Algorithmen und
Datenstrukturen

BALANCIERTE BÄUME

HOCHSCHULE AALEN
GRADUATE
CAMPUS

AVL-BÄUME

Eigenschaften

- Selbstbalancierende binäre Suchbäume
- Benannt nach Adelson-Velsky und Landis (sowjetische Mathematiker)

- Binärer Suchbaum mit veränderten Einfüge- und Löschoperationen
- Höhenbedingung für Teilbäume
  - Höhenunterschied zwischen linkem und rechtem Teilbaum ist maximal eins
  - Verhindert Degenerierung des Baums

2

EINFÜHRUNG

Definition

Ein binärer Suchbaum ist genau dann ein AVL-Baum, wenn gilt:

- Der Balance-Faktor eines Knotens v ist der Höhenunterschied zwischen seinem rechten Teilbaum (T_r) und seinem linken Teilbaum (T_l).

$$bal(v) = h(T_r) - h(T_l)$$

- Der Balance-Faktor an jedem Knoten ist maximal eins.

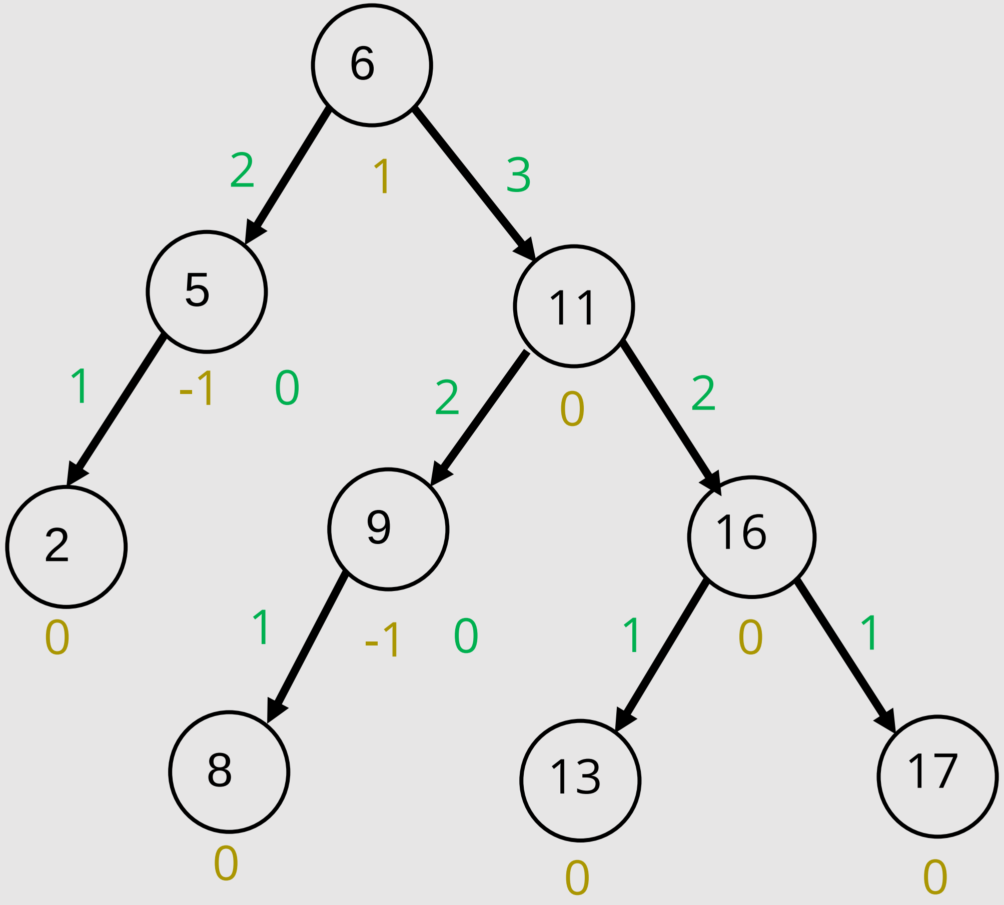
$$bal(v) \in \{-1, 0, 1\}$$

HOCHSCHULE AALEN GRADUATE CAMPUS

EINFÜHRUNG

Beispiel:

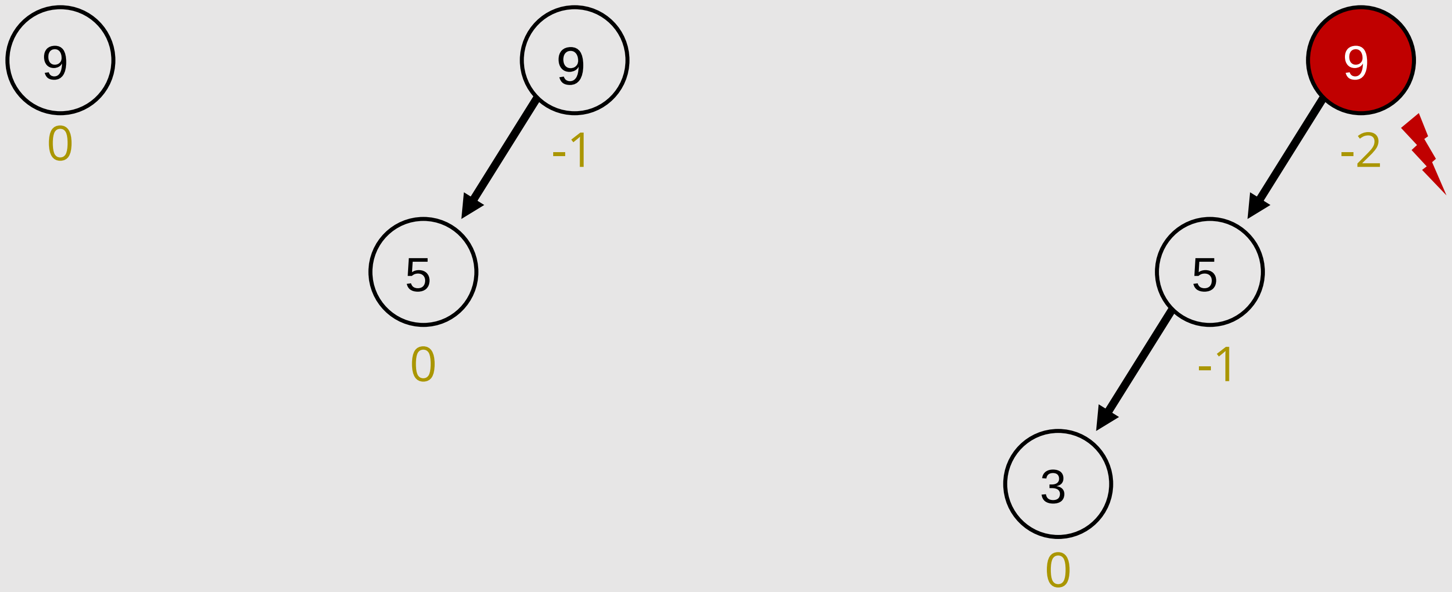
- max. Höhe d. Teilbaums
- Balance-Faktor



