Fachhochschule
Dortmund

University of Applied Sciences and Arts
Fachbereich Informatik

Algorithmen und Datenstrukturen

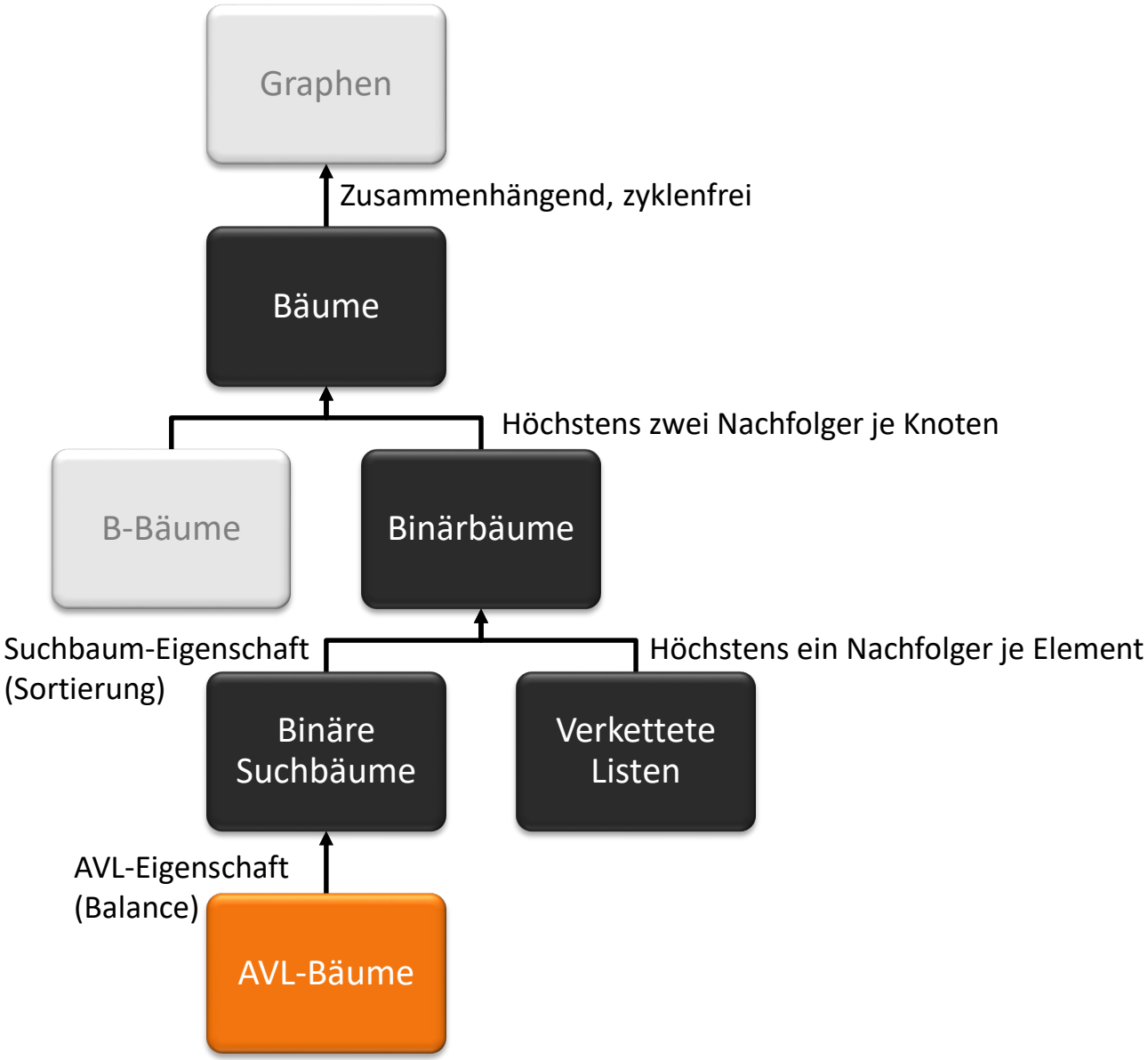
VL07 – AVL-Bäume

Inhalt

- AVL-Bäume
  - Motivation
  - Definition
  - Suchen, Einfügen und Löschen
  - Java-Programm
  - Komplexität

AVL-Bäume

Übersichtsgrafik



AVL-Bäume

Motivation

Motivation

Optimale und degenerierte Suchbäume

- Sie kennen Binäre Bäume als Datenstruktur für das Einfügen, Entfernen und Suchen von Daten.
- Die Reihenfolge der Schlüssel beim Einfügen bestimmt die Baumstruktur: zwischen optimal und degeneriert zu einer linearen Liste.
- Bei einer optimalen Struktur ist der Aufwand für das Suchen eines Schlüssels $O(\log_2 n)$, bei einem degenerierten Baum nur $O(n)$.

Motivation

Ausgeglichene (balancierte) Bäume

- Wie kann verhindert werden, dass ein Suchbaum bei einer ungünstigen Einfügereihenfolge entartet?
  - Ziel sind ausgeglichene (balancierte) Bäume, die bei vorgegebener Anzahl n von Elementen eine möglichst kleine Höhe h haben.
- Ansätze:
  - AVL-Bäume
  - Rot-Schwarz-Bäume
  - B-Bäume (VL08)

AVL-Bäume

Definition

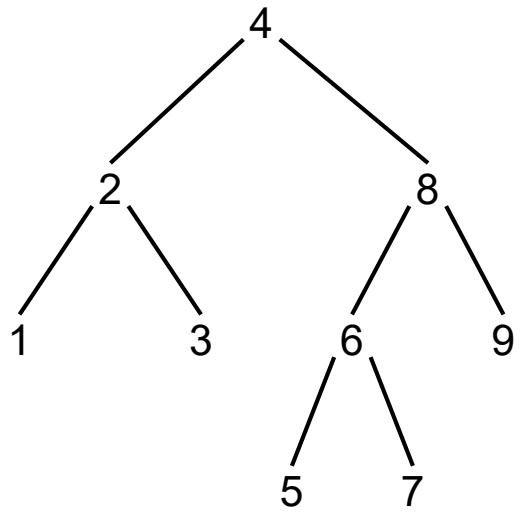
Definition

AVL-Bäume sind höhenbalanciert

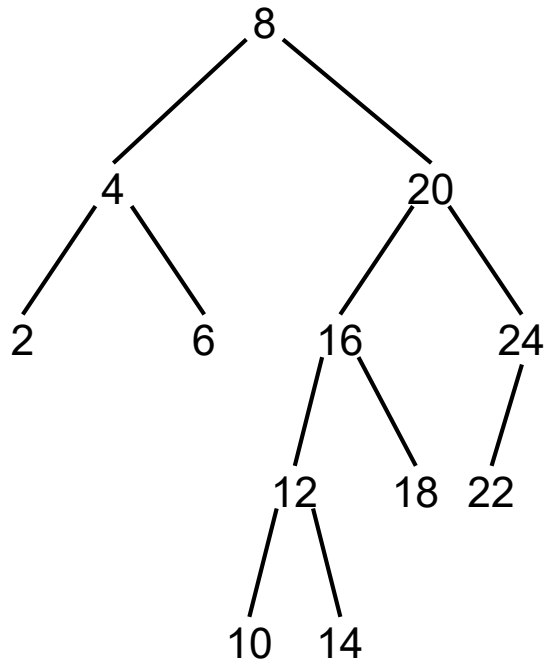
- Als typischen (und historisch ersten) Vertreter höhenbalancierter Bäume lernen wir AVL-Bäume kennen:
  - Diese wurden von Adelson-Velskij und Landis im Jahr 1962 vorgeschlagen.
  - Ein Binärer Suchbaum ist ein AVL-Baum, wenn sich für jeden Knoten die Höhen der zugehörigen Teilbäume um höchstens 1 unterscheiden (**Balancefaktor**).
  - Ein AVL-Baum mit n Knoten hat höchstens eine logarithmische Höhe, genauer: $h \leq 1{,}441 * \log_2 (n+2) - 0{,}327$.

