

# Einführung in die Programmierung

## AVL Bäume

# Wiederholung: Bäume

## *Ein grundlegendes Problem*

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

## *Beispiel*

- Stammbaum
- Dateisysteme
- Entscheidungsbäume
- Suchbäume, z.B. für Lexika Daten
- Rekursionsbäume

## *Anforderungen*

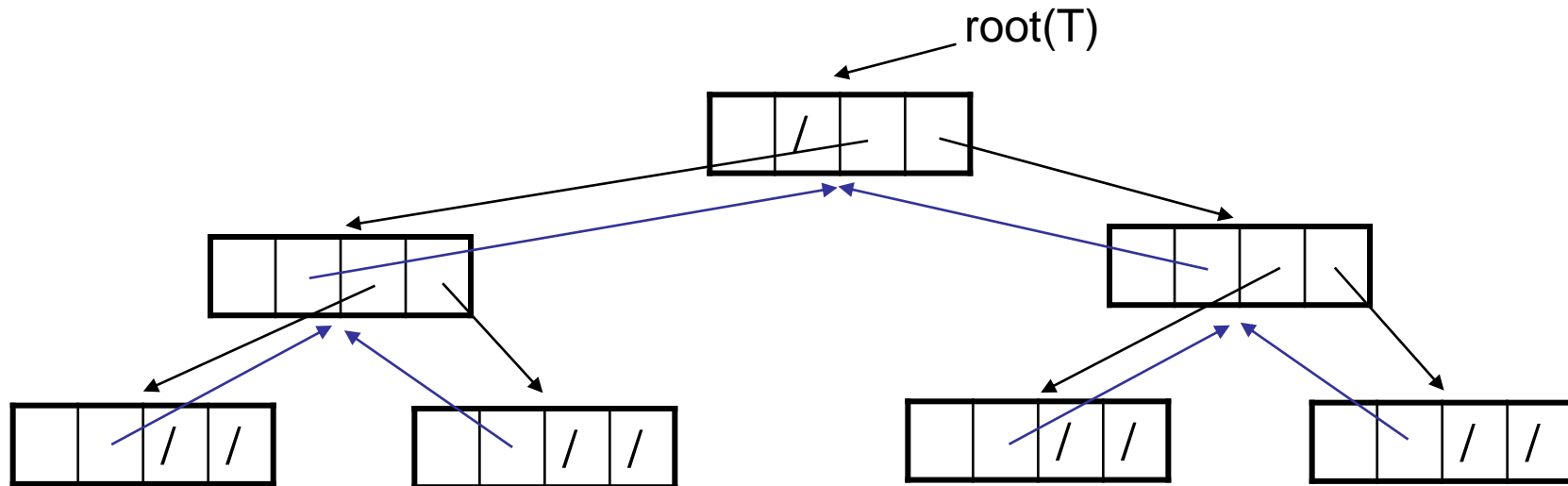
- Schneller Zugriff, d.h. schneller als  $O(n)$
- Einfügen neuer Datensätze
- Löschen bestehender Datensätze

# Wiederholung: Binärbaum

## Binärbäume (Darstellung im Rechner)

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

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



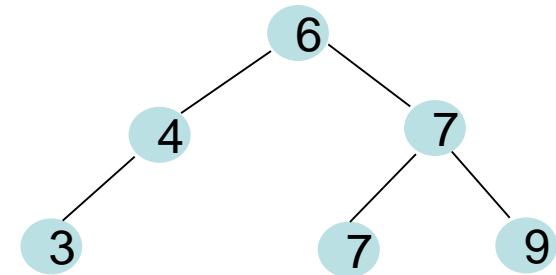
# Wiederholung: Binäre Suchbäume

## *Binäre Suchbäume*

- Verwende Binärbaum
- Speichere Schlüssel „geordnet“

## *Binäre Suchbaumeigenschaft:*

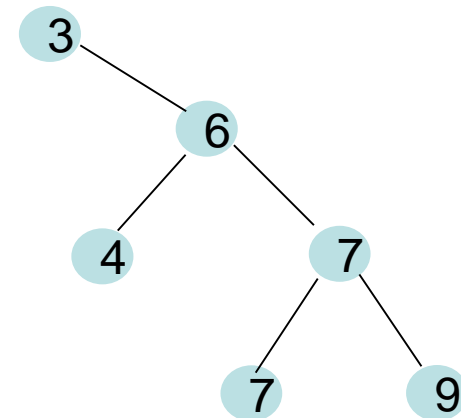
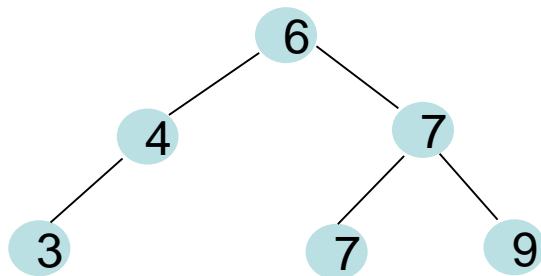
- Sei  $x$  Knoten im binären Suchbaum
- Ist  $y$  Knoten im **linken Unterbaum** von  $x$ , dann gilt  **$\text{key}(y) \leq \text{key}(x)$**
- Ist  $y$  Knoten im **rechten Unterbaum** von  $x$ , dann gilt  **$\text{key}(y) > \text{key}(x)$**



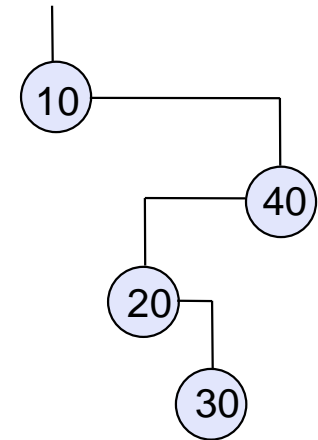
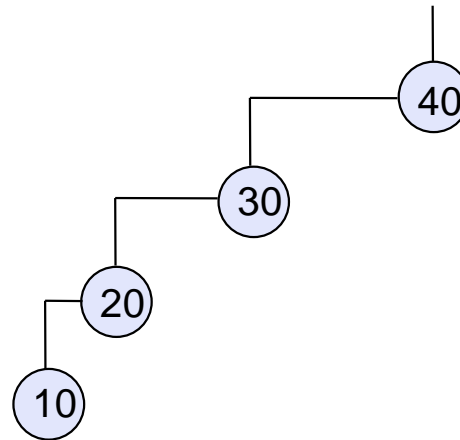
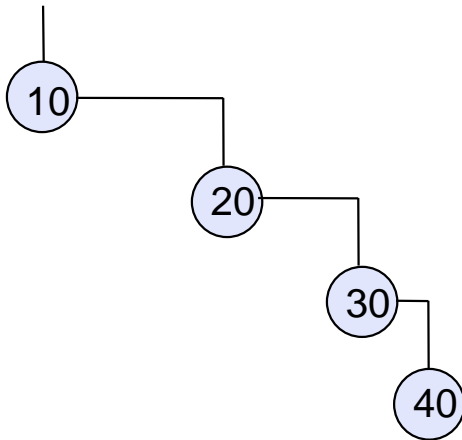
# Wiederholung: Binäre Suchbäume

