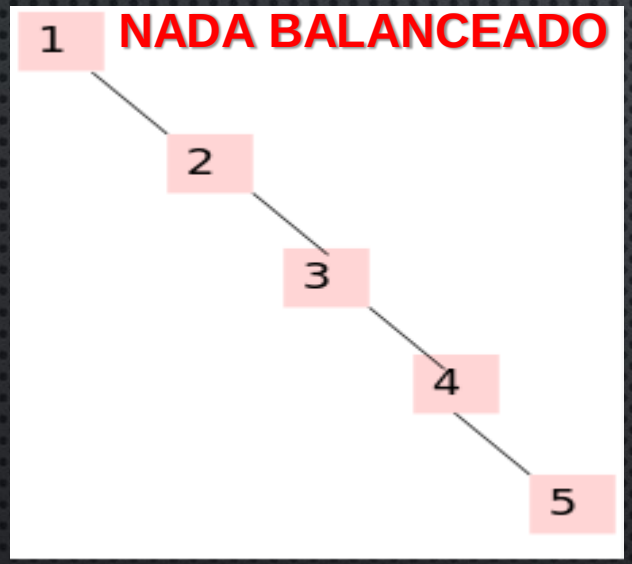
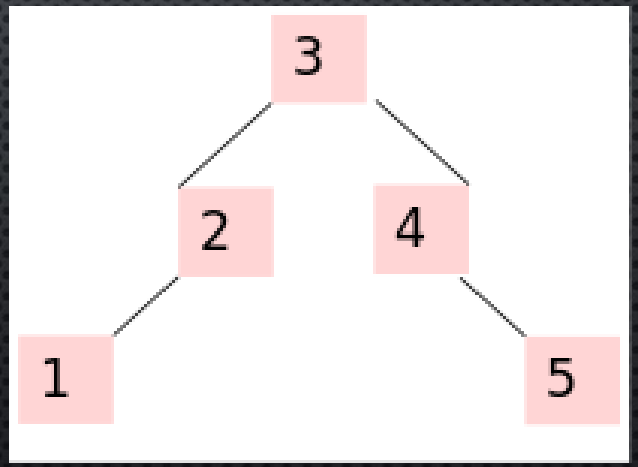
ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

Arbol Binarios de Búsqueda.- En promedio la eficiencia de búsqueda de una clave es $O(\log_2(n))$, pero en el peor de los casos es $O(n)$