Definition

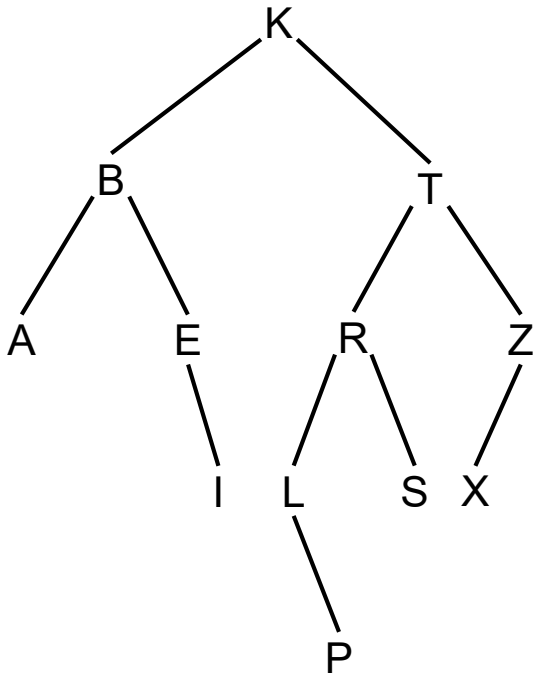
Beispiele und Gegenbeispiele



AVL-Baum



Kein AVL-Baum



AVL-Baum

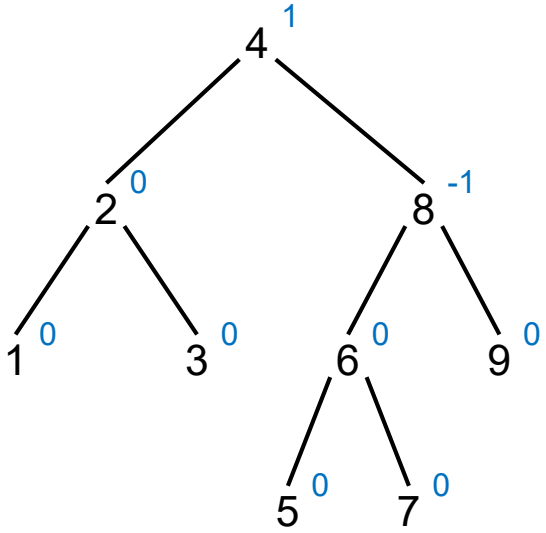
Definition

Balance-Faktor

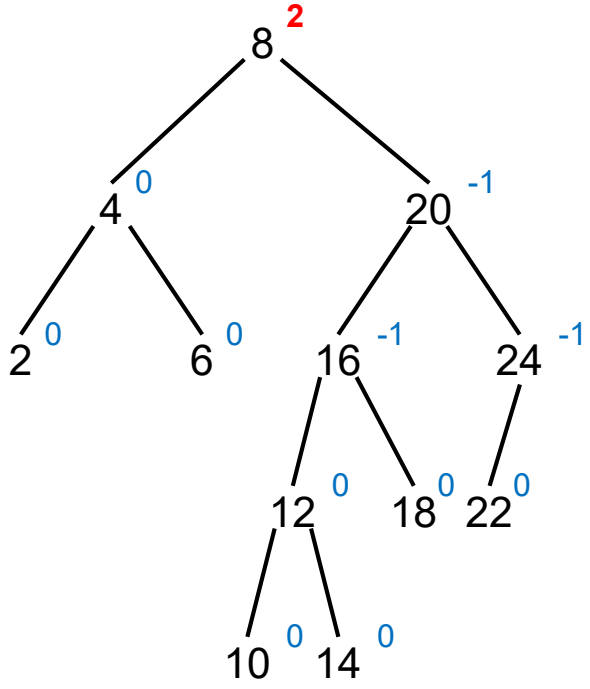
- Um die Ausgeglichenheit eines Knotens festzustellen, muss man nicht die Höhen beider Teilbäume abspeichern. Es genügt, den Balancefaktor $\mathrm{bal}(k)$ für jeden Knoten k zu speichern:
  - $\mathrm{bal}(k) = h(\text{rechter Teilbaum von } k) - h(\text{linker Teilbaum von } k)$
  - Wahlweise auch umgekehrt, das Vorzeichen von $\mathrm{bal}(k)$ ist irrelevant.

Definition

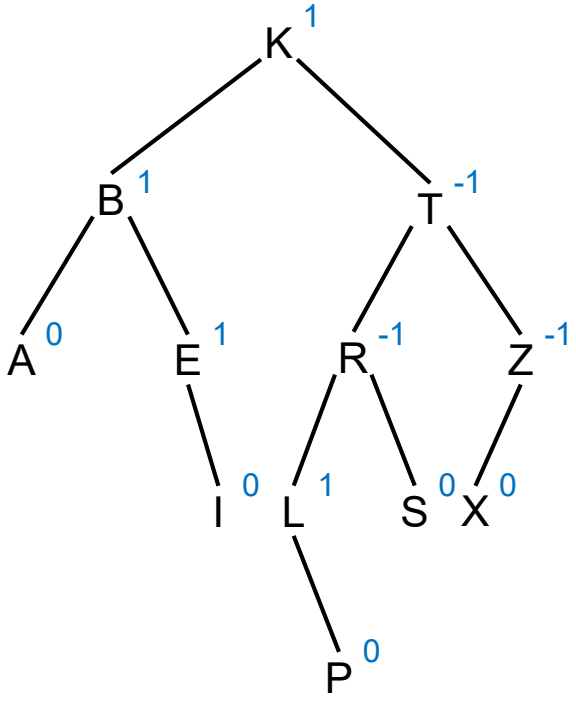
Balance-Faktor



AVL-Baum



Kein AVL-Baum



AVL-Baum

AVL-Bäume

Suchen, Einfügen und Löschen

AVL-Bäume

Suchen, Einfügen und Löschen

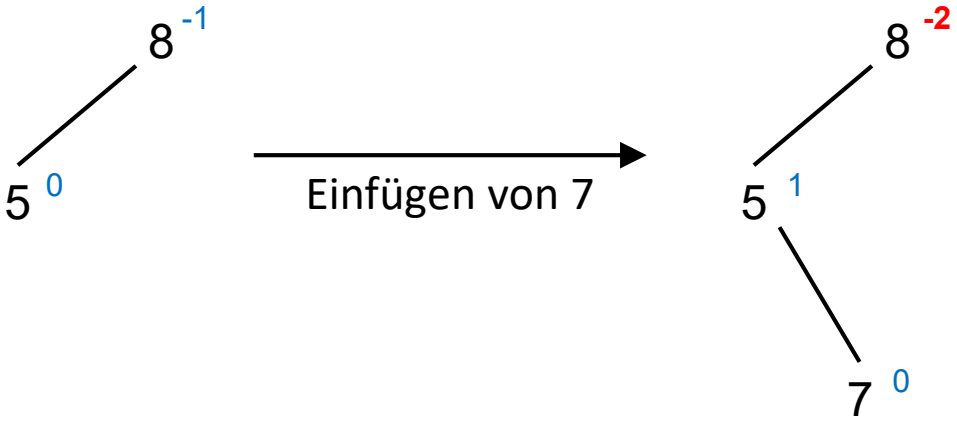
- Das Suchen nach Elementen erfolgt genau wie bei einem Binären Suchbaum (AVL-Bäume **sind** Binäre Suchbäume).

- Das Einfügen von Elementen erfolgt auch **zunächst** genau wie bei einem Binären Suchbaum:
  - Wenn das gesuchte Element noch nicht vorhanden ist, endet die Suche in einem Knoten, der den Schlüssel nicht enthält und dessen Teilbaum, in dem weiter gesucht werden müsste, leer ist.
  - Der einzufügende Schlüssel wird an der Stelle des leeren Teilbaums als neues Blatt eingefügt.

Suchen, Einfügen und Löschen

Einfügen und Verlust an Ausgeglichenheit

- Nach dem Einfügen kann es jedoch sein, dass der „AVL-Baum“ nicht mehr ausgeglichen ist:



- Die Ausgeglichenheit muss also nach dem naiven Einfügen wiederhergestellt werden, siehe Animation: https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

Suchen, Einfügen und Löschen

Wiederherstellen von Ausgeglichenheit

- Hierzu wird ausgehend von der Einfügestelle (dem neuen Blatt) der Baum rückwärts entlang des Suchpfades bis zur Wurzel durchlaufen.
- An jedem hierbei durchlaufenen Knoten wird geprüft, ob der mit diesem Knoten als Wurzel beginnende Teilbaum noch ausgeglichen ist.
- Ist dies **nicht** der Fall, so muss der entsprechende Teilbaum durch geeignete Operationen ausgeglichen werden!