## *Unterschiedliche Suchbäume*

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



# Problem – Degenerierte Suchbäume



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

# Lösung – Balancierte Suchbäume

- Ziel: Sicherstellen dass die Höhe der Teilbäume ungefähr gleich ist
- Resultat: Höhe des Baumes:  $O(\log n)$
- Wie: Rotation

# Der Weg zu balancierten Bäumen: Rotation

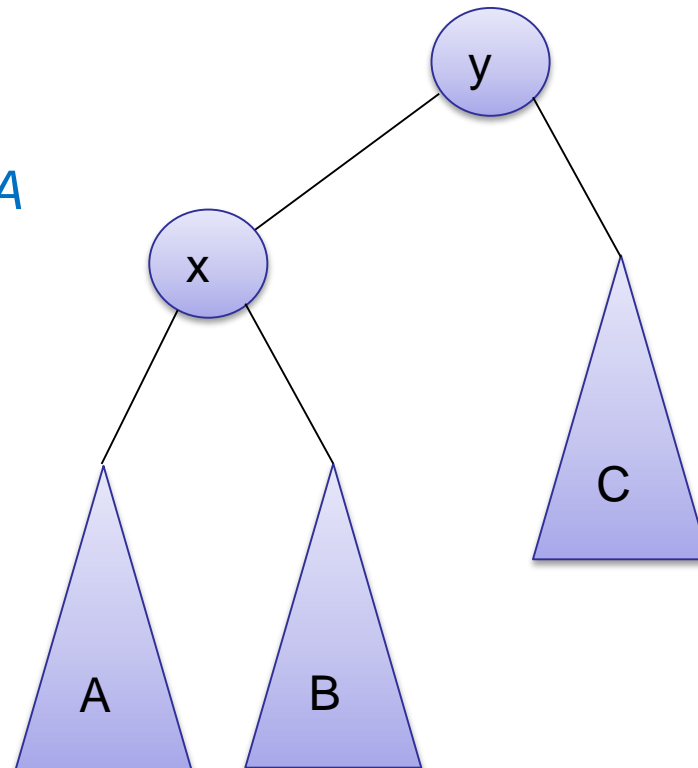
*Gegeben folgender Baum*

*Aufgabe:*

*Einfügen von a in Teilbaum A*

*Problem:*

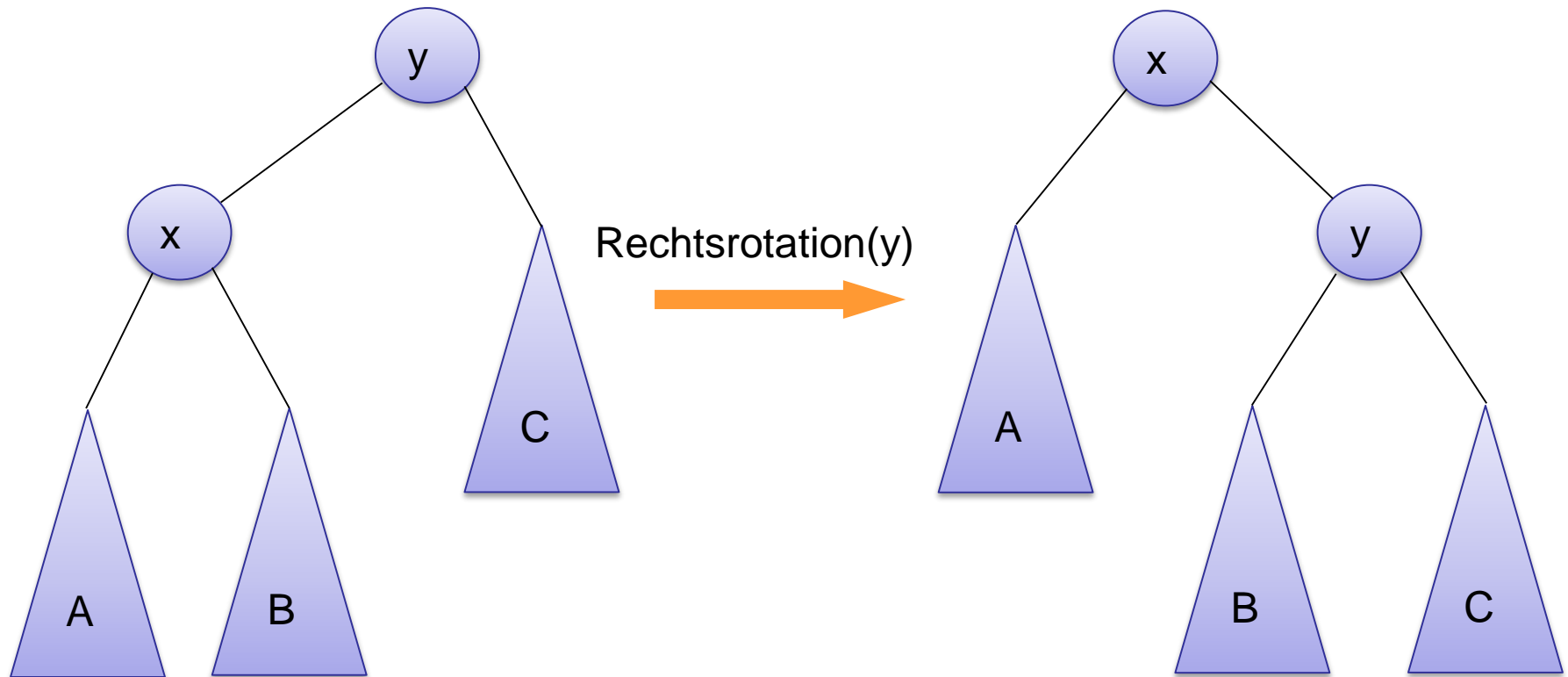
*Teilbaum von x wird zu groß*