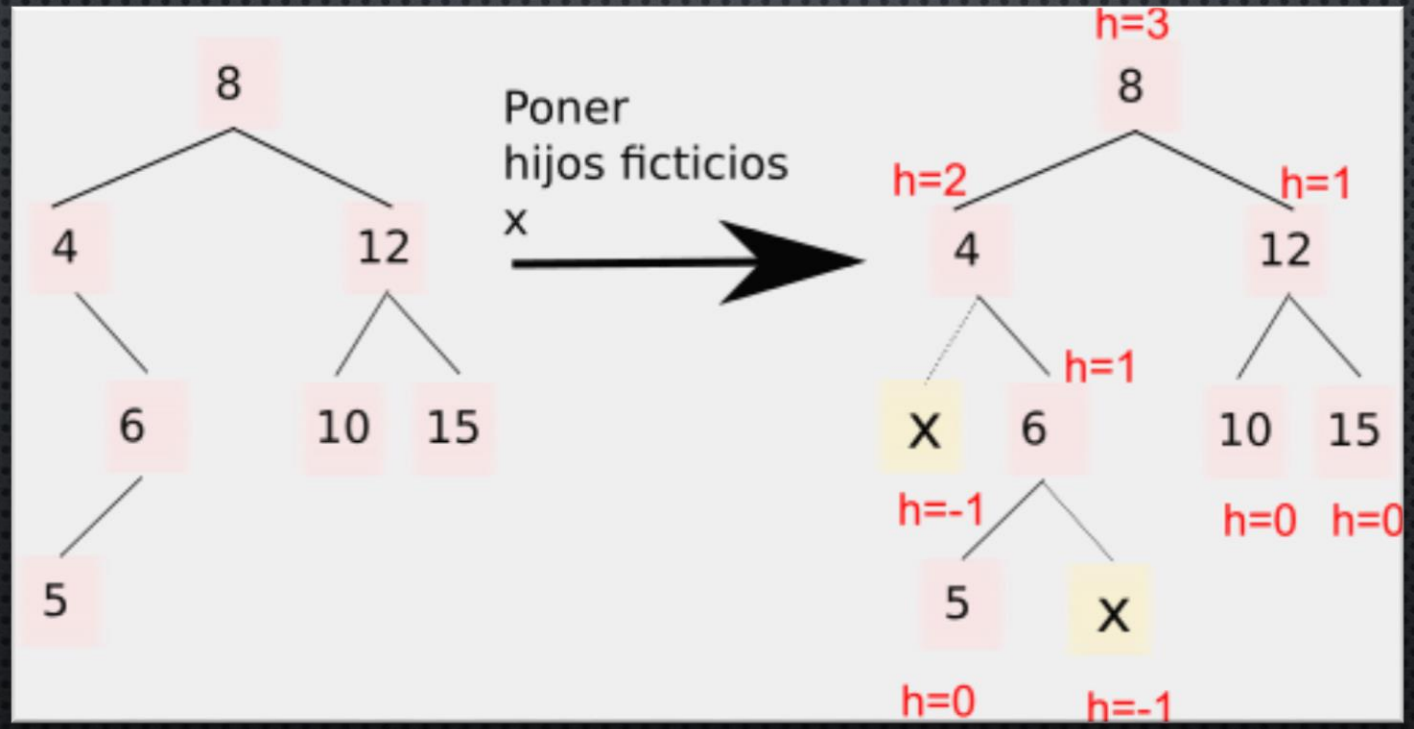


Objetivo: Balancear el ABB para lograr tiempos de búsqueda $O(\log_2(n))$



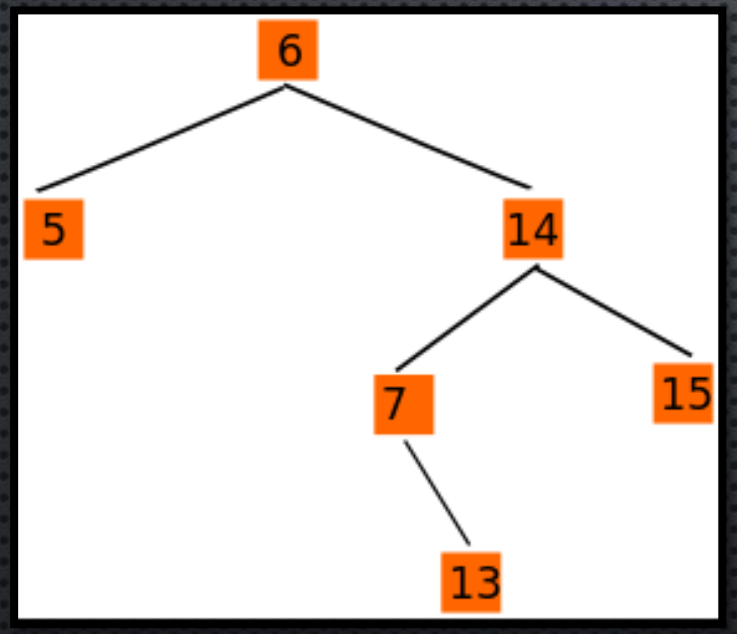
ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

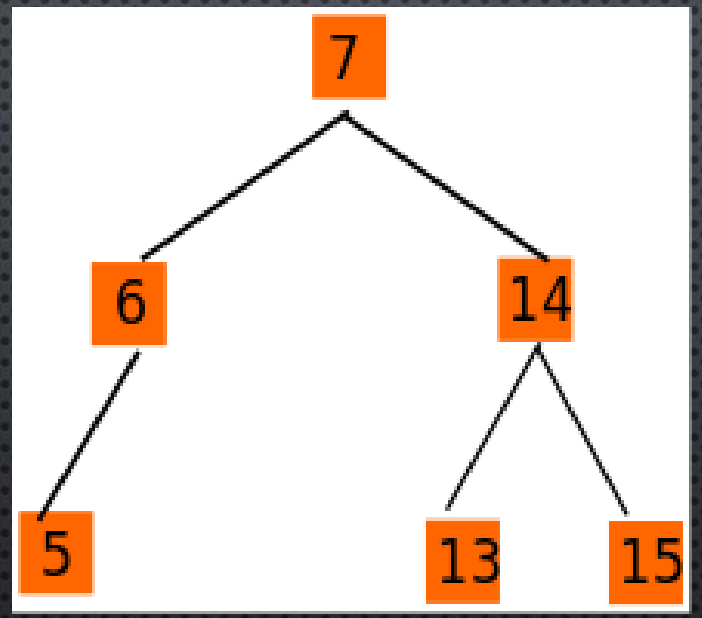
Arbol AVL (Adelson, Velskii,Landis).- Un AVL es un ABB que cumple que las diferencia en altura de los subárboles que cuelgan de un mismo nodo no puede ser mayor que 1.
La altura de un árbol vacío es -1.
ALTURA: Longitud del camino más largo que va desde un nodo a un nodo hoja.

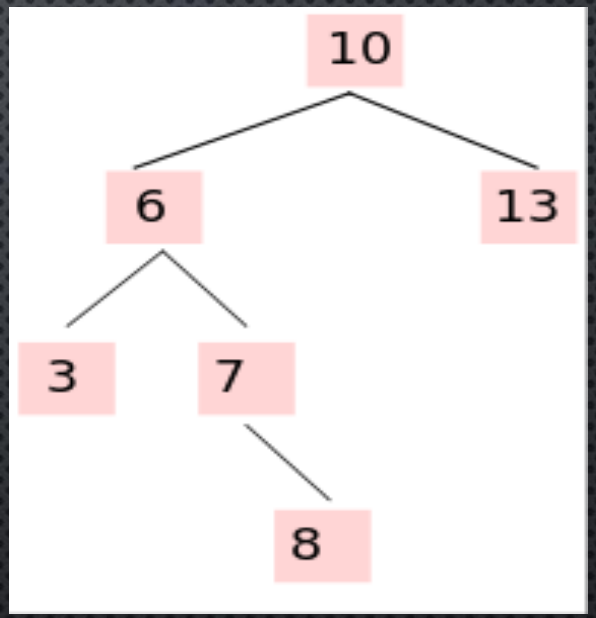


ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

Indicar cuales de los siguientes árboles son AVL.

