AVL Bäume

Manfred Hauswirth | Open Distributed Systems | Einführung in die Programmierung, WS 22/23

Organisatorisches

- Aktuell finden die Lehrevaluationen für die Lehrveranstaltungen des Wintersemesters 2022/2023 statt.

- Links zu der Umfrage in ISIS

- Die Teilnahme ist bis zum 27.01.2023 möglich.

Klausur

- Ersttermin am 27.02.2023 um 10:00 Uhr
  - Anmeldung über MTS vom 16.02.-23.02.2023
  - Techniktest am 20.02.2023 um 10:00 Uhr

- **Nach**klausur am 31.03.2023 um 16:00 Uhr
  - Anmeldung über MTS vom 20.03.-29.03.2023
  - Techniktest am 24.03.2023 um 16:00 Uhr

- Klausuren finden als Online-Klausuren via ISIS und Zoom statt
- Zulassung zur Klausur nur bei Erreichen des Kriteriums
- Weitere Ankündigungen zum Ablauf in ISIS beachten
- **Teilnahme am Ersttermin wird empfohlen**

Rückblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine

VL 1 „Algorithmen, Pseudocode, Sortieren I“: Insertion Sort

VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort

VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortierverfahren

VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue

VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen

VL 6 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C

VL 7 „Teile und Herrsche I“: Einführung der algorithmischen Methode, Merge Sort

VL 8 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise

VL 9 „Prioritätenschlangen/Halden/Heaps “: Heap Sort, Binärer Heap, Heap Operationen

VL 10 „Fortgeschrittene Sortierverfahren“: Quick Sort, Radix Sort

VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung

VL 12 „Teile und Herrsche II“: Generalisierung des algorithmischen Prinzips, Mastertheorem

VL 13 „Q & A“: Offene Vorlesung/Wiederholung

Wiederholung: Bäume

Ein grundlegendes Problem

- Speicherung von Datensätzen
- Listen oft nicht ausreichend, da O(n) zu langsam ist

Anforderungen

- Schneller Zugriff, d.h. schneller als O(n)
- Einfügen neuer Datensätze
- Löschen bestehender Datensätze

Beispiele:

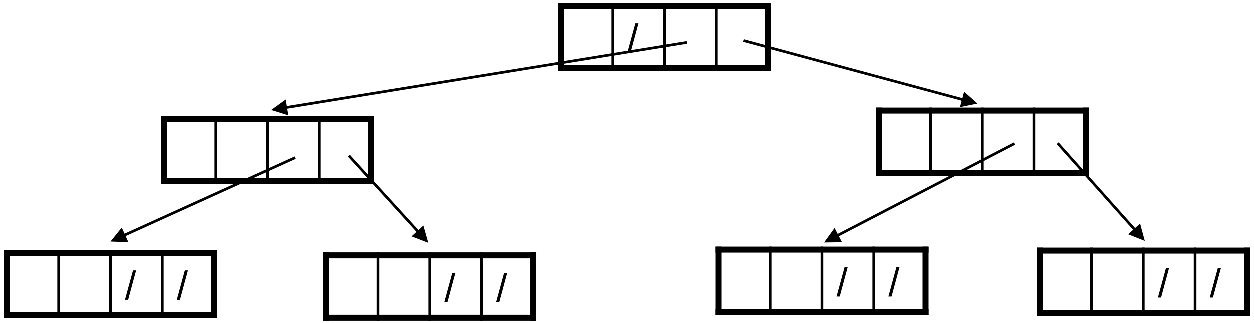
- Stammbaum
- Dateisysteme
- Entscheidungsbäume
- Suchbäume
  z.B. für Lexikadaten
- Rekursionsbäume

Wiederholung: Binärbaum

Darstellung im Rechner

key	p(v)	lc(v)	rc(v)

- Schlüssel key und ggf. weitere Daten
- Zeiger lc(v) (rc(v)) auf linkes (rechtes) Kind von v
- Elterzeiger p(v) auf Elter von v (blau)
- Wurzelzeiger root(T) auf die Wurzel des Baums T

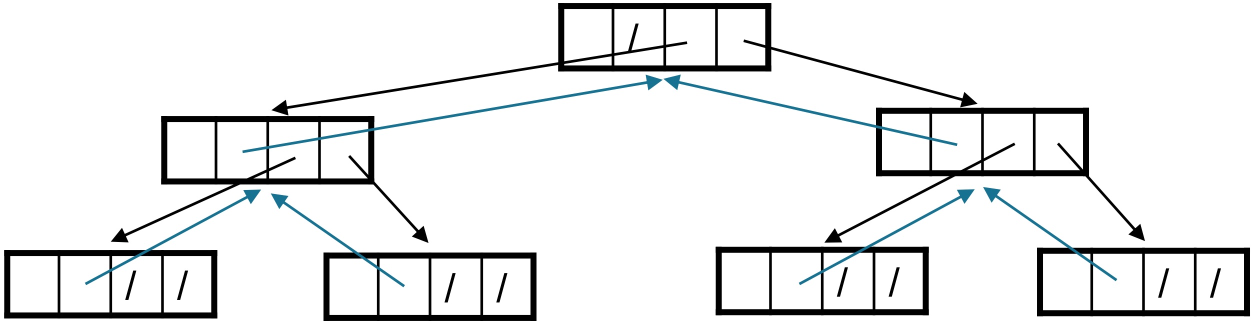


Wiederholung: Binärbaum

Darstellung im Rechner

key	p(v)	lc(v)	rc(v)

- Schlüssel key und ggf. weitere Daten
- Zeiger lc(v) (rc(v)) auf linkes (rechtes) Kind von v
- Elterzeiger p(v) auf Elter von v (blau)
- Wurzelzeiger root(T) auf die Wurzel des Baums T

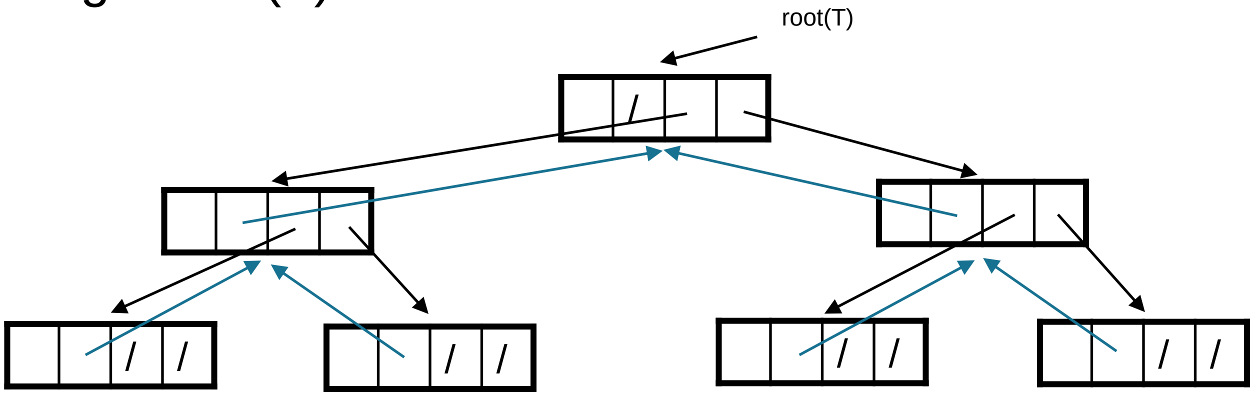


Wiederholung: Binärbaum

Darstellung im Rechner

- Schlüssel key und ggf. weitere Daten
- Zeiger lc(v) (rc(v)) auf linkes (rechtes) Kind von v
- Elterzeiger p(v) auf Elter von v (blau)
- Wurzelzeiger root(T) auf die Wurzel des Baums T

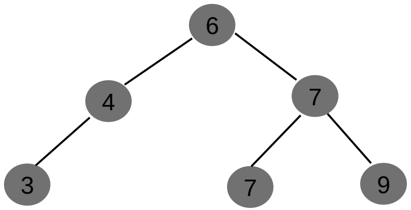
key	p(v)	lc(v)	rc(v)



Wiederholung: Binäre Suchbäume

Binäre Suchbäume

- Verwende Binärbaum
- Speichere Schlüssel „geordnet“



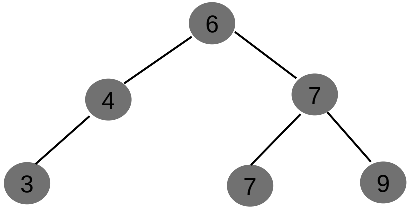
Binäre Suchbaumeigenschaft:

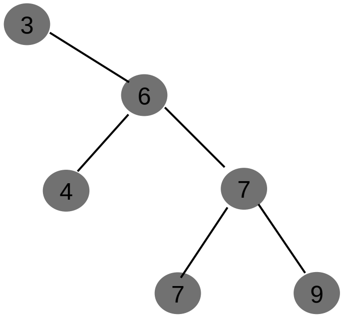
- Sei x Knoten im binären Suchbaum
- Ist y Knoten im linken Unterbaum von x, dann gilt key(y) ≤ key(x)
- Ist y Knoten im rechten Unterbaum von x, dann gilt key(y) > key(x)

Wiederholung: Binäre Suchbäume

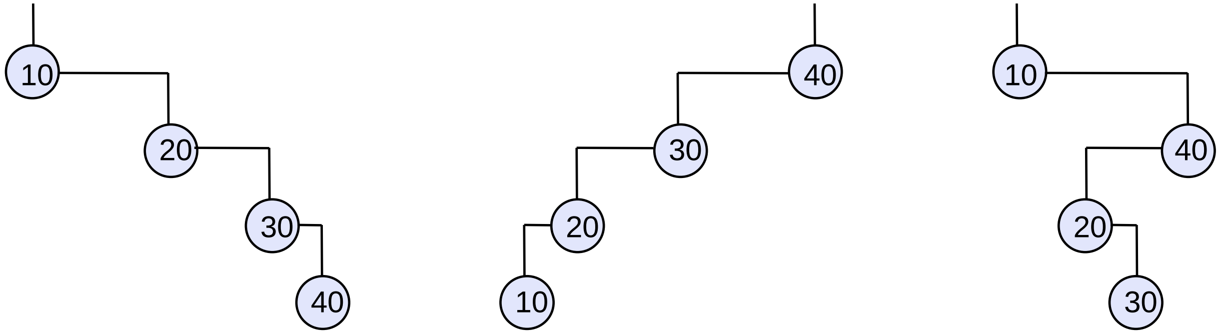
Unterschiedliche Suchbäume

- Schlüsselmenge 3,4,6,7,7,9
- Wir erlauben mehrfaches Vorkommen desselben Schlüssels





Problem – Degenerierte Suchbäume



- Komplexität der Operationen hängt von der Höhe der Bäume ab
- Problem: Degenerierte Bäume haben eine Höhe h = O(n)

Problem – Degenerierte Suchbäume

- Der Baum verliert seine besonderen Eigenschaften, wenn er degeneriert, d.h. kein angemessenes Breiten-Höhen-Verhältnis mehr besitzt, also z.B. zur linearen Liste wird.
  - Formgebung und –erhaltung sind essenziell für die Effizienz

- „Freie“ Bäume:
  - Hier kann sich der Baum frei entwickeln. Beliebige, auch degenerierte Formen können entstehen

Lösung – Balancierte Suchbäume

- Ziel: Sicherstellen, dass die Höhe der Teilbäume ungefähr gleich ist

- Resultat: Höhe des Baumes: O(log n)

Formoptimierung bei Bäumen

- Dynamisch optimierte Bäume
  - Die Operationen Entfernen und Einfügen wirken form-erhaltend:
  - Droht der Baum zu degenerieren, so wird eine Umstrukturierung durchgeführt, z.B: AVL.
- Dynamisch optimierte Bäume mit Gewichten
  - Wird auf die im Baum gespeicherten Elemente mit unterschiedlicher Häufigkeit zugegriffen, so kann dies bei der Gestaltung und Platzierung berücksichtigt werden, z.B. Splay Trees.
- Statisch konstruierte Bäume
  - Hier wird Formerhaltung nicht im laufenden Betrieb vorgenommen, sondern bei Bedarf der Baum neu strukturiert.

Selbst-balancierende Bäume

- Es wird bei jeder Operation auf dem Baum sichergestellt, dass dieser relativ balanciert bleibt

- Einfügen und Löschen werden teurer, aber dafür kann beim Suchen eine Komplexität von O(log(n)) garantiert werden

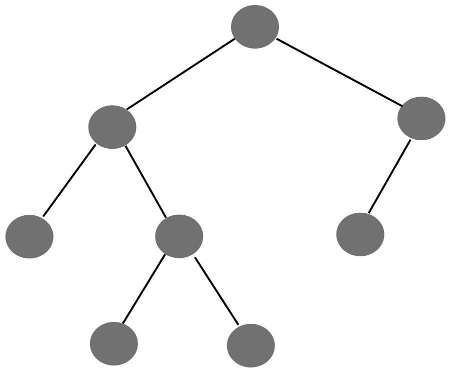
Beispiele

  - AVL Bäume
  - 2-3 Bäume
  - 2-3-4 Bäume
  - Rot-Schwarz Bäume (red-black trees)

AVL-Bäume

AVL-Bäume [Adelson-Velsky und Landis]

- Ein binärer Suchbaum heißt AVL-Baum, wenn für jeden Knoten gilt: Die Höhe seines linken und rechten Teilbaums unterscheidet sich höchstens um 1.



