Pseudocódigo: **Árbol Binario de Búsqueda (ABB) y Árbol AVL**

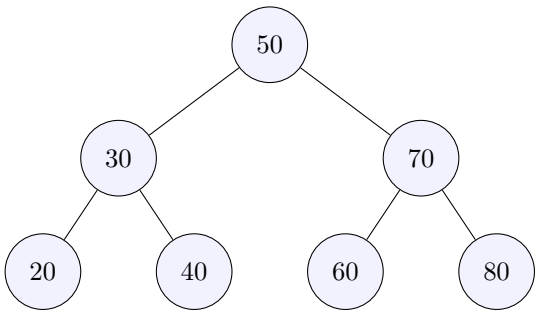
Árbol Binario de Búsqueda (ABB)

Definición: Un Árbol Binario de Búsqueda (ABB) es una estructura de datos jerárquica donde cada nodo tiene un máximo de dos hijos (izquierdo y derecho), y cumple las siguientes propiedades:

1. Para cada nodo N, todos los valores en el subárbol **izquierdo** son menores que N
2. Para cada nodo N, todos los valores en el subárbol **derecho** son mayores que N
3. Los subárboles izquierdo y derecho son también ABB

Características clave:

- **Ordenamiento:** El recorrido *in-order* produce los elementos ordenados
- **Operaciones básicas:** Inserción, eliminación y búsqueda
- **Complejidad:**
  - Caso promedio: $O(\log n)$ para operaciones básicas
  - Peor caso (árbol degenerado): $O(n)$
- **Aplicaciones:** Diccionarios, implementación de conjuntos, algoritmos de búsqueda



Ejemplo de ABB balanceado

Operaciones Básicas

Inserción

```
function insert(root, v):
    if root == NULL:
        return newNode(v)
    if v < root.value:
        root.left = insert(root.left, v)
    else if v > root.value:
        root.right = insert(root.right, v)
    return root
```

Búsqueda

```
function search(root, v):
    if root == NULL or root.value == v:
        return root
    if v < root.value:
        return search(root.left, v)
    else:
        return search(root.right, v)
```

Eliminación:

```
function delete(root, v):
    if root == NULL:
        return NULL

    // Buscar el nodo a eliminar
    if v < root.value:
        root.left = delete(root.left, v)
    else if v > root.value:
        root.right = delete(root.right, v)

    // Nodo encontrado
    else:
        // Caso 1: 0-1 hijo
        if root.left == NULL:
            return root.right
        else if root.right == NULL:
            return root.left

        // Caso 2: 2 hijos
        else:
            temp = findMin(root.right)
            root.value = temp.value
            root.right = delete(root.right, temp.value)

    return root
```

Ventajas y Limitaciones

Ventajas	Limitaciones
▪ Simple implementación ▪ Recorrido ordenado eficiente ▪ Búsqueda binaria natural	▪ Puede degenerar en lista enlazada ▪ Desempeño $O(n)$ en peor caso ▪ No garantiza balance automático

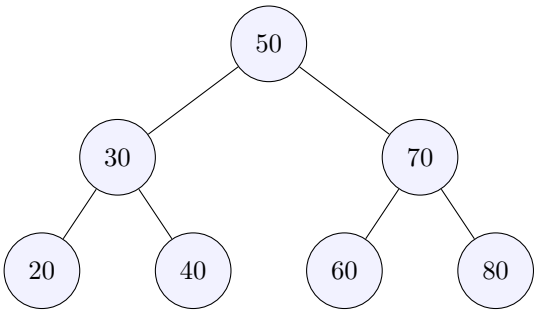
Árbol AVL (Adelson-Velsky y Landis)

Definición: ABB autobalanceado donde:

$$|\text{altura(izq)} - \text{altura(der)}| \leq 1$$

Características:

- **Balance automático:** Realiza rotaciones al insertar/eliminar para mantener equilibrio
- **Complejidad:** Garantiza operaciones en $O(\log n)$ incluso en peores casos
- **Ventaja:** Evita degradación a listas enlazadas (problema de ABB básicos)
- **Altura máxima:** $1{,}44 \log_2(n+2)$



Ejemplo de AVL balanceado

Implementación AVL

Nodo:

```
AVLNode:
    integer value
    AVLNode left
    AVLNode right
    integer height
```

Altura y Balance:

```
function height(node):
    if node == NULL: return 0
    return node.height

function updateHeight(node):
    node.height = 1 + max(
        height(node.left),
        height(node.right)
    )

function getBalance(node):
    if node == NULL: return 0
    return height(node.left) - height(node.right)
```

