INSERCION:

Para llevar a cabo la insercion de un nuevo elemento seguiremos los siguientes pasos:

1: Realizar la insercion igual que en los ABB

2: Equilibrar el arbol.

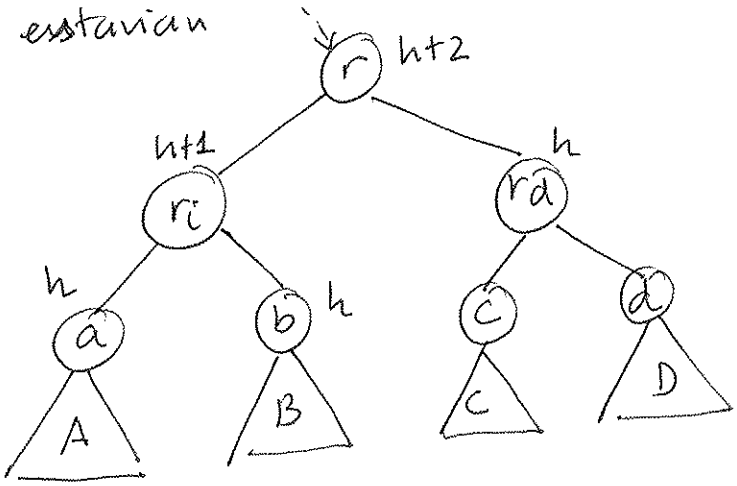
↳. Al realizar la insercion, recorremos los nodos que va desde el nuevo nodo insertado hasta la raiz, comprobando si ha cambiado la altura y si es necesario equilibrar el árbol.



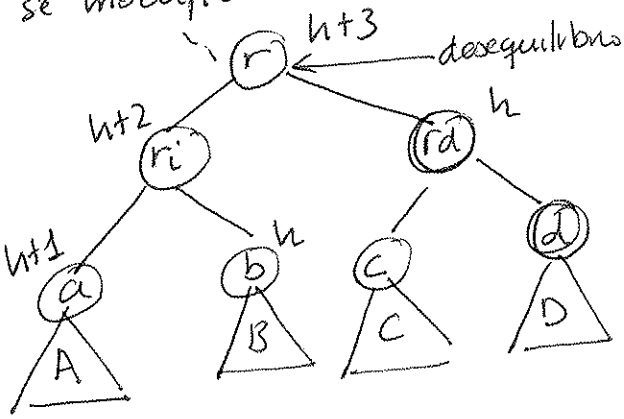
. El desequilibrio puede ocurrir porque el nuevo nodo se haya insertado en A o en B o en C o en D