AVL-Bäume: Beweis für Höhe

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: durch Fallunterscheidung

a) $n \leq 2^{h+1}-1$ (obere Schranke)

b) $(3/2)^h \leq n$ (untere Schranke)

AVL-Bäume: Beweis für Höhe Obere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: a) $n \leq 2^{h+1}-1$ (obere Schranke)

AVL-Bäume: Beweis für Höhe Obere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: a) $n \leq 2^{h+1} - 1$ (obere Schranke)

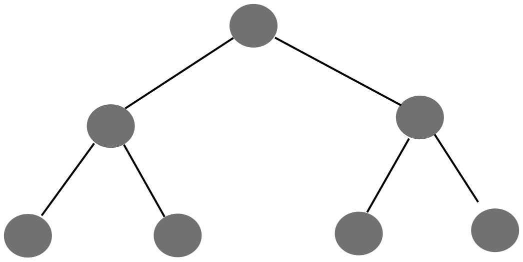
- AVL-Baum ist Binärbaum

AVL-Bäume: Beweis für Höhe Obere Schranke

Satz: Für jeden AVL-Baum der Höhe h≥0 mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: a) $n \leq 2^{h+1}-1$ (obere Schranke)



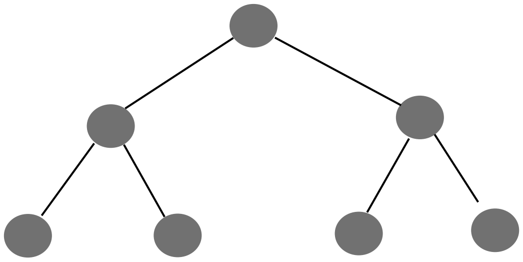
- AVL-Baum ist Binärbaum
- Ein vollständiger Binärbaum hat eine maximale Anzahl Knoten unter allen Binärbäumen der Höhe h

AVL-Bäume: Beweis für Höhe Obere Schranke

Satz: Für jeden AVL-Baum der Höhe h≥0 mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: a) $n \leq 2^{h+1}-1$ (obere Schranke)



- AVL-Baum ist Binärbaum
- Ein vollständiger Binärbaum hat eine maximale Anzahl Knoten unter allen Binärbäumen der Höhe h
- N(h) = Anzahl Knoten eines vollständigen Binärbaums der Höhe h

$$N(h) = 1 + 2 + 4 ... + 2^h = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- ➢ Beweis per Induktion über die Struktur von AVL-Bäumen

AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

➢ Beweis per Induktion über die Struktur von AVL-Bäumen

- (I.A.) Wir betrachten alle AVL-Bäume der Höhe 0 und 1.

AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- ➢ Beweis per Induktion über die Struktur von AVL-Bäumen
- (I.A.) Wir betrachten alle AVL-Bäume der Höhe 0 und 1.

  h=0: Der Baum hat einen Knoten. Es gilt $(3/2)^h = 1 \leq 1$.

AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

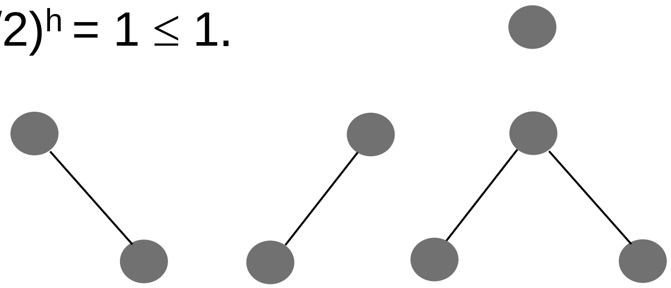
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- Beweis per Induktion über die Struktur von AVL-Bäumen
- (I.A.) Wir betrachten alle AVL-Bäume der Höhe 0 und 1.

  h=0: Der Baum hat einen Knoten. Es gilt $(3/2)^h = 1 \leq 1$.

  h=1: Der Baum hat 2 oder 3 Knoten.
  Es gilt $(3/2)^h = 3/2 \leq 2 \leq 3$.



AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} -1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- ➢ Beweis per Induktion über die Struktur von AVL-Bäumen
- (I.V.) Für jeden AVL-Baum der Höhe j, $0 \leq j \leq h$, gilt der Satz.

AVL-Bäume: Beweis für Höhe Untere Schranke

Technische Universität Berlin

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- ➢ Beweis per Induktion über die Struktur von AVL-Bäumen
- (I.V.) Für jeden AVL-Baum der Höhe j, $0 \leq j \leq h$, gilt der Satz.
- (I.S.) Sei $h \geq 1$. Betrachte AVL-Baum T der Höhe $h+1$ mit Wurzel v.

Open Distributed Sytems
ODS

AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- ➢ Beweis per Induktion über die Struktur von AVL-Bäumen
- (I.V.) Für jeden AVL-Baum der Höhe j, $0 \leq j \leq h$, gilt der Satz.
- (I.S.) Sei $h \geq 1$. Betrachte AVL-Baum T der Höhe $h+1$ mit Wurzel v.
  - Seien A,B linker bzw. rechter Teilbaum von v.

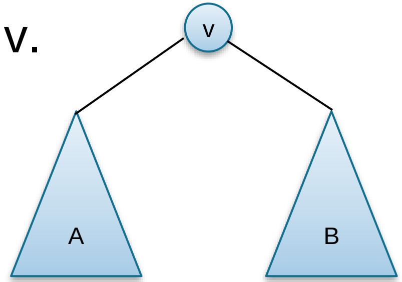
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- ➢ Beweis per Induktion über die Struktur von AVL-Bäumen
- (I.V.) Für jeden AVL-Baum der Höhe j, $0 \leq j \leq h$, gilt der Satz.
- (I.S.) Sei $h \geq 1$. Betrachte AVL-Baum T der Höhe h+1 mit Wurzel v.
  - Seien A,B linker bzw. rechter Teilbaum von v.



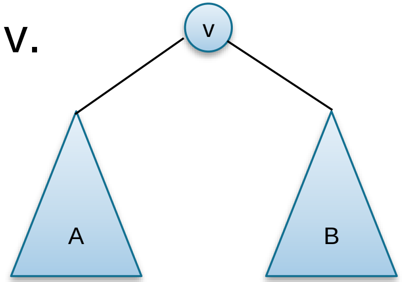
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- ➢ Beweis per Induktion über die Struktur von AVL-Bäumen
- (I.V.) Für jeden AVL-Baum der Höhe j, $0 \leq j \leq h$, gilt der Satz.
- (I.S.) Sei $h \geq 1$. Betrachte AVL-Baum T der Höhe h+1 mit Wurzel v.
  - Seien A,B linker bzw. rechter Teilbaum von v.
  - A oder B (oder beide) hat Tiefe h.



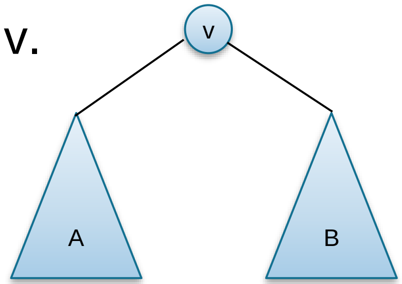
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke)

- ➢ Beweis per Induktion über die Struktur von AVL-Bäumen
- (I.V.) Für jeden AVL-Baum der Höhe j, $0 \leq j \leq h$, gilt der Satz.
- (I.S.) Sei $h \geq 1$. Betrachte AVL-Baum T der Höhe h+1 mit Wurzel v.
  - Seien A,B linker bzw. rechter Teilbaum von v.
  - A oder B (oder beide) hat Tiefe h.
  - Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.



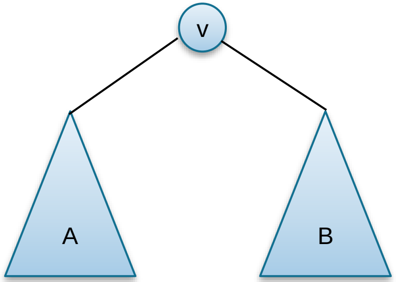
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

– Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.



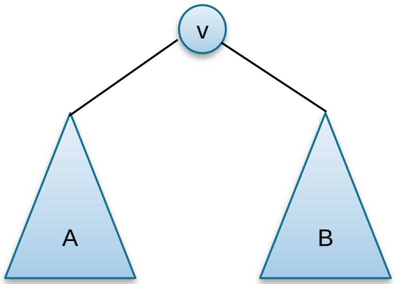
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

- Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.
- Da T ein AVL-Baum ist, sind auch A und B AVL-Bäume.



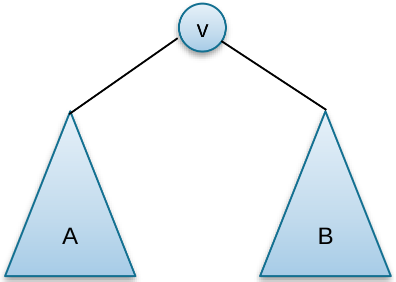
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

- Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.
- Da T ein AVL-Baum ist, sind auch A und B AVL-Bäume.
- Wir können (I.V.) anwenden, da A und B AVL-Bäume der Tiefe ≥ 0 sind



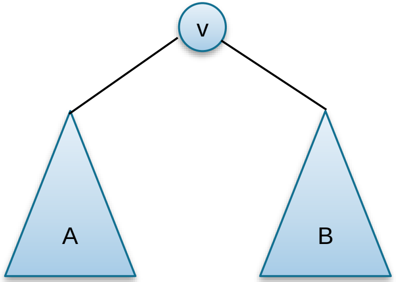
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

- Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.
- Da T ein AVL-Baum ist, sind auch A und B AVL-Bäume.
- Wir können (I.V.) anwenden, da A und B AVL-Bäume der Tiefe ≥ 0 sind
- Es gibt drei Fälle:



AVL-Bäume: Beweis für Höhe Untere Schranke

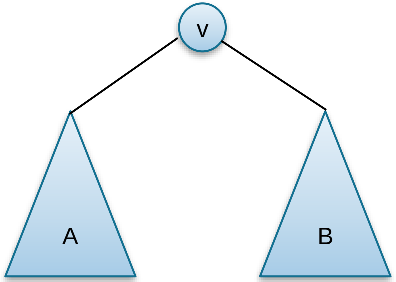
Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

- Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.
- Da T ein AVL-Baum ist, sind auch A und B AVL-Bäume.
- Wir können (I.V.) anwenden, da A und B AVL-Bäume der Tiefe ≥ 0 sind
- Es gibt drei Fälle:

1) A,B haben Höhe h



AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

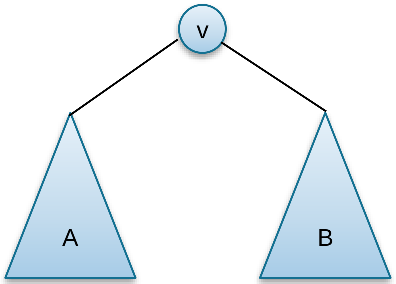
$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

- Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.
- Da T ein AVL-Baum ist, sind auch A und B AVL-Bäume.
- Wir können (I.V.) anwenden, da A und B AVL-Bäume der Tiefe ≥ 0 sind
- Es gibt drei Fälle:

  1) A,B haben Höhe h

  2) A hat Höhe h und B hat Höhe h-1



AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

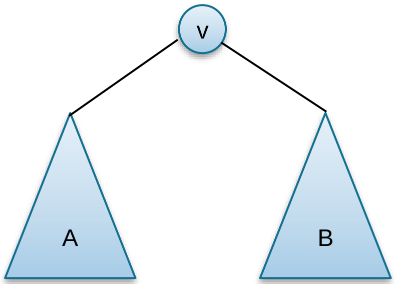
Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

- Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.
- Da T ein AVL-Baum ist, sind auch A und B AVL-Bäume.
- Wir können (I.V.) anwenden, da A und B AVL-Bäume der Tiefe ≥ 0 sind
- Es gibt drei Fälle:

  1) A,B haben Höhe h

  2) A hat Höhe h und B hat Höhe h-1

  3) A hat Höhe h-1 und B hat Höhe h



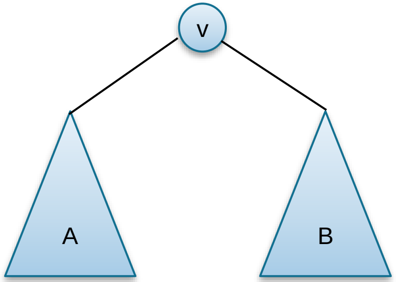
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe h≥0 mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

– Sei T(h) die minimale Anzahl Knoten in einem AVL-Baum der Tiefe h.



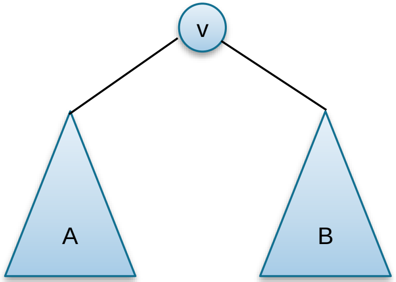
AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

- Sei T(h) die minimale Anzahl Knoten in einem AVL-Baum der Tiefe h.
- Nach (I.V.) gilt in allen drei Fällen



AVL-Bäume: Beweis für Höhe Untere Schranke

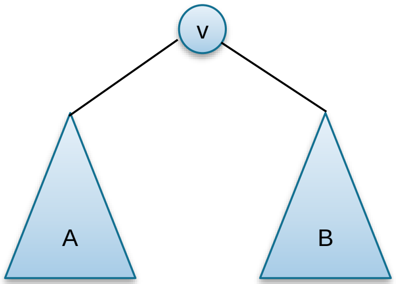
Satz: Für jeden AVL-Baum der Höhe h≥0 mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

- Sei T(h) die minimale Anzahl Knoten in einem AVL-Baum der Tiefe h.
- Nach (I.V.) gilt in allen drei Fällen

$$T(h+1) \geq T(h) + T(h-1) + 1 \geq \left(\frac{3}{2}\right)^h + \left(\frac{3}{2}\right)^{h-1} + 1$$



AVL-Bäume: Beweis für Höhe Untere Schranke

Satz: Für jeden AVL-Baum der Höhe h≥0 mit n Knoten gilt:

$$(3/2)^h \leq n \leq 2^{h+1}-1$$

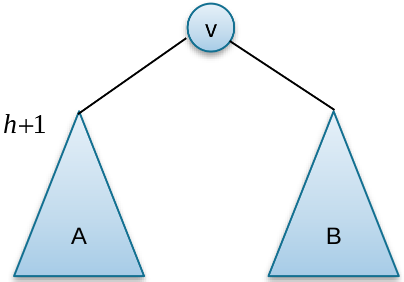
Beweis: b) $(3/2)^h \leq n$ (untere Schranke) – Fortsetzung Induktionsschritt

– Sei T(h) die minimale Anzahl Knoten in einem AVL-Baum der Tiefe h.