HOCHSCHULE AALEN GRADUATE CAMPUS

EINFÜGEN IN EINEN AVL-BAUM

- Überlegungen zum Herstellen einer AVL-Eigenschaft

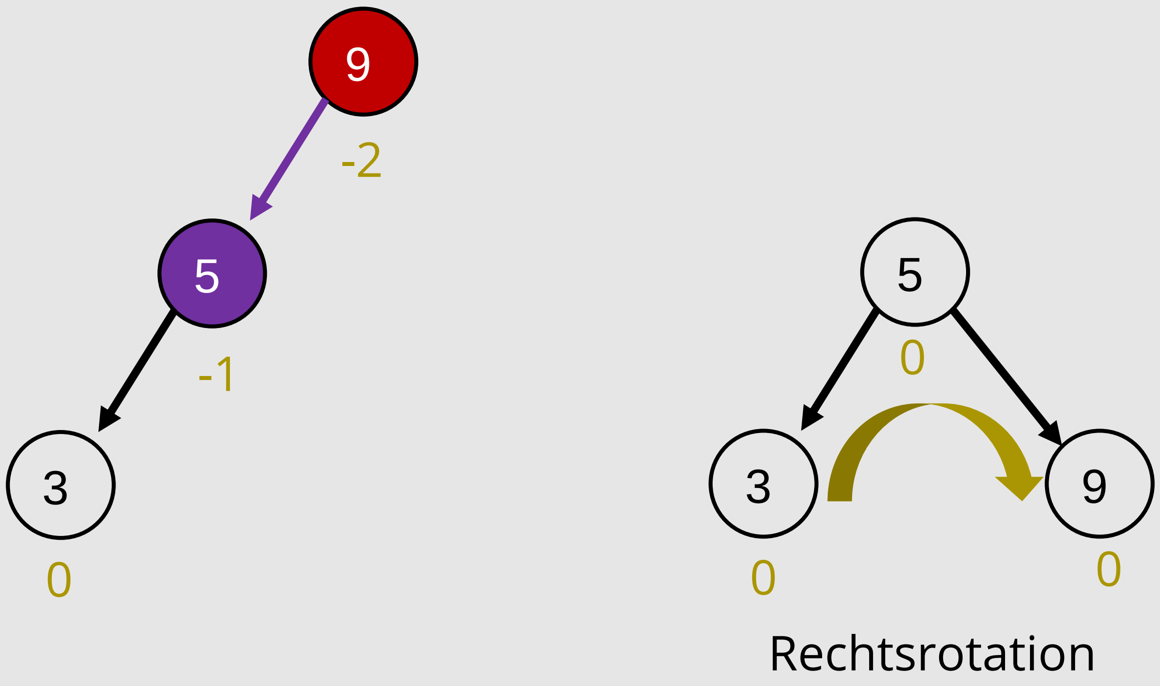


Das AVL-Kriterium ist verletzt

-> Rebalancierung nötig

EINFÜGEN IN EINEN AVL-BAUM: REBALANCIERUNG

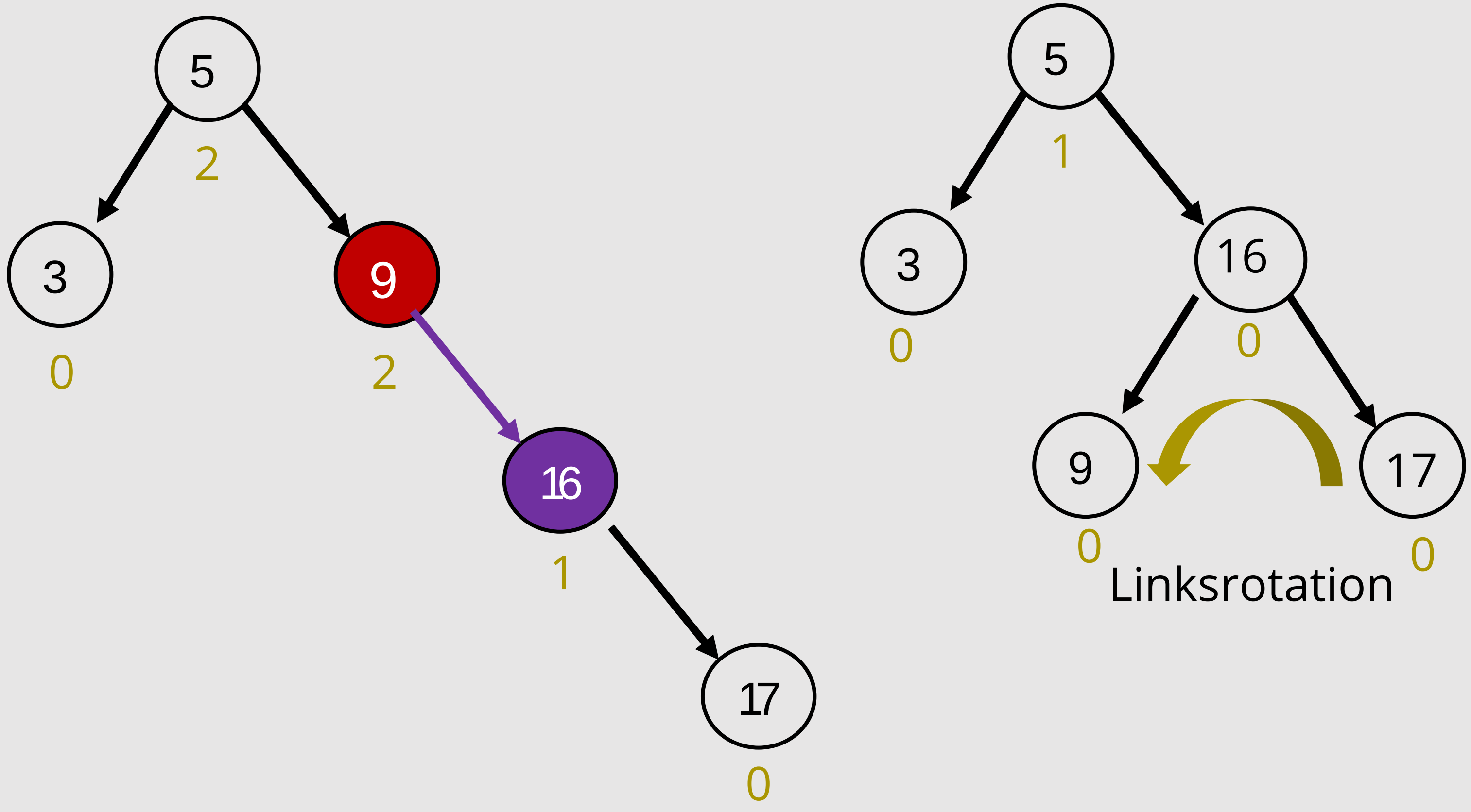
- Einfügen von 3 erfordert Rebalancierung