Suchen, Einfügen und Löschen

Beispielfall: links wird eingefügt

- Noch offen ist die Frage, wie genau der nach einem Einfügen evtl. notwendige Ausgleich zu erfolgen hat.
  - Wir wollen den Fall betrachten, dass ein Element in den linken Teilbaum eines Knotens `k` eingefügt und dabei dessen Höhe verändert wurde.
  - Der Fall „rechter Teilbaum“ verhält sich symmetrisch dazu und soll daher hier nicht näher ausgeführt werden.

- Ein neuer Knoten werde in den linken Teilbaum eingefügt. Vor dem Einfügen können drei verschiedene Fälle existieren:
  1. Der Balancefaktor von `k` war +1.
  2. Der Balancefaktor von `k` war 0.
  3. Der Balancefaktor von `k` war -1.
  - Andere Balancefaktoren (z.B. **-2** oder **2**) sind unzulässig, d.h. der Baum wäre dann bereits vor dem Einfügen kein gültiger AVL-Baum! (Widerspruch zur Voraussetzung des Algorithmus)

Rebalancierung

Der Algorithmus auf einer Folie (Übersicht und Lernhilfe)

- Weg zur Wurzel beschreiten:
  - Vom neu eingefügten Knoten (der immer ein Blatt ist) wird der Weg zur Wurzel des Baums verfolgt.
- Balancefaktoren neu berechnen:
  - Entlang des Weges werden die Balancefaktoren neu berechnet, bis ein Balancefaktor von **2** oder **-2** auftritt.
  - Größere Abweichungen sind nicht möglich, wenn der Baum vorher ein AVL-Baum war.
- Pivot-Element ermitteln:
  - Wähle aus der Menge des defekten Knotens und der zwei vorherigen Knoten des Weges den Knoten mit dem mittleren Wert aus (das **Pivot-Element** oder **Dreh-Element**). Dieses Element soll nach den Rotationen die Wurzel des betrachteten Teilbaums werden. Das Pivot-Element kann nie der defekte Knoten selbst sein (siehe unten).
- Rotationen bestimmen:
  - Handelt es sich beim Pivot-Element um den mittleren der drei betrachteten Knoten, so reicht eine Einfachrotation, ist es der unterste Knoten, so braucht es eine Doppelrotation (siehe unten).
- Richtung der Rotationen bestimmen:
  - Ist das Pivot-Element ein linkes Kind, so braucht es (zunächst) eine Rotation nach rechts. Ist das Pivot-Element ein rechtes Kind, muss nach links rotiert werden. Bei Doppelrotationen folgt eine weitere Rotation in die andere Richtung (siehe unten).

Rebalancierung

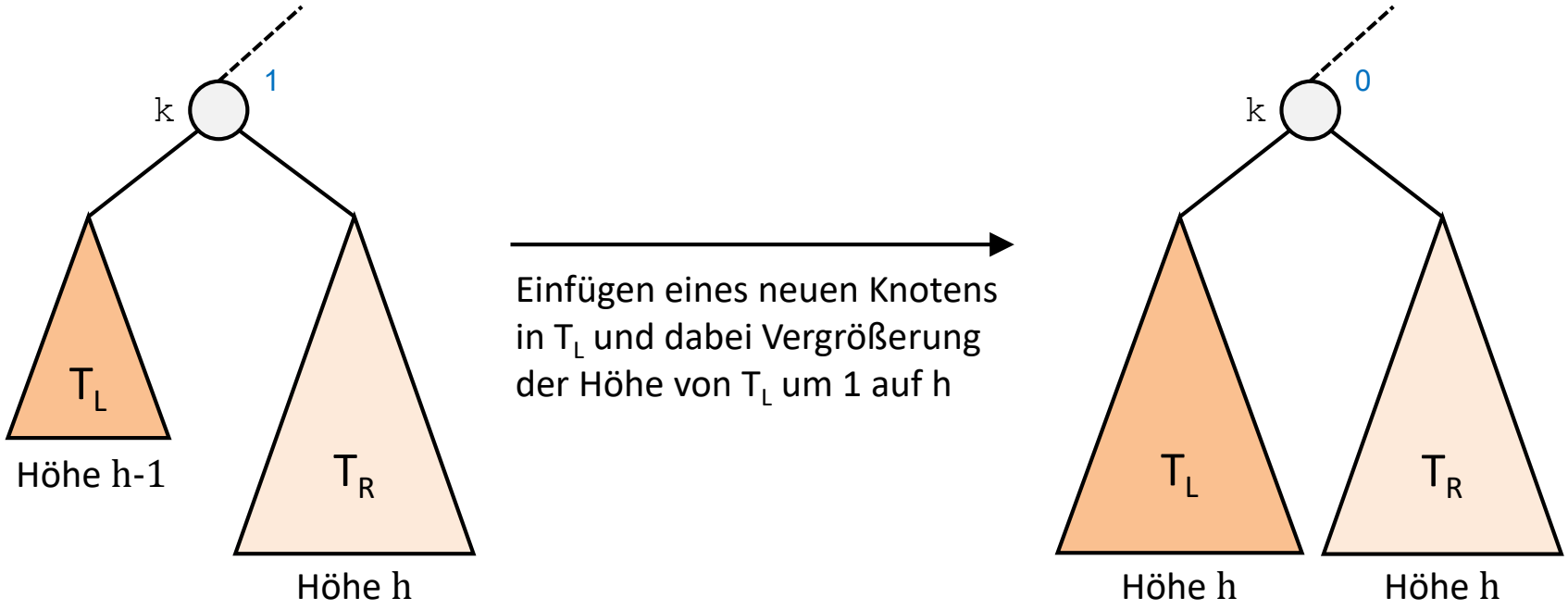
Der Algorithmus im Detail (Verstehen, Teil 1)

- Weg zur Wurzel beschreiten:
  - Vom neu eingefügten Knoten (der immer ein Blatt ist) wird der Weg zur Wurzel des Baums verfolgt.
- Balancefaktoren neu berechnen:
  - Entlang des Weges werden die Balancefaktoren neu berechnet, bis ein Balancefaktor von **2** oder **-2** auftritt.
  - Größere Abweichungen sind nicht möglich, wenn der Baum vorher ein gültiger AVL-Baum war.

Rebalancierung

Fall 1 (links wird eingefügt, Balancefaktor war +1)

