



Árboles

- Presentar la estructura no lineal más importante en computación
- Mostrar la especificación e implementación de varios tipos de árboles
- Algoritmos de manipulación de árboles

Contenido

- 1. Terminología fundamental**
 - 1.1. Recorridos de un árbol
- 2. Árboles binarios**
 - 2.1. Definición
 - 2.2. Especificación
 - 2.3. Implementación
- 3. Heap**
- 4. Árboles binarios de búsqueda**
 - 4.1. Definición
 - 4.2. Especificación
- 5. Árboles binarios equilibrados**
 - 5.1. Árboles AVL
- 6. Árboles generales**
 - 6.1. Especificación

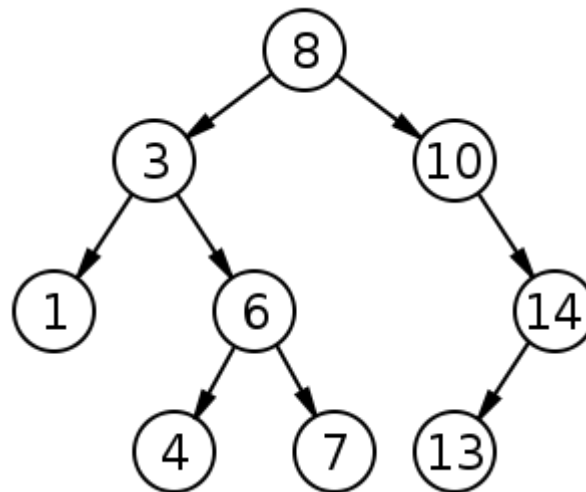
Árbol Binario de Búsqueda

- También llamados BST (acrónimo del inglés Binary Search Tree)
- Estructura de datos básica para almacenar elementos que están clasificados siguiendo algún orden lineal.
- **Propiedades:**
 - Para todo nodo N del árbol,
 - Todos los valores de los nodos del subárbol izquierdo de N deben ser *menores* al valor del nodo N, y
 - Todos los valores de los nodos del subárbol derecho de N deben ser *mayores o iguales* al valor del nodo N
- Un recorrido en *inorden* del árbol proporciona una lista en orden ascendente de los valores almacenados en los nodos

Árbol Binario de Búsqueda

■ Ejemplo:

Un árbol binario de búsqueda de tamaño 9 y profundidad 3, con raíz 8 y hojas 1, 4, 7 y 13



Árbol Binario de Búsqueda

Operaciones:

- *Búsqueda*: determinar si x está en el árbol
 - Si $x = \text{raíz} \Rightarrow$ localizado
 - Si $x < \text{raíz} \Rightarrow$ buscar en el subárbol izquierdo
 - Si $x \geq \text{raíz} \Rightarrow$ buscar en el subárbol derecho

Consiste en acceder a la raíz del árbol; si el elemento a localizar coincide con éste la búsqueda ha concluido con éxito; si el elemento es menor se busca en el subárbol izquierdo y si es mayor en el derecho. Si se alcanza un nodo hoja y el elemento no ha sido encontrado se supone que no existe en el árbol.

Cabe destacar que la búsqueda en este tipo de árboles es muy eficiente, representa una *función logarítmica*. El máximo número de comparaciones que necesitaríamos para saber si un elemento se encuentra en un árbol binario de búsqueda estaría entre $\lceil \log_2(N+1) \rceil$ y N , siendo N el número de nodos. La búsqueda de un elemento en un ABB (Árbol Binario de Búsqueda) se puede realizar de dos formas, iterativa o recursiva.