6

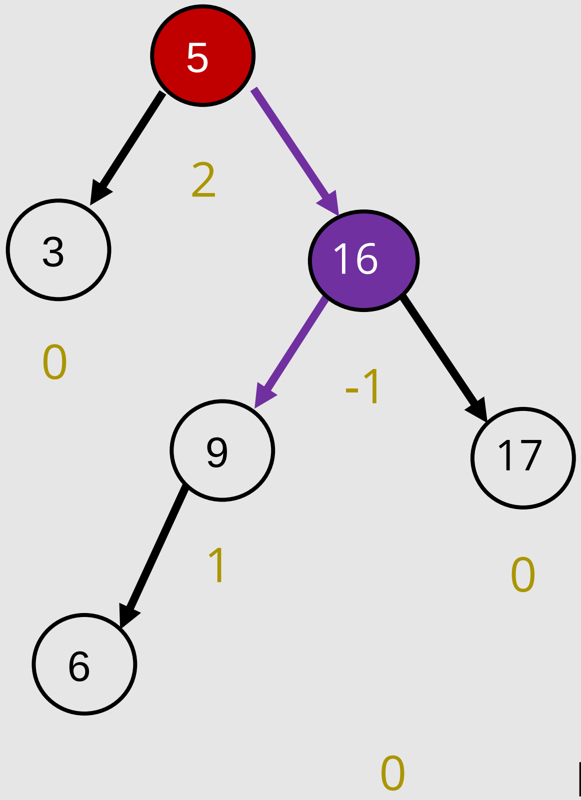
EINFÜGEN IN EINEN AVL-BAUM : REBALANCIERUNG

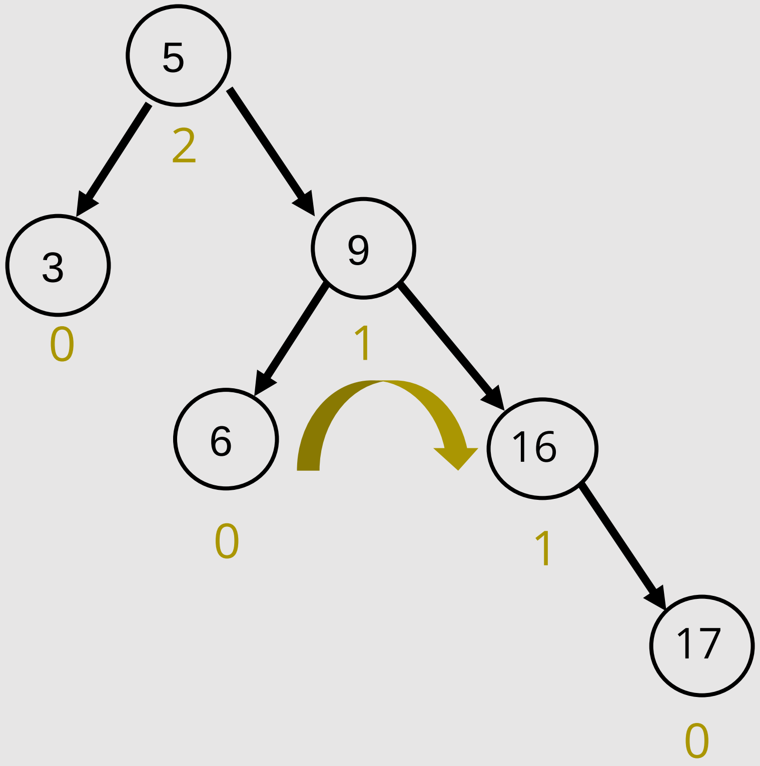
- Einfügen von 16 und danach 17



EINFÜGEN IN EINEN AVL-BAUM : REBALANCIERUNG

- Einfügen von 9

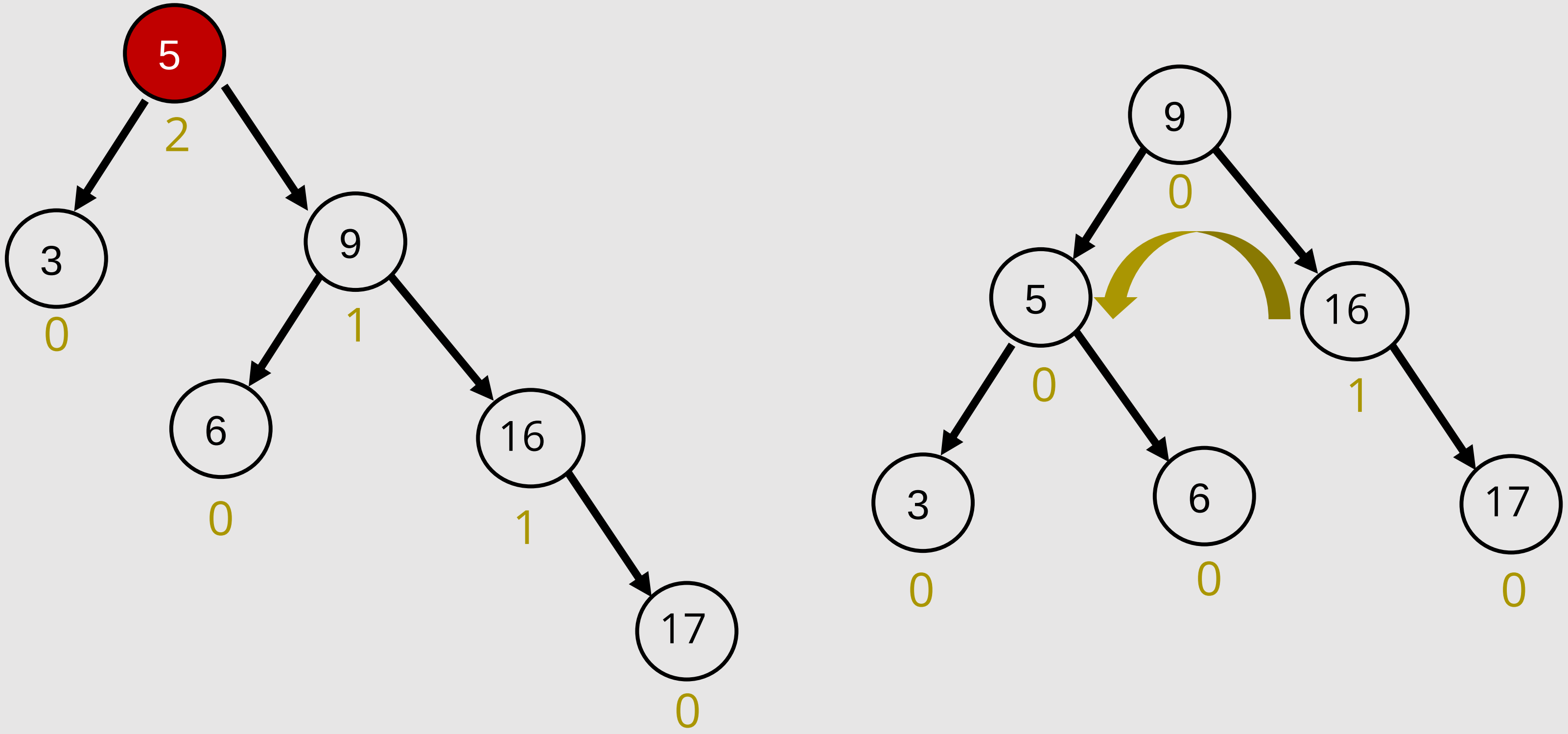




Doppelrotation (1)

EINFÜGEN IN EINEN AVL-BAUM : REBALANCIERUNG

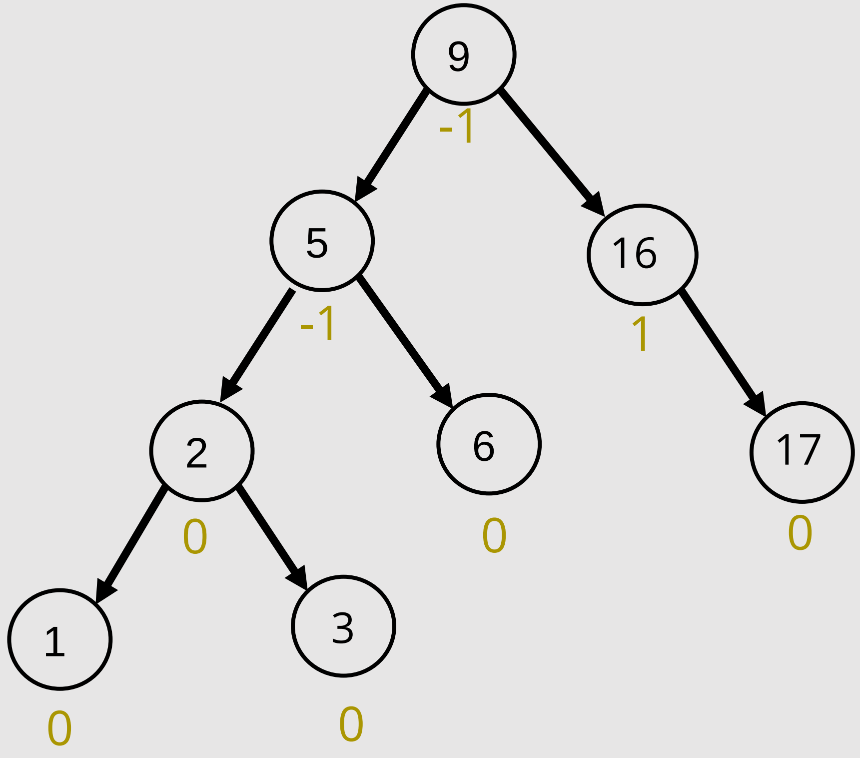
- Einfügen von 6: nach erster Rotation



Doppelrotation (2)