– Nach (I.V.) gilt in allen drei Fällen

$$T(h+1) \geq T(h) + T(h-1) + 1 \geq \left(\frac{3}{2}\right)^h + \left(\frac{3}{2}\right)^{h-1} + 1$$

$$\geq (1+3/2) \cdot \left(\frac{3}{2}\right)^{h-1} \geq \left(\frac{3}{2}\right)^2 \cdot \left(\frac{3}{2}\right)^{h-1} = \left(\frac{3}{2}\right)^{h+1}$$



AVL-Bäume: Höhe in Θ(log n)

Satz: Für jeden AVL-Baum der Höhe h≥0 mit n Knoten gilt:

$$(3/2)^h \le n \le 2^{h+1}-1$$

Korollar: Ein AVL-Baum mit n Knoten hat Höhe Θ(log n).

Beweis: (1) Zeige h=O(log n): Es gilt n ≥ $(3/2)^h$ nach Satz

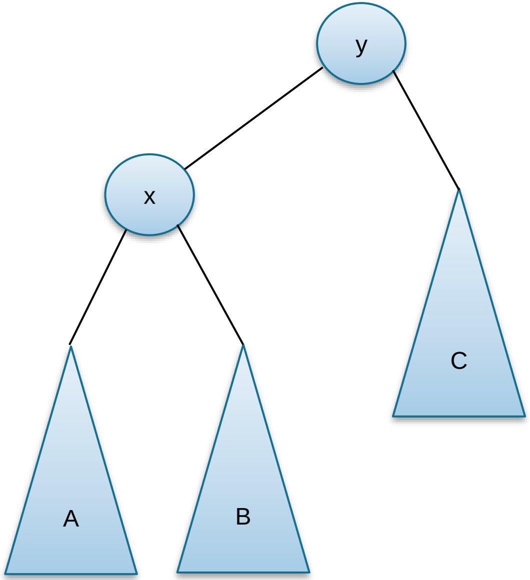
$$n \ge \left(\frac{3}{2}\right)^h \Rightarrow \log n \ge \log\left(\left(\frac{3}{2}\right)^h\right) \Rightarrow \log n \ge h \cdot \log(3/2) \Rightarrow h = O(\log n)$$

(2) Zeige h=Ω(log n): Es gilt n ≤ 2^{h+1} -1 ≤ 2^{h+1} nach Satz

$$n \le 2^{h+1} \Rightarrow \log n \le h+1 \Rightarrow \log n \le 2h \Rightarrow h = \Omega(\log n)$$

Der Weg zu balancierten Bäumen: Rotation

Gegeben sei folgender Baum



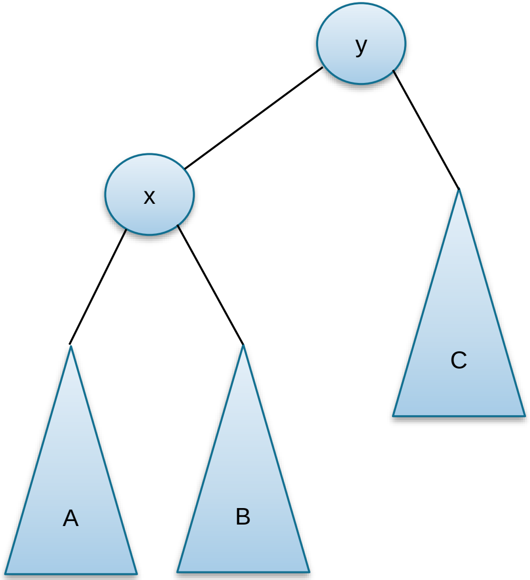
Aufgabe:

- Einfügen von a in Teilbaum A

Problem:

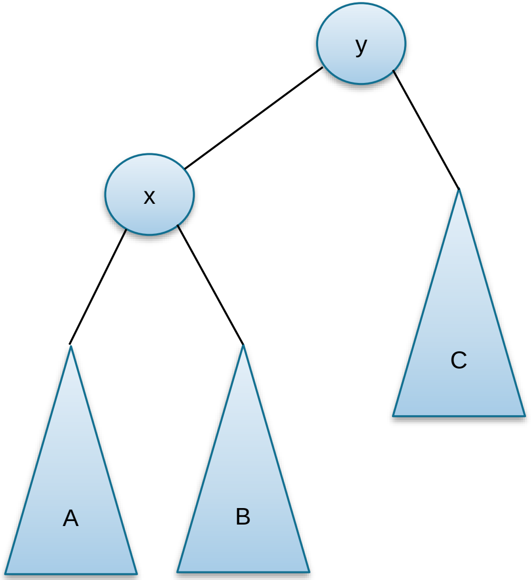
- Teilbaum von x wird zu groß

Der Weg zu balancierten Bäumen: Rotation

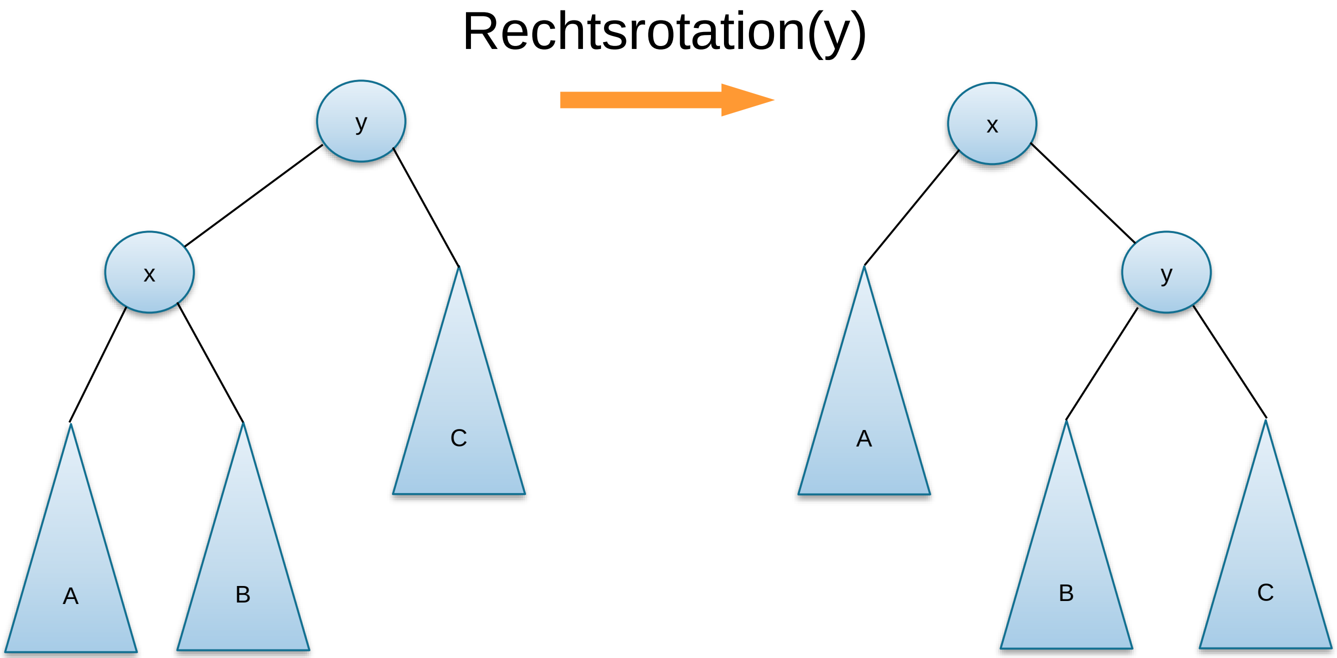


Der Weg zu balancierten Bäumen: Rotation

Rechtsrotation(y)



Der Weg zu balancierten Bäumen: Rotation



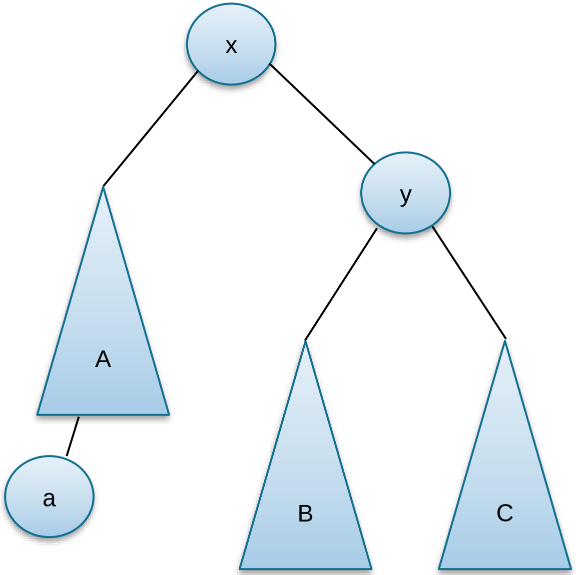
Der Weg zu balancierten Bäumen: Rotation

Aufgabe:

- Einfügen von a in Teilbaum A

Lösung:

- Rotation



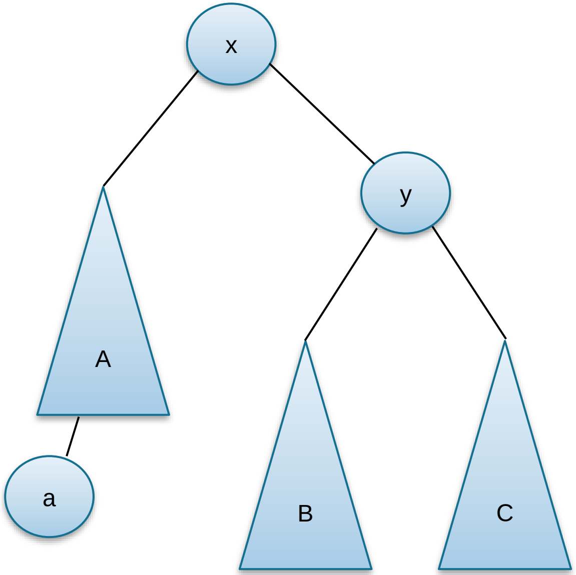
Der Weg zu balancierten Bäumen: Rotation

Aufgabe:

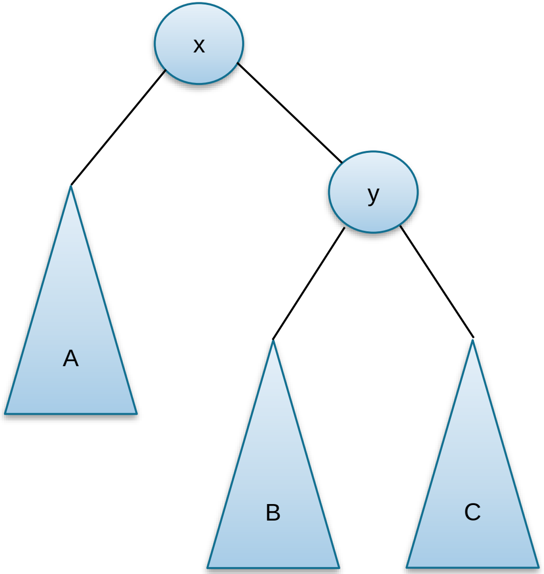
- Einfügen von a in Teilbaum A

Lösung:

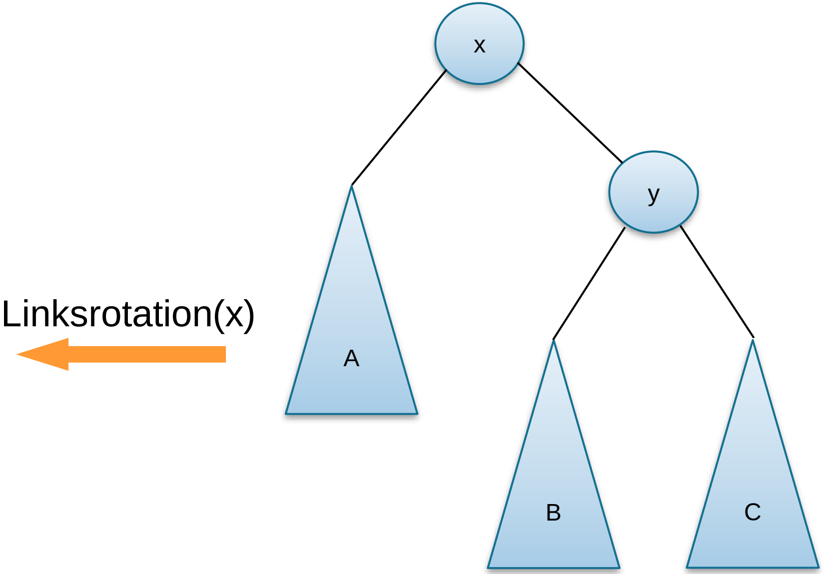
- Rotation
- Danach Einfügen von a in Teilbaum A