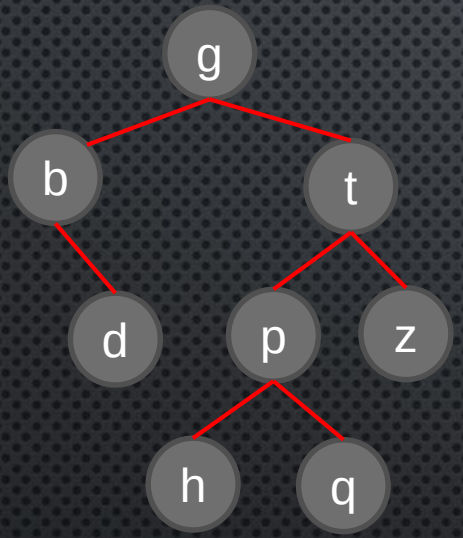


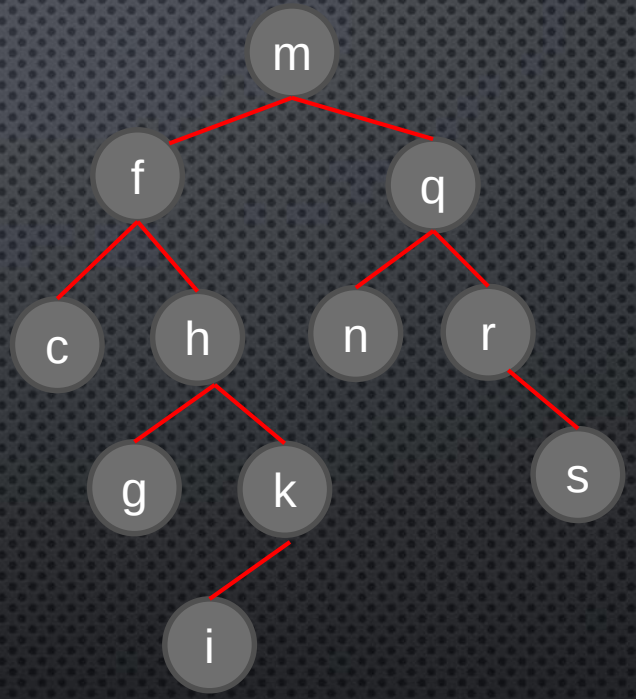




ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

Indicar cuales de los siguientes árboles son AVL.





ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

Funciones que nos interesan de un AVL.

- Buscar un elemento
- Insertar un elemento
- Borrar un elemento

Reprsentación de un AVL

```
template <class T>
struct info_nodo_AVL{
   T et:
   info_nodo_AVL<T>*padre,*hizq,*hder;
   int altura;
};
```

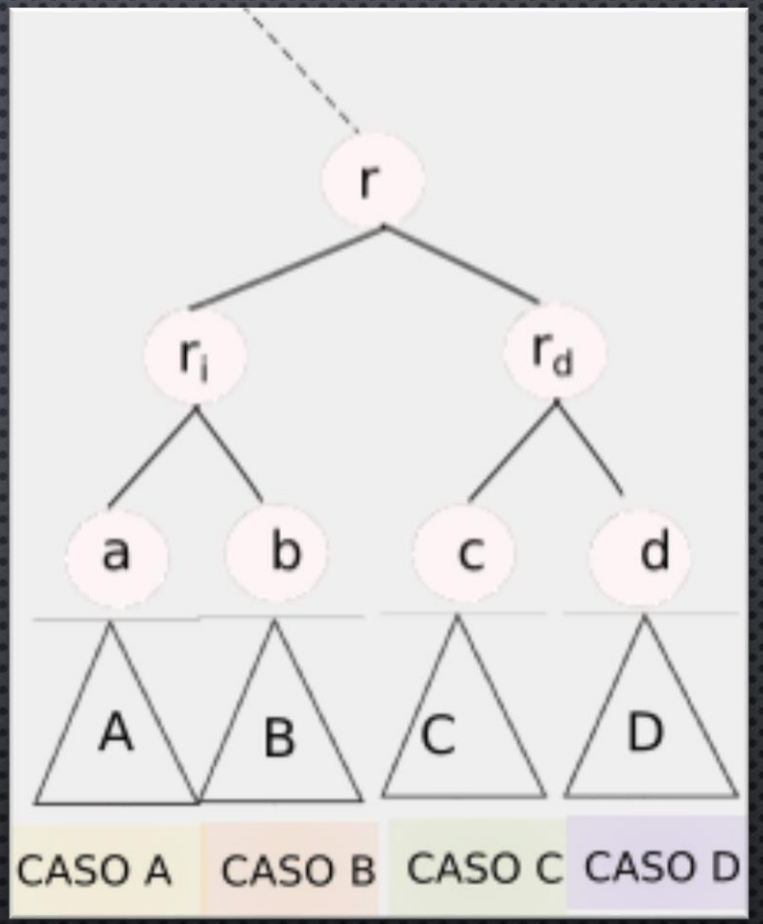
```
template <class T>
info_nodo_AVL<T>* Busqueda(info_nodo_AVL<T>*n,const T &x){
1.    if (n==nullptr) return nullptr;
2.    else
3.       if (n->et==x) return n;
4.       else
5.         if (n->et<x) return Busqueda(n->hder,x);
6.         else
7.           return Busqueda(n->hizq,x);
8. }
```

ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

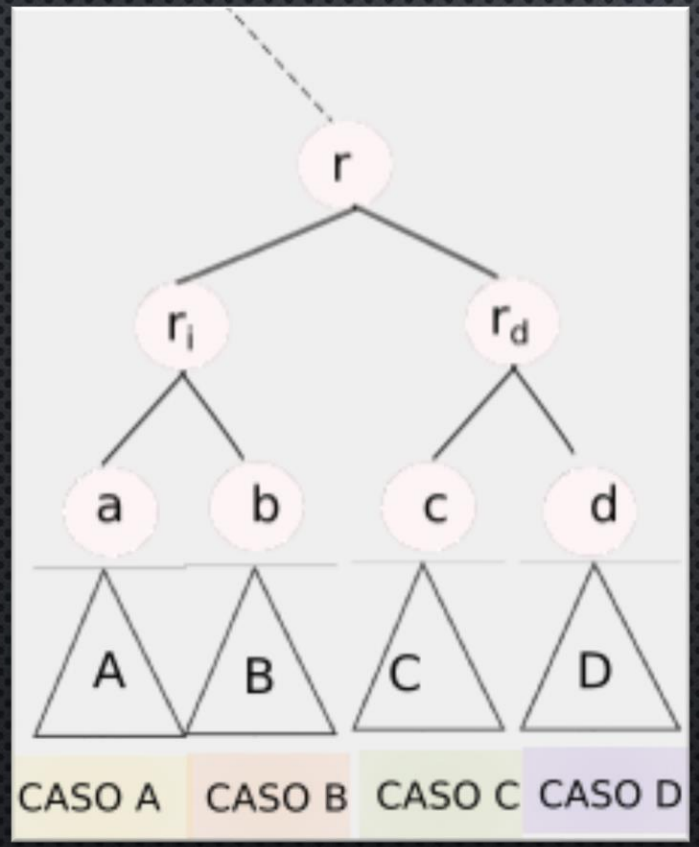
Insertar un elemento

1. Realizar la inserción igual que en un ABB
2. Equilibrar el árbol.-

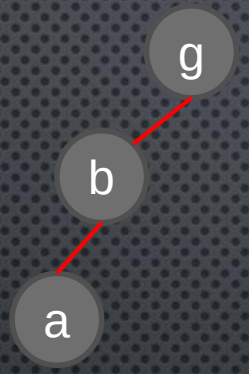
   Al realizar la inserción, recorremos los nodos que va desde el nuevo nodo insertado hasta la raíz comprobando si ha cambiado la altura y si es necesario equilibrar.