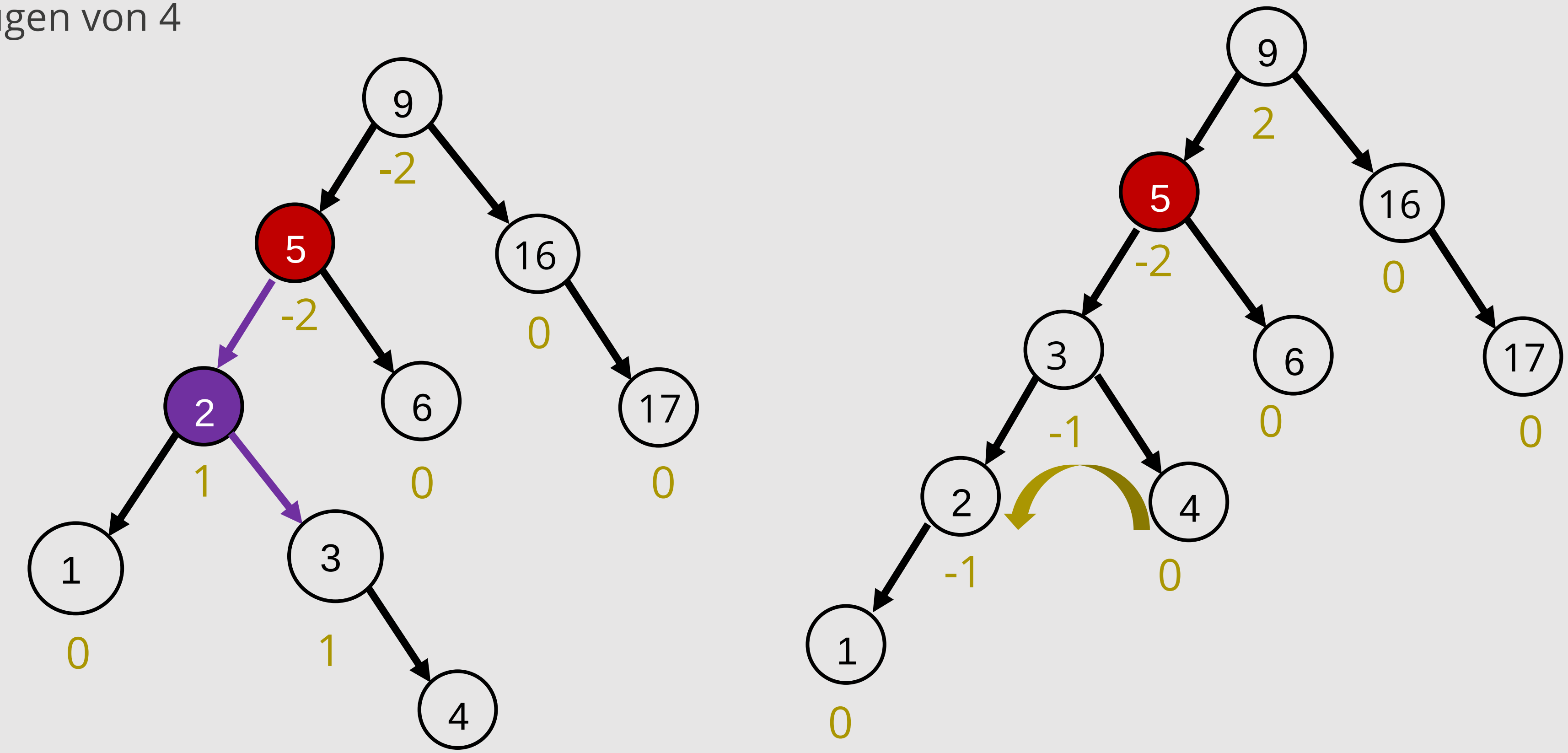
EINFÜGEN IN EINEN AVL-BAUM : REBALANCIERUNG

- Einfügen von 1 und 3, keine Verletzung der AVL-Kriterien



EINFÜGEN IN EINEN AVL-BAUM : REBALANCIERUNG

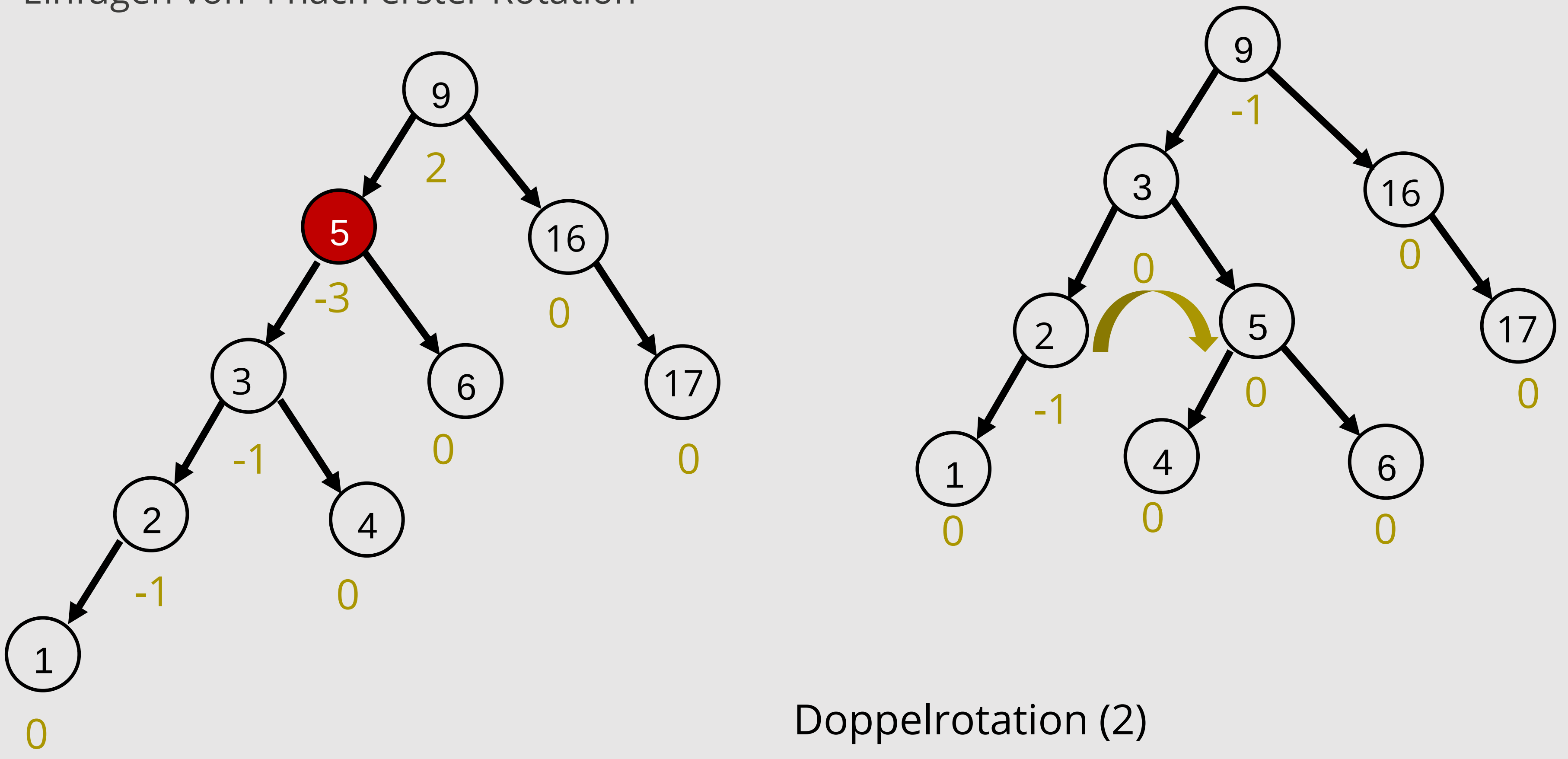
- Einfügen von 4



Doppelrotation (1)

EINFÜGEN IN EINEN AVL-BAUM : REBALANCIERUNG

- Einfügen von 4 nach erster Rotation



Doppelrotation (2)

EINFÜGEN IN EINEN AVL-BAUM : REBALANCIERUNG

Beobachtungen

- Balance-Faktoren ändern sich lediglich auf dem Pfad von der Wurzel zur Einfügeposition.
- Die Art der Rotation ist abhängig vom Pfad ausgehend vom tiefsten Balance-Faktor ± 2 zur Einfügeposition (rot gefärbt)
- Einer der darunter liegenden Knoten hat einen Balance-Faktor ± 1

EINFÜGEN IN EINEN AVL-BAUM

Fallunterscheidungen

Wenn eine Verletzung der AVL-Eigenschaft vorliegt, können vier Fälle unterschieden werden.

Es wurde eingefügt im

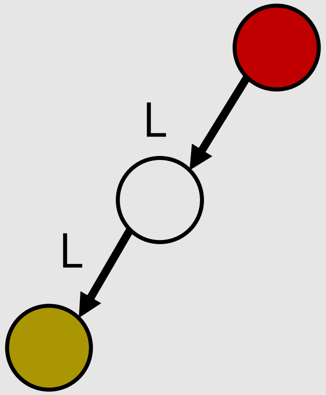
- 1. linken Teilbaum des linken Kindes (vgl. Seite 6, linkes Kind=5)
- 2. rechten Teilbaum des linken Kindes (vgl. Seite 11, linkes Kind=9)
- 3. linken Teilbaum des rechten Kindes (vgl. Seite 8, rechtes Kind=16)
- 4. rechten Teilbaum des rechten Kindes (vgl. Seite 7, rechtes Kind=16)

22

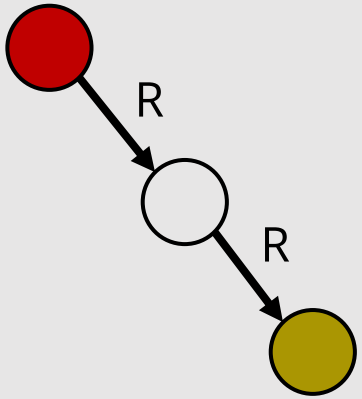
EINFÜGEN IN EINEN AVL-BAUM

Fallunterscheidungen

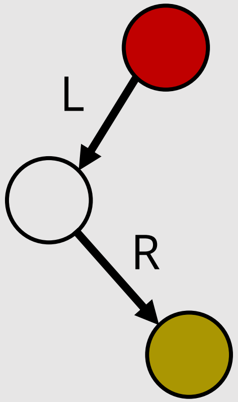
Je nach Einfügeposition bzw. Pfad vom verletzenden Knoten zur Einfügeposition wird eine Rotation angewendet.



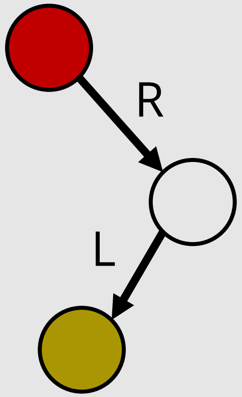
Rechtsrotation



Linksrotation



Doppelrotation rechts



Doppelrotation links

23

EINFÜGEN IN EINEN AVL-BAUM

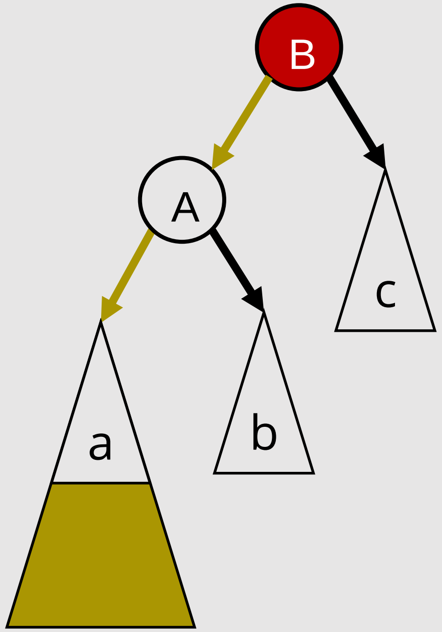
Anmerkungen zu Rotationen

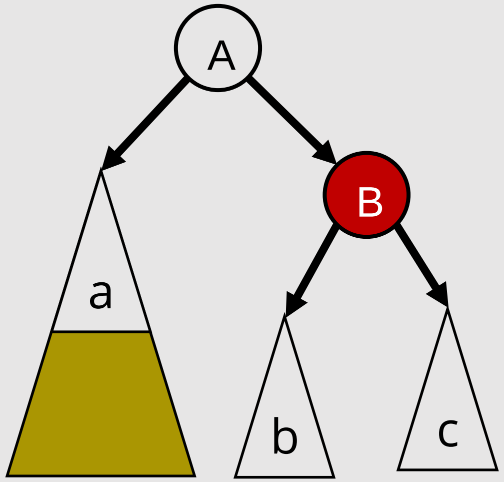
- Für die Rotation sind genau diese drei Knoten wichtig
- Die darunter liegenden Teilbäume werden durch Dreiecke angedeutet