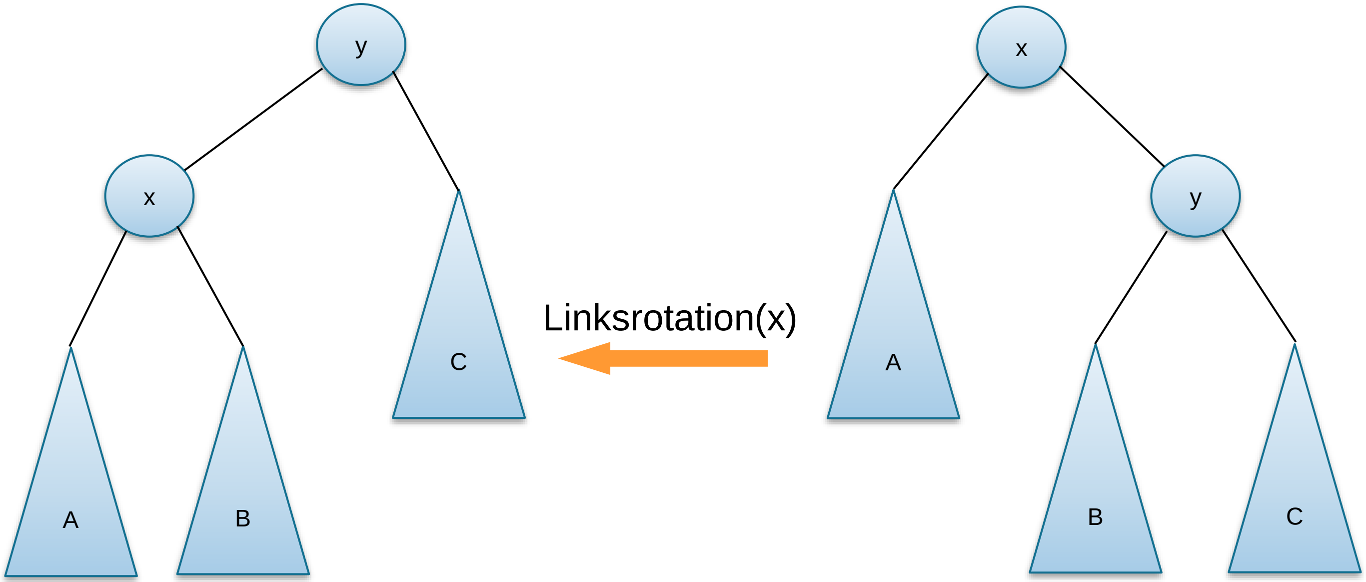
Der Weg zu balancierten Bäumen: Rotationen sind symmetrisch



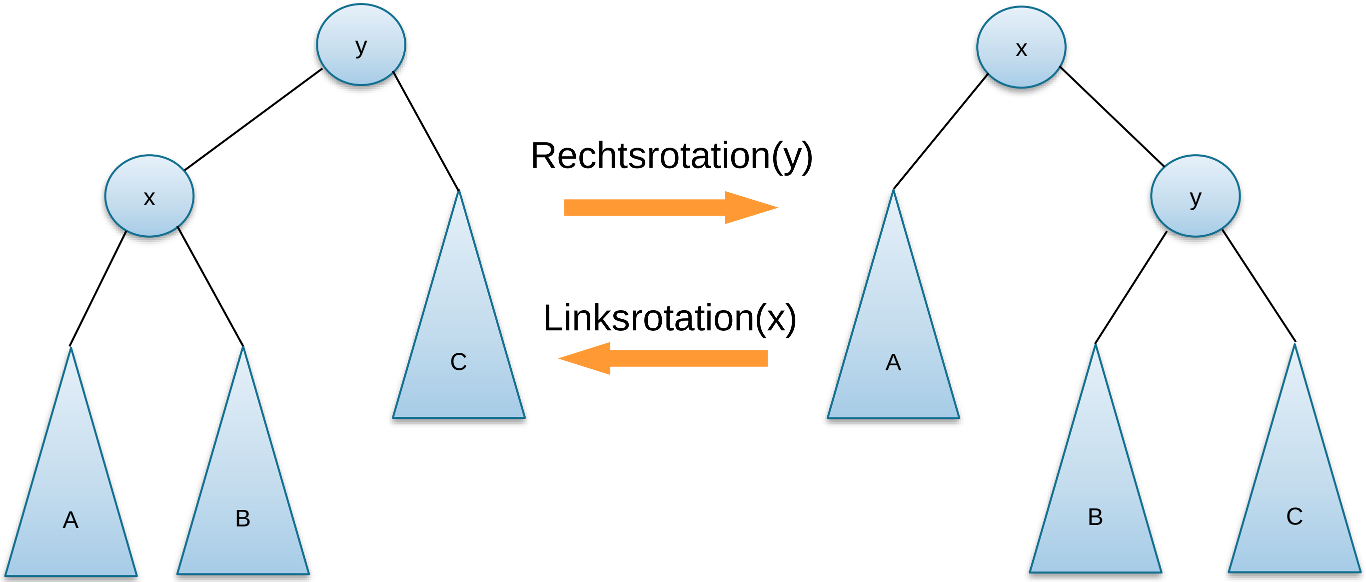
Der Weg zu balancierten Bäumen: Rotationen sind symmetrisch



Der Weg zu balancierten Bäumen: Rotationen sind symmetrisch

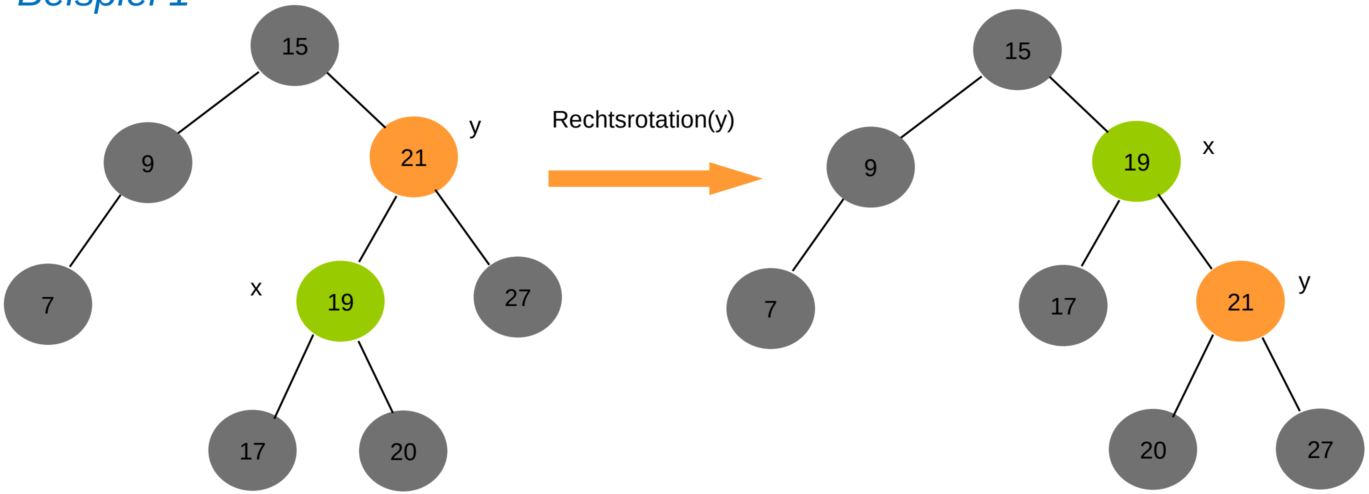


Der Weg zu balancierten Bäumen: Rotationen sind symmetrisch



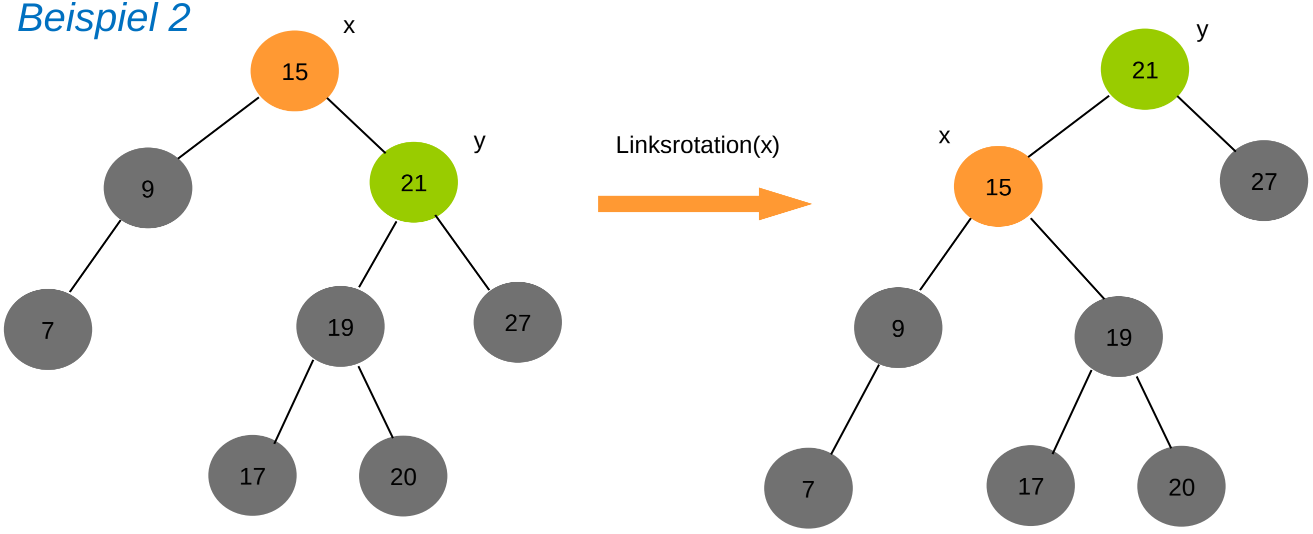
Rechtsrotation

Beispiel 1



Linksrotation

Beispiel 2



Linksrotation auf dem Baum T

Linksrotation(x)

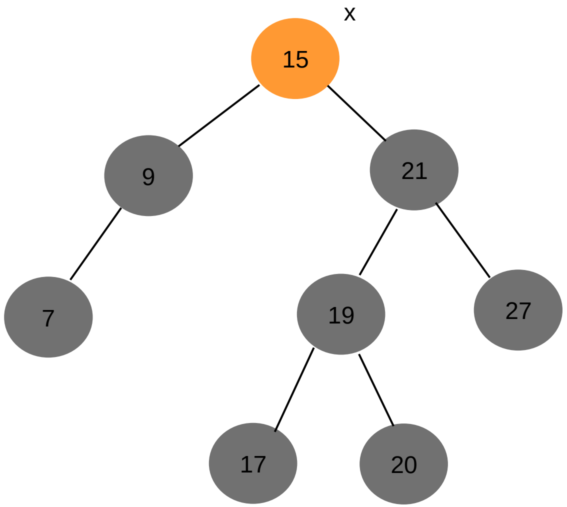
1. y $\leftarrow$ rc[x]
2. rc[x] $\leftarrow$ lc[y]
3. **if** lc[y]$\neq$**nil then** p[lc[y]] $\leftarrow$ x
4. p[y] $\leftarrow$ p[x]
5. **if** p[x]=**nil then** root[T] $\leftarrow$ y
6. **else if** x=lc[p[x]] **then** lc[p[x]] $\leftarrow$ y
7. **else** rc[p[x]] $\leftarrow$ y
8. lc[y] $\leftarrow$ x
9. p[x] $\leftarrow$ y

Linksrotation auf dem Baum T

Linksrotation(x)

Annahme: x hat rechtes Kind.

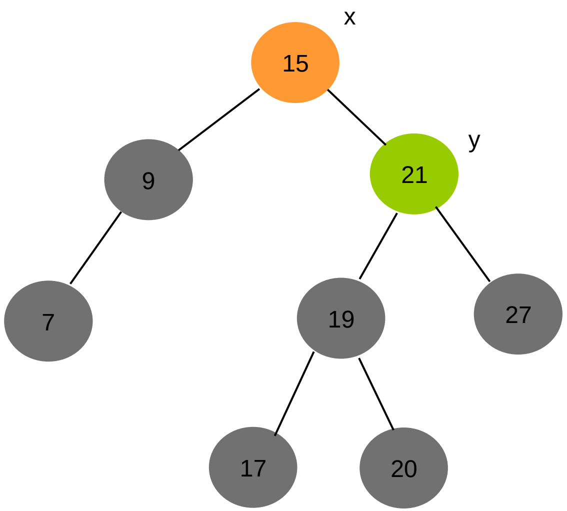
1. $y \leftarrow rc[x]$
2. $rc[x] \leftarrow lc[y]$
3. **if** $lc[y] \neq$ **nil then** $p[lc[y]] \leftarrow x$
4. $p[y] \leftarrow p[x]$
5. **if** $p[x]=$**nil then** $root[T] \leftarrow y$
6. **else if** $x=lc[p[x]]$ **then** $lc[p[x]] \leftarrow y$
7. **else** $rc[p[x]] \leftarrow y$
8. $lc[y] \leftarrow x$
9. $p[x] \leftarrow y$



Linksrotation auf dem Baum T

Linksrotation(x)

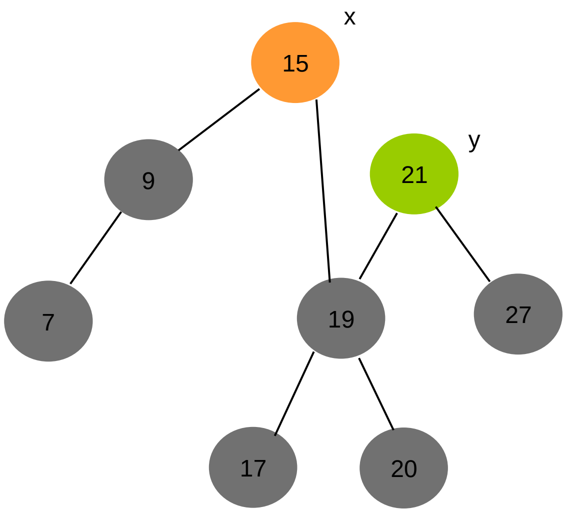
1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y



Linksrotation auf dem Baum T

Linksrotation(x)

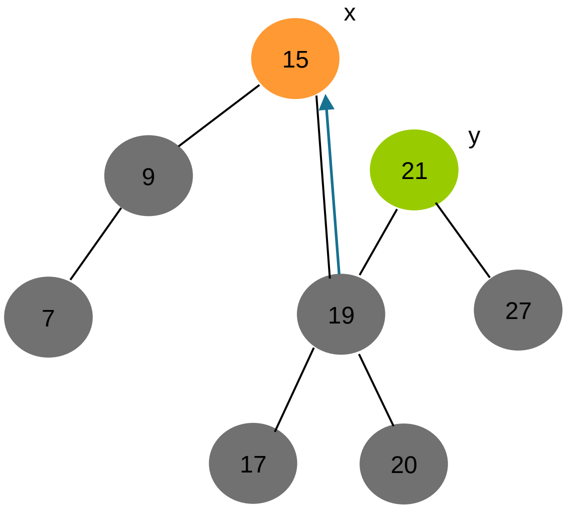
1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y