CASO A: El nuevo nodo se ha insertado en A generando un desequilibrio en r

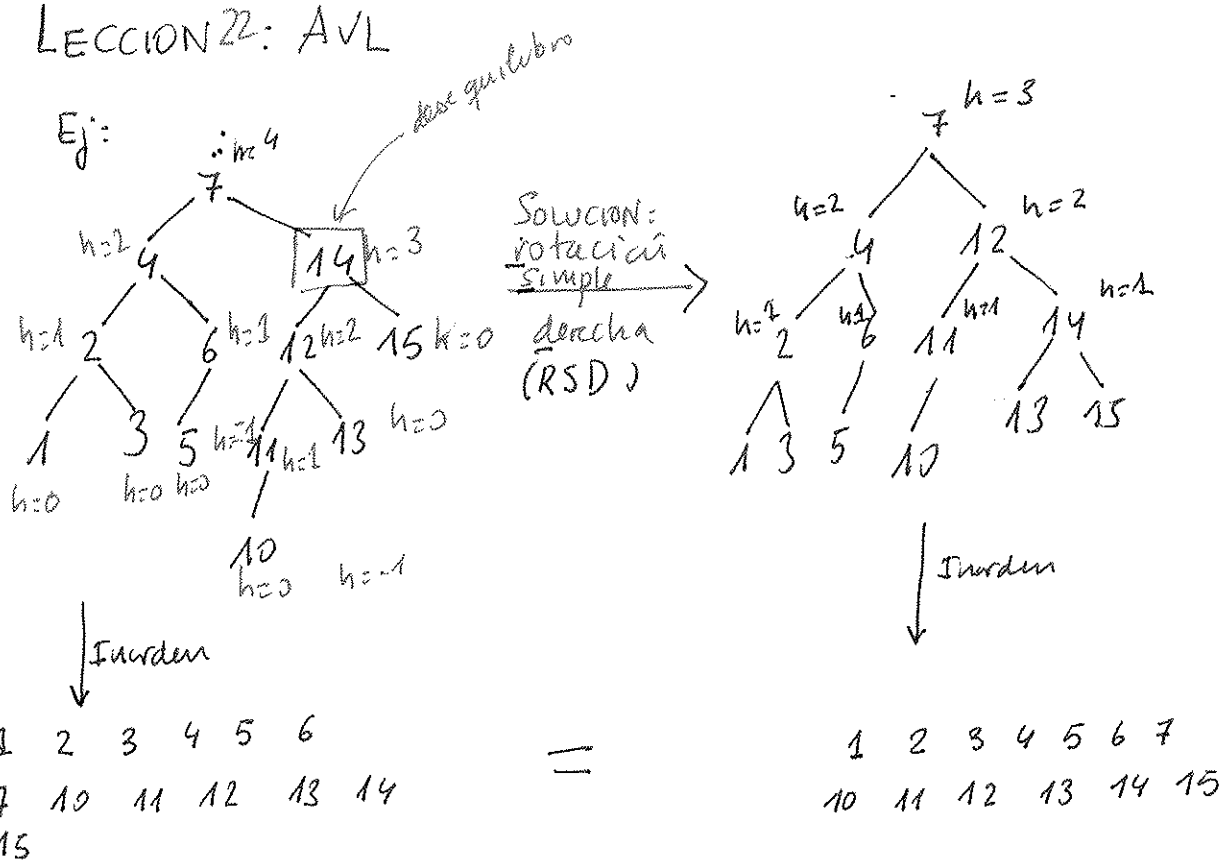
Antes de insertar las alturas estaban



Despues de insertar las alturas se modifican



LECCION 22: AVL



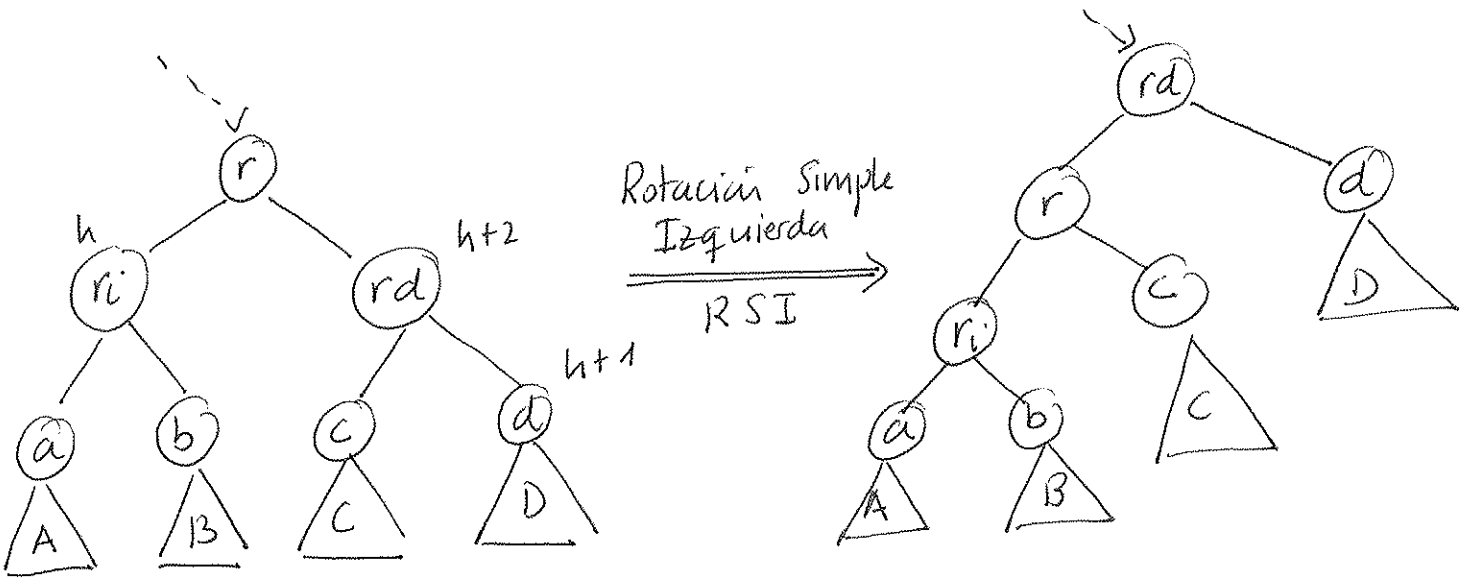
CODIGO PARA REALIZAR RSD

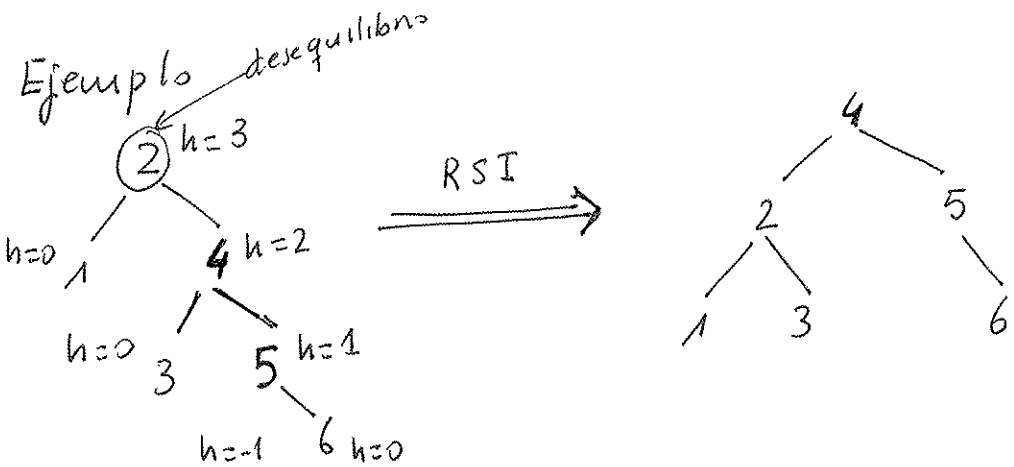
```
template <class T>
void SimpleDerecha (info_nodo_AVL <T> * &n){//en el ejemplo n=14
        info_nodo_AVL <T> * aux = n->hizq; // 12
        info_nodo_AVL <T> * padre = n->padre; // 7
        n->hizq = aux->hder; // a 14 le ponemos como hizq 13
        if (n->hizq != 0)
            n->hizq->padre = n; // el padre de 13 es 14.

        n->padre = aux; // el padre de 14 ahora sera 12
        aux->padre = padre; // el padre de 12 es 7
        aux->hder = n; // 12 tiene como hder 14
        n = aux; // modificamos n para que apunte a 12
        ActualizarAltura (n->hder);
}
template <class T>
void ActualizarAltura (info_nodo_AVL <T> *&n){
    if (n!=0){
        n->altura = std::max(Altura(n->hizq), Altura(n->hder)) + 1;
        ActualizarAltura (n->padre);
    }
}
```