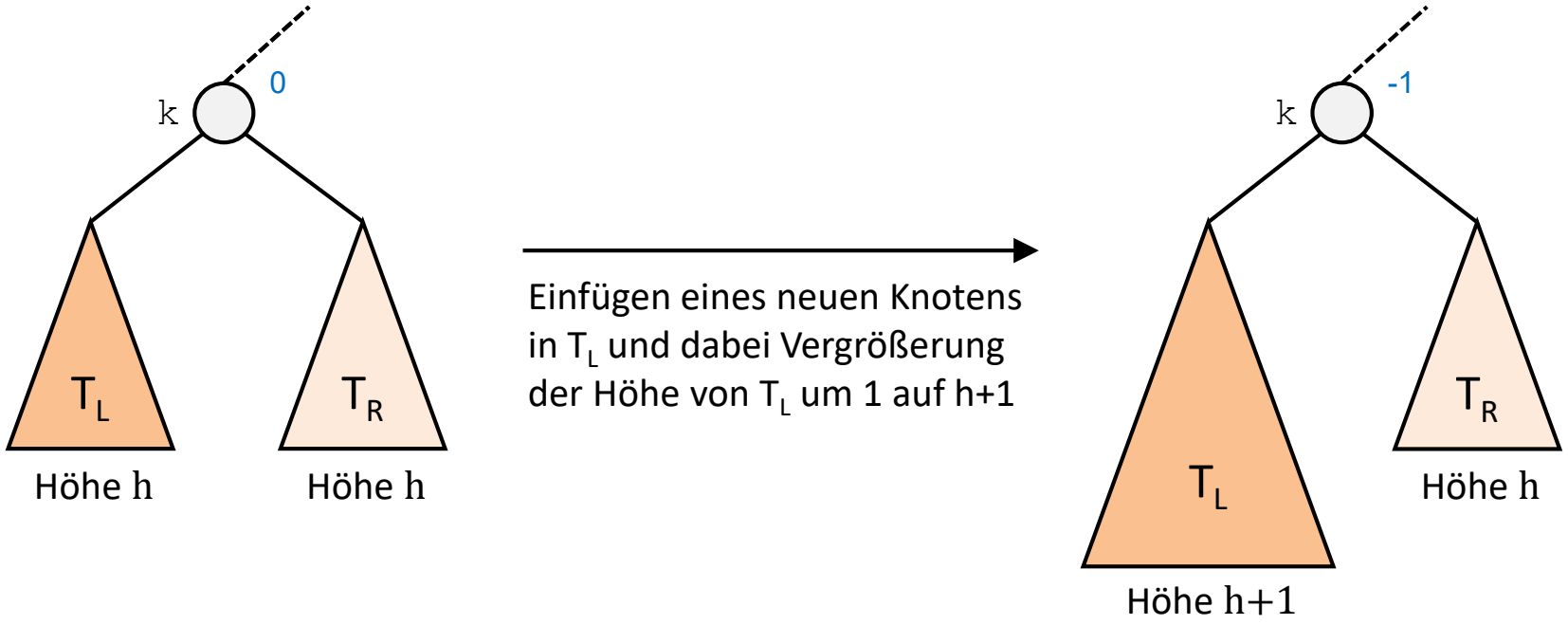
- Der Balancefaktor war +1:
  - Der rechte Teilbaum von k war vor dem Einfügen um ein Niveau höher als der linke Teilbaum.
  - Der Knoten k ist jetzt ausbalanciert, die Gesamthöhe von k hat sich nicht verändert.
  - Neuer Balancefaktor: 0 (keine weiteren Maßnahmen erforderlich)



Rebalancierung

Fall 2 (links wird eingefügt, Balancefaktor war 0)

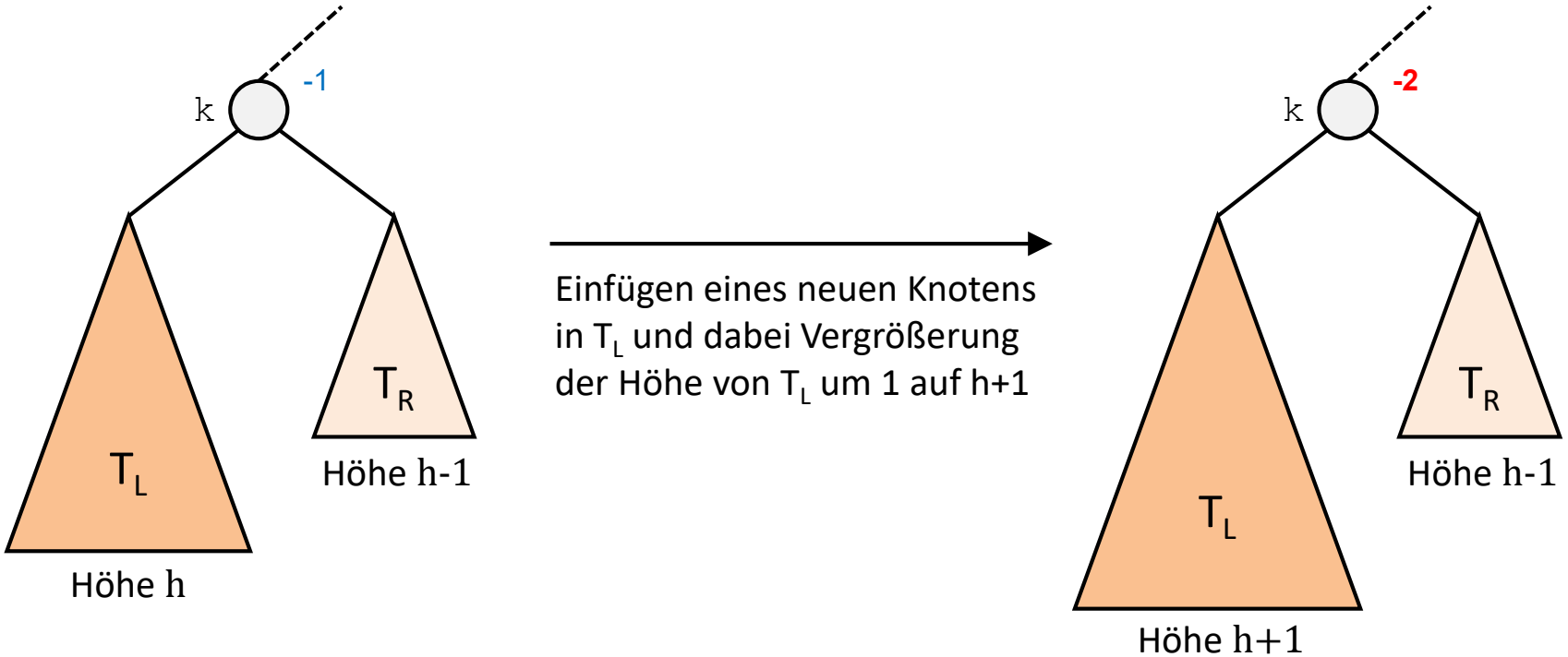
- Der Balancefaktor war 0:
  - Der rechte Teilbaum von k war vor dem Einfügen gleich hoch wie der linke Teilbaum.
  - Der linke Teilbaum von k ist jetzt um ein Niveau höher als der rechte Teilbaum, die Gesamthöhe von k hat sich verändert.
  - Neuer Balancefaktor: -1 (Änderung wird nach oben propagiert, keine weiteren Maßnahmen erforderlich)



Rebalancierung

Fall 3 (links wird eingefügt, Balancefaktor war -1)

- Der Balancefaktor war -1:
  - Der rechte Teilbaum von k war vor dem Einfügen um ein Niveau niedriger als der linke Teilbaum.
  - Der Knoten k hat jetzt einen ungültigen Balancefaktor, die Gesamthöhe von k hat sich verändert.
  - Neuer Balancefaktor: **-2**



Rebalancierung

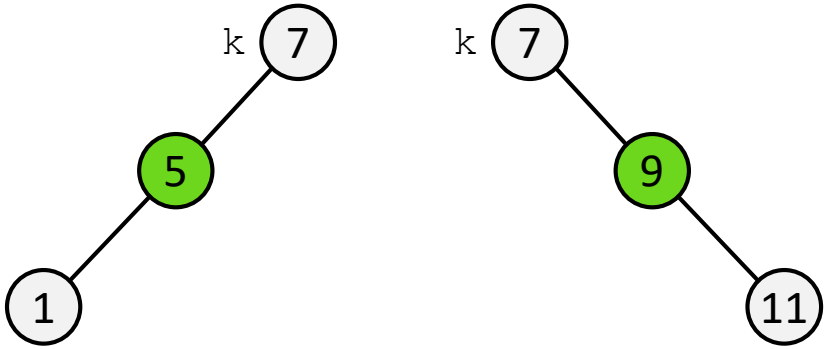
Der Algorithmus im Detail (Verstehen, Teil 2)

- Pivot-Element ermitteln:
  - Wähle aus der Menge des defekten Knotens und der zwei vorherigen Knoten des Weges den Knoten mit dem **mittleren Wert** aus (das **Pivot-Element** oder **Dreh-Element**). Dieses Element soll nach den Rotationen die Wurzel des betrachteten Teilbaums werden. Das Pivot-Element kann nie der defekte Knoten selbst sein (siehe unten).