Rotaciones:

```
function rotateRight(y):
    x = y.left
    T2 = x.right
    x.right = y
    y.left = T2
    updateHeight(y)
    updateHeight(x)
    return x

function rotateLeft(x):
    y = x.right
    T2 = y.left
    y.left = x
    x.right = T2
    updateHeight(x)
    updateHeight(y)
    return y
```

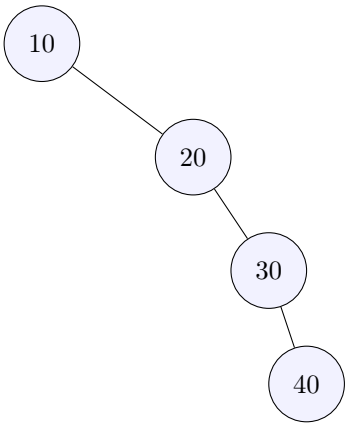
Inserción AVL

```
function insertAVL(node, value):
    // 1. Insercion ABB normal
    if node == NULL:
        return newAVLNode(value)
    if value < node.value:
        node.left = insertAVL(node.left, value)
    else if value > node.value:
        node.right = insertAVL(node.right, value)
    else:
        return node  // Duplicados no permitidos

    // 2. Actualizar altura
    updateHeight(node)

    // 3. Obtener factor de equilibrio
    balance = getBalance(node)

    // 4. Casos de desequilibrio:
    // Izquierda-Izquierda
    if balance > 1 and value < node.left.value:
        return rotateRight(node)
    // Derecha-Derecha
    if balance < -1 and value > node.right.value:
        return rotateLeft(node)
    // Izquierda-Derecha
```

```
if balance > 1 and value > node.left.value:
    node.left = rotateLeft(node.left)
    return rotateRight(node)
// Derecha-Izquierda
if balance < -1 and value < node.right.value:
    node.right = rotateRight(node.right)
    return rotateLeft(node)

return node
```

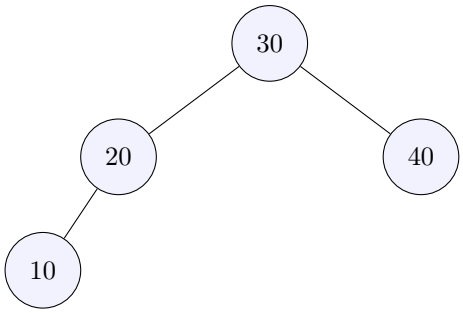
Comparación Visual: ABB vs AVL

ABB degenerado:



Altura: 4

AVL balanceado:



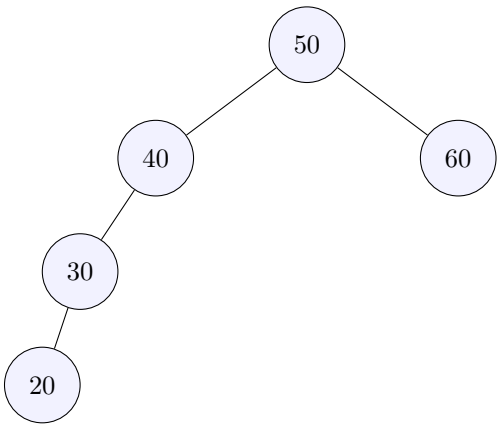
Altura: 2

Tabla Comparativa ABB vs AVL

Característica	**ABB**	**AVL**
Balance automático	No	Sí
Altura máxima	$O(n)$	$O(\log n)$
Búsqueda (peor caso)	$O(n)$	$O(\log n)$
Inserción (peor caso)	$O(n)$	$O(\log n)$
Memoria adicional	Ninguna	Altura + Balance
Dificultad implementación	Baja	Media-Alta

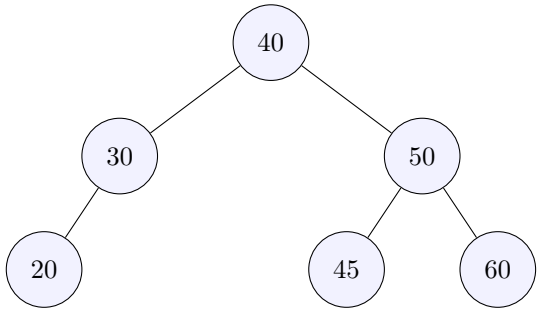
Ejemplo de Rotación AVL

Antes de rotación:



Factor de equilibrio: +2

Después de rotación:



Balance restaurado

```
// Ejemplo de rotacion izquierda-izquierda
root = insertAVL(root, 50)
root = insertAVL(root, 40)
root = insertAVL(root, 30)  // Desbalance detectado
root = insertAVL(root, 20)  // Rotacion simple (RR)
// Arbol balanceado automaticamente
```