Linksrotation auf dem Baum T

Linksrotation(x)

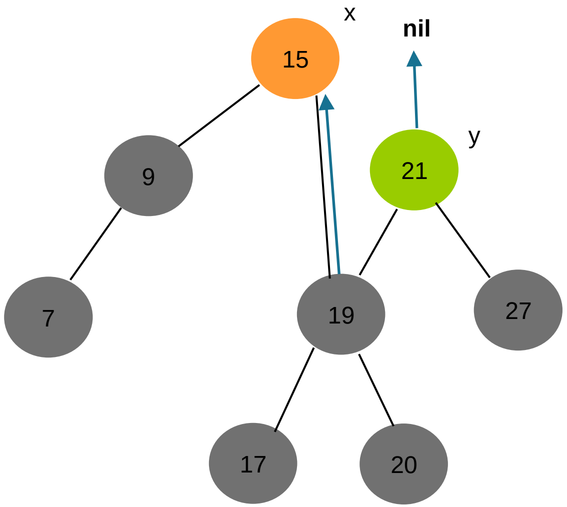
1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y



Linksrotation auf dem Baum T

Linksrotation(x)

1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y

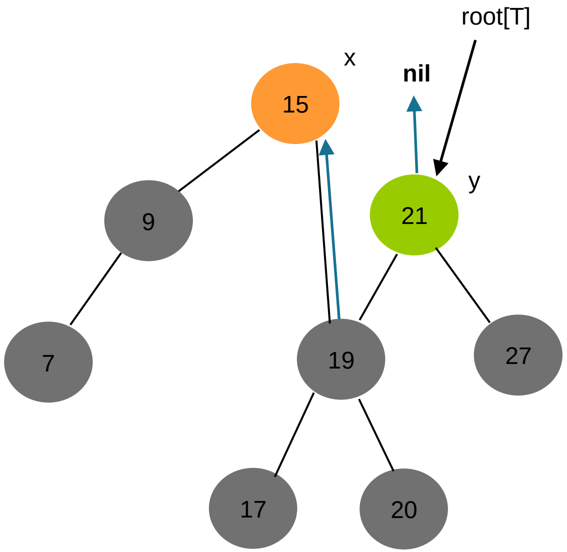


Linksrotation auf dem Baum T

Linksrotation(x)

1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y

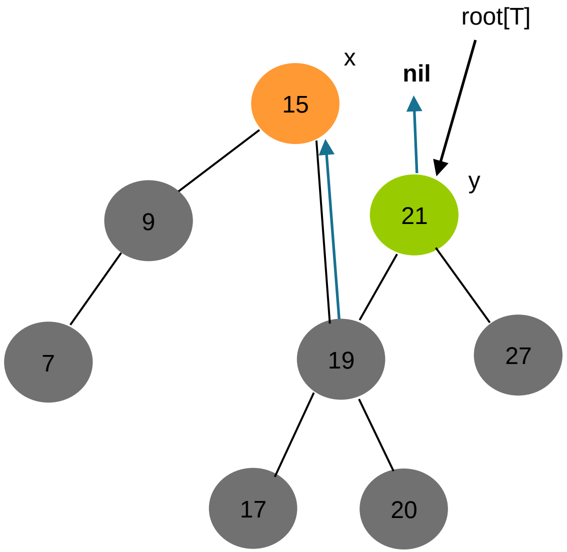
Achtung: die Wurzel des Baumes muss angepasst werden!



Linksrotation auf dem Baum T

Linksrotation(x)

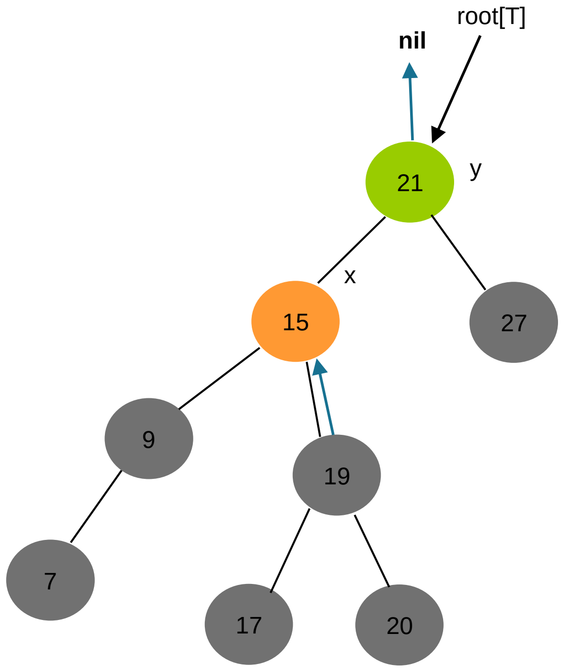
1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y



Linksrotation auf dem Baum T

Linksrotation(x)

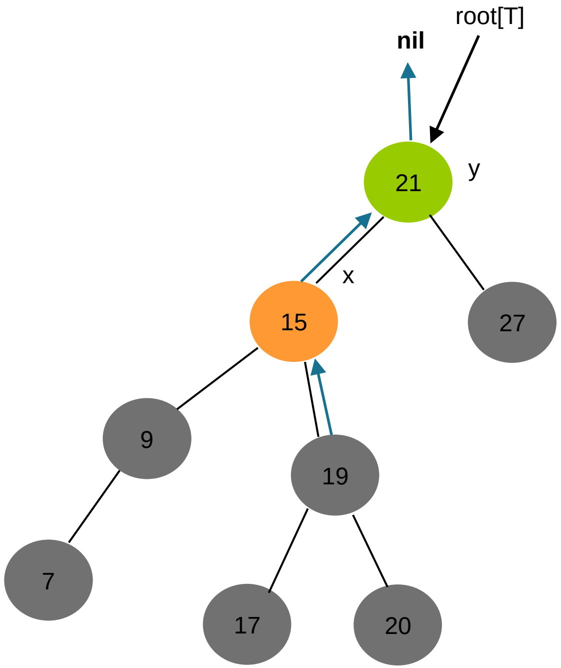
1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y



Linksrotation auf dem Baum T

Linksrotation(x)

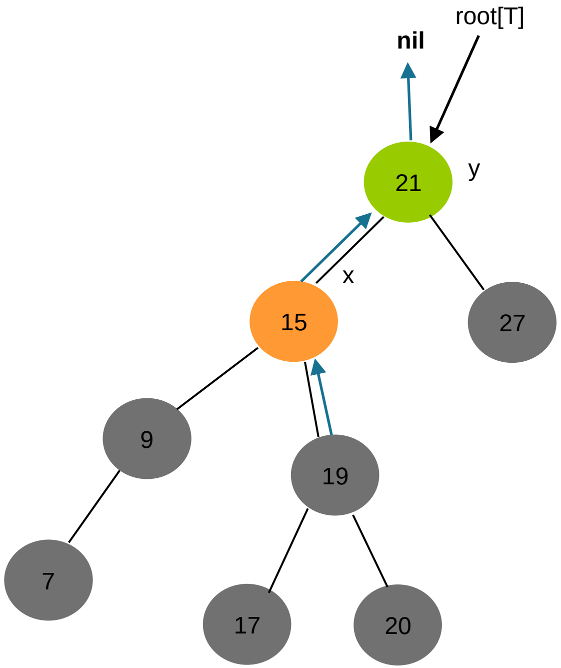
1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y



Linksrotation auf dem Baum T

Linksrotation(x)

1. y ← rc[x]
2. rc[x] ← lc[y]
3. **if** lc[y]≠**nil then** p[lc[y]] ← x
4. p[y] ← p[x]
5. **if** p[x]=**nil then** root[T] ← y
6. **else if** x=lc[p[x]] **then** lc[p[x]] ← y
7. **else** rc[p[x]] ← y
8. lc[y] ← x
9. p[x] ← y



Der Weg zu dynamischen AVL-Bäumen

Wiederholung: Binäre Bäume

- Operationen: Suche, Einfügen, Löschen, Min/Max, Vorgänger/Nachfolger,…
- Laufzeit O(h)

Beobachtung: Nur Einfügen/Löschen verändern Struktur des Baums

Idee: Dynamische AVL-Bäume

- Wir müssen die AVL-Eigenschaft nach jedem Einfügen/Löschen wiederherstellen.
- Dann unterscheiden sich die Höhen aller Teilbäume um maximal 1
- Somit gilt die AVL Eigenschaft und wir haben h = O(log n)
- Entsprechend sind die Laufzeiten für die Operationen O(log n)

Beinahe-AVL-Baum

Definition: Ein Baum heißt beinahe-AVL-Baum, wenn die AVL-Eigenschaft in jedem Knoten außer der Wurzel erfüllt ist und sich die Höhe der Unterbäume der Wurzel um höchstens 2 unterscheidet.

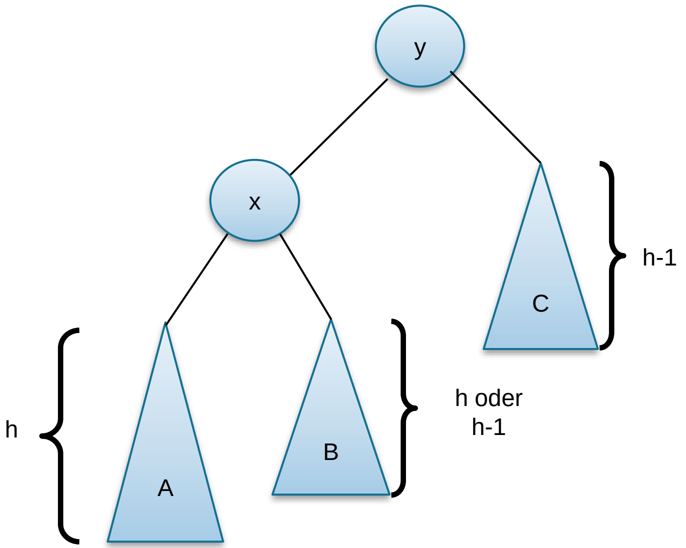
Umformung von Beinahe-AVL-Baum zu AVL-Baum

Unterproblem

- Umformen eines beinahe-AVL-Baums in einen AVL-Baum mithilfe von Rotationen
- O.B.d.A.: Linker Teilbaum der Wurzel höher als der rechte

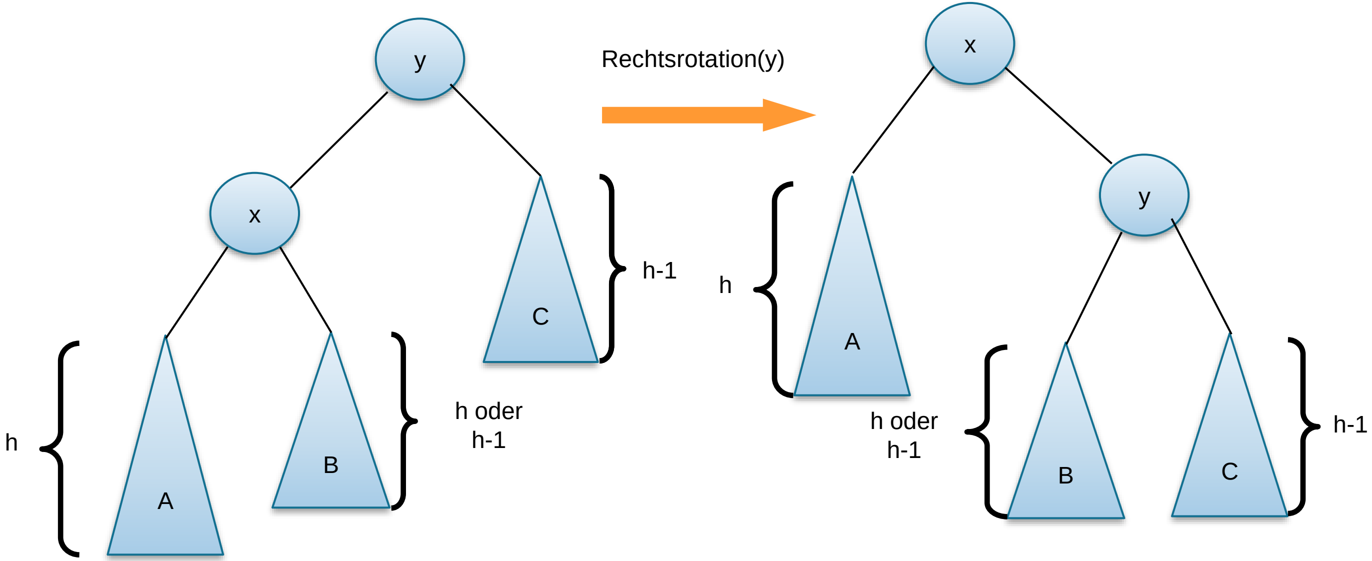
Umformung von Beinahe-AVL-Baum zu AVL-Baum

Fall 1: Einfache Rechtsrotation



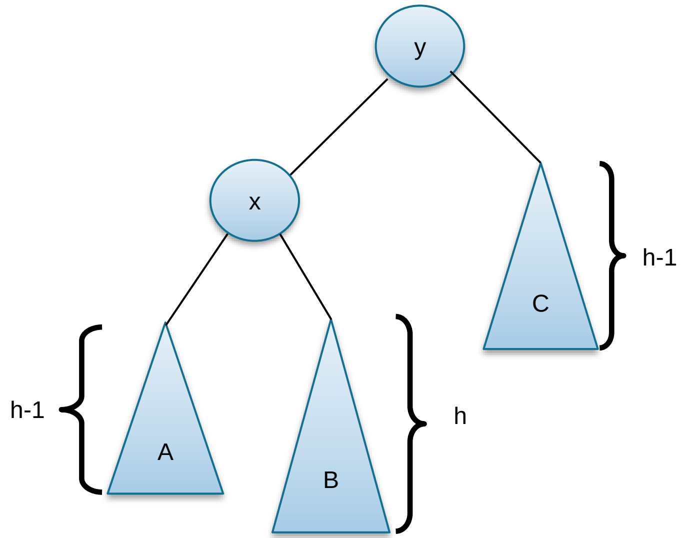
Umformung von Beinahe-AVL-Baum zu AVL-Baum