24

EINFÜGEN IN EINEN AVL-BAUM

LL: Rechtsrotation



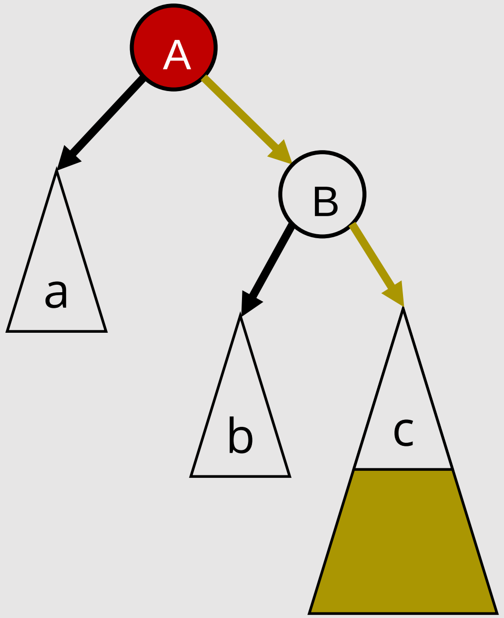


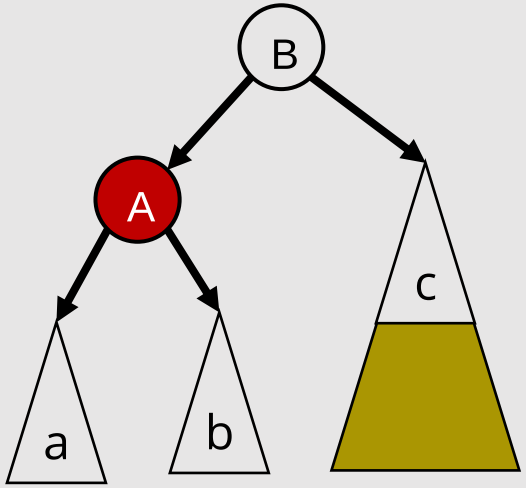
HOCHSCHULE AALEN
GRADUATE
CAMPUS

25

EINFÜGEN IN EINEN AVL-BAUM

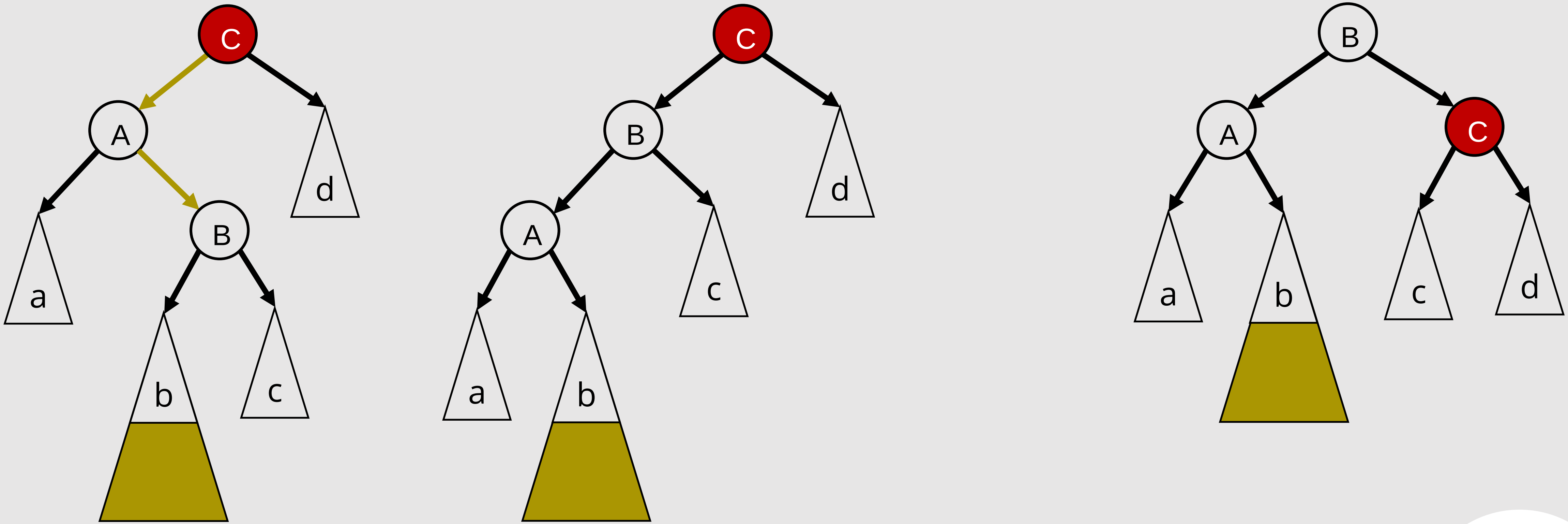
RR: Linksrotation





EINFÜGEN IN EINEN AVL-BAUM

LR: Doppelrotation nach rechts

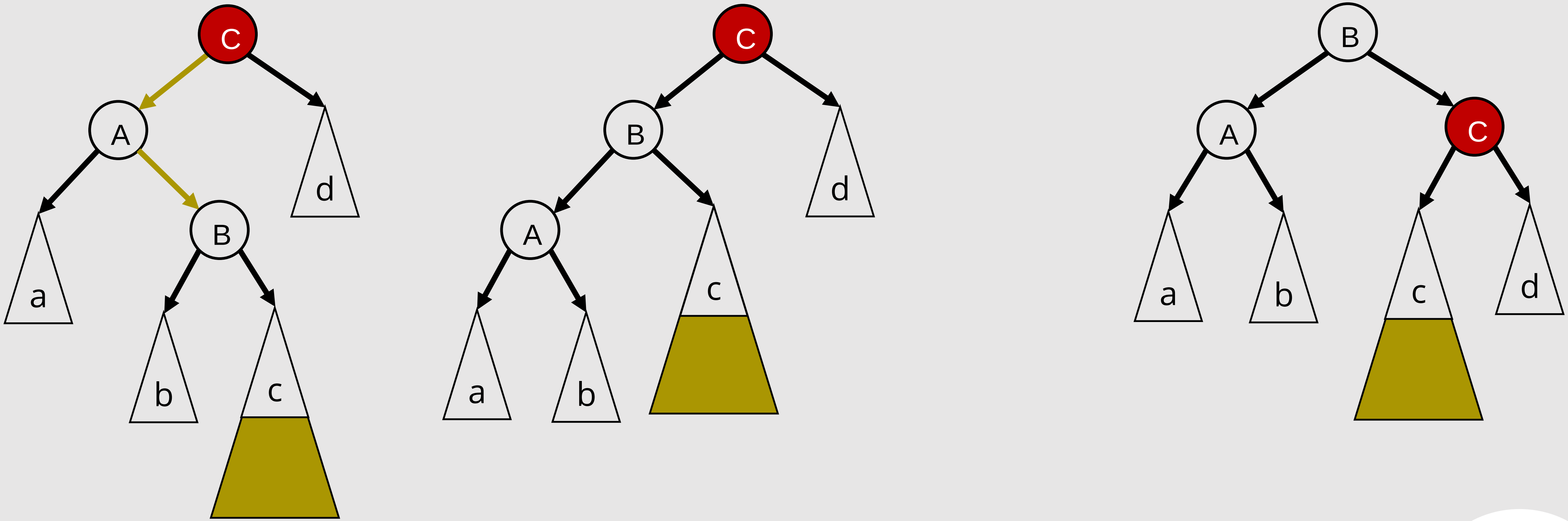


HOCHSCHULE AALEN GRADUATE CAMPUS

27

EINFÜGEN IN EINEN AVL-BAUM

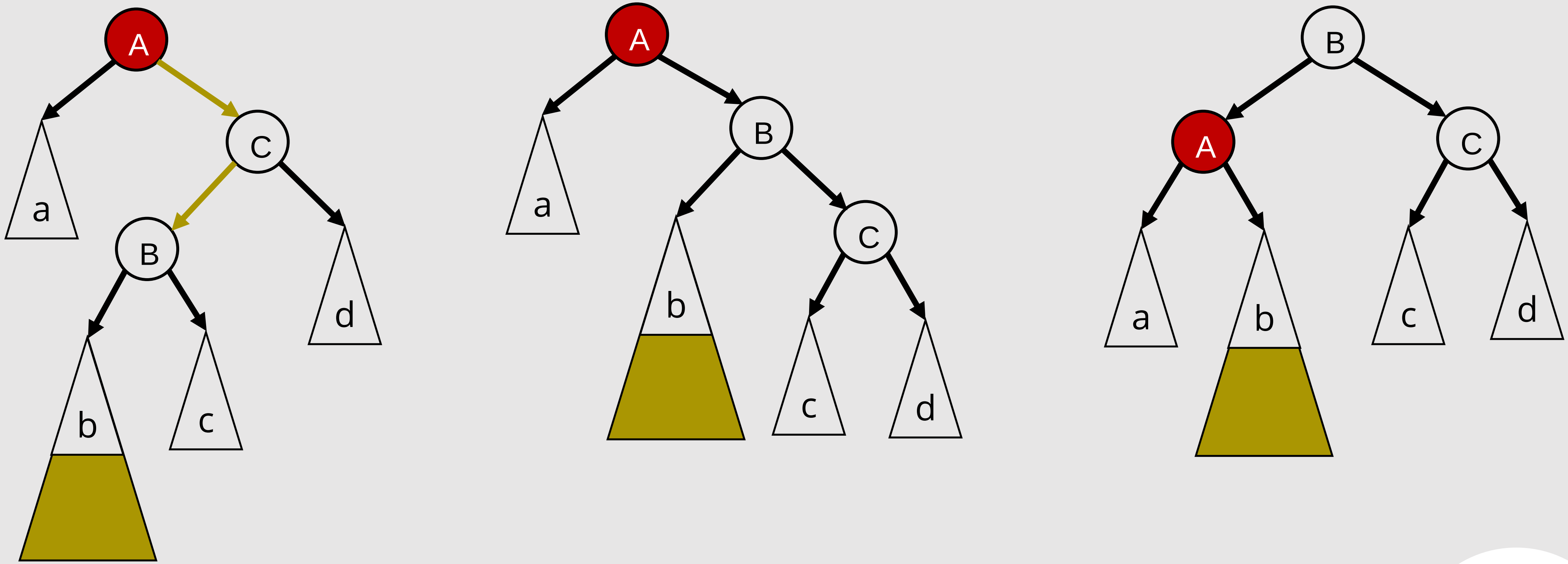
LR: Doppelrotation nach rechts (andere Position im Teilbaum)



28

EINFÜGEN IN EINEN AVL-BAUM

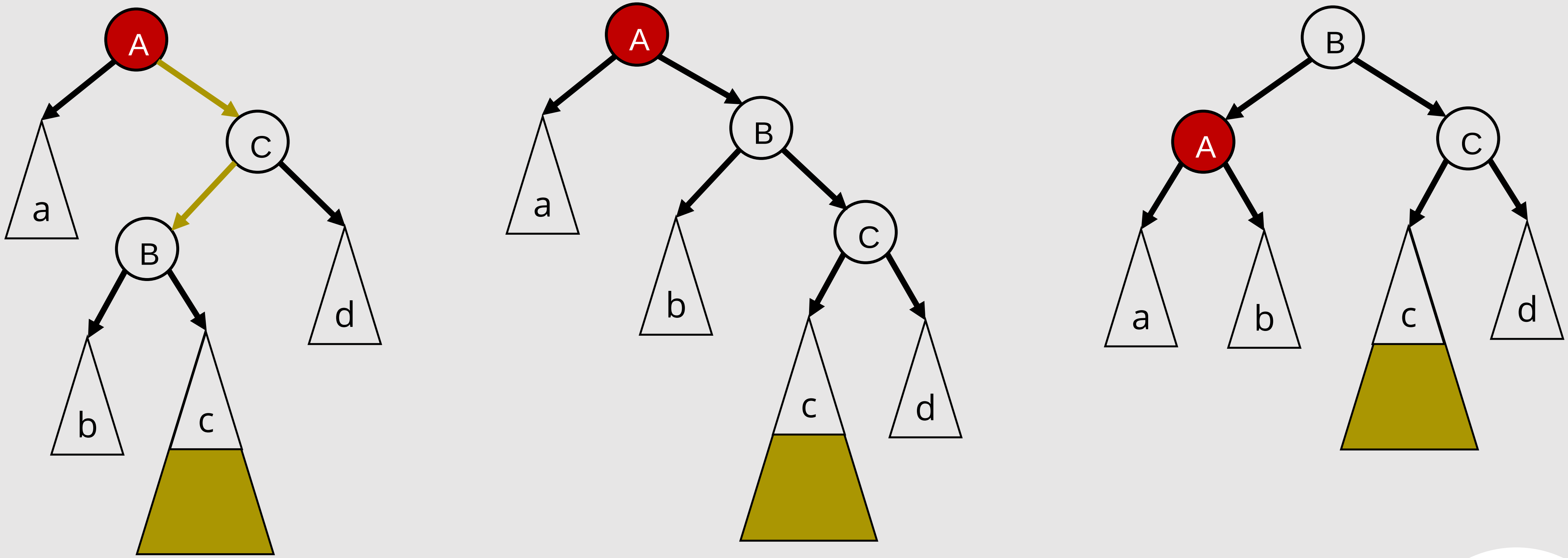
RL: Doppelrotation nach links



29

EINFÜGEN IN EINEN AVL-BAUM

RL: Doppelrotation nach links (andere Position im Teilbaum)