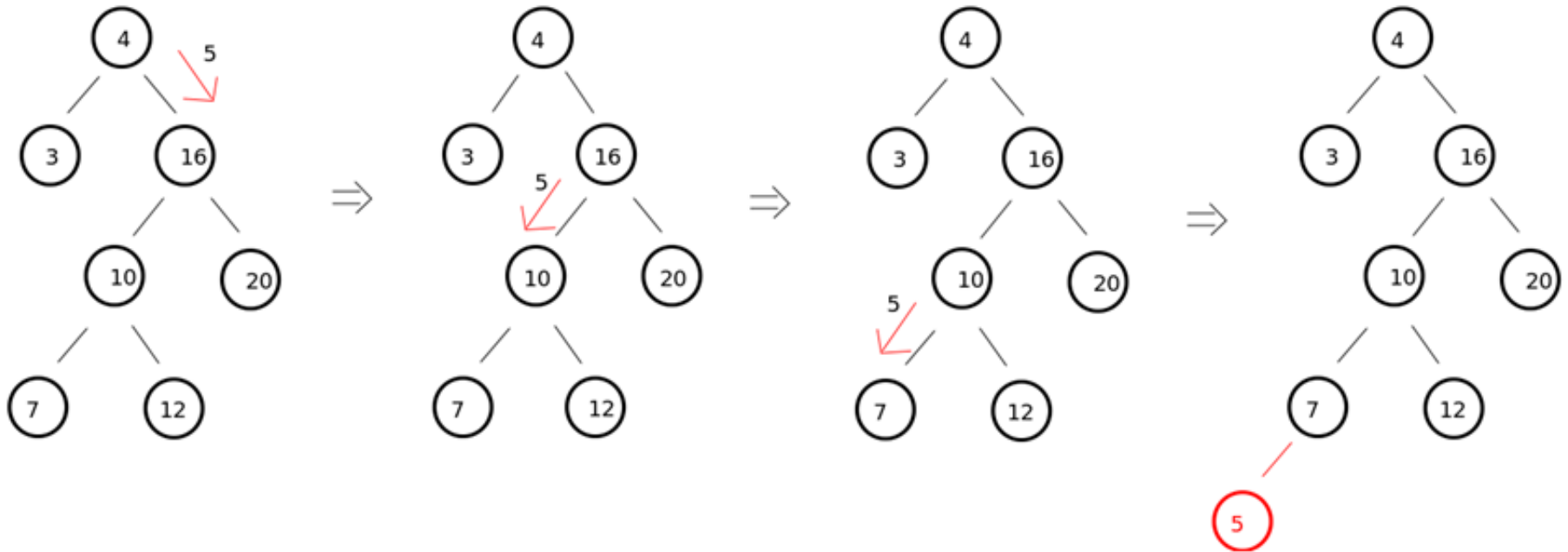
Árbol Binario de Búsqueda

- *Inserción:* añadir un nuevo elemento x al árbol
 - Avanzar en el árbol comparando x con raíz
 - Repetir el paso 1 hasta que el subárbol donde deba insertarse sea el árbol vacío
 - Insertar el elemento

La inserción es similar a la búsqueda y se puede dar una solución tanto iterativa como recursiva. Si tenemos inicialmente un árbol vacío se crea un nuevo nodo con el elemento a insertar. Si no lo está, se comprueba si el elemento dado es menor que la raíz del árbol inicial con lo que se inserta en el subárbol izquierdo y si es mayor o igual se inserta en el subárbol derecho.

Árbol Binario de Búsqueda

■ Ejemplo de inserción



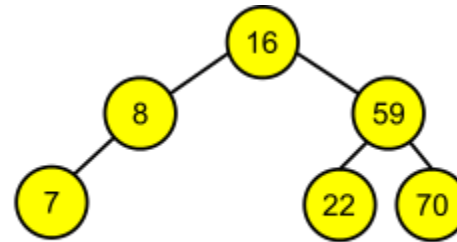
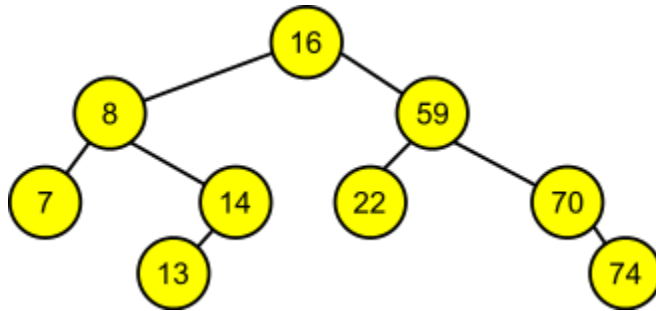
Árbol Binario de Búsqueda

- *Eliminación*: borrar un elemento x del árbol
 - Si x es hoja \Rightarrow se suprime x .
 - Si el elemento a borrar tiene un solo descendiente \Rightarrow se sustituye por ese descendiente.
 - Si el elemento a borrar tiene los dos descendientes \Rightarrow se sustituye por el menor elemento del subárbol derecho. Después se elimina el nodo correspondiente a dicho elemento.