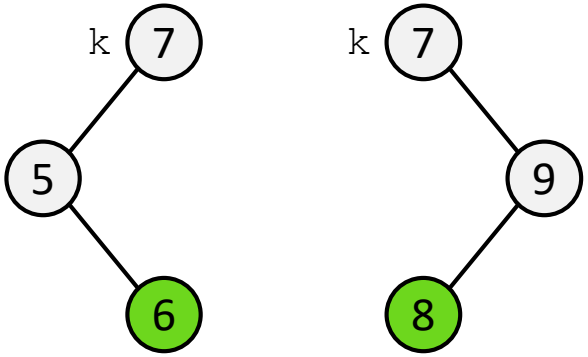
Rebalancierung

Auftretende Fälle (einfaches Beispiel ohne Teilbäume)

- Gerader Pfad = Knoten mit mittlerem Wert (**Pivot-Element**) steht in der Mitte:



- Geknickter Pfad = Knoten mit mittlerem Wert (**Pivot-Element**) steht unten:



Rebalancierung

Rotationen

- Das Pivot-Element kann niemals ganz oben stehen!
  - Der Weg vom Blatt trifft in jedem Fall entweder von links oder rechts her auf einen Knoten, d.h. alle bisher betrachteten Elemente sind entweder alle größer oder alle kleiner als der neue Knoten.

- Aufgabe: durch geeigneten Umbau des Baumes (Rotationen) das Pivot-Element zur Wurzel des Teilbaums machen
  - Da es den mittleren Wert hat, ist es ideal zur Balance!

- Jede einfache Rotation bewegt das Pivot-Element eine Ebene nach oben:
  - Gerader Pfad = Pivot-Element in der Mitte: Einfachrotation
  - Geknickter Pfad = Pivot-Element unten: Doppelrotation

Rebalancierung

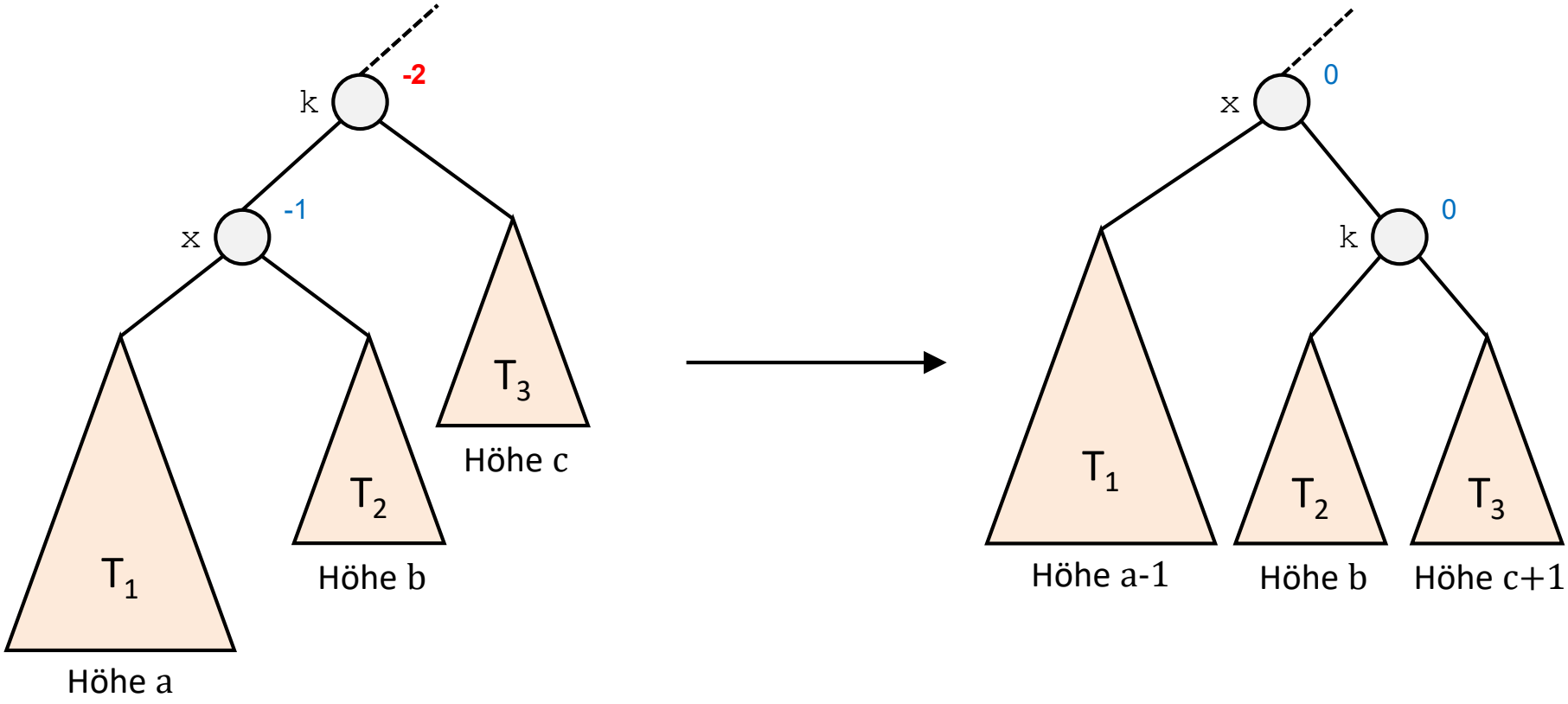
Der Algorithmus im Detail (Verstehen, Teil 3)

- Rotationen bestimmen:
  - Handelt es sich beim Pivot-Element um den mittleren der drei betrachteten Knoten, so reicht eine Einfachrotation, ist es der unterste Knoten, so braucht es eine Doppelrotation (siehe unten).
- Richtung der Rotationen bestimmen:
  - Ist das Pivot-Element ein linkes Kind, so braucht es (zunächst) eine Rotation nach rechts. Ist das Pivot-Element ein rechtes Kind, muss nach links rotiert werden. Bei Doppelrotationen folgt eine weitere Rotation in die andere Richtung (siehe unten).

Rebalancierung

Rotation nach rechts (isolierte Betrachtung)

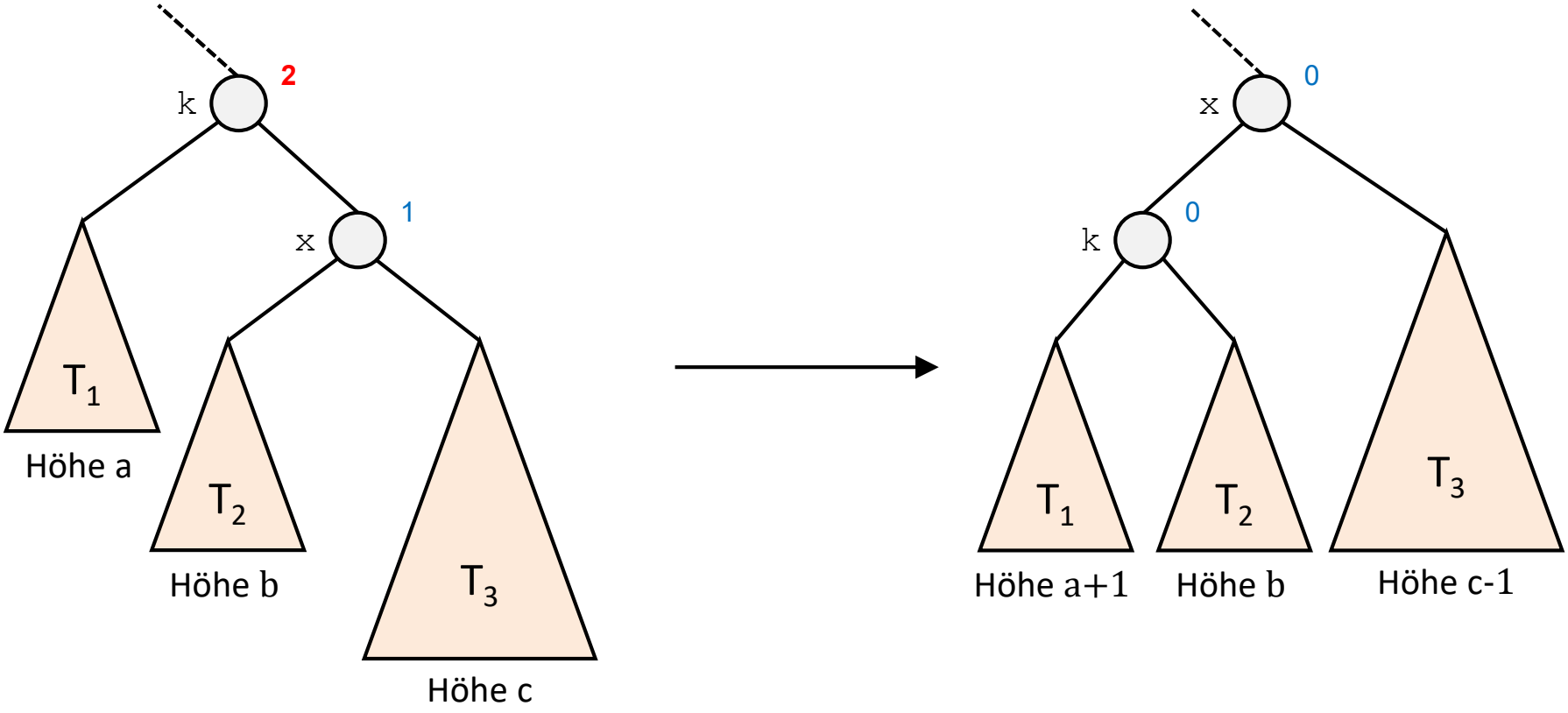
- Rotation nach rechts um den Knoten `k` (siehe Folien 29, 37):
  - Der Knoten `x` wird für den betrachteten Teilbaum die neue Wurzel.
  - Der Knoten `k` wandert eine Ebene nach unten und wird ein Kind von `x`.
  - Der Teilbaum T_2 wird neuer linker Teilbaum von `k`.