Insertar un elemento.- Caso A



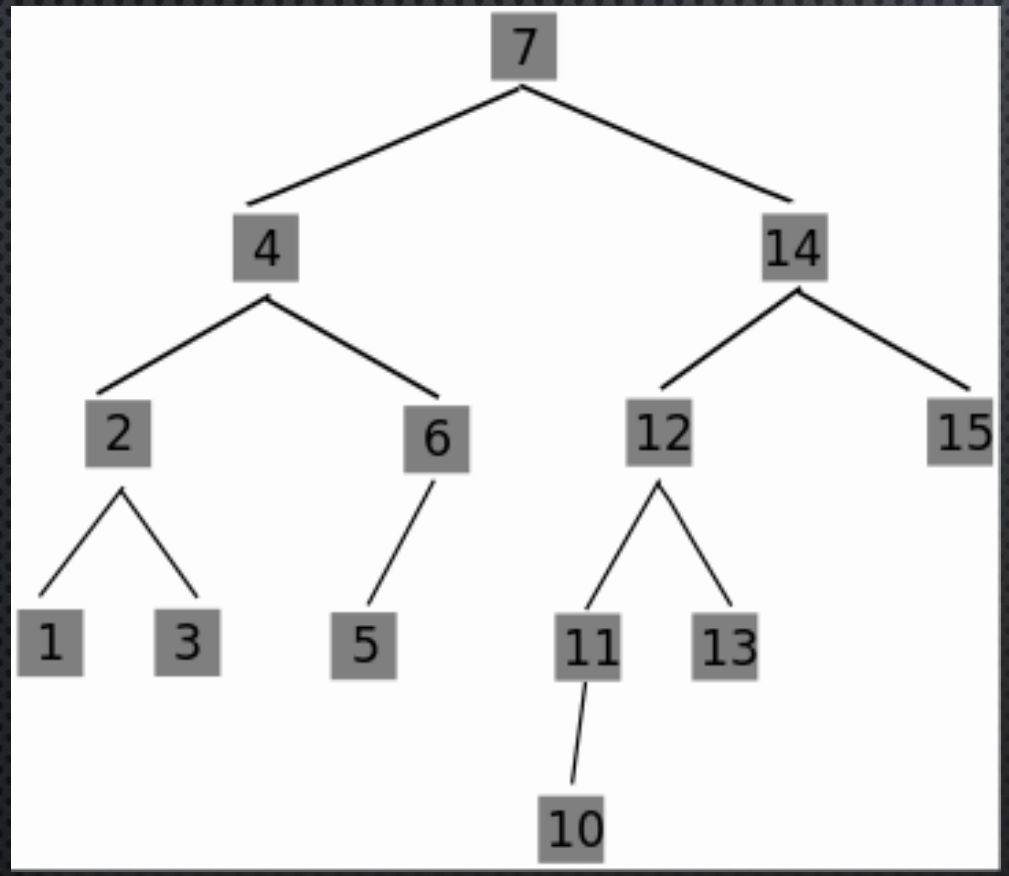
Ejemplo. La forma más trivial.



Se aplica Rotación Simple a Derecha (RSD)

ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

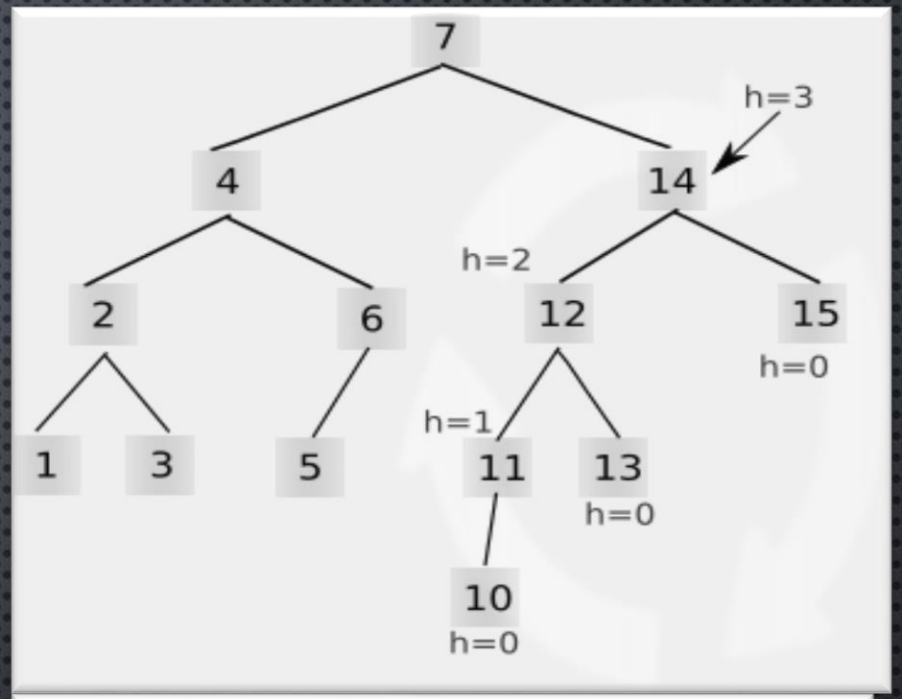
Insertar un elemento.- Caso A

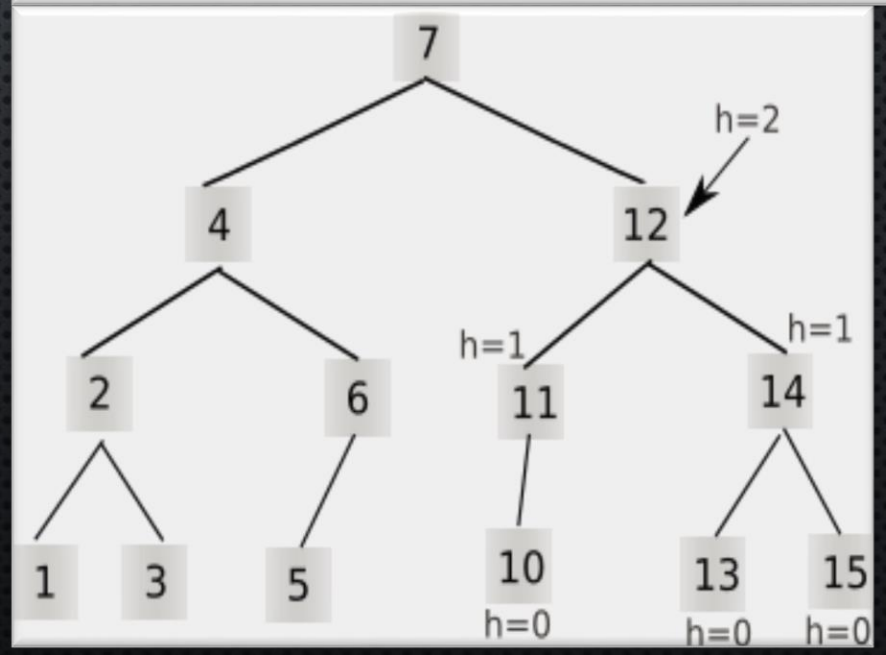


Insertar un elemento.- Rotación simple a derecha

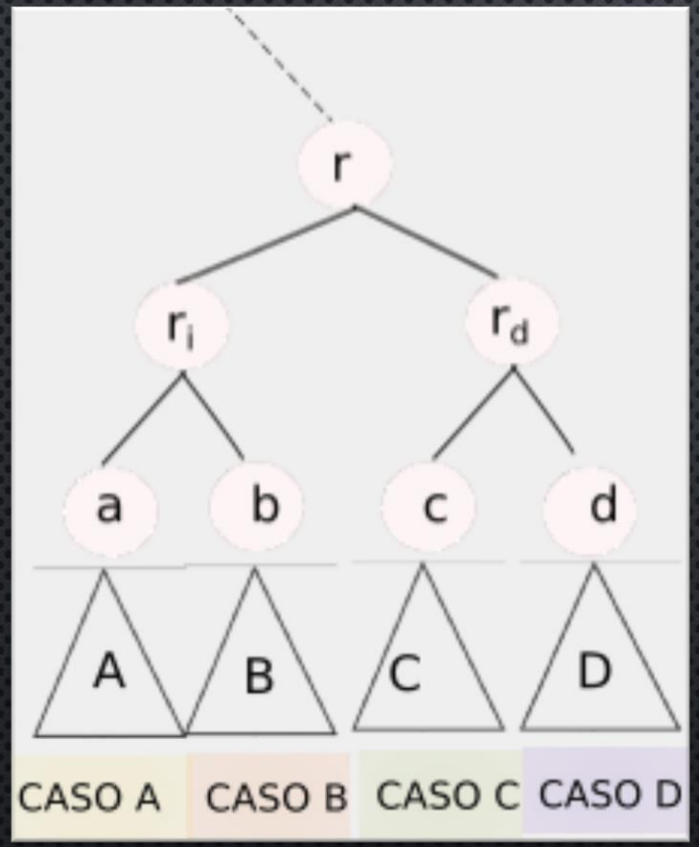
```
template <class T>
void SimpleDerecha (info_nodo_AVL<T> * & n)
{
1.      info_nodo_AVL<T> * aux = n->izq;
2.      info_nodo_AVL<T> * padre = n->padre;
3.      n->hijoizq = aux->hder;
4.      if (n->hijoizq != nullptr)
5.          n->hizq->padre = n;
6.      n->padre = aux;
7.      aux->padre = padre;
8.      aux->hder = n;
9.      n = aux;
10.    ActualizarAltura(n->hder);
}

template <class T>
void ActualizarAltura (info_nodo_AVL<T> * & n) {
{
 if (n != 0) {
  n->altura = std::max(Altura(n->hijoizq),
                       Altura(n->hijoder))+1;
  ActualizarAltura(n->padre);}}
```