30

EINFÜGEN IN EINEN AVL-BAUM

```
function LEFT-ROTATE(A)                    // AVL-Kriterium in A verletzt, A ist alter Wurzelknoten
    B = A.right                            // linkes Kind von A
    b = B.left                             // rechter Teilbaum von B
    // Perform rotation
    B.left = A                             // A wird zu linkem Kind von B
    A.right = b                            // zwischengespeicherter Teilbaum von B wird zu linkem Teilbaum von A
    // Höhen für Balancefaktor neu berechnen
    A.height = MAXIMUM(HEIGHT(A.left),  HEIGHT(A.right)) + 1
    B.height = MAXIMUM(HEIGHT(B.left),  HEIGHT(B.right)) + 1
    // neuen Wurzelknoten zurückgeben
    return B
end function
```


31

EINFÜGEN IN EINEN AVL-BAUM

```
function RIGHT-ROTATE(B)          // AVL-Kriterium in B verletzt, B ist alter Wurzelknoten
    A = B.left                    // linkes Kind von B
    b = A.right                   // rechter Teilbaum von A
    // Rotation durchführen
    A.right = B                   // B wird zu rechtem Kind von A
    B.left = b                    // zwischengespeicherter Teilbaum von A wird zu linkem Teilbaum von B
    // Höhen für Balancefaktor neu berechnen
    B.height = MAXIMUM(HEIGHT(B.left),  HEIGHT(B.right)) + 1
    A.height = MAXIMUM(HEIGHT(A.left),  HEIGHT(A.right)) + 1
    // neuen Wurzelknoten zurückgeben
    return A
end function
```


32

EINFÜGEN IN EINEN AVL-BAUM

```
function LEFT-RIGHT-ROTATE(C)
    // AVL-Kriterium in C verletzt, C ist Wurzelknoten des zu ändernden Baums
    C.left = LEFT-ROTATE(C.left)        // Linksrotation über linkem Teilbaum von C (in der Grafik A)
    return RIGHT-ROTATE(C)              // Rechtsrotation über C, neuen Wurzelknoten zurückgeben
end function
```

```
function RIGHT-LEFT-ROTATE(A)
    // AVL-Kriterium in C verletzt, C ist Wurzelknoten des zu ändernden Baums
    A.right = RIGHT-ROTATE(A.right)          // Linksrotation über rechtem Teilbaum von A (in der Grafik C)
    return LEFT-ROTATE(A)                    // Rechtsrotation über A, neuen Wurzelknoten zurückgeben
end function
```


33

EINFÜGEN IN EINEN AVL-BAUM

```
function AVL-INSERT(root, x)
    // Rekursiver Aufruf zum Einfügen
    if x.key < root.key then
        root.left = AVL-INSERT(root.left, x)
    else
        root.right = AVL-INSERT(root.right, x)
    end if
    // Höhe des Knotens aktualisieren und Balance-Faktor berechnen
    root.height = MAXIMUM(HEIGHT(root.left), HEIGHT(root.right.right)) + 1
    bal = HEIGHT(root.right) - HEIGHT(root.left)
```


37

EINFÜGEN IN EINEN AVL-BAUM

```
// Fall 1: Left Left
if  bal > 1 and x.key < root.left.key then
    return RIGHT-ROTATE(x)
end if
// Fall 2: Right Right
if  bal < −1 and x.key > root.right.key then
    return LEFT-ROTATE(x)
end if
```


HOCHSCHULE AALEN GRADUATE CAMPUS

38

EINFÜGEN IN EINEN AVL-BAUM

```
    // Fall 3: Left Right
    if bal > 1 and x.key > root.left.key then
        return LEFT-RIGHT-ROTATE(x)
    end if
    // Fall 4: Right Left
    if bal < -1 and x.key < root.right.key then
        return RIGHT-LEFT-ROTATE(x)
    end if
end function
```


39

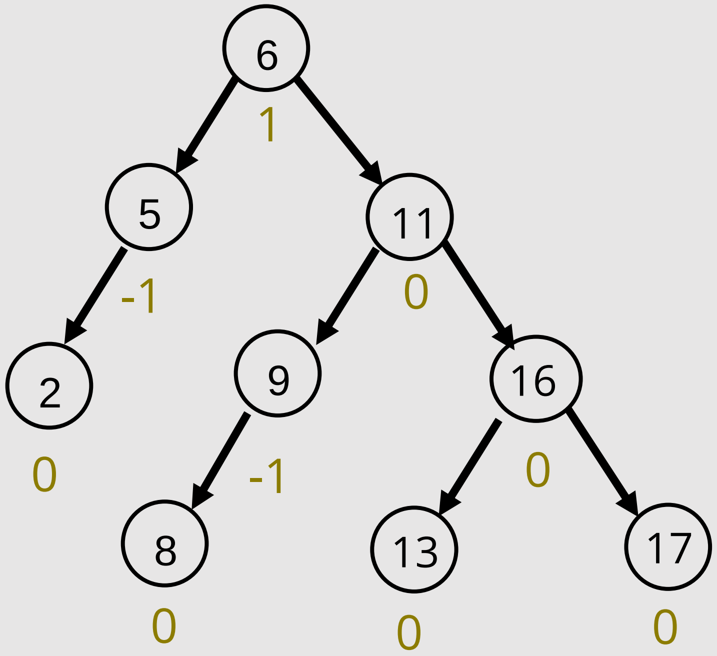
LÖSCHEN AUS EINEM AVL-BAUM

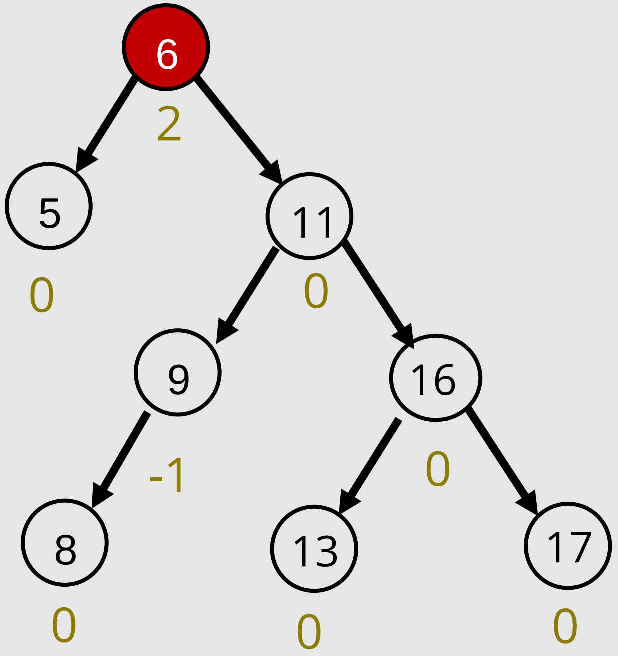
Vorgehen

- gleiche Vorgehensweise wie beim binären Suchbaum (BST)
- Vorgänger des gelöschten Knoten ersetzt diesen
- anschließend muss ggf. rebalanciert werden
- Rebalancieren durch Rotation

LÖSCHEN AUS EINEM AVL-BAUM

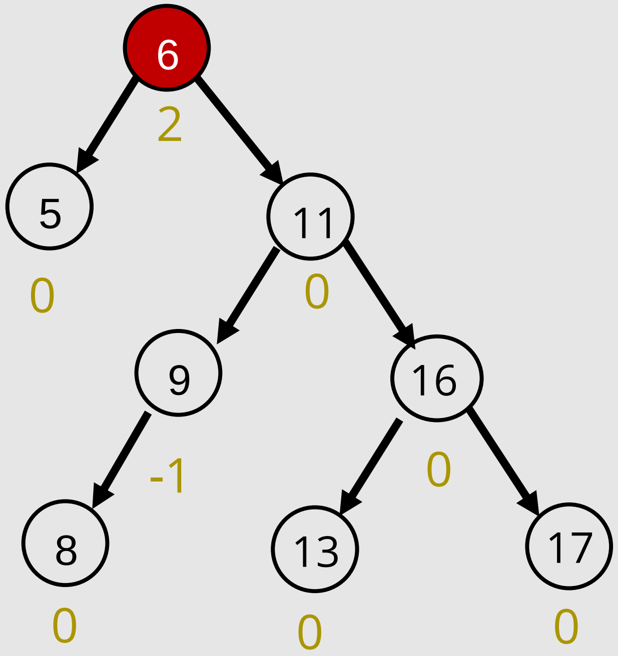
- Löschen von 2 (Blatt-Knoten!)