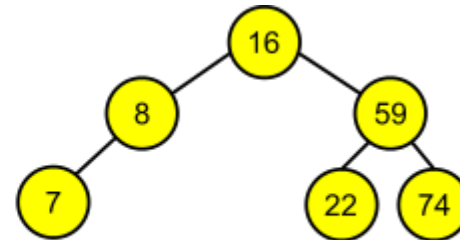
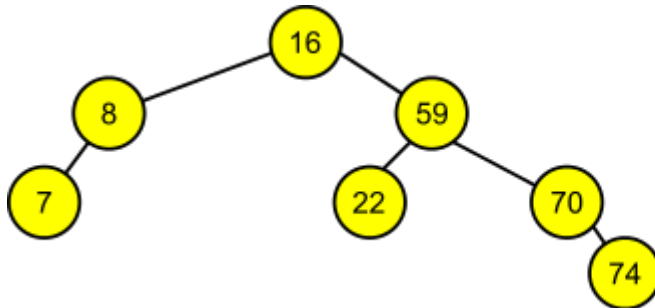
Árbol Binario de Búsqueda

Ejemplo de borrado

- Sin descendientes (nodos 13, 14, 74):

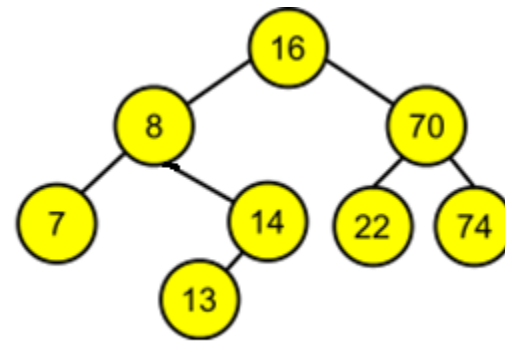
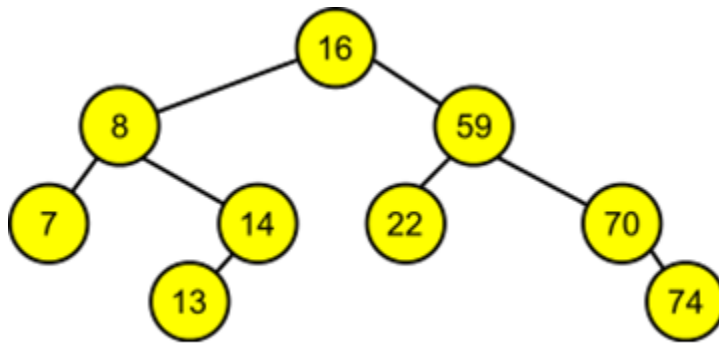


- Con 1 descendiente (nodo 70)



Árbol Binario de Búsqueda

- Con 2 descendientes (nodo 59)



TAD Árbol Binario de Búsqueda

Especificación

```
public class ArbolBusqueda<E> {  
    // Declaración de tipos: ArbolBusqueda  
    // Características: Es un árbol binario donde para cada nodo se cumple la propiedad  
    // de que todos los nodos con clave menor que la suya están en  
    // su subárbol izquierdo y todos los nodos con clave mayor o igual  
    // se encuentran en el subárbol derecho.  
    // Los objetos son modificables  
  
    public ArbolBusqueda();  
        // Produce: Un árbol vacío  
    public boolean esVacio();  
        // Produce: Cierto si el árbol está vacío. Falso en caso contrario.  
    public E raiz() throws ArbolVacioExcepcion;  
        // Produce: Si el árbol está vacío lanza la excepción ArbolVacioExcepcion,  
        // sino devuelve el objeto almacenado en la raíz  
    public ArbolBusqueda<E> hijolzq() throws ArbolVacioExcepcion;  
        // Produce: Si el árbol está vacío lanza la excepción ArbolVacioExcepcion,  
        // sino devuelve el subárbol izquierdo
```