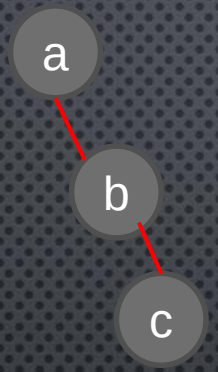




Insertar un elemento.- Caso D



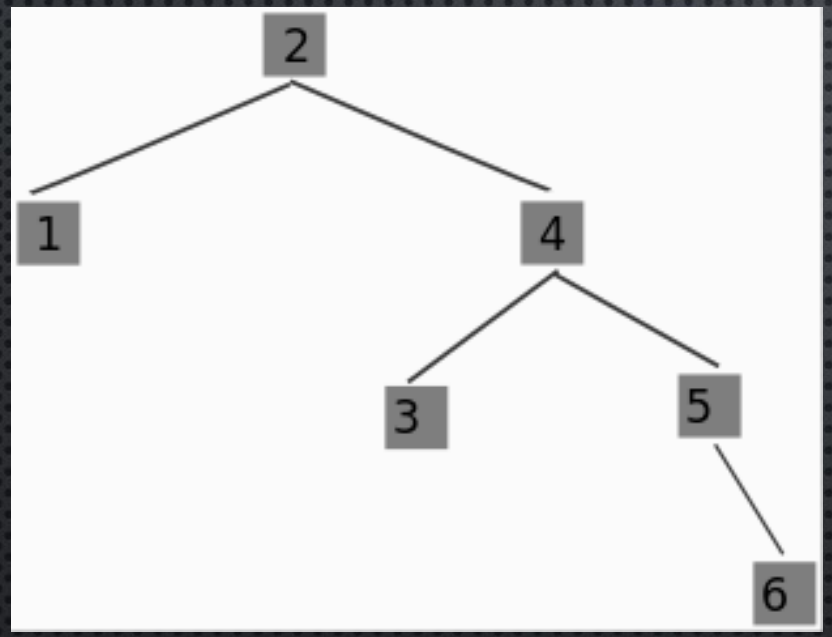
Ejemplo. La forma más trivial.



Se aplica Rotación Simple a Izquierda (RSI)

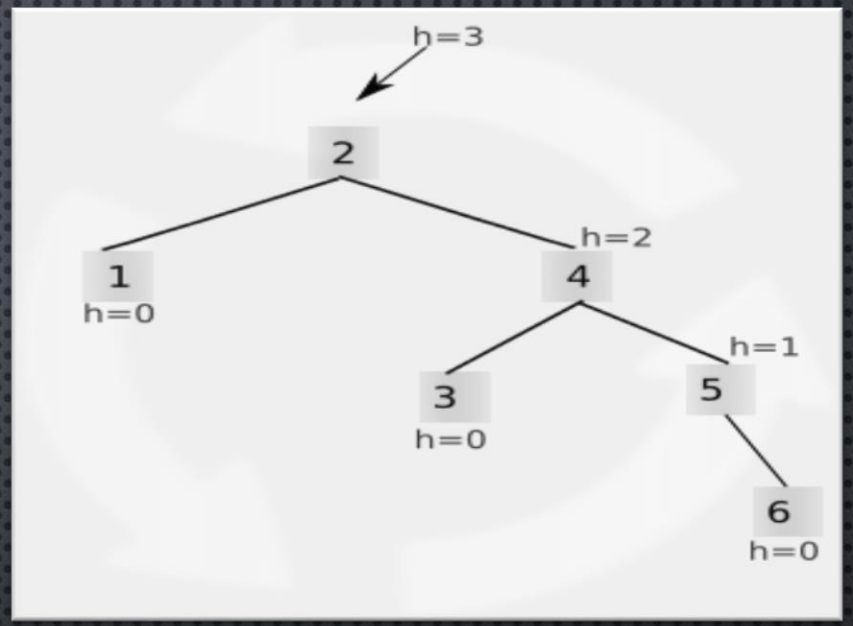
ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

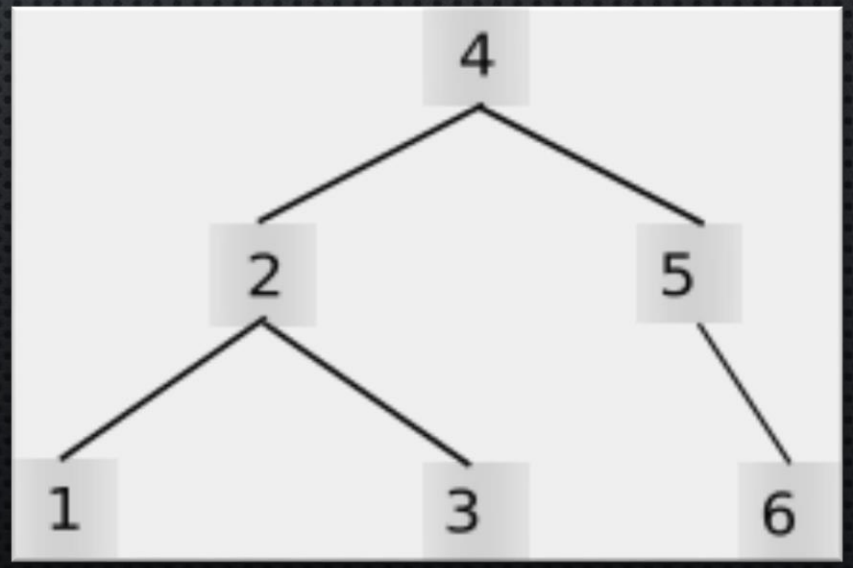
Insertar un elemento.- Caso D



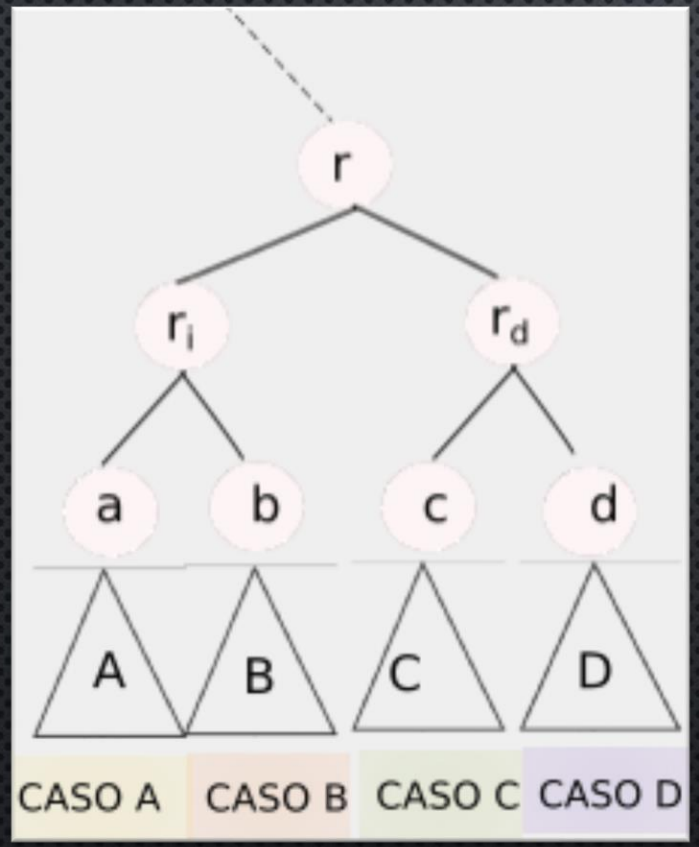
Insertar un elemento.- Rotación simple a derecha

```
template <class T>
void SimpleIzquierda (info_nodo_AVL<T> * & n)
{
1.      info_nodo_AVL<T> * aux = n->hder;
2.      info_nodo_AVL<T> * padre = n->padre;
3.      n->hder = aux->hizq;
4.      if (n->hder != nullptr)
5.          n->hder->padre = n;
6.      n->padre = aux;
7.      aux->padre = padre;
8.      aux->hizq = n;
9.      n = aux;
10.    ActualizarAltura(n->hizq);
}
```