Fall 1: Einfache Rechtsrotation



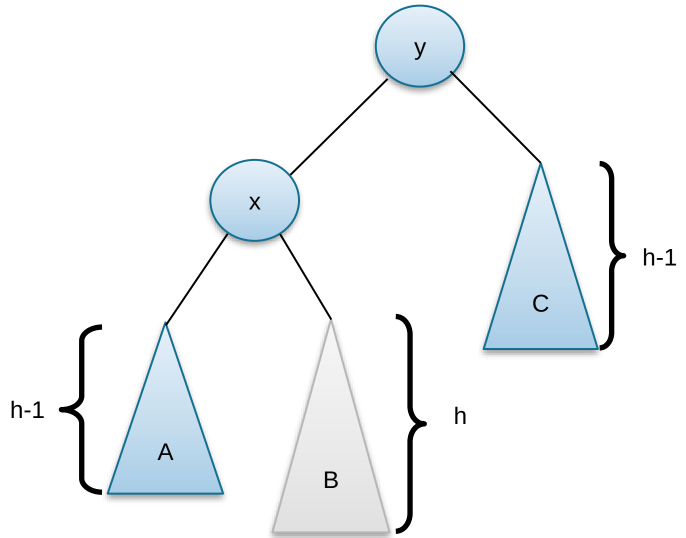
Umformung von Beinahe-AVL-Baum zu AVL-Baum

Fall 2: Doppelrotation



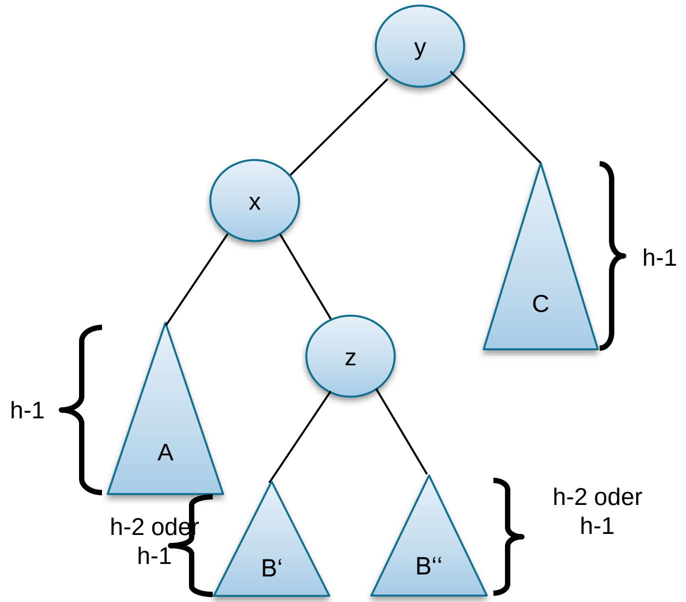
Umformung von Beinahe-AVL-Baum zu AVL-Baum

Fall 2: Doppelrotation



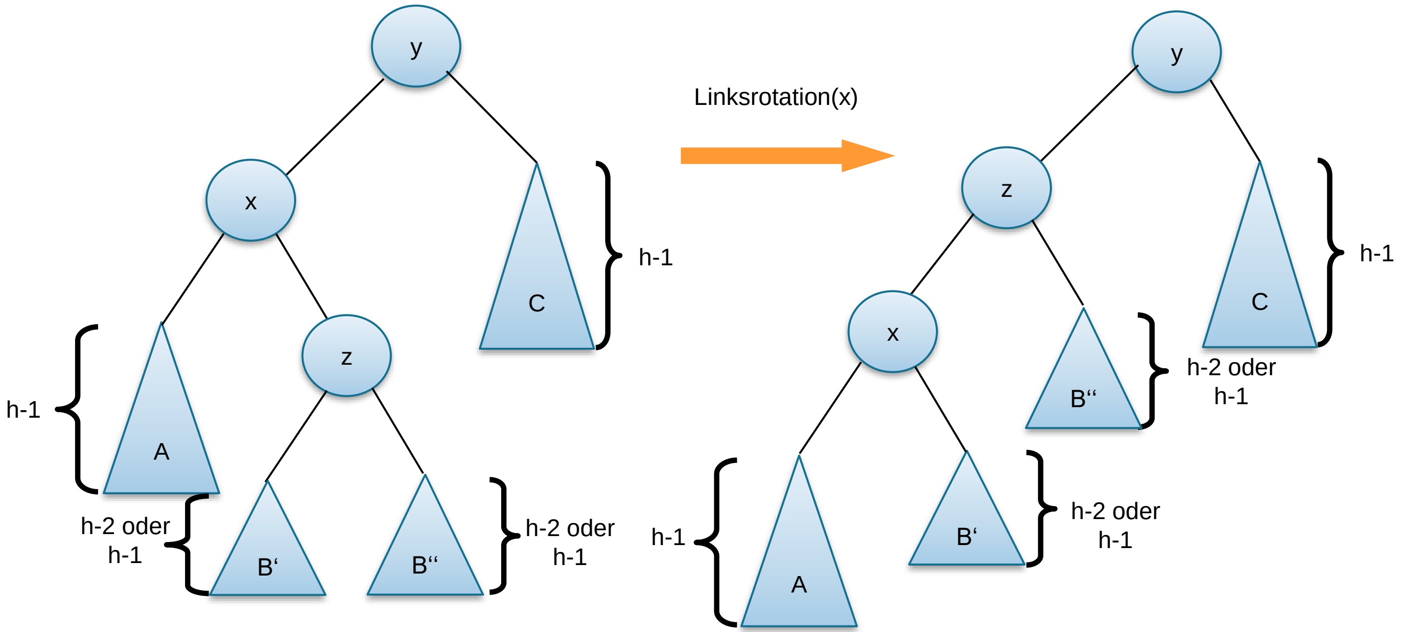
Umformung von Beinahe-AVL-Baum zu AVL-Baum

Fall 2: Doppelrotation



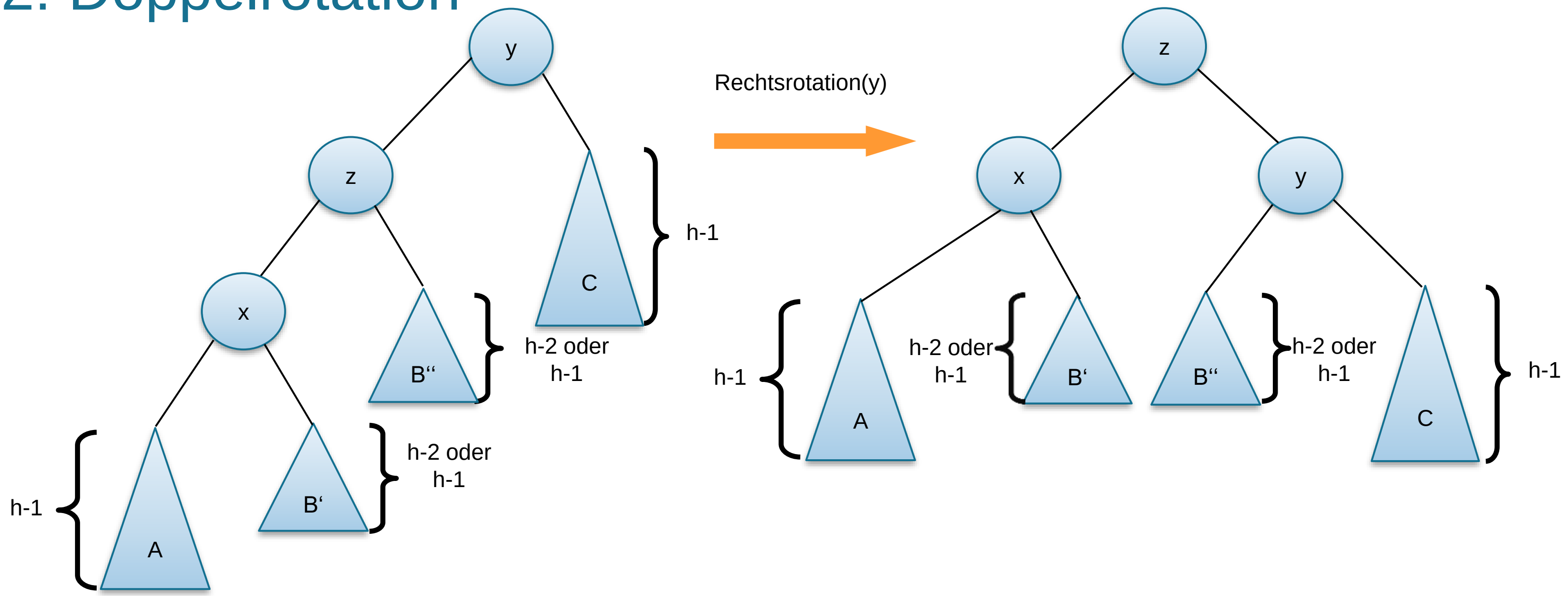
Umformung von Beinahe-AVL-Baum zu AVL-Baum

Fall 2: Doppelrotation



Umformung von Beinahe-AVL-Baum zu AVL-Baum

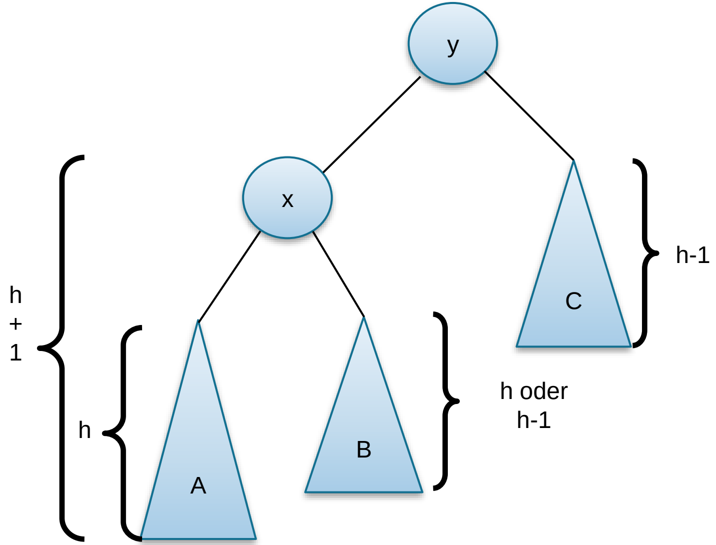
Fall 2: Doppelrotation



Umformung von Beinahe-AVL-Baum zu AVL-Baum: Balance

Balance(y)

1. **if** h[lc[y]] > h[rc[y]]+1 **then**
2.   **if** h[lc[lc[y]]]< h[rc[lc[y]]] **then**
3.   Linksrotation(lc[y])
4.   Rechtsrotation(y)
5. **else if** h[rc[y]]> h[lc[y]]+1 **then**
6.   **if** h[rc[rc[y]]]< h[lc[rc[y]] **then**
7.   Rechtsrotation(rc[y])
8.   Linksrotation(y)

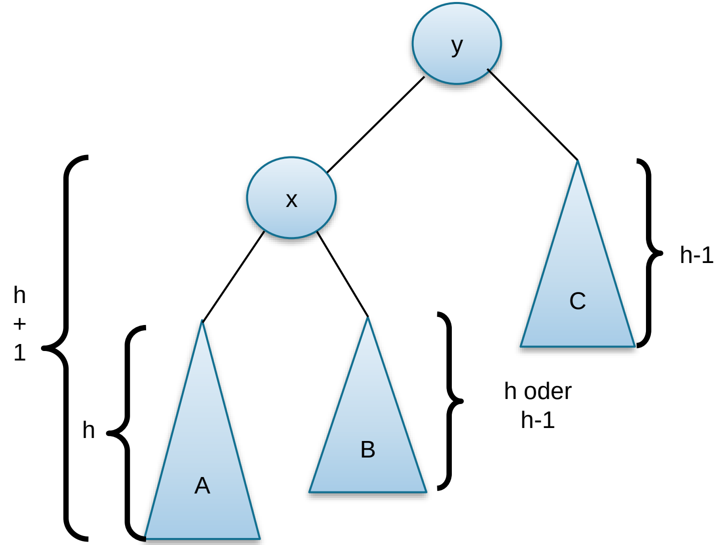


Umformung von Beinahe-AVL-Baum zu Balance

h gibt die Höhe des Teilbaums an. Dies müssen wir zusätzlich in unserer Datenstruktur aufrechterhalten.