TAD Árbol Binario de Búsqueda

Especificación

```
public ArbolBusqueda<E> hijoDer() throws ArbolVacioExcepcion;  
    // Produce: Si el árbol está vacío lanza la excepción ArbolVacioExcepcion,  
    //          sino devuelve el subárbol derecho  
  
public void insertar(E elemento);  
    // Modifica:      this  
    // Produce:       Añade el objeto elemento a this  
  
public void eliminar(E elemento) throws ElementoIncorrecto;  
    // Modifica:      this  
    // Produce:       Si elemento no existe en el árbol, lanza la excepción  
    //               ElementoIncorrecto sino elimina el objeto de this.  
  
public boolean buscar(E elemento) ;  
    // Produce:       Devuelve cierto si el objeto está en el árbol y  
    //               falso en otro caso  
}
```

TAD Árbol Binario de Búsqueda

Implementación

- ¿Cómo comparar los elementos de los nodos?

Solución: los elementos deben ser instancias de una clase que implemente la interface Comparable<E> existente en java.

```
public interface Comparable<E>{  
    public int compareTo(E e);  
}
```

- Paso 1: Definición interfaz

```
public interface ArbolBusqueda <E extends Comparable<E>> {  
    public boolean esVacio();  
    public E raiz() throws ArbolVacioExcepcion;  
    public ArbolBusqueda<E> hijoIzq() throws ArbolVacioExcepcion;  
    public ArbolBusqueda<E> hijoDer() throws ArbolVacioExcepcion;  
    public void insertar(E elemento);  
    public void eliminar(E elemento) throws ElementoIncorrecto;  
    public boolean buscar(E elemento);  
}
```

- Paso 2: Clase implemente la interfaz

- ☐ Mediante estructuras enlazadas genéricas

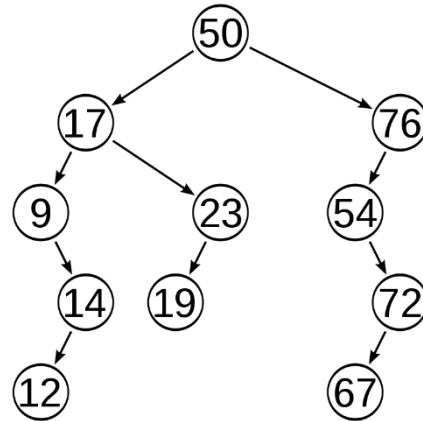
- **public class** ArbolBinarioBusqueda<E **extends** Comparable<E>> **implements** ArbolBusqueda<E>

Árbol Binario Equilibrado

- **Los árboles binarios de búsqueda** son una estructura sobre la cual se pueden realizar eficientemente las operaciones de búsqueda, inserción y eliminación
 - La eficiencia de las operaciones depende exclusivamente de la altura del árbol
 - Árbol de N nodos perfectamente equilibrado \Rightarrow el coste de acceso es de orden logarítmico: $O(\log N)$

Árbol Binario Equilibrado

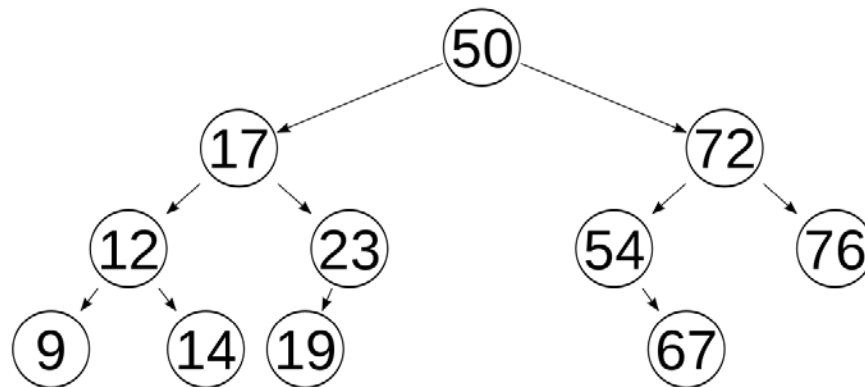
- Sin embargo, si el árbol crece o decrece descontroladamente, el rendimiento puede disminuir considerablemente



- ☐ Caso más desfavorable: insertar un conjunto de claves ordenadas en forma ascendente o descendente
- ☐ Coste de acceso: $O(N)$
- **Árboles equilibrados:**
 - ☐ Aseguran un coste logarítmico sin exigir que el árbol sea completo
 - ☐ Idea central: realizar acomodos o equilibrios después de inserciones o eliminaciones de elementos

Árbol AVL

- Un **árbol AVL** es un árbol binario de búsqueda en el que para todo nodo A del árbol la altura de los subárboles izquierdo y derecho no debe diferir en más de una unidad.
- La condición de equilibrio asegura una profundidad del árbol de $O(\log N)$.