Insertar un elemento.- Caso B

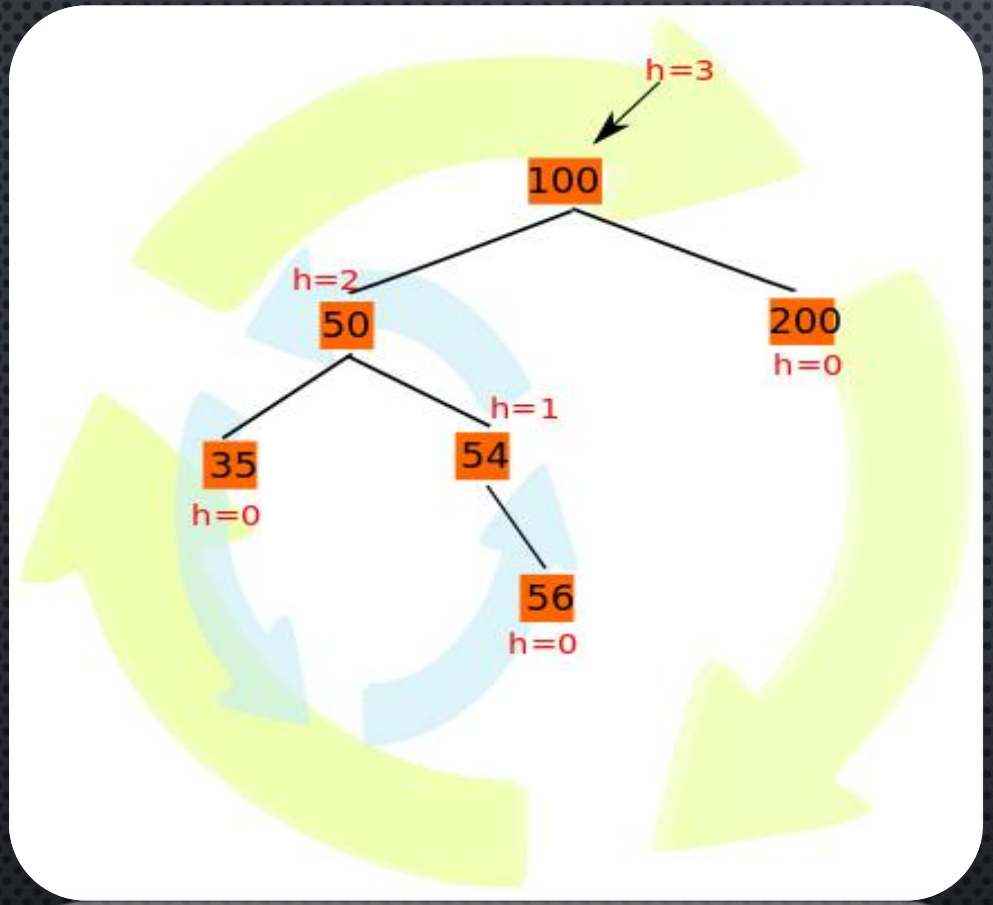


Ejemplo. La forma más trivial.

Se aplica:

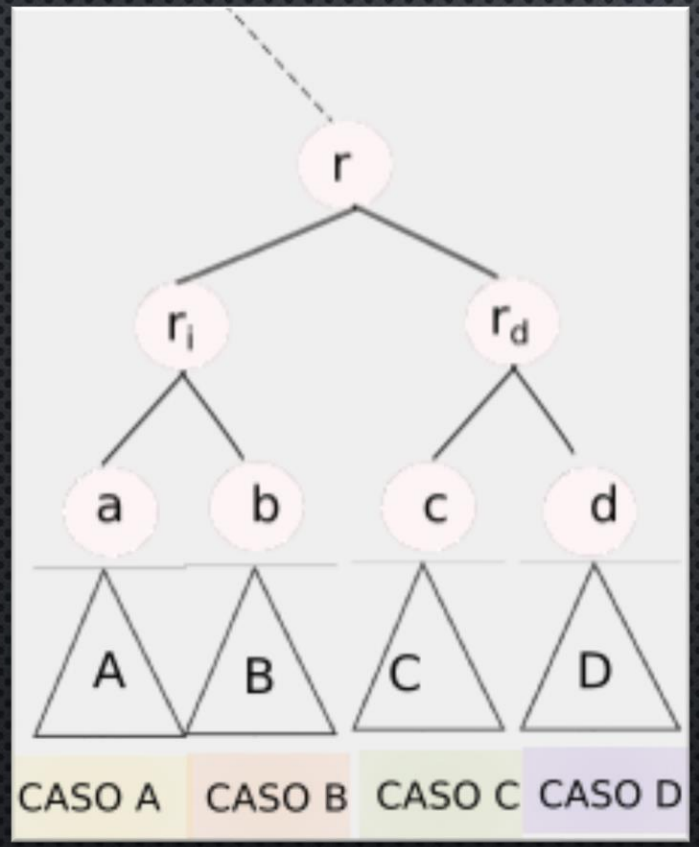
1. Rotación Simple a Izquierda (RSI)
2. Rotación Simple a Derecha (RSD)