# Der Weg zu balancierten Bäumen: Rotation

## Rotationen



# Der Weg zu balancierten Bäumen: Rotation

## Rotationen

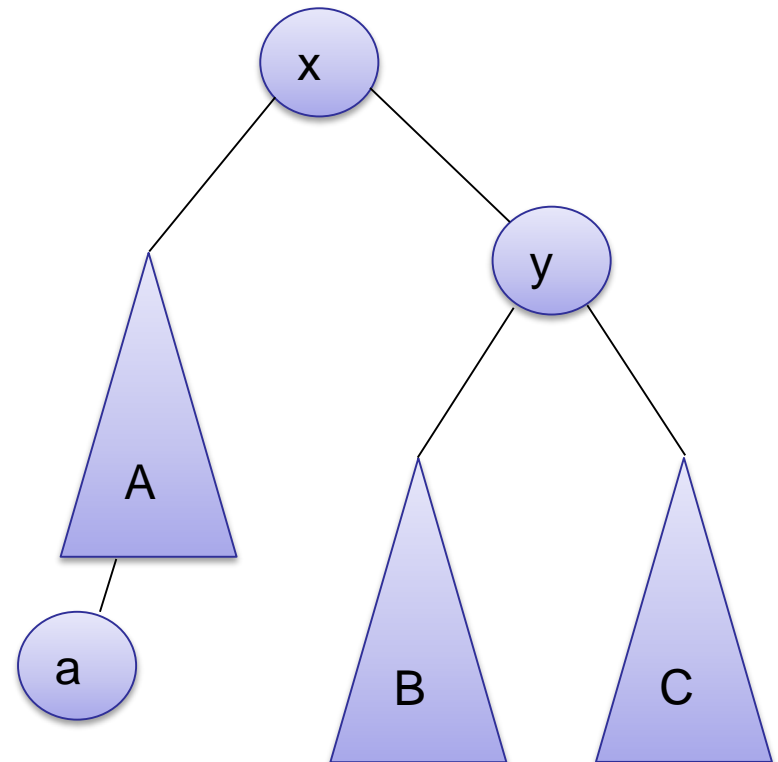
*Aufgabe:*

*Einfügen von a in Teilbaum A*

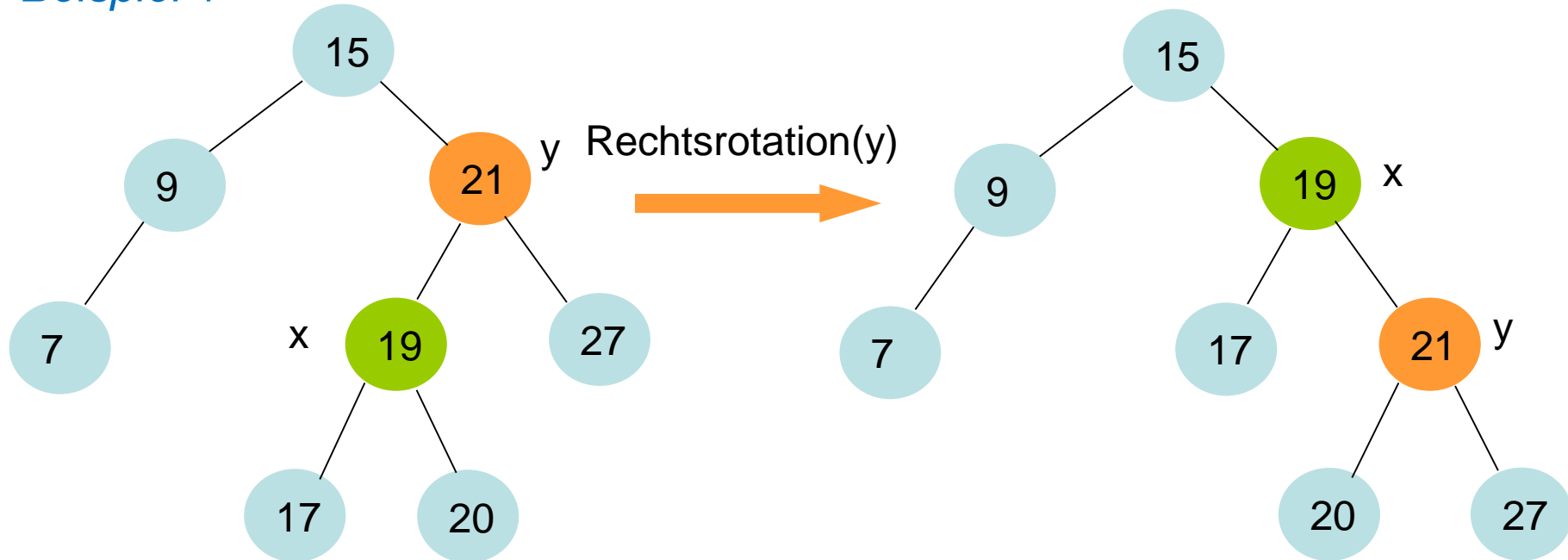
*Lösung:*

*Rotation*

*Danach Einfügen von a in Teilbaum A*



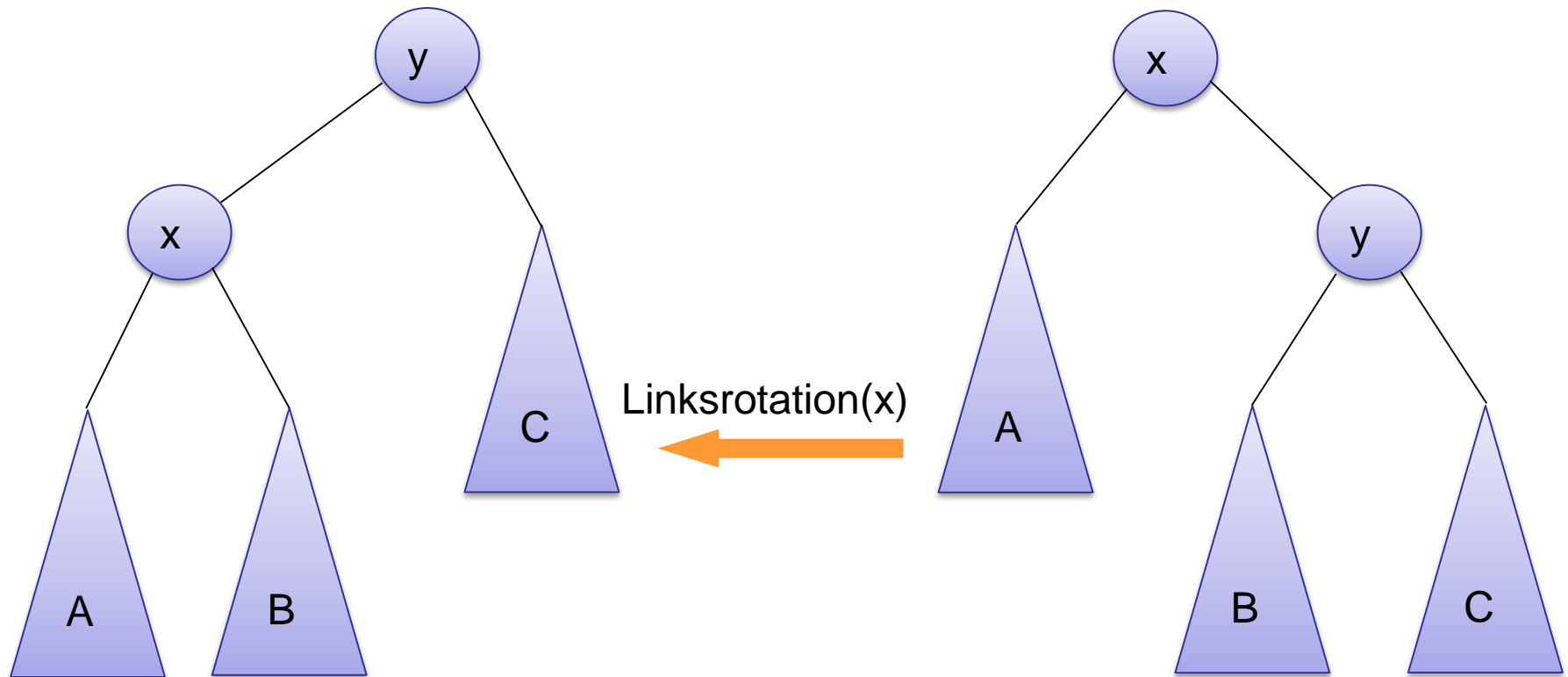
## Beispiel 1



# Der Weg zu balancierten Bäumen: Rotation

## Rotationen sind symmetrisch

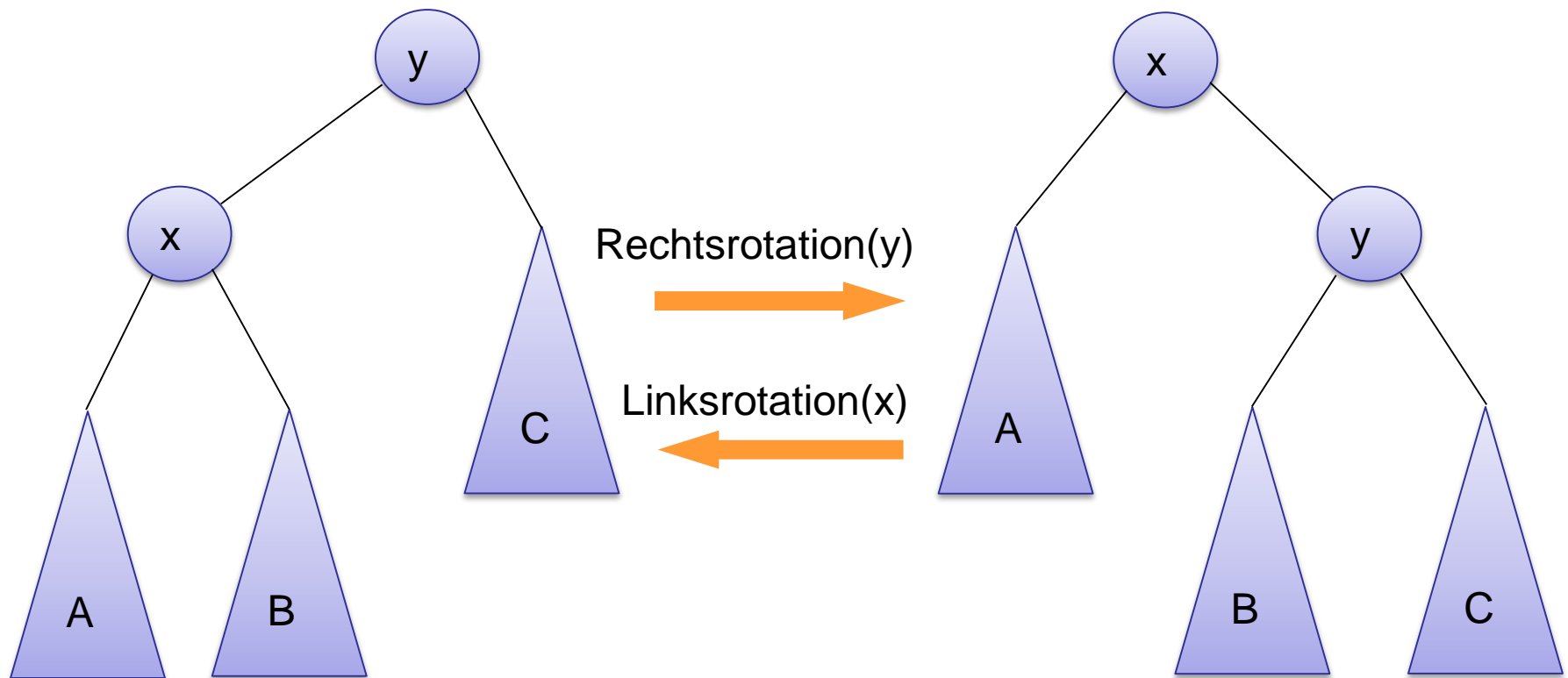
### Rotationen



# Der Weg zu balancierten Bäumen: Rotation

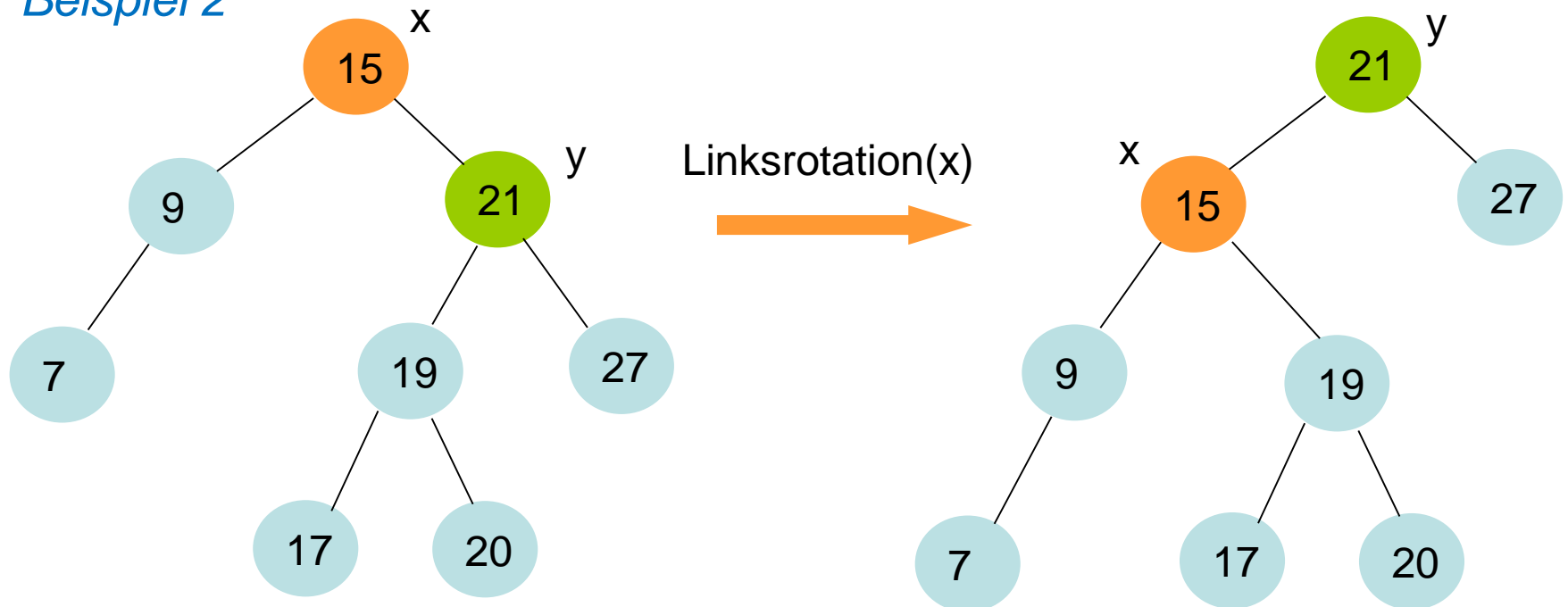