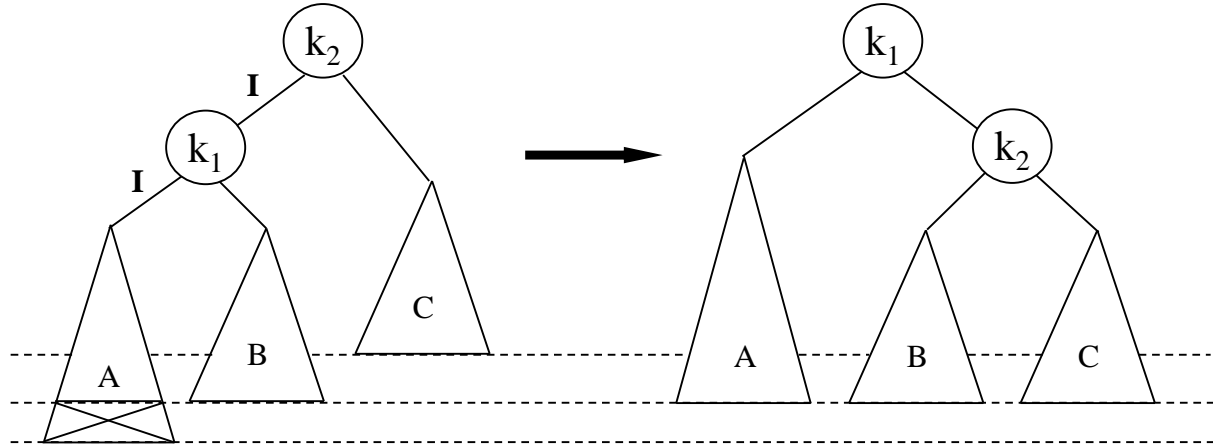


Árbol AVL

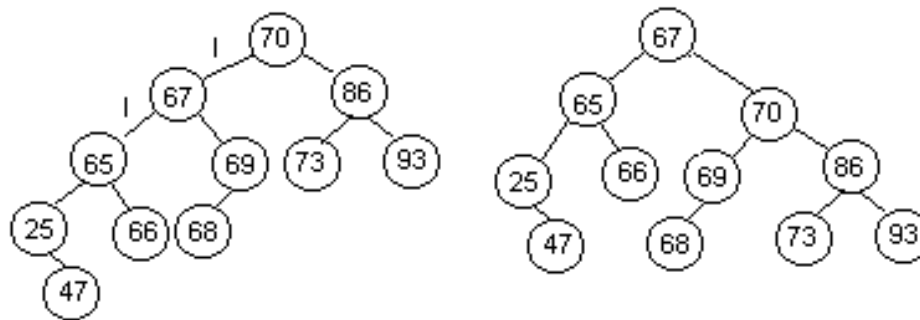
- **Factor de equilibrio (FE)** de un nodo: diferencia entre la altura de los subárboles. Los valores que puede tomar son -1, 0, 1. Si llegara a tomar los valores -2 o 2 \Rightarrow debe reestructurarse el árbol
 - Reestructurar el árbol significa rotar los nodos del mismo
 - Rotación simple: involucra dos nodos
 - Por la rama izquierda
 - Por la rama derecha
 - Rotación compuesta: afecta a tres nodos
 - Por las ramas izquierda y derecha
 - Por las ramas derecha e izquierda

Árbol AVL

■ Rotación simple (l-l)