Rebalancierung

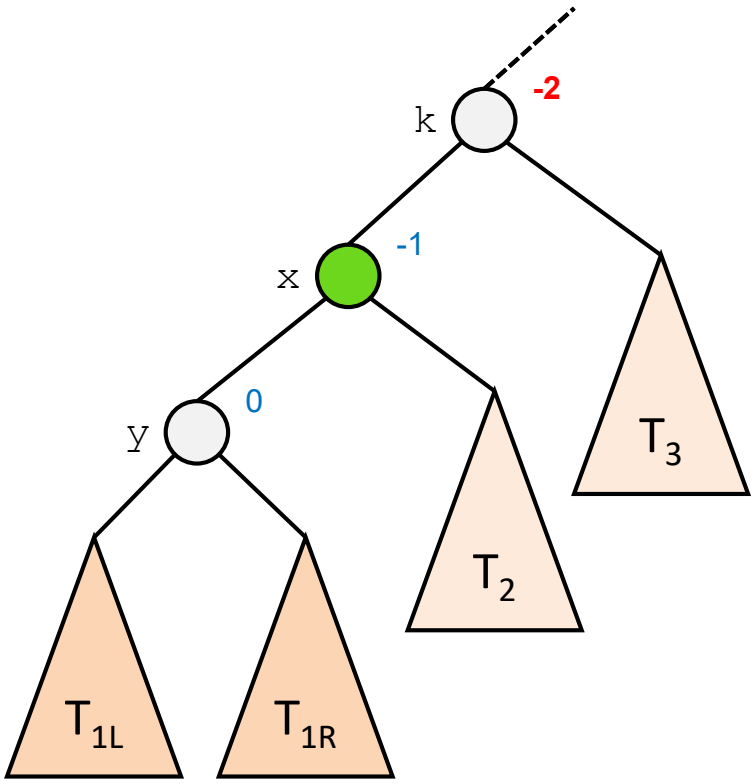
Rotation nach links (isolierte Betrachtung)

- Rotation nach links um den Knoten `k`:
  - Der Knoten `x` wird für den betrachteten Teilbaum die neue Wurzel.
  - Der Knoten `k` wandert eine Ebene nach unten und wird ein Kind von `x`.
  - Der Teilbaum T_2 wird neuer rechter Teilbaum von `k`.



Rebalancierung

Fall 3a (Pivot-Element ist mittlerer Knoten)

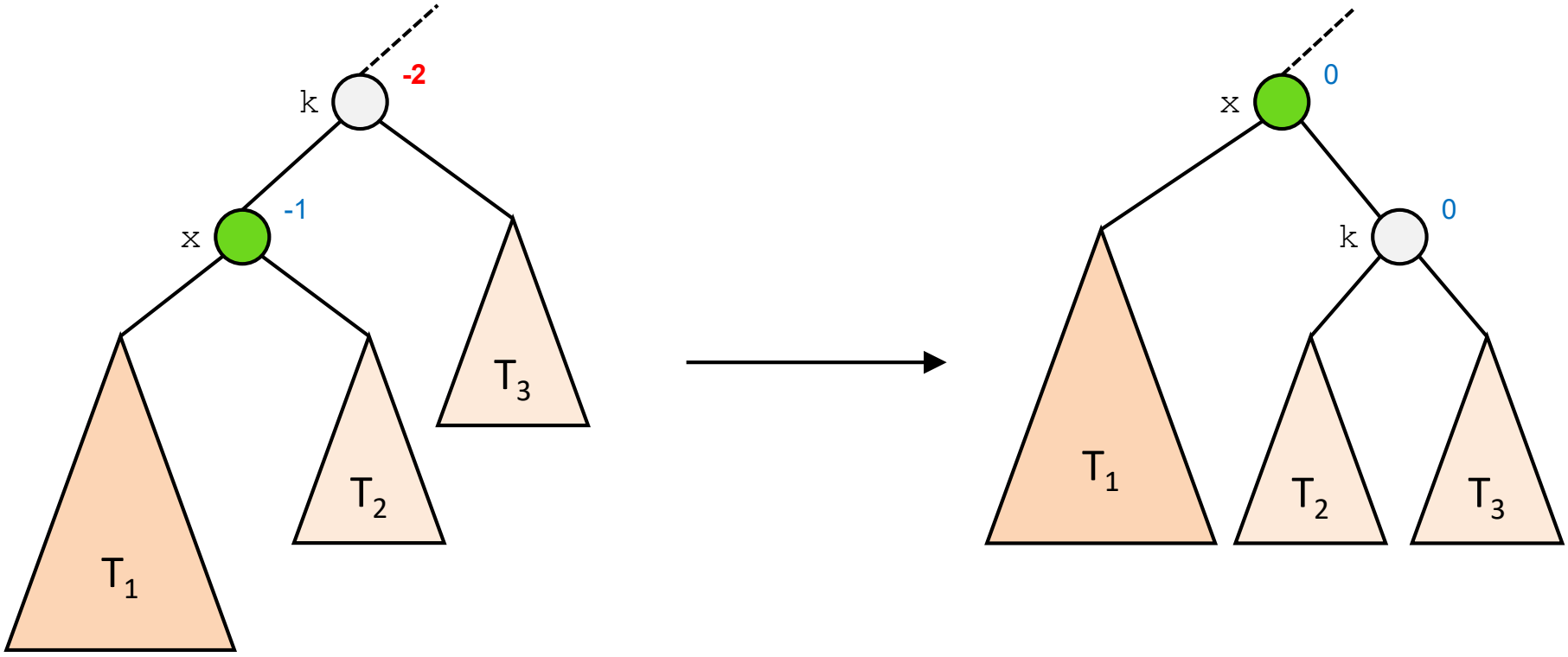


- k ist der erste Knoten auf dem Weg zur Wurzel mit ungültiger Balance (**-2**).
- Zu betrachtende Menge von Knoten: k, x, y
- Das **Pivot-Element** (Knoten mit dem mittlerem Wert) ist x.
  - Position innerhalb der drei Knoten: Mittlerer Knoten = Gerader Pfad
- Durchzuführende Operation ist eine Einfachrotation nach rechts:
  - Pivot-Element muss ein Niveau höher platziert werden, also Einfachrotation.
  - x ist im Beispiel ein linkes Kind.