Rotationen sind symmetrisch

## Rotationen



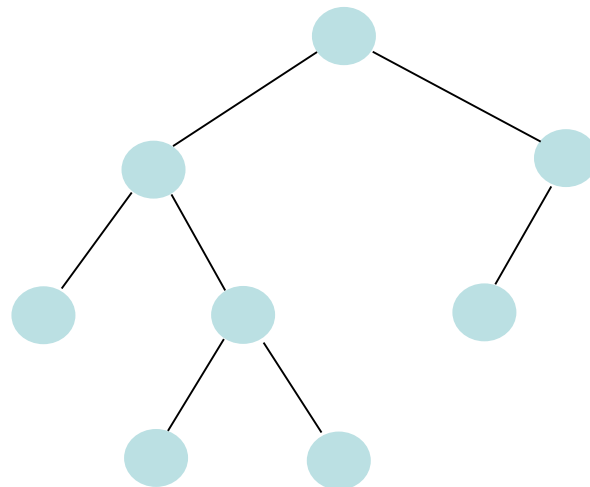
# Linksrotation

## Beispiel 2



## AVL-Bäume [Adelson-Velsky und Landis]

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



## Satz

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

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



## Satz

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

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

## Korollar

Ein AVL-Baum mit  $n$  Knoten hat Höhe  $\Theta(\log n)$ .

## Satz

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