LECCION 22: AVL

CASO D - La insercion se ha realizado en el árbol D

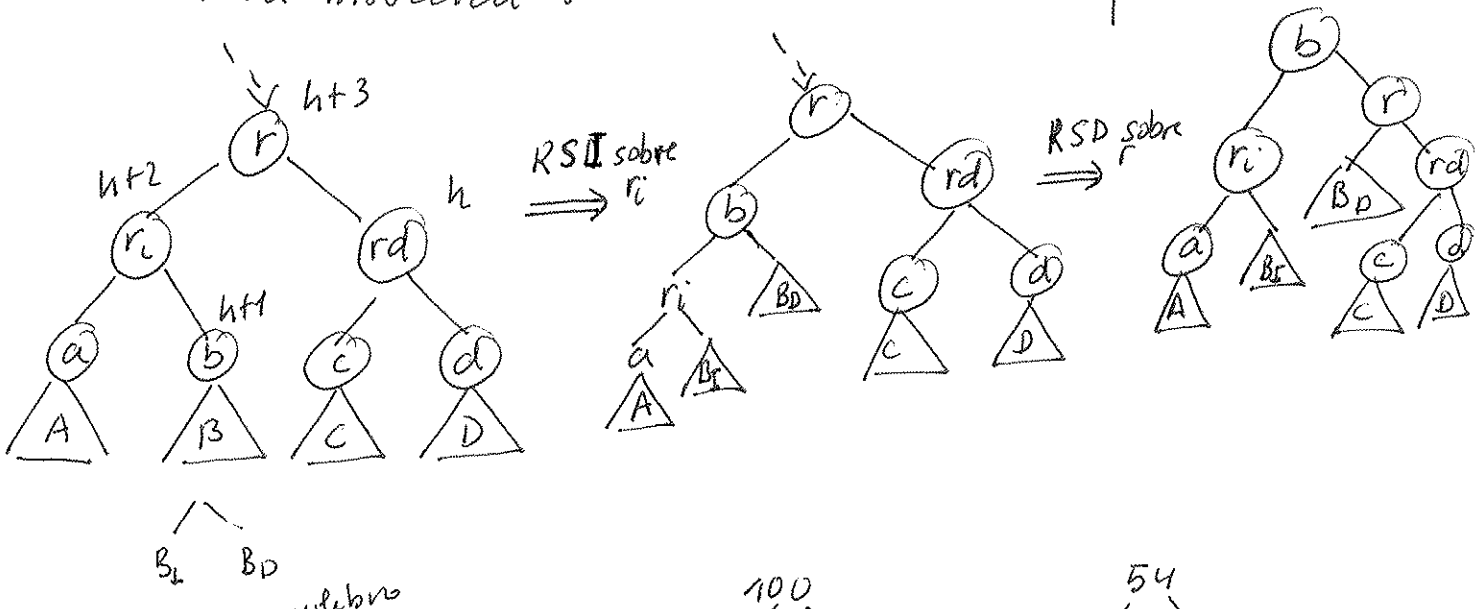




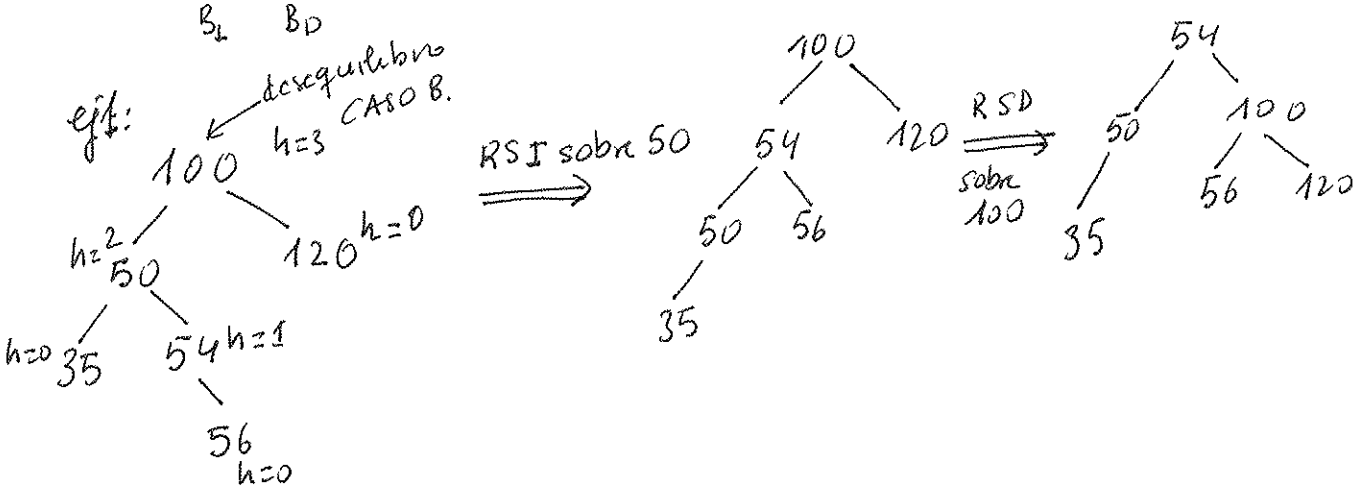

```
template <class T>
void SimpleIzquierda(info_nodo_AVL<T>* &n){ //en el ej. 2
    info_nodo_AVL<T> * aux = n->hder; // 4 en el ej
    info_nodo_AVL<T>* padre = n->padre; // nulo en el ej
    n->hder = aux->hizq; // a 2 se le pone como hder 3
    if (n->hder != 0)
        n->hder->padre = n; //el padre de 3 ahora es 2
    n->padre = aux; // a 2 se le pone como padre 4
    aux->padre = padre; // el padre de 4 es nulo
    aux->hizq = n;
    n = aux;
    ActualizarAltura(n->hizq);
```