Rebalancierung

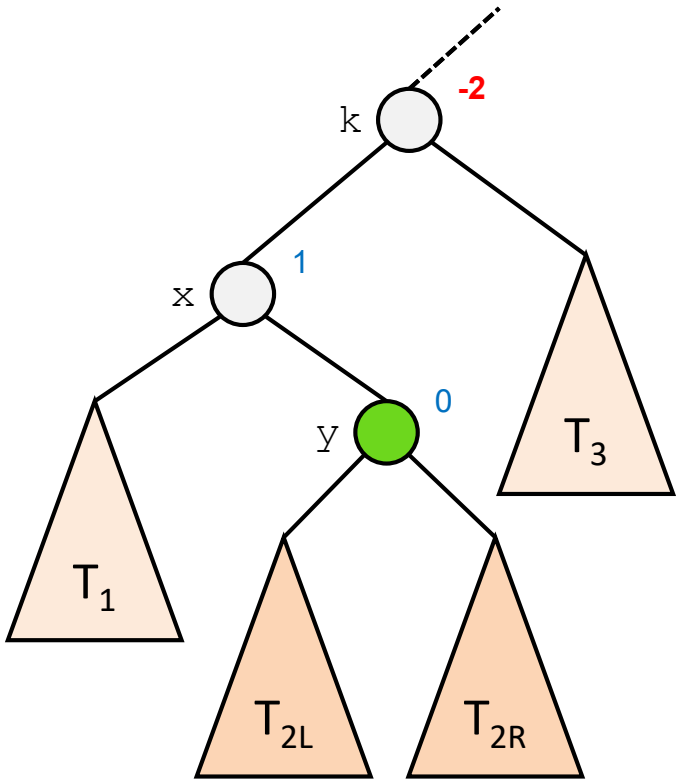
Fall 3a: Rotation nach rechts

- Rotation nach rechts um den Knoten `k` (siehe Folie 37):
  - Der Knoten `x` wird für den betrachteten Teilbaum die neue Wurzel.
  - Der Knoten `k` wandert eine Ebene nach unten und wird ein Kind von `x`.
  - Der Teilbaum T_2 wird neuer linker Teilbaum von `k`.



Rebalancierung

Fall 3b (Pivot-Element ist unterer Knoten)

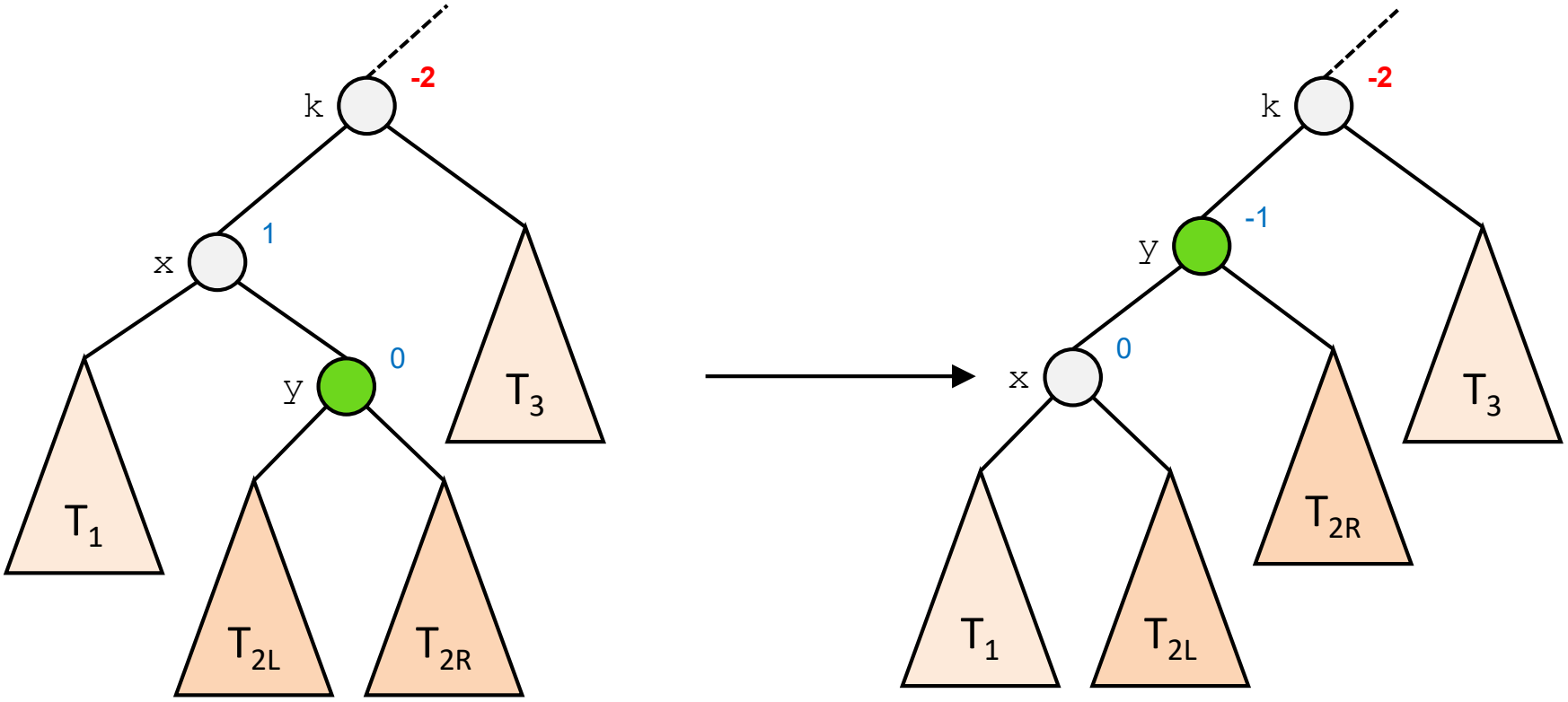


- k ist der erste Knoten auf dem Weg zur Wurzel mit ungültiger Balance (**-2**).
- Zu betrachtende Menge von Knoten: k, x, y
- Das **Pivot-Element** (Knoten mit dem mittlerem Wert) ist y.
  - Position innerhalb der drei Knoten: Unterer Knoten = Genickter Pfad
- Durchzuführende Operation ist eine Links-Rechts-Doppelrotation:
  - Pivot-Element muss zwei Niveaus höher platziert werden, also Doppelrotation.
  - y ist im Beispiel ein rechtes Kind.

Rebalancierung

Fall 3b: Links-Rechts-Doppelrotation

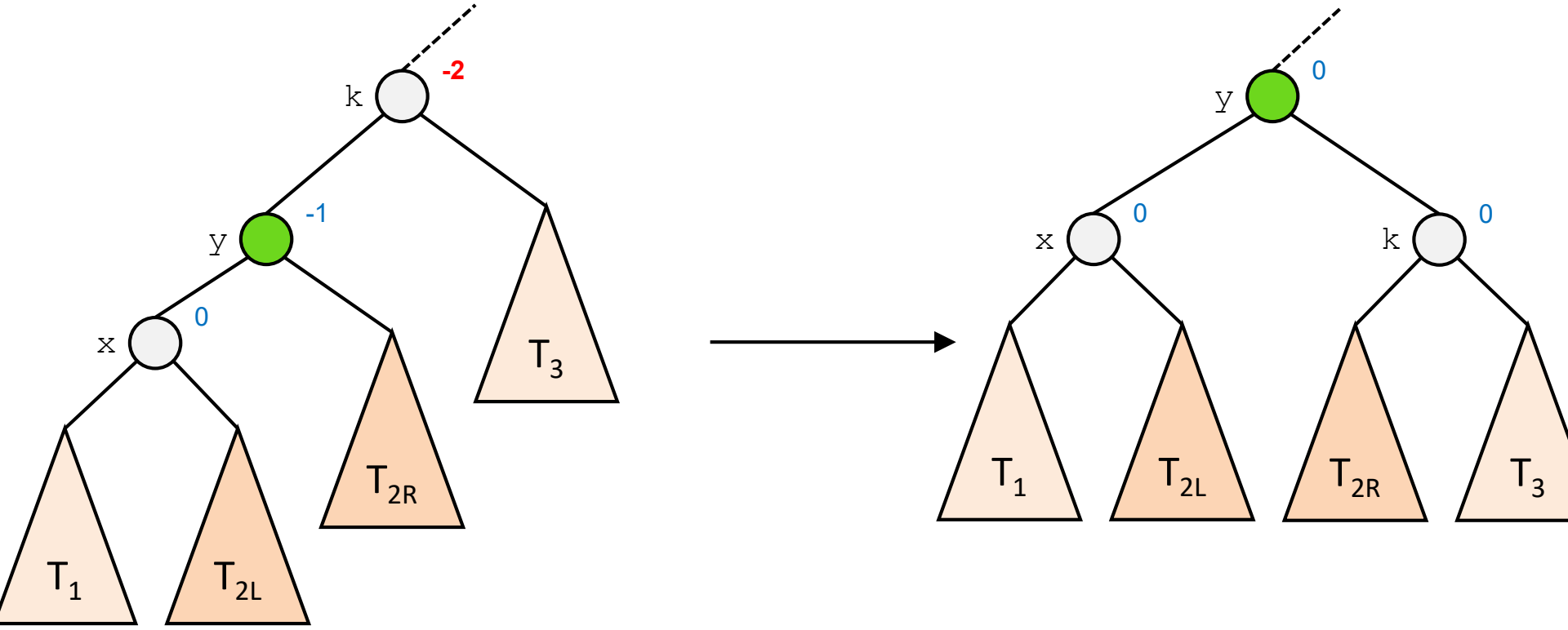
- Zunächst Rotation nach links um den Knoten `x`:
  - Der Knoten `y` wird für den linken Teilbaum von `k` die neue Wurzel.
  - Der Knoten `x` wandert eine Ebene nach unten und wird ein Kind von `y`.
  - Der Teilbaum T_{2L} wird neuer rechter Teilbaum von `x`.