Balance(y)

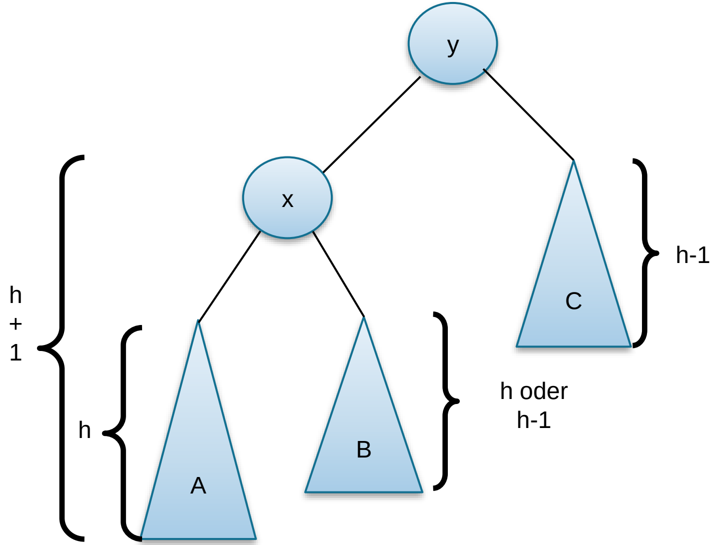
1. **if** h[lc[y]] > h[rc[y]]+1 **then**
2.   **if** h[lc[lc[y]]]< h[rc[lc[y]]] **then**
3.   Linksrotation(lc[y])
4.   Rechtsrotation(y)
5. **else if** h[rc[y]]> h[lc[y]]+1 **then**
6.   **if** h[rc[rc[y]]]< h[lc[rc[y]] **then**
7.   Rechtsrotation(rc[y])
8.   Linksrotation(y)



Umformung von Beinahe-AVL-Baum zu AVL-Baum: Balance

Balance(y)

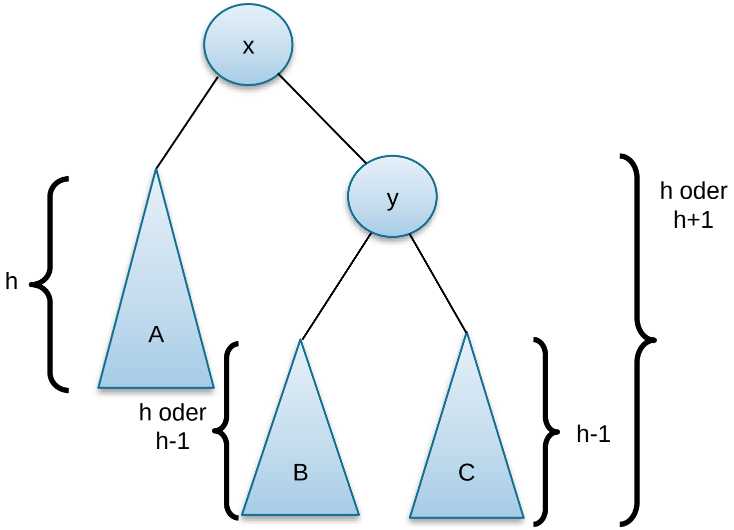
1. **if** h[lc[y]] > h[rc[y]]+1 **then**
2. **if** h[lc[lc[y]]]< h[rc[lc[y]]] **then**
3. Linksrotation(lc[y])
4. Rechtsrotation(y)
5. **else if** h[rc[y]]> h[lc[y]]+1 **then**
6. **if** h[rc[rc[y]]]< h[lc[rc[y]] **then**
7. Rechtsrotation(rc[y])
8. Linksrotation(y)



Umformung von Beinahe-AVL-Baum zu AVL-Baum: Balance

Balance(y)

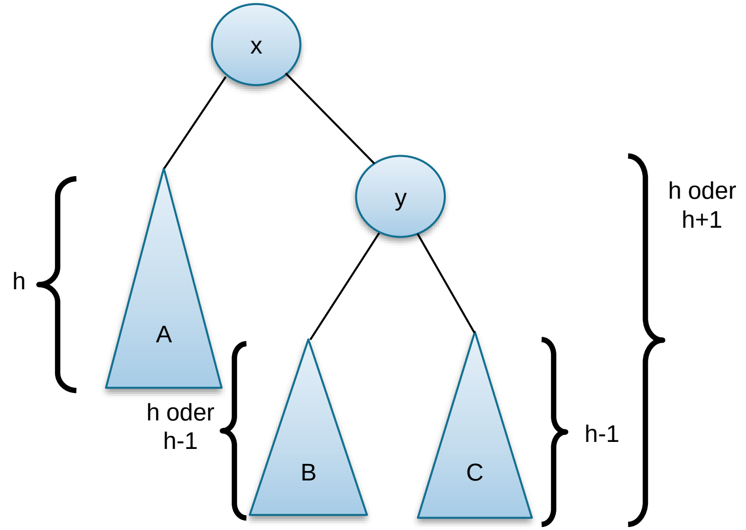
1. **if** h[lc[y]] > h[rc[y]]+1 **then**
2. **if** h[lc[lc[y]]]< h[rc[lc[y]]] **then**
3. Linksrotation(lc[y])
4. Rechtsrotation(y)
5. **else if** h[rc[y]]> h[lc[y]]+1 **then**
6. **if** h[rc[rc[y]]]< h[lc[rc[y]]] **then**
7. Rechtsrotation(rc[y])
8. Linksrotation(y)



Umformung von Beinahe-AVL-Baum zu AVL-Baum: Balance

Balance(y)

1. **if** h[lc[y]] > h[rc[y]]+1 **then**
2.   **if** h[lc[lc[y]]]< h[rc[lc[y]]] **then**
3.   Linksrotation(lc[y])
4.   Rechtsrotation(y)
5. **else if** h[rc[y]]> h[lc[y]]+1 **then**
6.   **if** h[rc[rc[y]]]< h[lc[rc[y]] **then**
7.   Rechtsrotation(rc[y])
8.   Linksrotation(y)



Achtung: Nach allen Rotationen müssen die Höhen der Knoten x und y angepasst werden!!!

Umformung von Beinahe-AVL-Baum zu AVL-Baum

Kurze Zusammenfassung

- Wir können aus einem beinahe-AVL-Baum mithilfe von maximal 2 Rotationen einen AVL-Baum machen
- Dabei erhöht sich die Höhe des Baums nicht

Einfügen

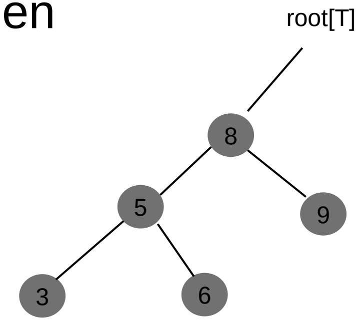
- Wir fügen ein wie früher
- Dann laufen wir den Pfad zur Wurzel zurück (z.B. als Teil der Rekursion)
- An jedem Knoten balancieren wir, falls der Unterbaum ein beinahe-AVL-Baum ist

AVL-Bäume: Einfügen

Technische Universität Berlin

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)



Einfügen(2)

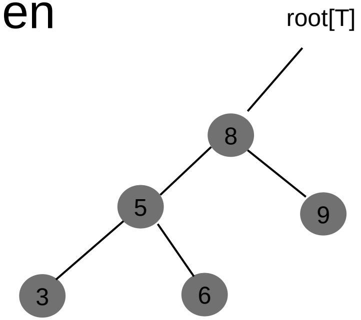
Open Distributed Sytems
ODS

AVL-Bäume: Einfügen

Wird im Beispiel aufgerufen mit
AVL-Einfügen(root[T], 2)

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)



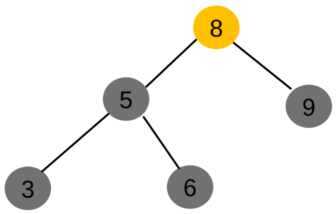
Einfügen(2)

AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➤ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)

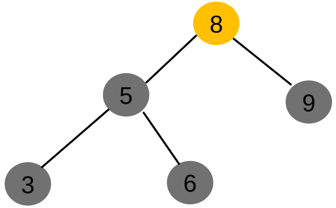


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

if t=**nil** **then**
    t ← new node(x); h[t]←0 ; **return**
else if x<key[t] **then** AVL-Einfügen(lc[t],x)
else if x>key[t] **then** AVL-Einfügen(rc[t],x)
else return ➢ Schlüssel schon vorhanden
h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)

Einfügen(2)

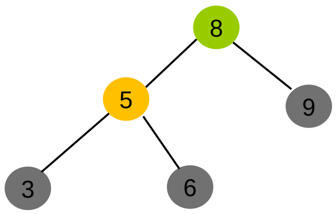


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=nil **then**
2.   t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)

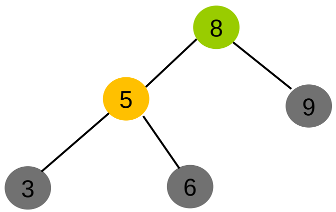


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=nil **then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➤ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)

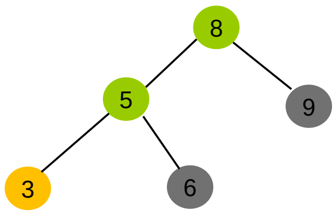


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)

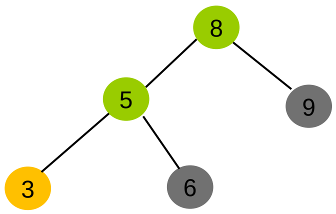


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=**nil** **then**
2. t $\leftarrow$ new node(x); h[t]$\leftarrow$0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] $\leftarrow$ 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)

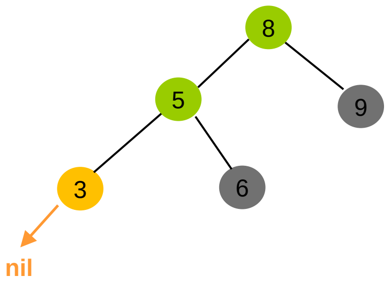


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t $\leftarrow$ new node(x); h[t]$\leftarrow$0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] $\leftarrow$ 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)



AVL-Bäume: Einfügen

Technische Universität Berlin

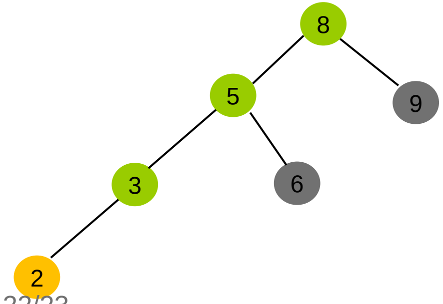
AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Neuen Knoten erzeugen. Zusätzlich noch Zeiger lc[t] und rc[t] auf **nil** setzen, sowie p[t] und den Zeiger von p[t] setzen.

Hinweis: Eventuell muss die Wurzel des Baums (root[T]) angepasst werden

Einfügen(2)



Open Distributed Sytems
ODS

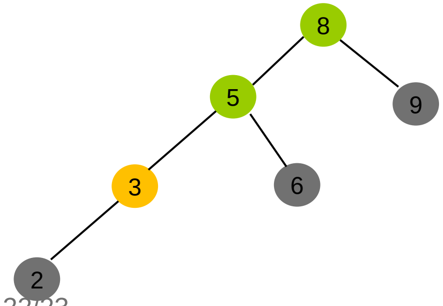
AVL-Bäume: Einfügen

Technische Universität Berlin

AVL-Einfügen(t,x)

if t=**nil** **then**
    t ← new node(x); h[t]←0 ; **return**
else if x<key[t] **then** AVL-Einfügen(lc[t],x)
else if x>key[t] **then** AVL-Einfügen(rc[t],x)
else return ➢ Schlüssel schon vorhanden
h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)

Einfügen(2)



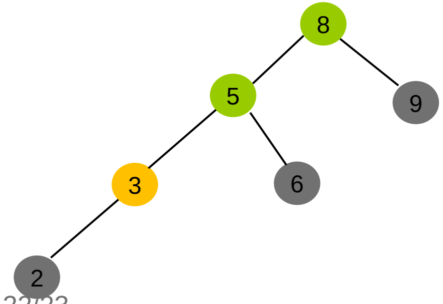
Open Distributed Sytems
ODS

AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

if t=**nil** **then**
    t ← new node(x); h[t]←0 ; **return**
else if x<key[t] **then** AVL-Einfügen(lc[t],x)
else if x>key[t] **then** AVL-Einfügen(rc[t],x)
else return ⮚ Schlüssel schon vorhanden
h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)

Einfügen(2)

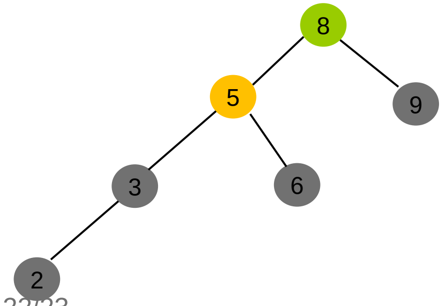


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)

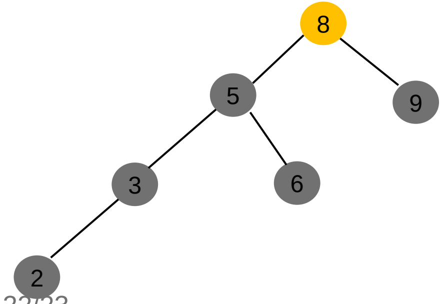


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)

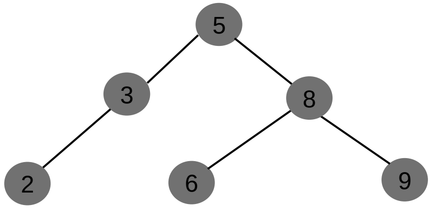


AVL-Bäume: Einfügen

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t $\leftarrow$ new node(x); h[t]$\leftarrow$0 ; **return**
3. **else if** x<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** x>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] $\leftarrow$ 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Einfügen(2)



AVL-Bäume: Einfügen

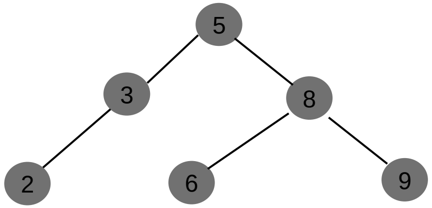
AVL-Einfügen(t,x)

if t=**nil then**
    t ← new node(x); h[t]←0 ; **return**
else if x<key[t] **then** AVL-Einfügen(lc[t],x)
else if x>key[t] **then** AVL-Einfügen(rc[t],x)
else return ➤ Schlüssel schon vorhanden
h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)

Laufzeit

- O(h) = O(log n)

Einfügen(2)



AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

AVL-Einfügen(t,x)
1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

Beweis: *per* Induktion über die Höhe des Baumes.

AVL-Einfügen(t,x)
1. **if** t=**nil then**
2. t $\leftarrow$ new node(x); h[t]$\leftarrow$0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] $\leftarrow$ 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

Beweis: *per* Induktion über die Höhe des Baumes.

- (I.A.): Für Bäume der Höhe 0 und 1 ist die Aussage korrekt.

AVL-Einfügen(t,x)
1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

Beweis: *per* Induktion über die Höhe des Baumes.

