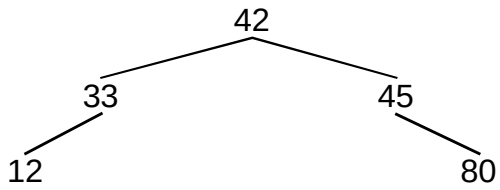
VL07, Aufgabe 1 (Übung)

- Auf welche Art und Weise unterscheidet sich der Algorithmus für das Suchen in einem AVL-Baum von der Suche in einem Binären Suchbaum?
- Auf welche Art und Weise unterscheidet sich der Algorithmus für das Traversieren eines AVL-Baums von der Traversierung eines Binären Suchbaums?
- Warum kann die Suche in einem Binären Suchbaum und einem AVL-Baum auch **leicht** iterativ anstatt rekursiv programmiert werden?
- Warum kann das vollständige Traversieren in einem Binären Suchbaum und einem AVL-Baum zwar auch iterativ anstatt rekursiv programmiert werden, aber nicht mehr leicht ohne besonderen Programmieraufwand?

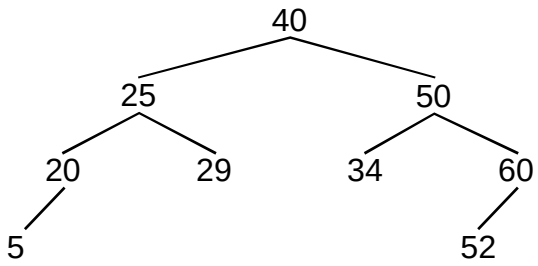
VL07, Aufgabe 2 (Übung)

Gegeben sind folgende Bäume. Beurteilen Sie jeweils, ob es sich um einen AVL-Baum handelt und begründen Sie Ihre Entscheidung. Falls es sich nicht um einen AVL-Baum handelt, prüfen Sie ob sich der Baum durch eine Rotation in einen AVL-Baum überführen lässt.

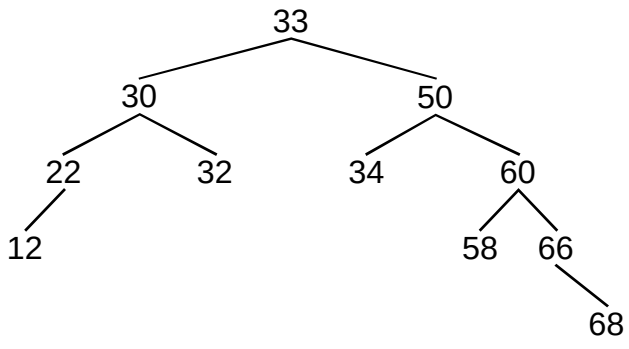
a)



b)



c)



VL07, Aufgabe 3 (Übung)

Konstruieren Sie zwei AVL-Bäume der Höhen 4 und 5 mit der maximal möglichen Anzahl an Knoten im rechten Teilbaum der Wurzel und der minimal möglichen Anzahl Knoten im linken Teilbaum der Wurzel.

VL07, Aufgabe 4 (Übung)

Bauen Sie einen AVL-Baum auf, indem Sie der Reihe nach die Schlüssel `3`, `2`, `1`, `7`, `4`, `5` und `6` in einen anfangs leeren AVL-Baum einfügen.

- Zeichnen Sie nach jedem Einfügeschritt den entstehenden Baum einschließlich der Balance-Faktoren.
- Falls eine Balancierung notwendig sein sollte, geben Sie den Baum vor und nach dem Balancieren an. Geben Sie außerdem jeweils den Namen der verwendeten Balancierungsoperation (Links-/Rechts-Rotation, LR-/RL-Doppelrotation) an.

VL07, Aufgabe 5 (Praktikum)

Implementieren Sie in der Datei `AVLBaum.java` aus dem Beispiel-Programm `BSP07-AVLBaum.zip` die Methode `public boolean suchen(T daten)`. Die Methode soll dabei analog zur gleichnamigen Methode aus der vorigen Vorlesung (`BSP06-Suchbaum.zip`, Datei `Baum.java`) angeben, ob die übergebenen Daten im Baum vorhanden sind oder nicht. **Die Implementierung in dieser Woche soll dabei jedoch iterativ vorgehen, nicht rekursiv!**

Hinweis: Gehen Sie davon aus, dass der Datentyp `T` das Interface `Comparable<T>` implementiert. Orientieren Sie sich hierzu am bereits bestehenden Code in `AVLBaum.java`.

VL07, Aufgabe 6 (Praktikum)

Implementieren Sie in der Datei `AVLBaum.java` aus dem Beispiel-Programm zur Vorlesung die Methode `traversierenPreOrder()`, die den Baum **iterativ** vollständig in Pre-Order-Reihenfolge traversieren soll.

Verwenden Sie zur Lösung dieser Aufgabe einen Stack mit Elementtyp `AVLKnoten`. Die Schnittstelle `Deque`, die durch `LinkedList` implementiert wird, stellt die nötigen Stack-Methoden zur Verfügung. Das folgende Programm zeigt zur Orientierung die Verwendung von `Deque` als Stack mit Elementtyp `String`.

```
import java.util.*;

public class StackBeispiel
{
    public static void main(String[] args)
    {
        Deque<String> s = new LinkedList<String>();

        s.push("Erstes Element");
        s.push("Zweites Element");
        s.push("Drittes Element");

        while (!s.isEmpty())
            System.out.println(s.pop());
    }
}
```