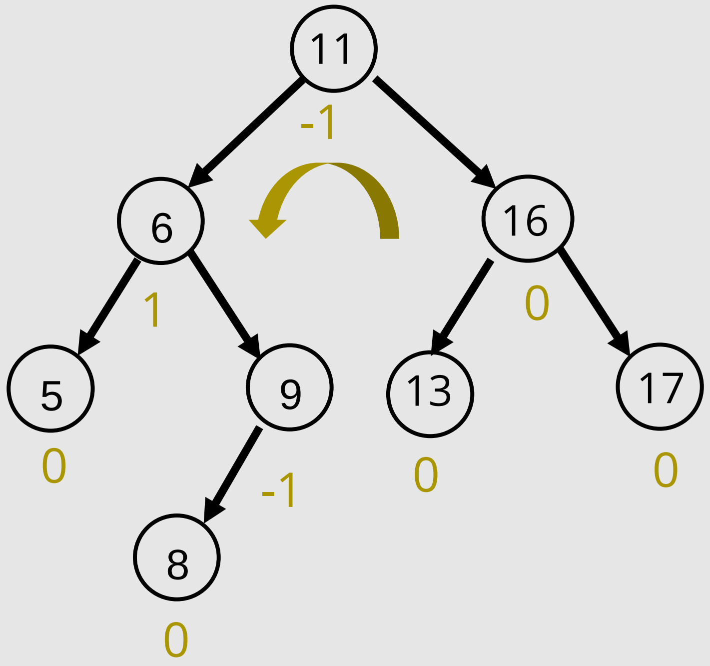




LÖSCHEN AUS EINEM AVL-BAUM

- Rebalancieren nach Löschen von 2. Der Balancefaktor bei 6 ist positiv, es wird eine Linksrotation durchgeführt.

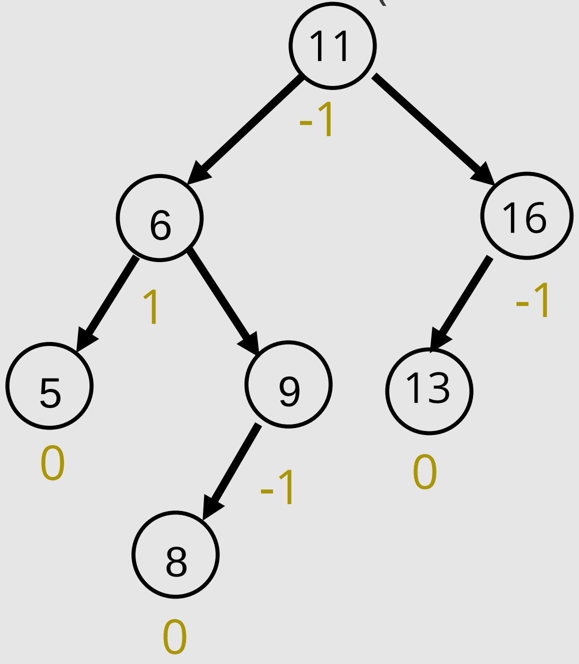


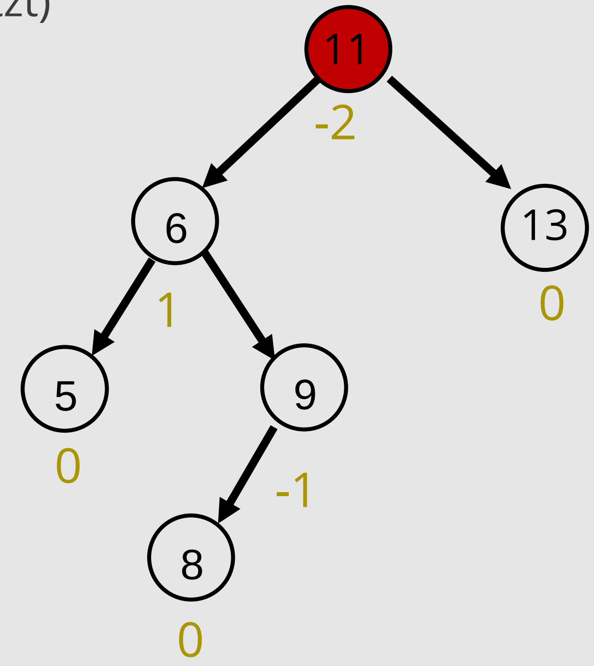


HOCHSCHULE AALEN GRADUATE CAMPUS

LÖSCHEN AUS EINEM AVL-BAUM

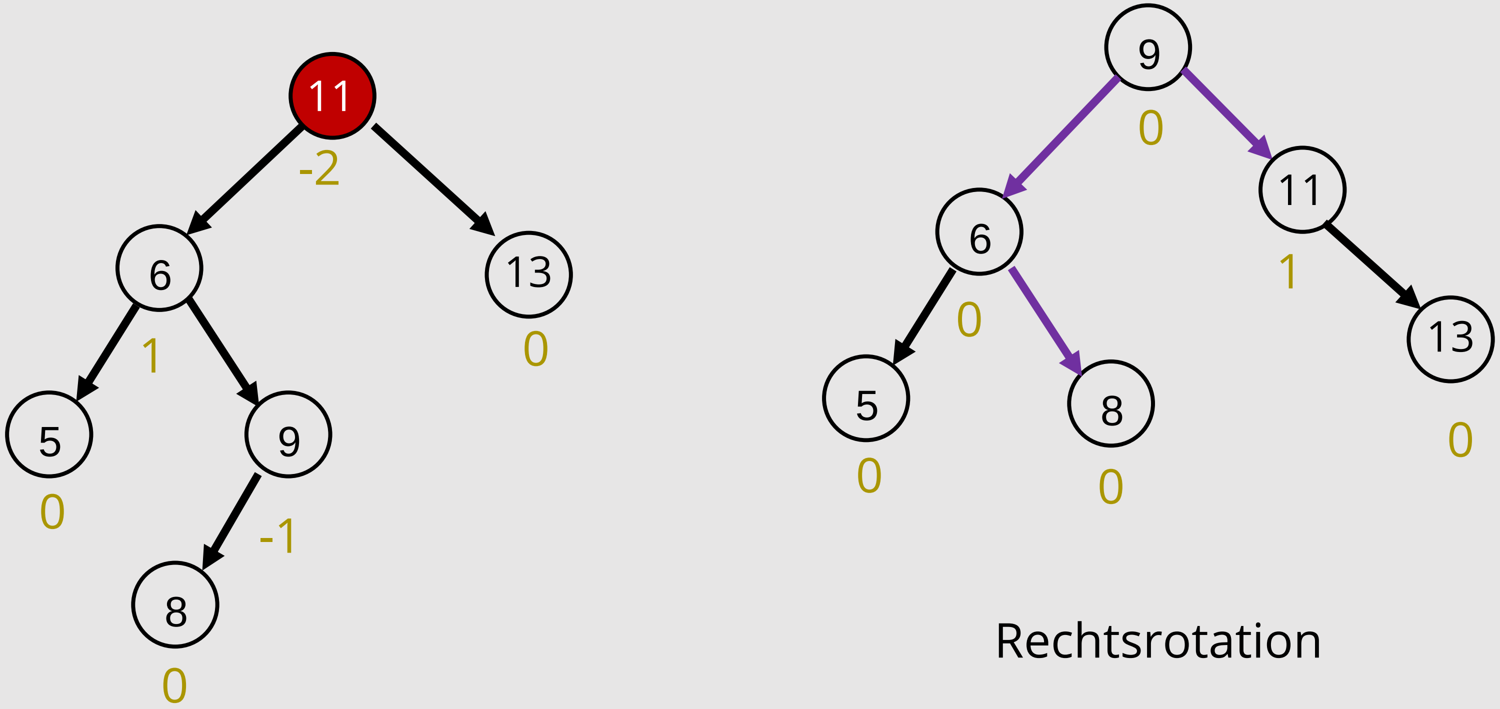
- Löschen von 17 und dann 16 (ein Kindknoten wird versetzt)





LÖSCHEN AUS EINEM AVL-BAUM

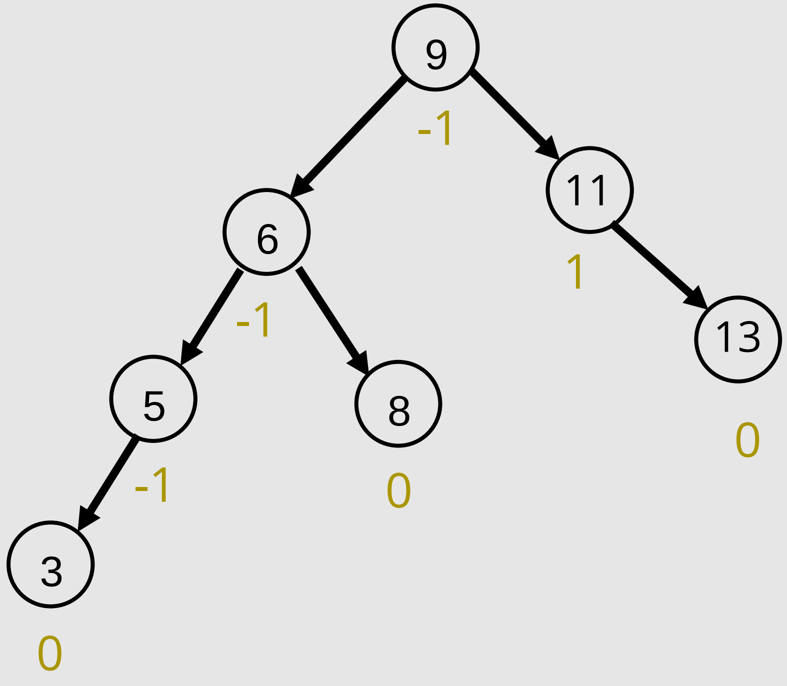
- Rebalancieren nach Löschen von 16 nötig. Negative Balance: Rotation nach rechts über Vorgänger