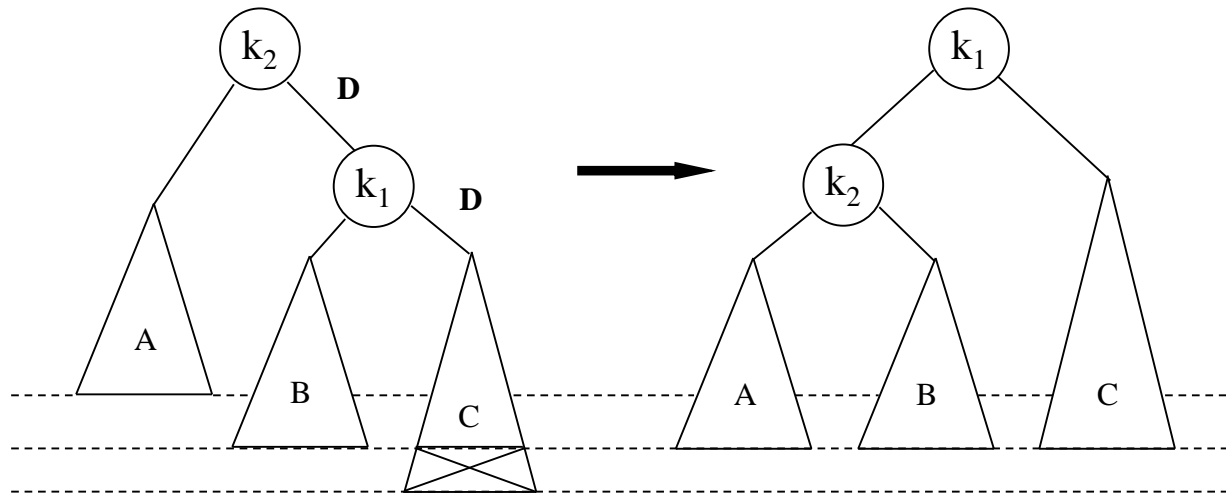


■ Ejemplo: Inserción del nodo 47 en un árbol equilibrado



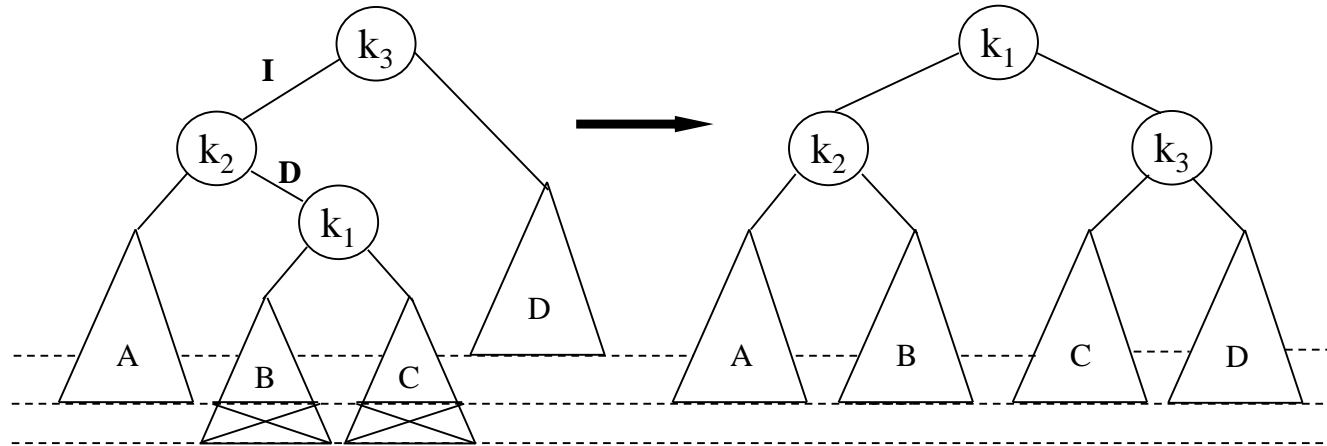
Árbol AVL

■ Rotación simple (D-D)

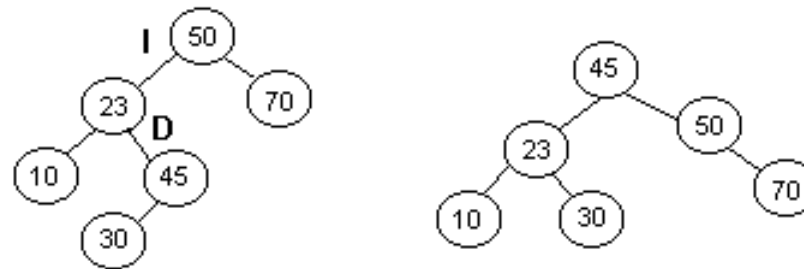


Árbol AVL

- Rotación doble (I-D)

