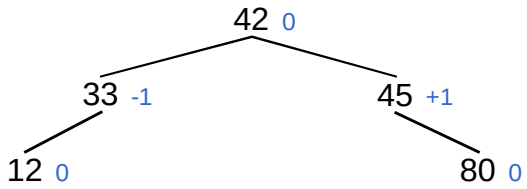
VL07, Lösung 1

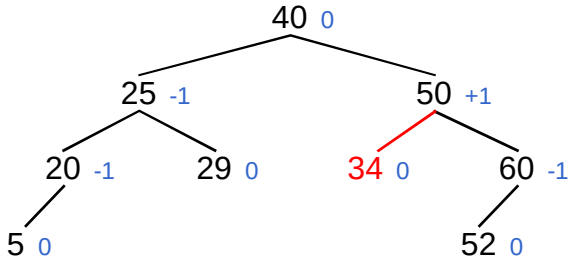
- Jeder AVL-Baum **ist** auch ein Binärer Suchbaum. Daher unterscheiden sich die Algorithmen für die Suche und das Traversieren **nicht**. Die Suche in einem AVL-Baum ist jedoch in der Regel effizienter, da die Höhe auf $O(\log n)$ begrenzt ist. Für das Travervieren gilt dies nicht, da dort ohnehin alle Knoten besucht werden.
- Die Suche in einem AVL-Baum (und einem Binären Suchbaum) verfolgt nur einen einzigen Pfad von der Wurzel bis zum gesuchten Knoten bzw. über ein Blatt hinaus. Eine Rückkehr zu Elternknoten ist nicht notwendig. Eine rekursive Implementierung enthält daher nur eine Endrekursion, die leicht durch eine `while`-Schleife ersetzt werden kann (siehe Aufgabe 5).
- Bei der vollständigen Traversierung eines Baumes (einschließlich AVL-Bäume und Binärer Suchbäume) muss bei einer rekursiven Implementierung regelmäßig zum Elternknoten eines Teilbaums zurückgekehrt werden (Backtracking), um danach weitere Teilbäume zu besuchen. Daher ist eine iterative Implementierung aufwändiger, ggf. unter Benutzung eines eigenen Stacks (siehe Aufgabe 6).

VL07, Lösung 2

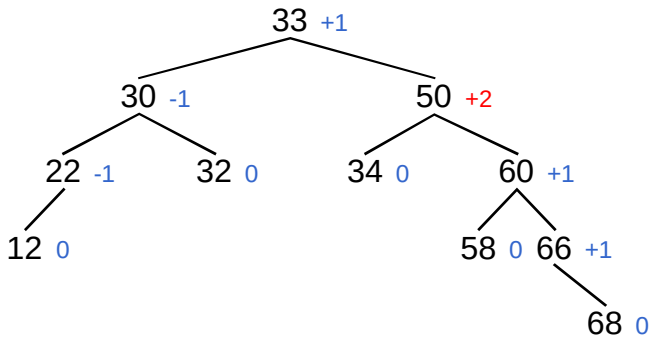
a) Es handelt sich um einen AVL-Baum. Der Balance-Faktor ist an jedem Knoten -1, 0 oder +1. Außerdem ist die Suchbaum-Eigenschaft erfüllt (die Schlüssel im linken Teilbaum eines Knotens sind alle kleiner als der Schlüssel des Knotens, und die Schlüssel im rechten Teilbaum sind alle größer als der Schlüssel des Knotens).



