

# Aufgabe 03 - Bäume

## Binärer Suchbaum

**Implementieren** Sie einen *binären Suchbaum* in Java (oder in einer anderen objekt-orientierten Sprache). In einem binären Suchbaum werden die *Werte geordnet abgelegt*. Dabei ist immer der *Vaterknoten größer als alle Knoten seines linken Teilbaums* und *kleiner als alle Knoten seines rechten Teilbaums*. Abbildung 1 zeigt den Ablauf während des Einfügens von Werten. *Duplikate Einträge im Baum* werden von unserer Implementierung *nicht unterstützt*. Unterstützen Sie dabei alle Typen für die eine Ordnung definiert ist, in Java also zum Beispiel das *Comparable-Interface* implementieren.

Implementieren Sie die Methoden *insert*, *search*, *minimum*, *maximum*, *predecessor*, *successor* und *remove*. Überlegen Sie sich wie Sie den Baum *visualisieren* können (in der Konsole genügt).

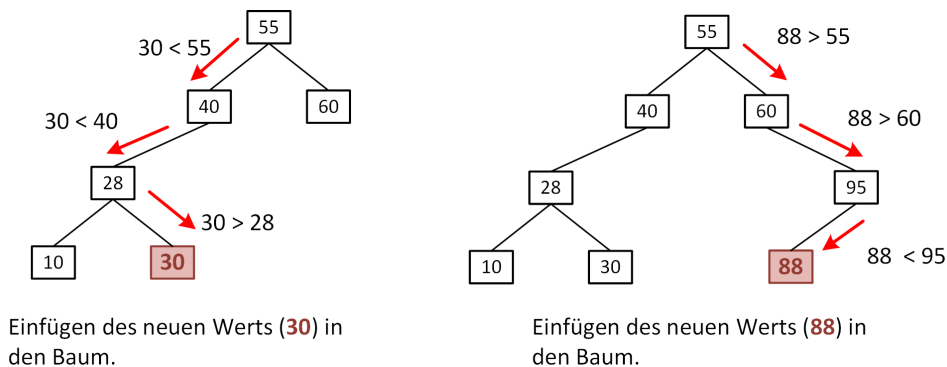


Abbildung 1: Einfügen in einen binären Suchbaum

Schaffen Sie eine Möglichkeit den *Wert* eines *bestimmten Knotens zu aktualisieren*. Dabei müssen Sie die *Grenzen* für eine *zulässige Aktualisierung beachten* und dürfen die *Struktur* des binären Suchbaumes *nicht verletzen*. Wenn Sie einen *Wert aktualisieren* darf er sich daher nur *zwischen* dem *Predecessor* und *Successor* befinden. Abbildung 2 veranschaulicht das Aktualisieren von Knoten.

**Implementieren** Sie eine *Traversierung auf dem Baum* welcher diesen in einer *geordneten List* ausgibt.

## AVL-Baum

**Erweitern** Sie Ihre *Implementierung* auf einen *AVL-Baum*. Beim Einfügen müssen Sie in diesem Fall, *falls notwendig*, eine *Rebalancierung durchführen*. Implementieren Sie dazu das *Einfügen* als *rekursive Funktion* und *prüfen Sie auf jeder Ebene*, *nachdem der Wert eingefügt wurde*, *ob die Bedingungen eines AVL-Baums verletzt* sind. Dies ist der Fall *wenn sich die Höhe des linken und rechten Teilbaums um mehr als 1 unterscheiden*. Ist dies der Fall, führen Sie eine *Rebalancierung* durch. Dabei müssen Sie die zwei in der Vorlesung vorgestellten Fälle beachten und eine *Rotation* (*einfach* oder *doppelt*) durchführen.