Rebalancierung

Fall 3b: Links-Rechts-Doppelrotation

- Dann Rotation nach rechts um den Knoten `k`:
  - Der Knoten `y` wird für den betrachteten Teilbaum die neue Wurzel.
  - Der Knoten `k` wandert eine Ebene nach unten und wird ein Kind von `y`.
  - Der Teilbaum T_{2R} wird neuer linker Teilbaum von `k`.



Suchen, Einfügen und Löschen

Löschen

- Es scheint ein Gesetz zu sein, dass das Entfernen von Elementen oft aufwändiger ist als das Einfügen:
  - Siehe Praktikumsaufgabe zu verketteten Listen (VL03)
  - Das gilt auch für Hashsets (VL11) und AVL-Bäume!

- Das Löschen von Elementen erfolgt **zunächst** genau wie bei einem Binären Suchbaum:
  - Wenn das Opfer höchstens einen Nachfolger hat, wird der Knoten durch Umsetzen der Zeiger aus dem AVL-Baum entfernt.
  - Hat das Opfer zwei Nachfolger, wird es z.B. durch den größten Knoten im linken Teilbaum ersetzt. Danach wird der nun verwaiste Knoten gelöscht (er kann höchstens einen Nachfolger haben).
  - Danach muss analog zum Einfügen der Weg zur Wurzel zurückgegangen werden. Dabei muss ggf. **jeder Knoten** auf diesem Weg (nicht nur höchstens einer wie beim Einfügen) rebalanciert werden!

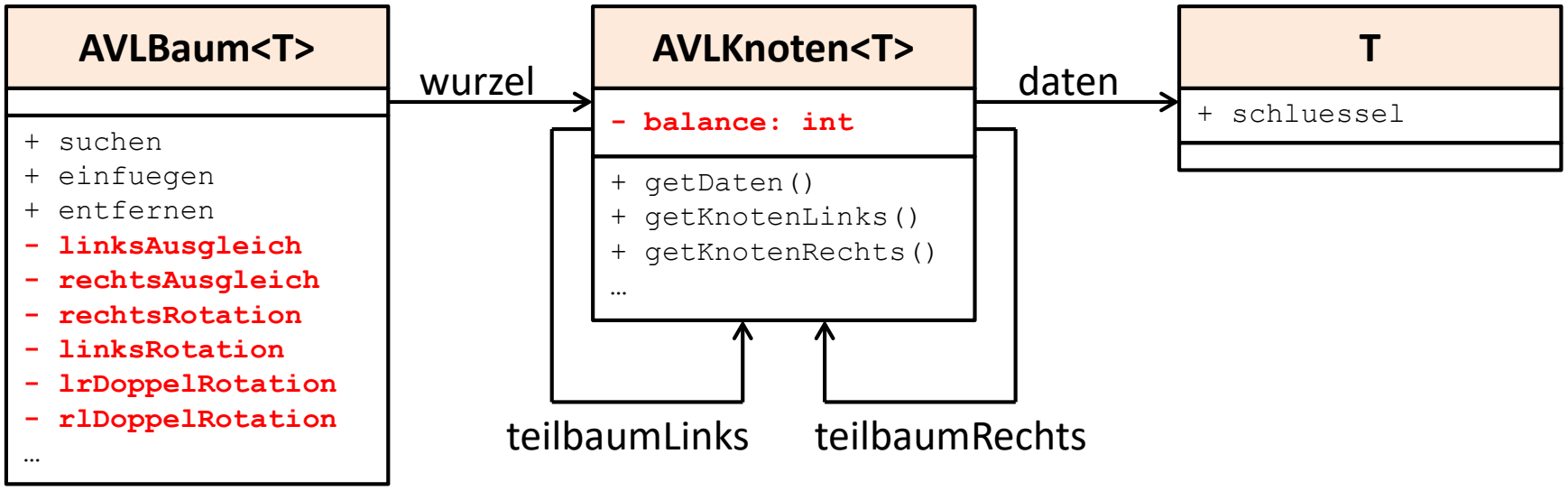
AVL-Bäume

Java-Programm

Java-Programm

Klassendiagramm

- Geeignete Struktur zur Speicherung von Daten mit Schlüsseln:
  - Der Knoten des Binären Suchbaums wird um den Balancefaktor `balance` vom Typ `int` erweitert:



Java-Programm

Übersicht

- Die Methode `einfuegen` ist rekursiv programmiert:
  - In einem Binären Suchbaum kann das Einfügen iterativ programmiert werden, da immer nur ein Pfad verfolgt wird (siehe VL05 und VL06).
  - Dies gilt auch bei einem AVL-Baum. Allerdings müssen noch auf dem Weg zur Wurzel die Balance-Faktoren überprüft werden!
- Zusätzlich wird die Methode `einfuegen` des Binären Suchbaums um den Links- bzw. Rechtsausgleich erweitert:
  - Methode `einfuegen` in Datei `AVLBaum.java` (`BSP07-AVLBaum.zip`)
  - Methode `linksAusgleich` in Datei `AVLBaum.java`:
    - Fall 1: siehe Methode `rechtsRotation` in Datei `AVLBaum.java`
    - Fall 2: siehe Methode `lrDoppelRotation` in Datei `AVLBaum.java`
  - Die Methode `rechtsAusgleich` und die dort verwendeten Methoden `linksRotation` und `rlDoppelRotation` sind symmetrisch dazu.

Java-Programm

Ausschitt: Rechts-Rotation

- Methode `rechtsRotation` aus `AVLBaum.java`:

```
private AVLKnoten<T> rechtsRotation(AVLKnoten<T> k,
    AVLKnoten<T> x)
{
  k.setKnotenLinks(x.getKnotenRechts());
  x.setKnotenRechts(k);
  k.setBalance(0);

  return x;
}
```

AVL-Bäume

Komplexität

AVL-Bäume

Komplexität

- Die Höhe eines AVL-Baums wächst logarithmisch mit der Anzahl der Schlüssel.

- Suchen:
  - $O(\log_2 n)$ wegen Intervallhalbierung
- Einfügen:
  - $O(\log_2 n)$ wegen Überprüfen der Knoten bis zur Wurzel
- Entfernen:
  - $O(\log_2 n)$
  - Im Gegensatz zum Einfügen müssen im schlechtesten Fall Rotationen bzw. Doppelrotationen **an jedem Knoten** des Suchpfades durchgeführt werden.

Lernziele

- Sie können entscheiden, ob es sich bei einem gegebenen Baum um einen AVL-Baum handelt oder nicht
- Sie können Balance-Faktoren berechnen
- Sie können Situationen identifizieren, bei denen rotiert werden muss um die AVL-Eigenschaft wiederherzustellen
- Sie können angeben, welche Schritte bei einer Rotation notwendig sind
- Sie können aus einer Folge von Schlüsseln einen AVL-Baum konstruieren
- Sie können Problemstellungen für AVL-Bäume durch Bewegung in oder das Traversieren von AVL-Bäumen lösen und die Lösung in Java implementieren

Literatur

- Quellen:
  - Ottmann, T. und Widmayer, P.: Algorithmen und Datenstrukturen (Kapitel 5.2: Balancierte Binärbäume)
  - Saake, G.: Algorithmen und Datenstrukturen (Kapitel 14.4: Ausgeglichene Bäume)