LECCION 11: AVL

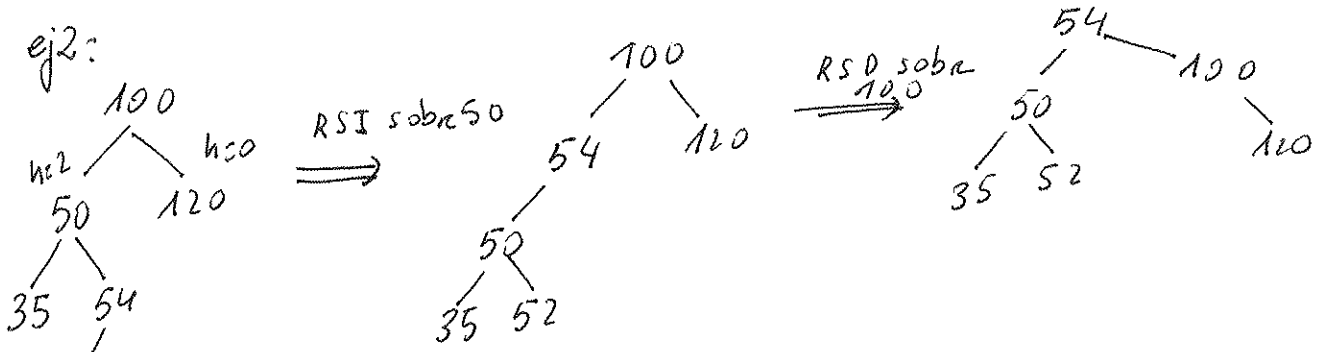
CASO B.- La inserccion se realiza en el arbol B_i { RSI sobre hizq / RSD sobre el nodo



ejt:



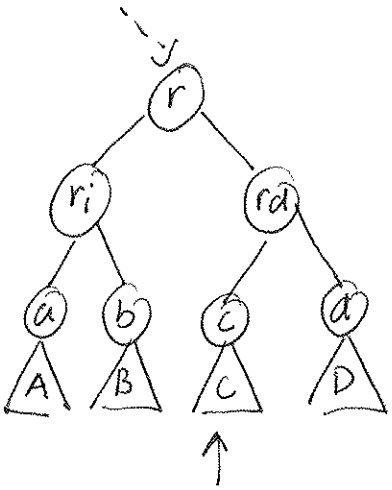
ej2:



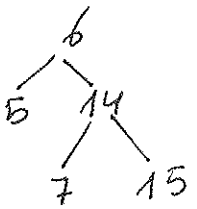

```
template <class T>
void DobleIzquierdaDerecha(info_nodo_AVL<T>* &n) {
    SimpleIzquierda(n->hizq);
    SimpleDerecha(n);
}
```