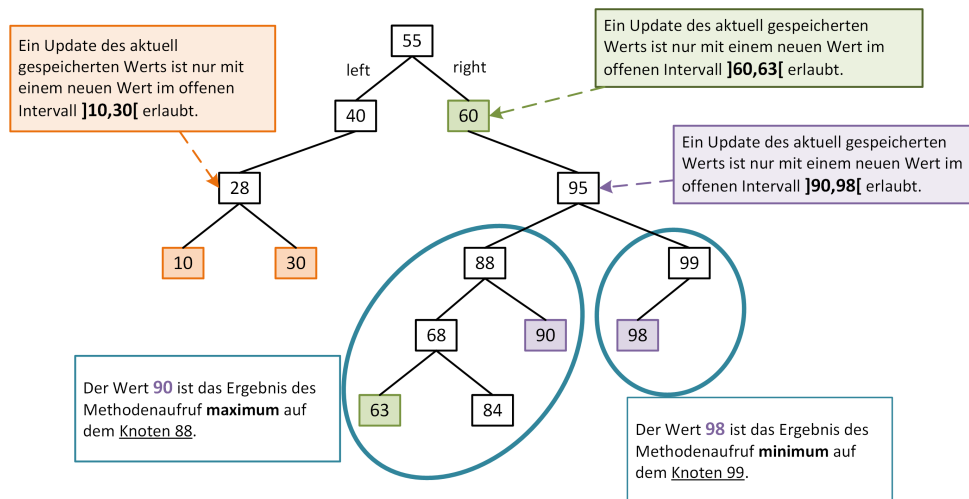


Abbildung 2: Aktualisieren in einen binären Suchbaum

Im **einfacheren Fall** muss eine **einzelne Rotation** nach **links oder rechts** durchgeführt werden. Abbildung 3 veranschaulicht diese Rotation.

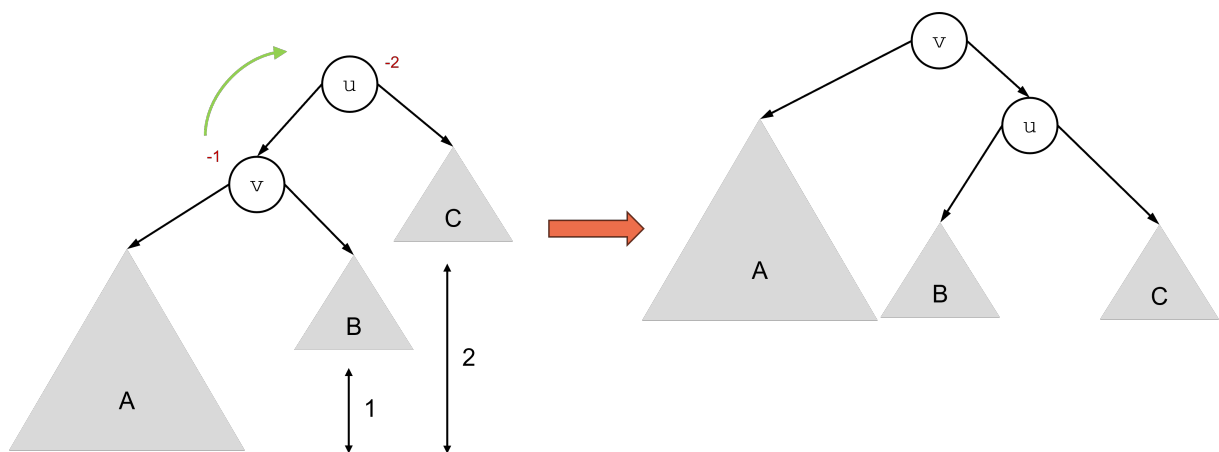


Abbildung 3: Einfache Rotation in einem AVL-Baum

Im anderen Fall muss eine **doppelte Rotation** (**Links-Rechts oder Rechts-Links**) durchgeführt werden. Abbildung 4 veranschaulicht diese Rotation.

*Optional:* **Implementieren** Sie das **Löschen** eines **Knotens** in einem AVL-Baum.

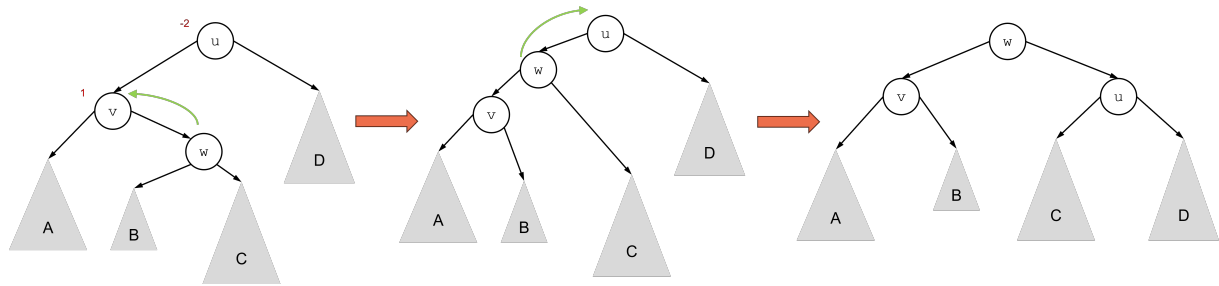


Abbildung 4: Doppelte Rotation in einem AVL-Baum

## Aufgabe

Basierend auf den Beschreibungen oben:

1. Implementierung eines binären Suchbaums (insert, search, remove, minimum, maximum, predecessor und successor)
2. Erweiterung des binären Suchbaums zu einem AVL-Baum (remove optional)