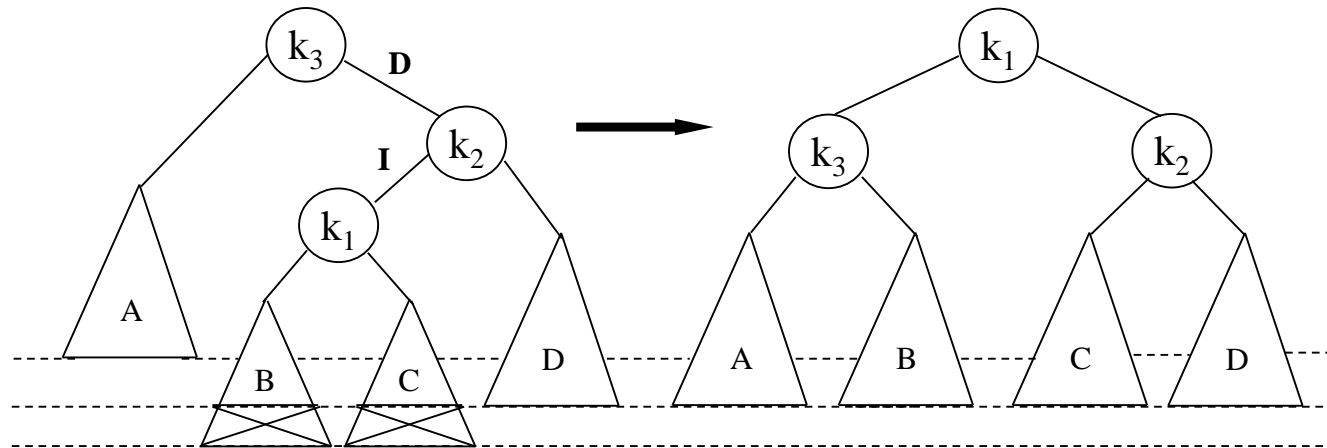


- Ejemplo: Inserción del nodo 30 en un árbol equilibrado

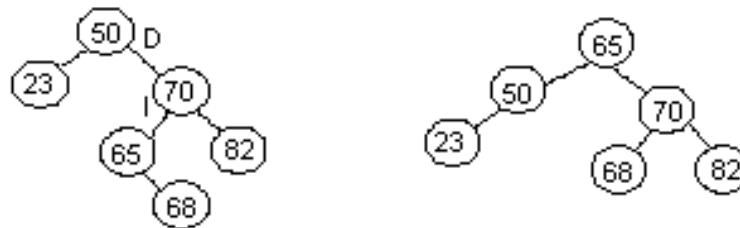


Árbol AVL

- Rotación doble (D-I)



- Ejemplo: Inserción del nodo 68 en un árbol equilibrado



Árbol AVL

■ Resumen operación insertar:

- Insertar el nodo como en un árbol binario de búsqueda
- En el regreso por el camino de inserción se comprueba el FE de los nodos
- Si un nodo presenta un FE incorrecto (2 o -2) se detiene el retroceso en este punto y se reestructura el árbol
- Una inserción provoca una única reestructuración

Árbol AVL

■ Resumen operación suprimir:

- Suprimir el nodo como en un árbol binario de búsqueda
- En el regreso por el camino de supresión se comprueba el FE de los nodos
- Si un nodo presenta un FE incorrecto (2 o -2) se reestructura el árbol y se continua el ascenso hasta llegar a la raíz
- Una supresión puede provocar varias reestructuraciones

Árbol AVL

■ Ejercicio:

- Dibuja el árbol AVL que resulta después de insertar los elementos: 10, 33, 58, 40, 75, 49, 7, 18, 25, 15, 36, 3.
- Dibuja el árbol AVL que resulta después de eliminar los siguientes elementos del árbol anterior: 40, 75, 10, 49, 18, 33, 25, 58.
- Dibuja el árbol AVL que resulta después de insertar los elementos: 3, 5, 4, 1, 9, 2, 6 y 7.
- Dibuja el árbol AVL que resulta después de eliminar los siguientes elementos del árbol anterior: 2, 3, 6, 7