LECCION 2: AVL

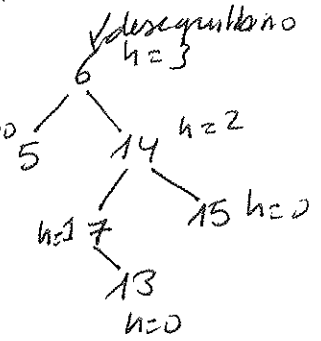
Caso C: La inserción se realiza sobre el árbol C. { 1º RSD sobre hder
2º RSI sobre el nodo desequilibrado



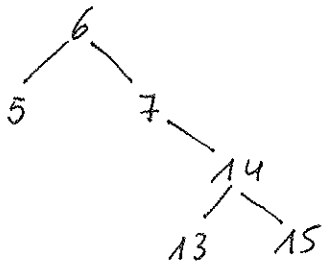
Ejemplo:



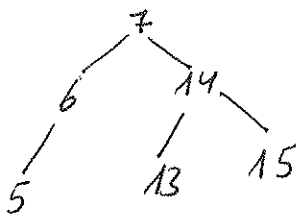
Insertamos 13



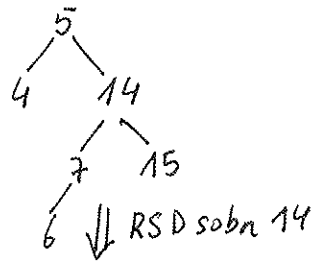
1º RSD sobre 14



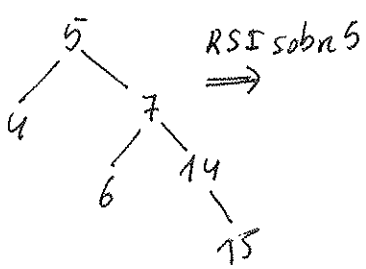
2º RSI sobre 6



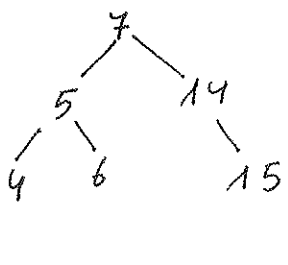
Ejemplo 2



⇓ RSD sobre 14



RSI sobre 5 ⟹




```
template <class T>
void DobleDerecha-Izquierda(info_nodo_AVL<T>
                            &n){

    SimpleDerecha(n->hder);
    SimpleIzquierda(n);
}
```

LECCION 22: AVL

Funcion de Insercion (recursiva)

1) Buscamos el pto de insercion 2) Aprovechamos la vuelta de la recursion para realizar los ajustes.

true: si aumenta la h

```
bool  InsertarAVL ( info_nodo_AVL * & raiz, TX) {

    bool hcrecido = false; ← quick.
    if (raiz == 0){
        raiz = new info_nodo AVL(x);
        raiz-> altura = 0;
        hcrecido = true;
        return hcrecido;

    }
    else  if (x < raiz -> et      ){
        if (InsertarAVL ( raiz->hizq, x)){
        switch ( Altura( raiz->hizq) - Altura (raiz->hder)) {

            case 0: return false;
                    break;

            case 1: raiz->altura++;
                    return true;

            case 2:
    /* CASO A */ if ( Altura(raiz->hizq->hizq) > Altura(raiz->hizq->hder))
                        SimpleDerecha ( raiz);

                 else  DobleIzquierda - Derecha (raiz)
    /* CASO B */
                 return false;


        }
    }
}
else if (x > raiz -> et ){
    if (InsertarAVL (raiz->hder, x)){
     switch (Altura (raiz->hder) - Altura (raiz->hizq) ) {
        case 0 : return false;

        case 1 : raiz->altura++;
                 return true;

        case 2:
```

9AVL

LECCION : AVL

```
        case 2:
/*CASO D*/ if (Altura(raiz->hder->hizq) < Altura(raiz->hder->hder))
                SimpleIzquierda(raiz)
            else
/*CASO C*/      DobleDerecha_Izquierda(raiz)
            return false;
    }
  }
}
```

Ejemplo: {1, 2, 3, 4, 5, 6, 7, 15, 14, 13, 12, 11, 10, 9}