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

## Korollar

Ein AVL-Baum mit  $n$  Knoten hat Höhe  $\Theta(\log n)$ .

## Beweis

- (1) Zeige  $h = O(\log n)$ : Es gilt  $n \geq (3/2)^h$  nach Satz

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

## Satz

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

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

## Korollar

Ein AVL-Baum mit  $n$  Knoten hat Höhe  $\Theta(\log n)$ .

## Beweis

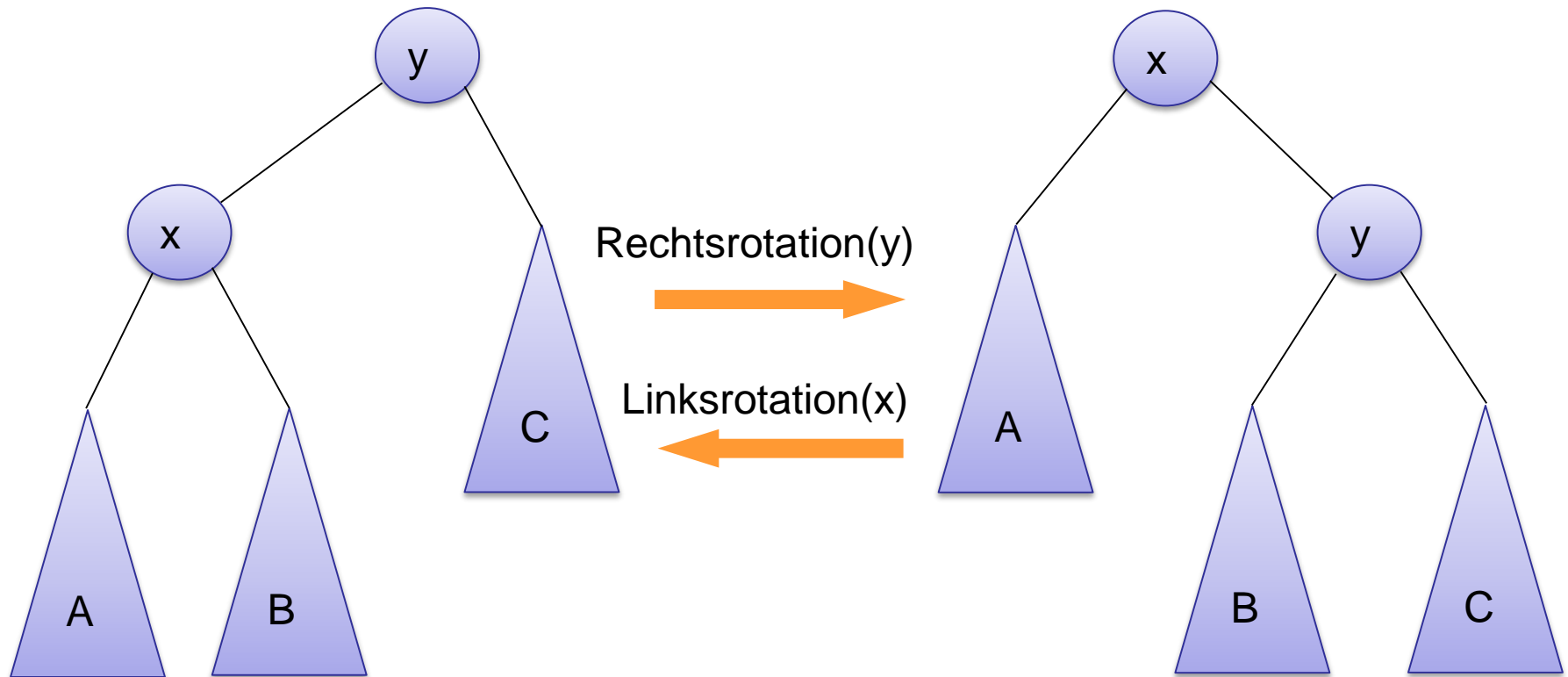
- (2) Zeige  $h = \Omega(\log n)$ : Es gilt  $n \leq 2^{h+1} - 1 \leq 2^{h+1}$  nach Satz