Insertar un elemento.- Caso B

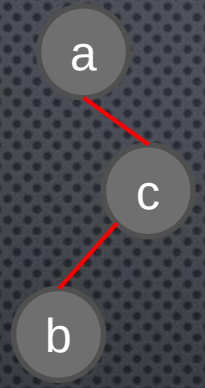



```
template <class T>
void Doble_IzquierdaDerecha (info_nodo_AVL<T> * & n)
{  SimpleIzquierda (n->hizq);
   SimpleDerecha(n)
};
```

Insertar un elemento.- Caso C



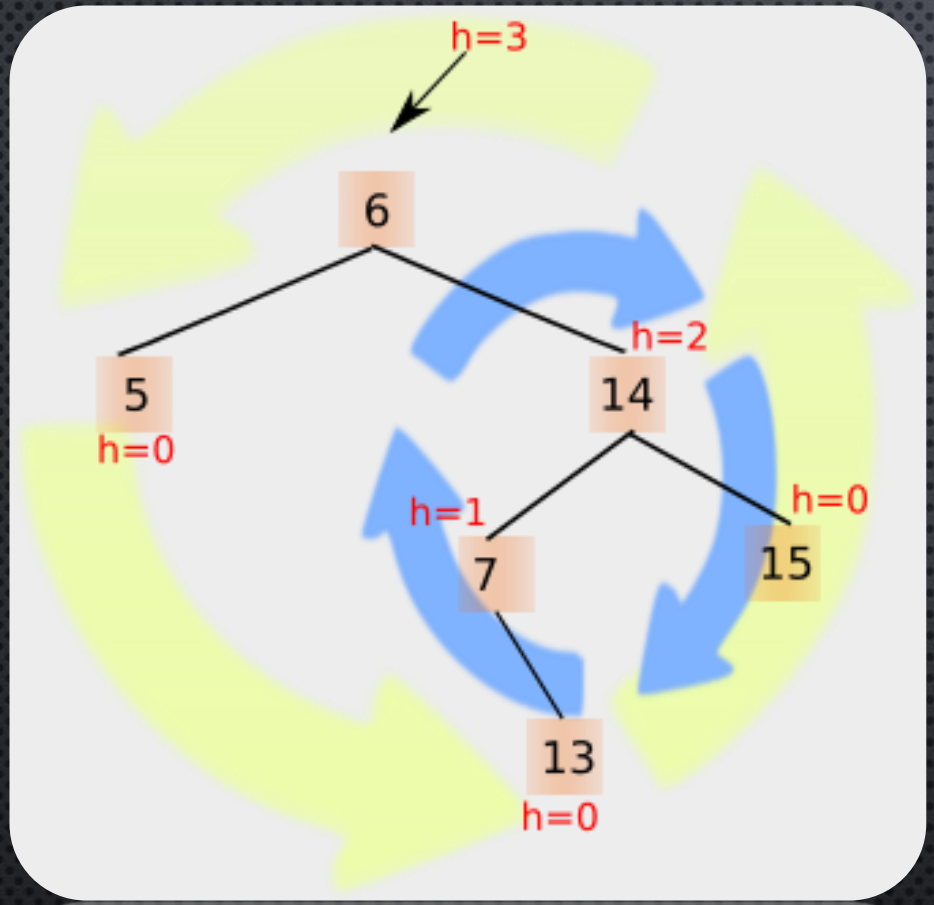
Ejemplo. La forma más trivial.



Se aplica:

1. Rotación Simple a Derecha(RSD)
2. Rotación Simple a Izquierda (RSI)

Insertar un elemento.- Caso C




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

Algoritmo de Inserción

```
template <class T>
bool InsertarAVL (info_nodo_AVL<T> * & raiz, T x) {
 if (raiz == nullptr) {
  raiz = new info_nodo_AVL(x);
  raiz->altura = 0;
  return true;
 }
 else {
  if (x < raiz->et){
     if (InsertarAVL(raiz->hizq, x)) {
       if (raiz->hizq !=nullptr)
        raiz->hizq->padre = raiz;
       switch (Altura(raiz->hizq) - Altura(raiz->hder)){
        case 0:return false;
        case 1:
          raiz->altura++;
          return true;
         case 2:
          //CASO A
         if (Altura(raiz->hizq->hizq) > Altura(raiz->hizq->hder))
           SimpleDerecha(raiz);
         else //CASO B
```

Algoritmo de Inserción

```
template <class T>
bool InsertarAVL (info_nodo_AVL<T> * & raiz, T x) {
 ….
else //CASO B
 Doble_IzquierdaDerecha(raiz);
return false;
}}}
else { // x es mayor que la etiqueta
  if (InsertarAVL(raiz->hder, x)) {
   if (raiz->hder != nullptr)
    raiz->hder->padre = raiz;
   switch (Altura(raiz->hder) - Altura(raiz->hizq)){
   case 0: return false;
   case 1: raiz->altura++;  return true;
   case 2:
   /* CASO D */
   if (Altura(raiz->hder->hder)>Altura(raiz->hder->hizq))
    SimpleIzquierda(raiz);
   else /* CASO C */
    Doble_DerechoIzquierda(raiz);
   return false;}}}}
}
```

ARBOLES: ARBOLES BINARIOS DE BUSQUEDA EQUILIBRADOS (AVL)

Ejercicio Construir el AVL con las siguientes claves {1,2,3,4,5,6,7,15,14,13,12,11,10,9}. Cuando haya que aplicar rotaciones especificar el tipo de rotación que se debe realizar.