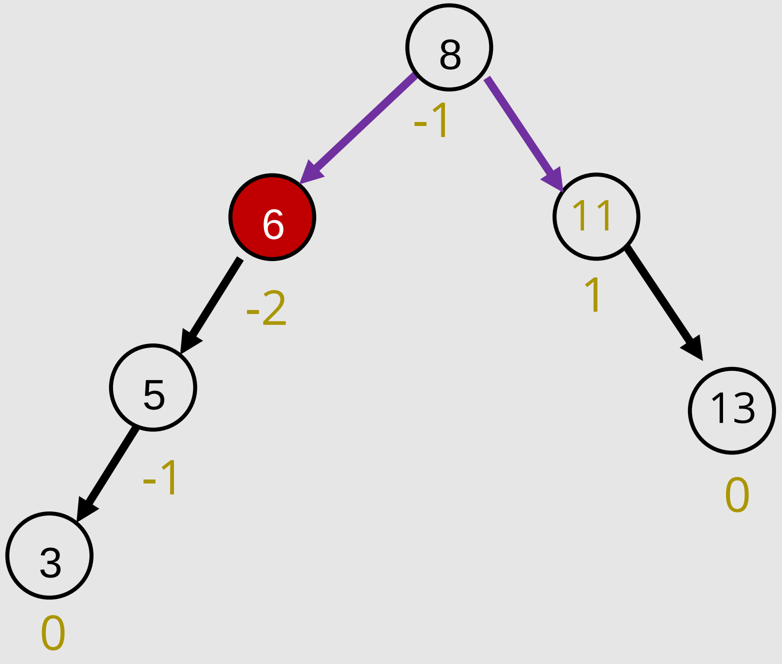


Rechtsrotation

LÖSCHEN AUS EINEM AVL-BAUM

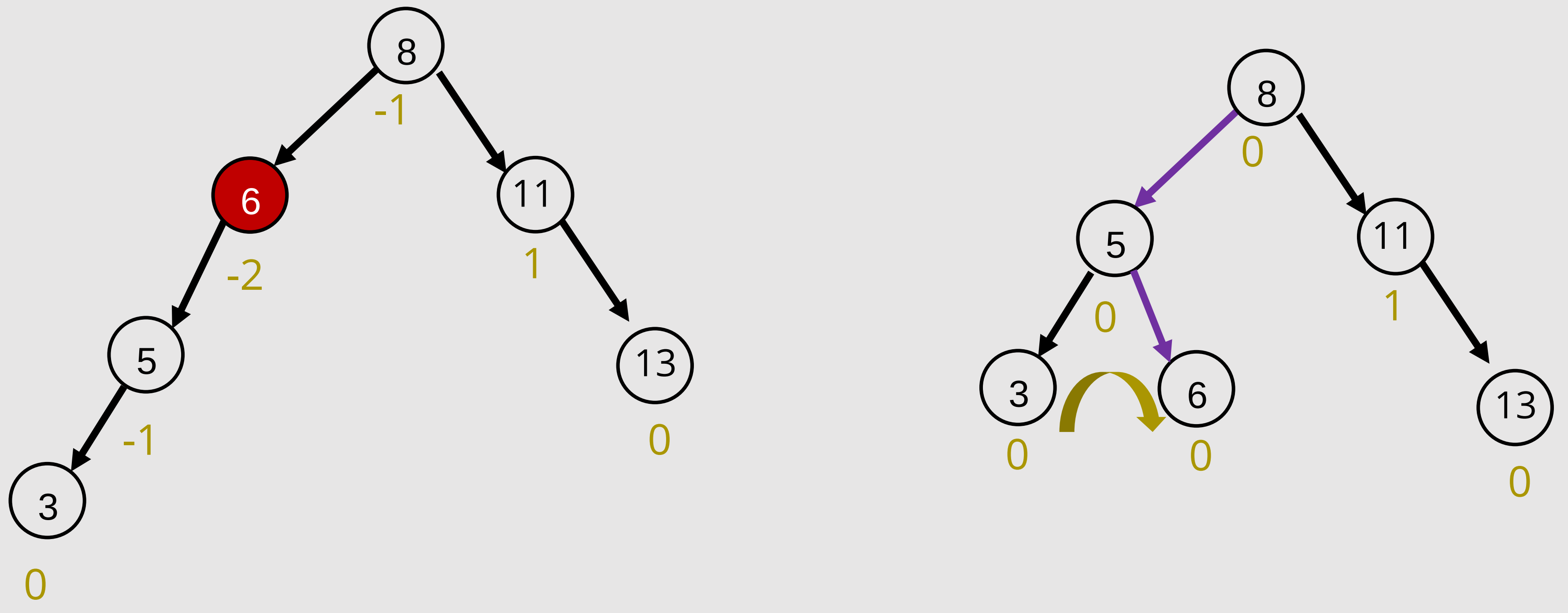
- Einfügen von 3, dann Löschen von 9, Vorgänger von 9 (=8) wird nach oben geholt, da gelöschter Knoten zwei Kinder hat.





LÖSCHEN AUS EINEM AVL-BAUM

- Konfliktknoten 6 führt zu Rotation nach rechts (über Vorgänger von 6), da negative Balance



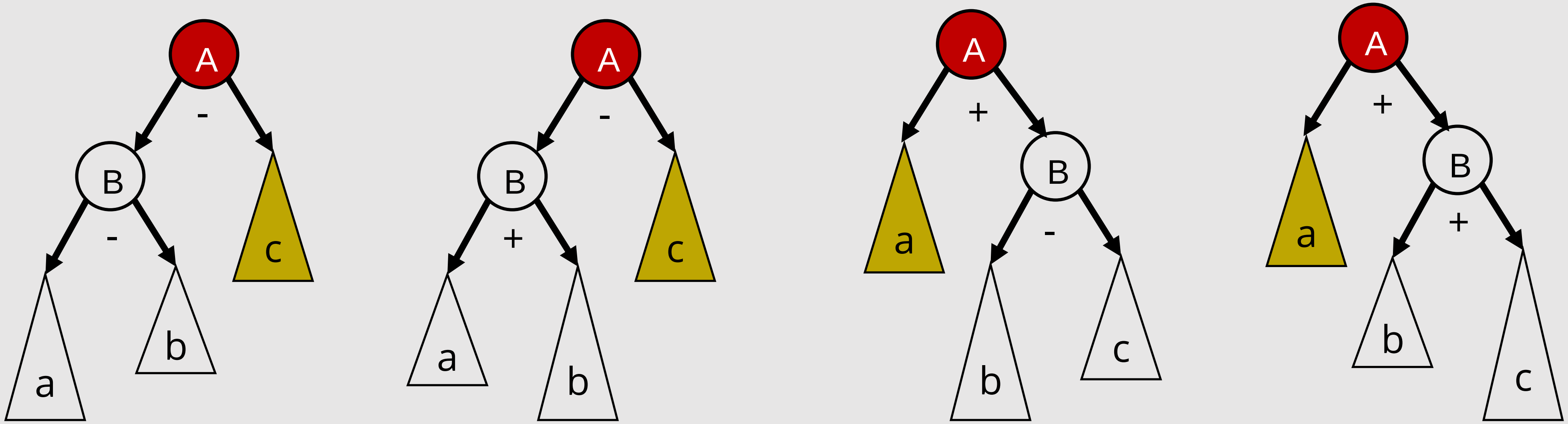
46

LÖSCHEN AUS EINEM AVL-BAUM

Beobachtungen

- Balance-Faktoren ändern sich wieder nur auf dem Pfad von der Wurzel zur Löschposition, nachdem „normales“ Löschen aus BST-Baum durchgeführt wurde.
- Die Rebalancierung findet ausgehend vom tiefsten Balance-Faktor ± 2 (rot gefärbt) in dem Teilbaum statt, in dem nicht gelöscht wurde.

ÜBERSICHT ZUM LÖSCHEN AUS EINEM AVL-BAUM



Balancefaktoren beide negativ => Rechtsrotation	Balancefaktor in A negativ, in B positiv => Doppelrotation rechts	Balancefaktor in A positiv, in B negativ => Doppelrotation links	Balancefaktoren beide positiv => Linksrotation

AVL-BAUM-VISUALISIERUNG ONLINE

- https://www.cs.usfca.edu/~galles/visualization/AVLtree.html
- Funktioniert auf Smartphones nicht so gut (sehr kleine Darstellung)

55

ZUSAMMENFASSUNG

- Bäume und Implementierungsmöglichkeiten
- Anwendung: Verwendung dieses Wissens zur Implementation einer effizienteren Wörterbuchstruktur.
- Alle Wörterbuchoperationen konnten effizient realisiert werden.
  - Ungünstigste Laufzeit: $T(n) = O(n)$
  - Günstigste Laufzeit: $T(n) = O(h) = O(\log n)$

- Wir wissen wenig über die Struktur (v.a. Höhe) eines Baums im allgemeinen Fall.
- Man kann zeigen, dass die erwartete Höhe eines BST im mittleren Fall $O(\log n)$ ist.
- Dennoch: Hinzufügen und Löschen kann im Verlauf der Lebenszeit eines Baums zu ungünstigen, nicht balancierten Bäumen führen.
- AVL-Bäume sind balancierte Bäume
- Sie können nicht zu Listen entarten
- Höhenunterschied einzelner Teilbäume maximal 1
- Zum Wiederherstellen der Balance beim Einfügen und Löschen können Rotationen durchgeführt werden