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

# Der Weg zu balancierten Bäumen: Rotation

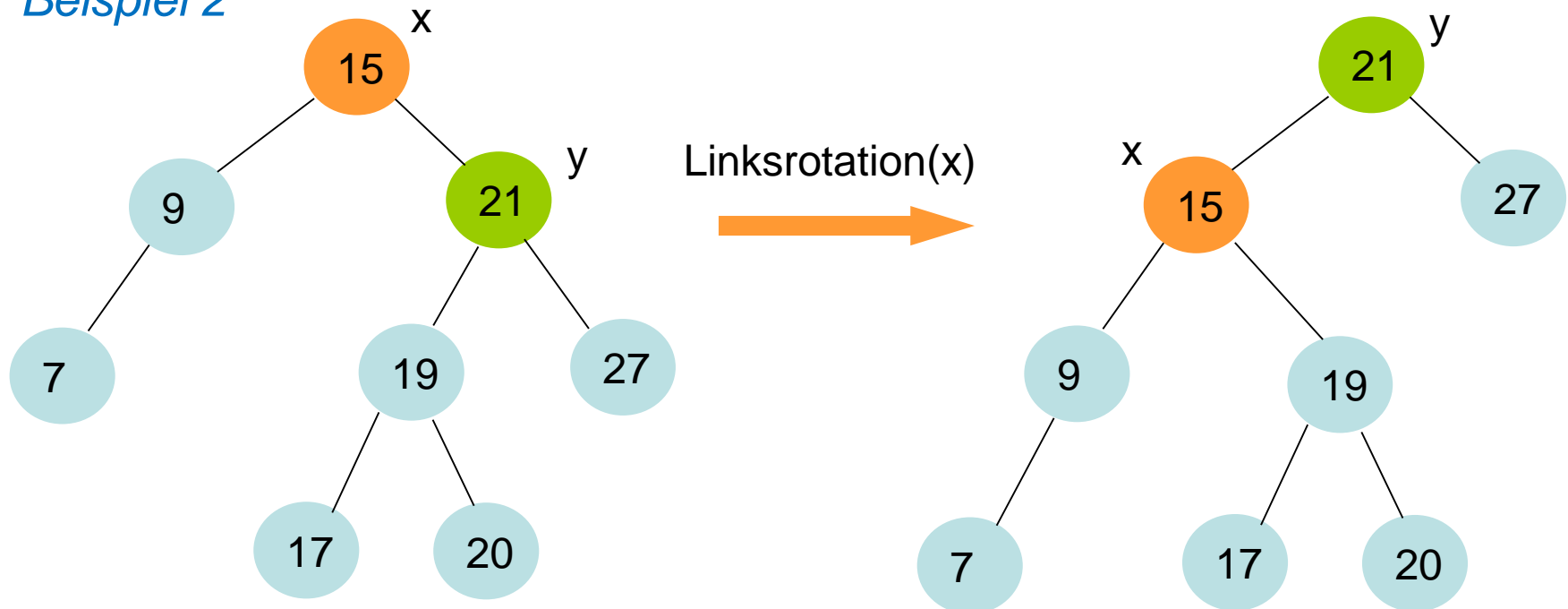
Links- und Rechtsrotation sind symmetrisch

## Rotationen



# Hier: Linksrotation (Rechtsrotation symmetrisch)

## Beispiel 2



# Linksrotation (Rechtsrotation symmetrisch)

Linksrotation( $x$ )

1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
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 (Rechtsrotation symmetrisch)

## Linksrotation(x)

Annahme: x hat  
rechtes Kind.

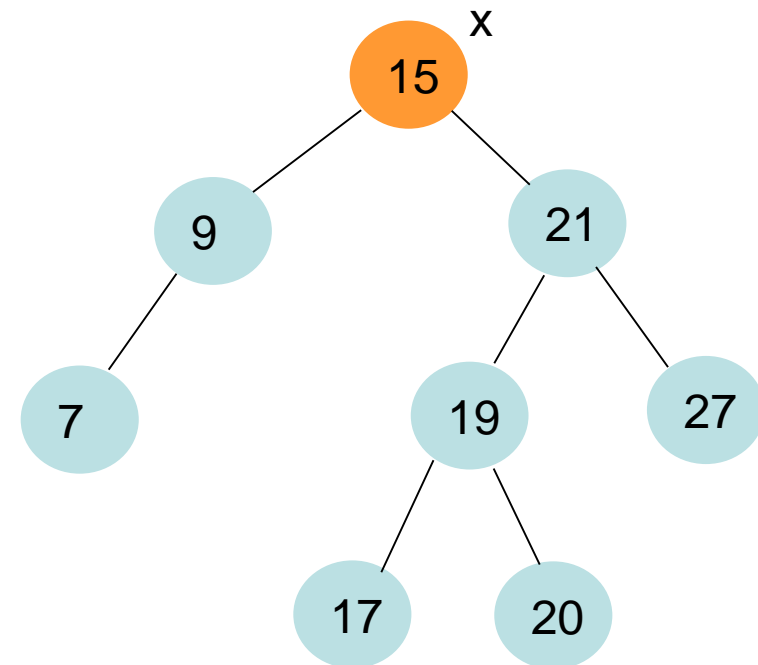
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7.     **else**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

# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

## Linksrotation(x)

1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
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.  $p[x] \leftarrow y$

Annahme: x hat  
rechtes Kind.

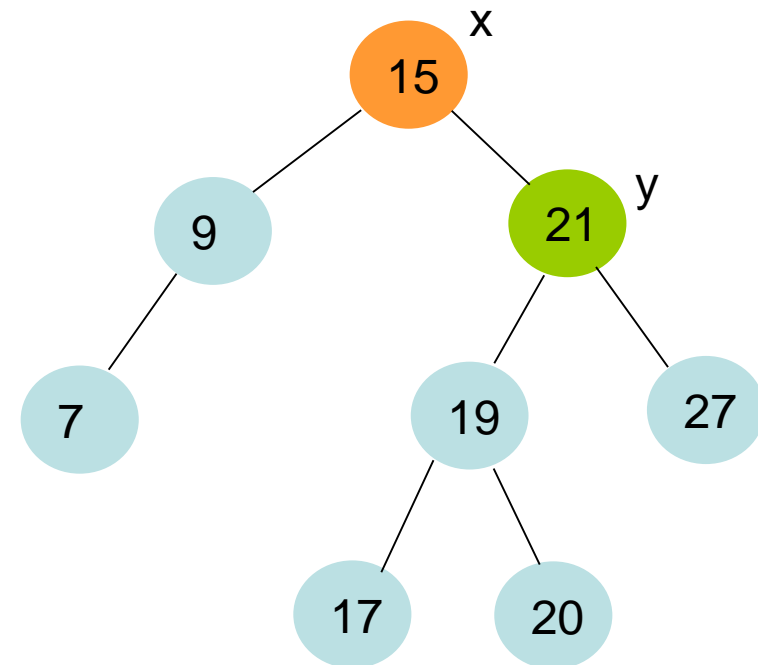




# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

Linksrotation(x)