- (I.A.): Für Bäume der Höhe 0 und 1 ist die Aussage korrekt.
- (I.V.): Der Satz gilt für Bäume der Höhe j, $1 \leq j \leq h$.

```
AVL-Einfügen(t,x)
if t=nil then
    t ← new node(x); h[t]←0 ; return
else if key[x]<key[t] then AVL-Einfügen(lc[t],x)
else if key[x]>key[t] then AVL-Einfügen(rc[t],x)
else return     ➢ Schlüssel schon vorhanden
h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

Beweis: *per* Induktion über die Höhe des Baumes.

- (I.A.): Für Bäume der Höhe 0 und 1 ist die Aussage korrekt.
- (I.V.): Der Satz gilt für Bäume der Höhe j, 1≤j≤h.
- (I.S.): Betrachte den Aufruf von AVL-Einfügen in einem AVL-Baum der Höhe h+1≥0.

```
AVL-Einfügen(t,x)
if t=nil then
    t ← new node(x); h[t]←0 ; return
else if key[x]<key[t] then AVL-Einfügen(lc[t],x)
else if key[x]>key[t] then AVL-Einfügen(rc[t],x)
else return    ➢ Schlüssel schon vorhanden
h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

AVL-Bäume Einfügen: Beweis für die Höhe

Technische Universität Berlin

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

```
AVL-Einfügen(t,x)
1. if t=nil then
2.     t ← new node(x); h[t]←0 ; return
3. else if key[x]<key[t] then AVL-Einfügen(lc[t],x)
4. else if key[x]>key[t] then AVL-Einfügen(rc[t],x)
5. else return    ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)
```

Beweis: *per* Induktion über die Höhe des Baumes.

- (I.A.): Für Bäume der Höhe 0 und 1 ist die Aussage korrekt.
- (I.V.): Der Satz gilt für Bäume der Höhe j, $1 \leq j \leq h$.
- (I.S.): Betrachte den Aufruf von AVL-Einfügen in einem AVL-Baum der Höhe $h+1 \geq 0$.
  - Sei o.B.d.A. key[x]<key[t] (der andere Fall ist symmetrisch).

Open Distributed Sytems
ODS

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

AVL-Einfügen(t,x)
1. **if** t=nil **then**
2. t $\leftarrow$ new node(x); h[t]$\leftarrow$0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] $\leftarrow$ 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Beweis: *per* Induktion über die Höhe des Baumes.

- (I.A.): Für Bäume der Höhe 0 und 1 ist die Aussage korrekt.
- (I.V.): Der Satz gilt für Bäume der Höhe j, 1≤j≤h.
- (I.S.): Betrachte den Aufruf von AVL-Einfügen in einem AVL-Baum der Höhe h+1≥0.
  - Sei o.B.d.A. key[x]<key[t] (der andere Fall ist symmetrisch).
  - Da h+1≥0 ist, wird Zeile (3) ausgeführt.

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

Beweis: (Induktion über die Höhe des Baumes. – IS)

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

Beweis: (Induktion über die Höhe des Baumes. – IS)

– Nach (I.V.) ist der Baum lc[t] nach Einfügen ein AVL-Baum mit Höhe r oder r+1, wobei r die Höhe *vor* dem Einfügen war.

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t $\leftarrow$ new node(x); h[t]$\leftarrow$0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] $\leftarrow$ 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

Beweis: (Induktion über die Höhe des Baumes. – IS)

- Nach (I.V.) ist der Baum lc[t] nach Einfügen ein AVL-Baum mit Höhe r oder r+1, wobei r die Höhe *vor* dem Einfügen war.
- Hat lc[t] nach dem Einfügen Höhe h oder h-1, so ist t ein AVL-Baum.

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

AVL-Einfügen(t,x)

1. **if** t=nil **then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Beweis: (Induktion über die Höhe des Baumes. – IS)

- Nach (I.V.) ist der Baum lc[t] nach Einfügen ein AVL-Baum mit Höhe r oder r+1, wobei r die Höhe *vor* dem Einfügen war.
- Hat lc[t] nach dem Einfügen Höhe h oder h-1, so ist t ein AVL-Baum.
- Hat lc[t] nach dem Einfügen Höhe h+1, so ist t u.U. ein beinahe-AVL-Baum.

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

Beweis: (Induktion über die Höhe des Baumes. – IS)

– Nach (I.V.) ist der Baum lc[t] nach Einfügen ein AVL-Baum mit Höhe r oder r+1, wobei r die Höhe *vor* dem Einfügen war.

– Hat lc[t] nach dem Einfügen Höhe h oder h-1, so ist t ein AVL-Baum.

– Hat lc[t] nach dem Einfügen Höhe h+1, so ist t u.U. ein beinahe-AVL-Baum.

– Dies wird in Zeile 7 durch Balance korrigiert.

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t ← new node(x); h[t]←0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➤ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

AVL-Einfügen(t,x)

1. **if** t=**nil then**
2. t $\leftarrow$ new node(x); h[t]$\leftarrow$0 ; **return**
3. **else if** key[x]<key[t] **then** AVL-Einfügen(lc[t],x)
4. **else if** key[x]>key[t] **then** AVL-Einfügen(rc[t],x)
5. **else return** ➢ Schlüssel schon vorhanden
6. h[t] $\leftarrow$ 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)

Beweis: (Induktion über die Höhe des Baumes. – IS)

- Nach (I.V.) ist der Baum lc[t] nach Einfügen ein AVL-Baum mit Höhe r oder r+1, wobei r die Höhe *vor* dem Einfügen war.
- Hat lc[t] nach dem Einfügen Höhe h oder h-1, so ist t ein AVL-Baum.
- Hat lc[t] nach dem Einfügen Höhe h+1, so ist t u.U. ein beinahe-AVL-Baum.
- Dies wird in Zeile 7 durch Balance korrigiert.
- Außerdem erhöht Balance die Höhe nicht und verringert sie maximal um 1.

AVL-Bäume Einfügen: Beweis für die Höhe

Satz Wird ein Element in einen AVL-Baum der Höhe h eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h+1.

```
AVL-Einfügen(t,x)
1. if t=nil then
2.     t ← new node(x); h[t]←0 ; return
3. else if key[x]<key[t] then AVL-Einfügen(lc[t],x)
4. else if key[x]>key[t] then AVL-Einfügen(rc[t],x)
5. else return    ➢ Schlüssel schon vorhanden
6. h[t] ← 1 + max{h[lc[t]], h[rc[t]]}
7. Balance(t)
```

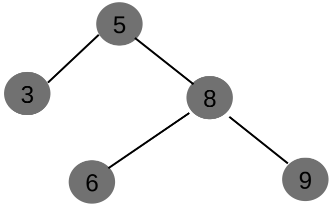
Beweis: (Induktion über die Höhe des Baumes. – IS)

- Nach (I.V.) ist der Baum lc[t] nach Einfügen ein AVL-Baum mit Höhe r oder r+1, wobei r die Höhe *vor* dem Einfügen war.
- Hat lc[t] nach dem Einfügen Höhe h oder h-1, so ist t ein AVL-Baum.
- Hat lc[t] nach dem Einfügen Höhe h+1, so ist t u.U. ein beinahe-AVL-Baum.
- Dies wird in Zeile 7 durch Balance korrigiert.
- Außerdem erhöht Balance die Höhe nicht und verringert sie maximal um 1.
- Also hat der Baum nach dem Einfügen Höhe h+1 oder h+2.

AVL-Bäume: Löschen

AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```



AVL-Bäume: Löschen

Technische Universität Berlin

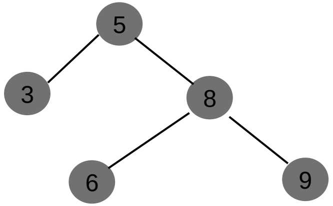
AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

- x bezeichnet den Schlüssel des zu löschenden Elements
- t ist der Teilbaum.

Wird aufgerufen mit AVL-Löschen(root[T], 3)

Löschen(3)



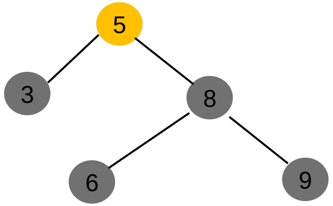
Open Distributed Sytems ODS

AVL-Bäume: Löschen

AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

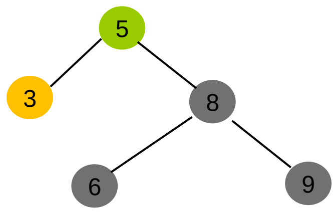
Löschen(3)



AVL-Bäume: Löschen

```
AVL-Löschen(t,x)
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
      Kopiere Informationen von u nach t
      AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

Löschen(3)



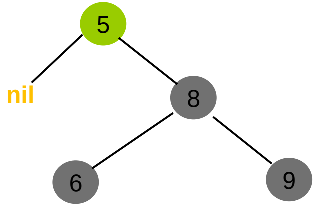
AVL-Bäume: Löschen

AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

Erfordert, einige Zeiger zu aktualisieren

Löschen(3)



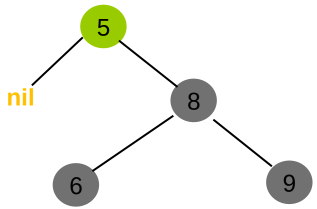
AVL-Bäume: Löschen

AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

Nichts zu tun,
da Baum leer.

Löschen(3)



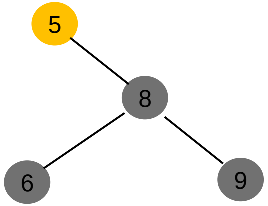
AVL-Bäume: Löschen

AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

Anpassen der Höhe.

Löschen(3)



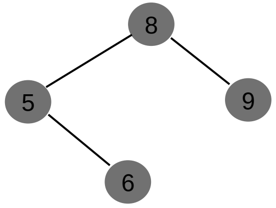
AVL-Bäume: Löschen

Technische Universität Berlin

AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

Löschen(3)



Open Distributed Sytems
ODS

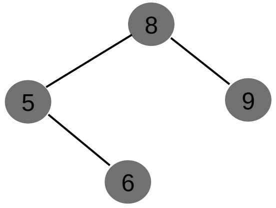
AVL-Bäume: Löschen

AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

Hinweis: Eventuell muss die Wurzel des Baums (root[T]) angepasst werden

Löschen(3)



AVL-Bäume Löschen : Beweis für die Höhe

Satz Wird ein Element aus einem AVL-Baum der Höhe h gelöscht, so ist der resultierende Baum ein AVL-Baum der Höhe h oder h-1.

Beweis: analog zum Einfügen

AVL-Löschen(t,x)

```
if t=nil then return ➢ x nicht im Baum
else if x<key[t] then AVL-Löschen(lc[t],x)
else if x>key[t] then AVL-Löschen(rc[t],x)
else if lc[t]=nil then ersetze t durch rc[t]
else if rc[t]=nil then ersetze t durch lc[t]
else u=MaximumSuche(lc[t])
        Kopiere Informationen von u nach t
        AVL-Löschen(lc[t],key[u])
if t≠nil then h[t] = 1 + max{h[lc[t]], h[rc[t]]}
Balance(t)
```

Laufzeit von Operationen auf AVL-Bäumen

Satz: Mithilfe von AVL-Bäumen kann man
Suche, Einfügen, Löschen, Minimum und Maximum in einer Menge von n Zahlen in $\Theta(\log n)$ Laufzeit durchführen.

Bäume (allgemein)

- *Def.:* Ein **t-ärer Baum** ist entweder die leere Menge oder ein Knoten (ein Objekt), welcher ein Datum und t Kindbäume enthält, welche t-äre Baume sind.

- *Def.:* Ein **Binär-Baum** ist entsprechend ein 2-ärer Baum.

Bäume (allgemein)

- Knotenorientierte Bäume: Daten liegen in den Knoten.
- Blattorientierte Bäume: Daten liegen nur in den Blättern.
  - Innere Knoten enthalten nur Zugriffsinformation
- Kantenorientierte Bäume: Daten in den Kanten

Bäume – Implementierung

- In einem Array – Dichte Speicherung
  - Vorgänger und Nachfolger werden durch Indexrechnung bestimmt
  - Gut für linksvolle oder rechtsvolle Bäume

- Per Pointer – Gestreute Speicherung
  - Baumklasse hat eine Referenz auf die Wurzel
  - Zeiger zu den Kindern
  - Möglichkeiten
    - Mit/ohne Nullelement
    - Mit/ohne Rückverzeigerung von den Blättern zur Wurzel
    - Statische oder dynamische Methoden zur Veränderung

Traversierung von Bäumen

- Pre-order: Knoten -> Links -> Rechts

- Post-order: Links -> Rechts -> Knoten

- In-order: Links -> Knoten -> Rechts

- Level-order: Schicht nach Schicht von der Wurzel aus

Traversierung von Bäumen

- Pre-order: Knoten -> Links -> Rechts

- Post-order: Links -> Rechts -> Knoten

- In-order: Links -> Knoten -> Rechts

- Level-order: Schicht nach Schicht von der Wurzel aus

- Erste drei Varianten: Depth first
- Letzte Variante: Breadth first

Bäume – Zusammenfassung

- Abbildung von Daten in einer Baumstruktur
  - Natürliche Ordnung der Daten.
  - Für effiziente Verarbeitung: O(log(n)) anstelle von O(n)

- Vier Haupttypen der Traversierung
  - Implementierbar: Rekursiv, stack/queue

- Suchbäume ermöglichen das Suchen in O(log(n))
  - Zusätzlicher Aufwand notwendig, um den Baum beim Einfügen, Löschen balanciert zu halten

Ausblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine

VL 1 „Algorithmen, Pseudocode, Sortieren I“: Insertion Sort

VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort

VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortierverfahren

VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue

VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen

VL 6 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C

VL 7 „Teile und Herrsche I“: Einführung der algorithmischen Methode, Merge Sort

VL 8 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise

VL 9 „Prioritätenschlangen/Halden/Heaps “: Heap Sort, Binärer Heap, Heap Operationen

VL 10 „Fortgeschrittene Sortierverfahren“: Quick Sort, Radix Sort

VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung

VL 12 „Teile und Herrsche II“: Generalisierung des algorithmischen Prinzips, Mastertheorem

VL 13 „Q & A“: Offene Vorlesung/Wiederholung