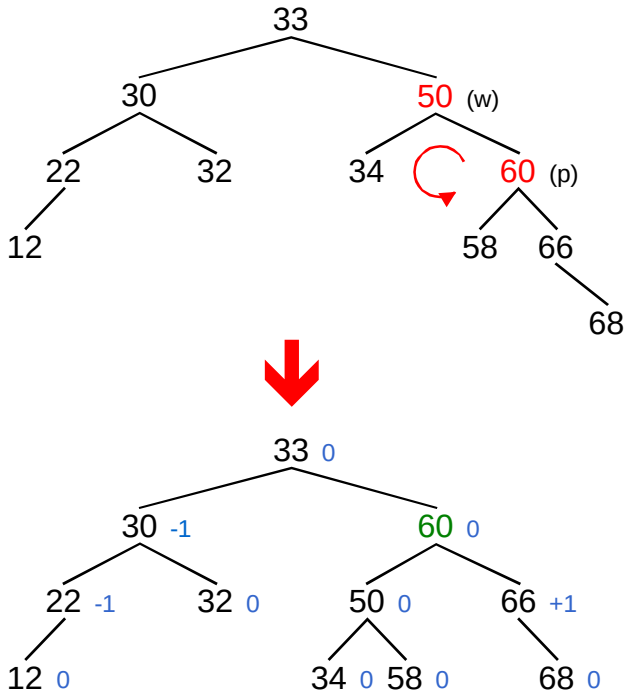
b) Der Baum ist zwar so ausbalanciert, dass alle Balance-Faktoren -1, 0 oder +1 sind. Allerdings ist der Knoten `34` falsch einsortiert, so dass die Suchbaum-Eigenschaft verletzt ist. Es handelt sich somit **nicht** um einen AVL-Baum.



c) Der Balance-Faktor am Knoten mit dem Schlüssel `50` beträgt +2 und ist somit ungültig. Es handelt sich daher **nicht** um einen AVL-Baum.

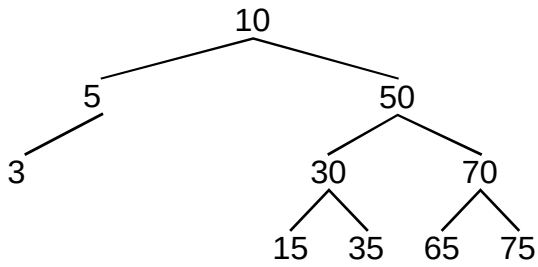


Da der rechte Teilbaum von Knoten `60` größer ist als dessen linker Teilbaum, wird eine einfache Linksrotation ausgeführt:

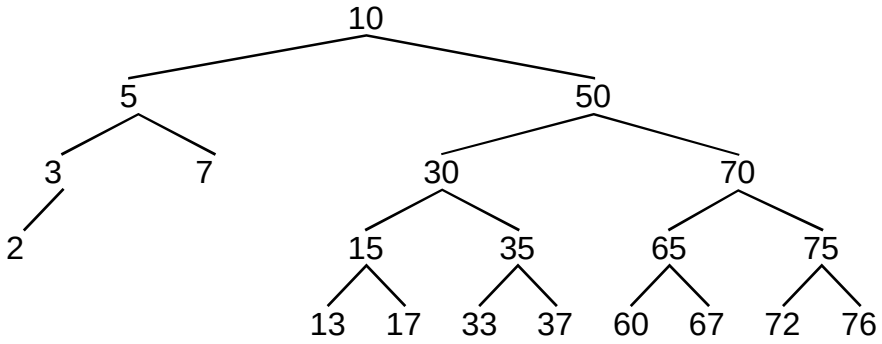


VL07, Lösung 3

Höhe 4:



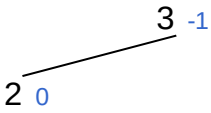
Höhe 5:



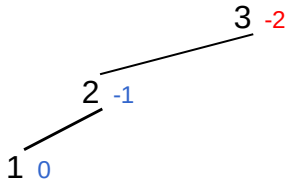
VL07, Lösung 4

Einfügen von `3`:

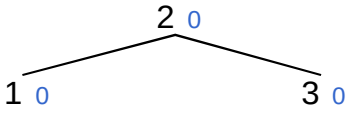
Einfügen von `2`:



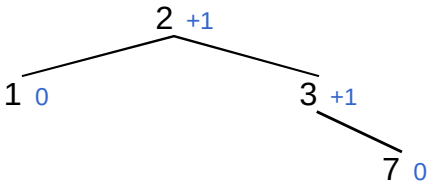
Einfügen von `1`:



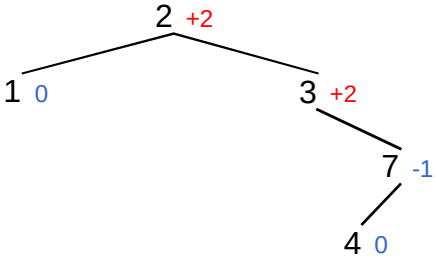
Nach Rechtsrotation:



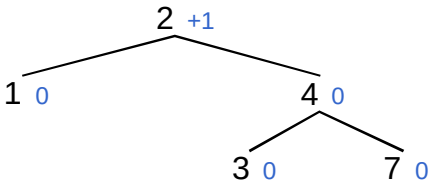
Einfügen von `7`:



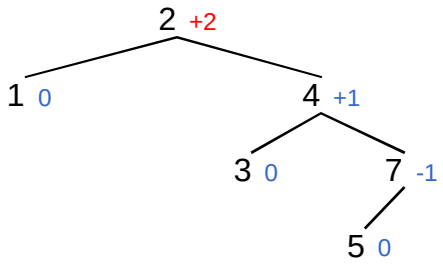
Einfügen von `4`:



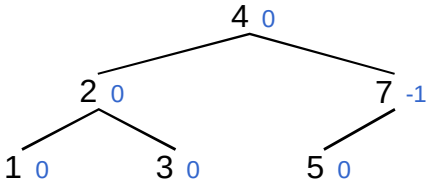
Nach RL-Doppelrotation:



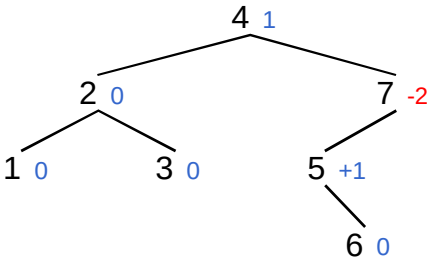
Einfügen von `5`:



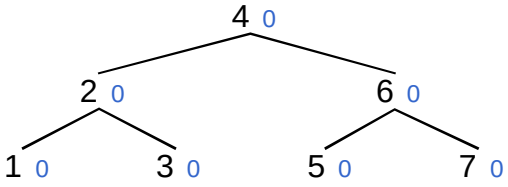
Nach Linksrotation:



Einfügen von `6`:



Nach LR-Doppelrotation:



VL07, Lösung 5

```
public boolean suchen(final T daten)
{
    AVLKnoten<T> teilbaum = wurzel;

    while (teilbaum != null)
    {
        // Vergleichs-Ergebnis zwischenspeichern, da compareTo()
        // aufwändig sein kann, und das Ergebnis mehrfach benötigt
        // wird
        final int cmp = daten.compareTo(teilbaum.getDaten());

        if (cmp == 0)
            return true; // Verlässt sofort die Methode

        teilbaum = (cmp < 0) ? teilbaum.getKnotenLinks() :
            teilbaum.getKnotenRechts();
        // Effiziente Kurzform für:
        // if (cmp < 0) { teilbaum = teilbaum.getKnotenLinks(); }
        // else { teilbaum = teilbaum.getKnotenRechts(); }
    }

    // Die Suche ist bei einem Blatt angelangt, ohne dass die Daten
    // auf dem Weg dorthin gefunden werden konnten
    return false;
}
```

VL07, Lösung 6

```
// Pre-Order
public String traversierePreOrder()
{
    if (wurzel == null)
        return "Der Baum ist leer.";

    String text = "";

    // Stack mit Elementtyp AVLKnoten anlegen
    Deque<AVLKnoten<T>> s = new LinkedList<AVLKnoten<T>>();
    s.push(wurzel);

    while (!s.isEmpty())
    {
        AVLKnoten<T> einKnoten = s.pop();

        text += einKnoten.getDaten();

        // Achtung: erst rechten Teilbaum auf den Stack legen
        if (einKnoten.getKnotenRechts() != null)
            s.push(einKnoten.getKnotenRechts());

        if (einKnoten.getKnotenLinks() != null)
            s.push(einKnoten.getKnotenLinks());
    }

    return text;
}
```