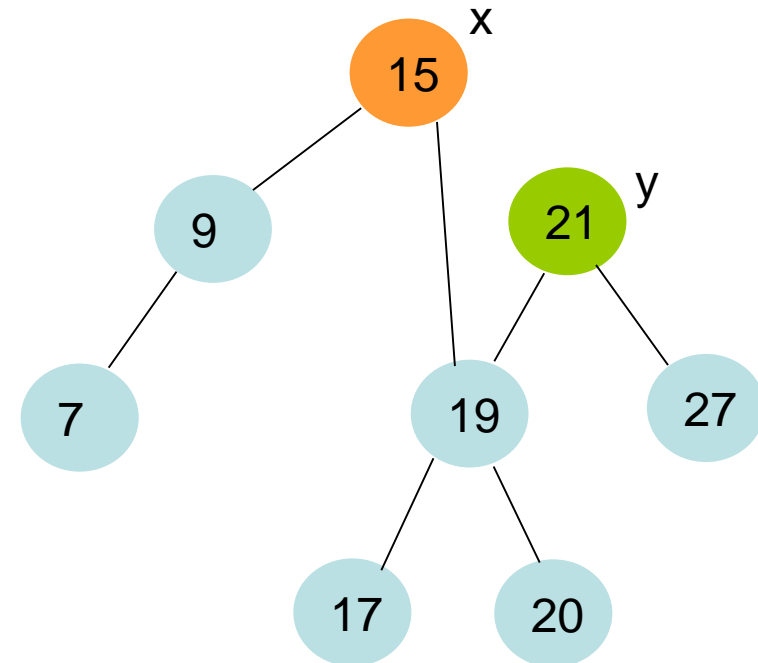
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7. **else**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



# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

Linksrotation(x)

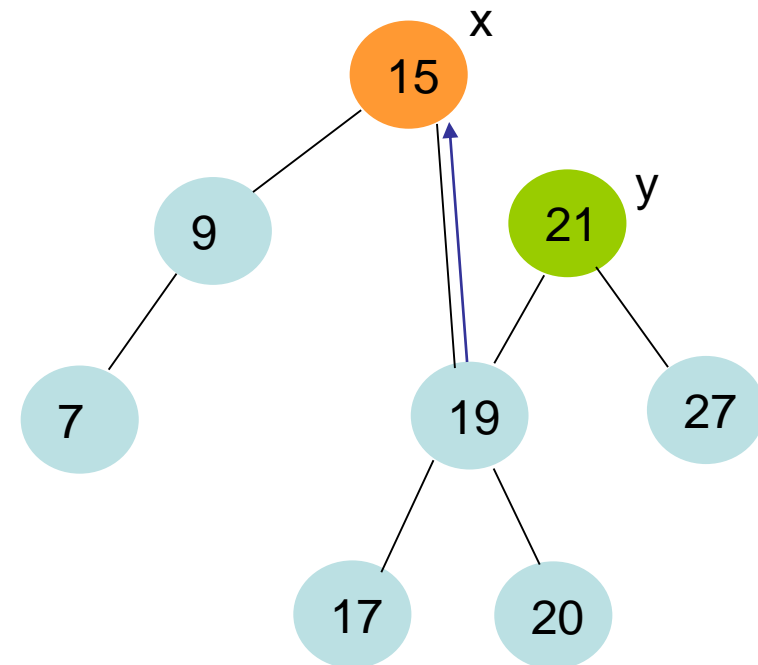
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7. **else**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



# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

Linksrotation(x)

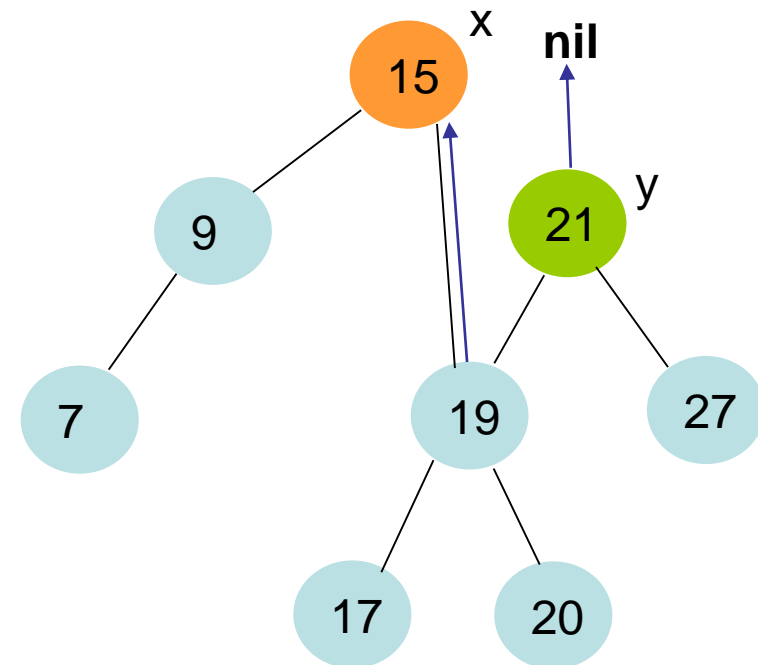
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7. **else**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



# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

Linksrotation(x)

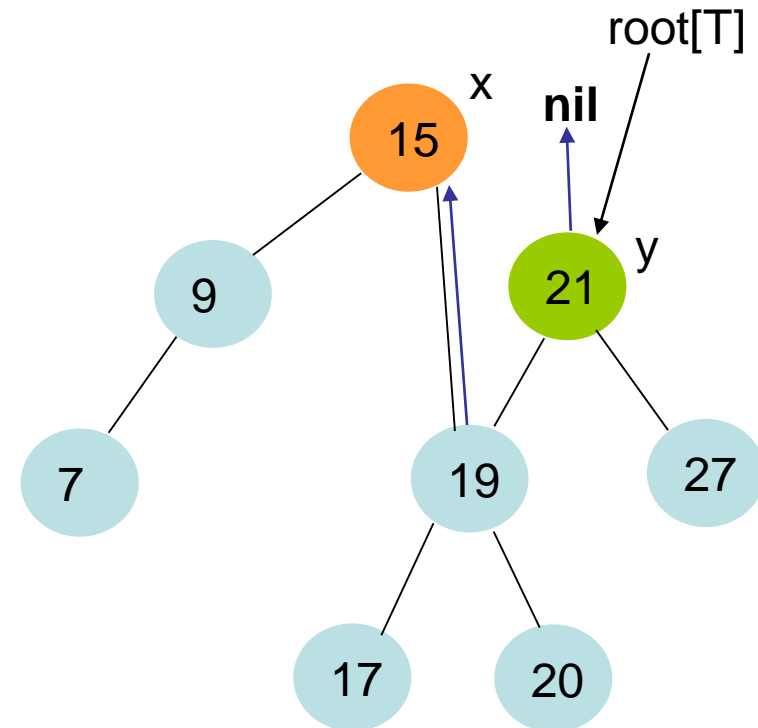
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7. **else**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



# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

Linksrotation(x)

1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **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.  $p[x] \leftarrow y$



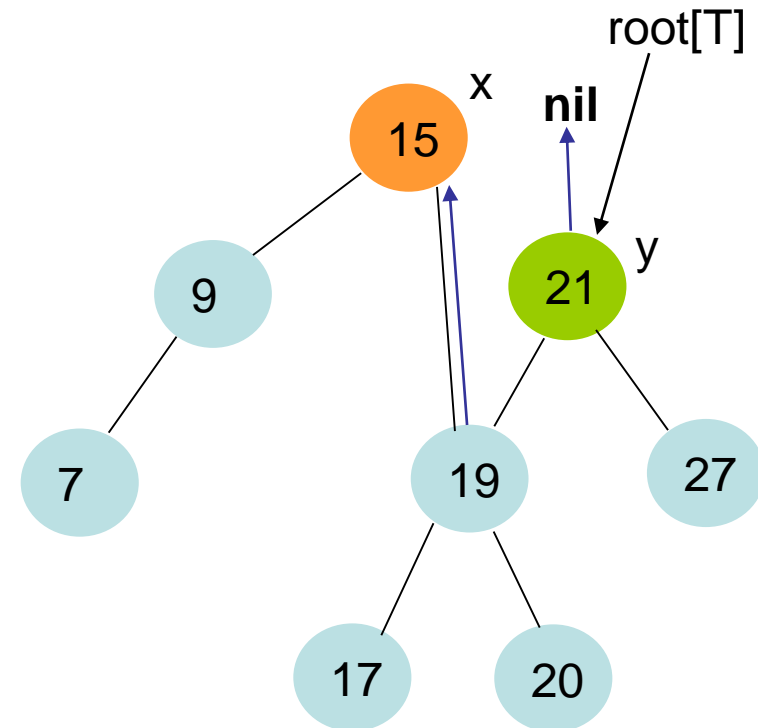
## Wichtiger Hinweis

- In diesem Fall muss die Wurzel des Baums ( $root[T]$ ) angepasst werden

# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

Linksrotation(x)

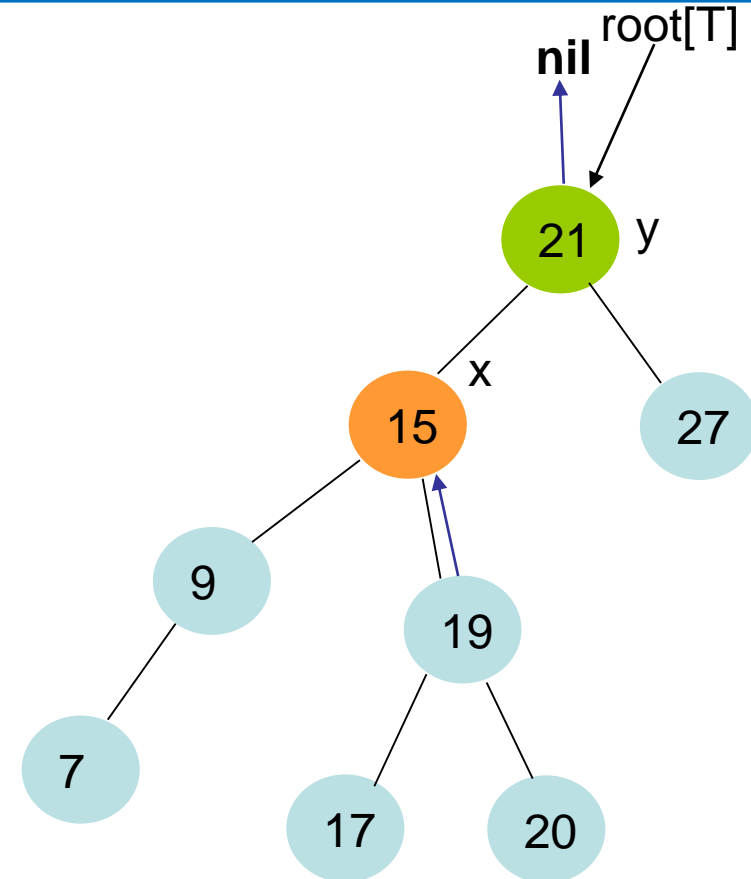
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7. **else**  $rc[p[x]]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



# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

Linksrotation(x)

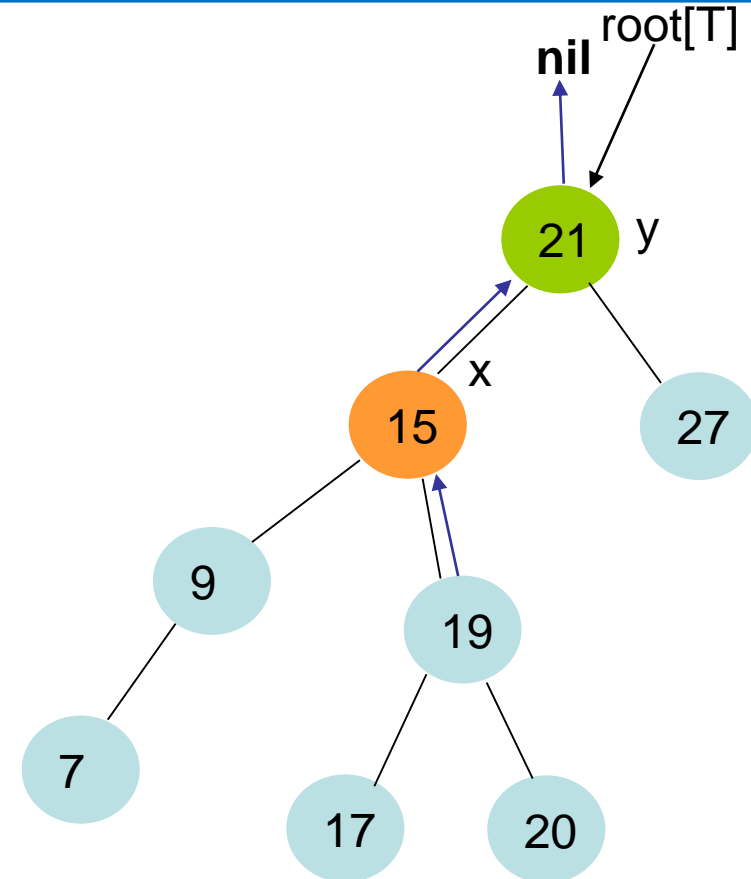
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7. **else**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



# Linksrotation (Rechtsrotation symmetrisch)

Linksrotation(x)

1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
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.  $p[x] \leftarrow y$

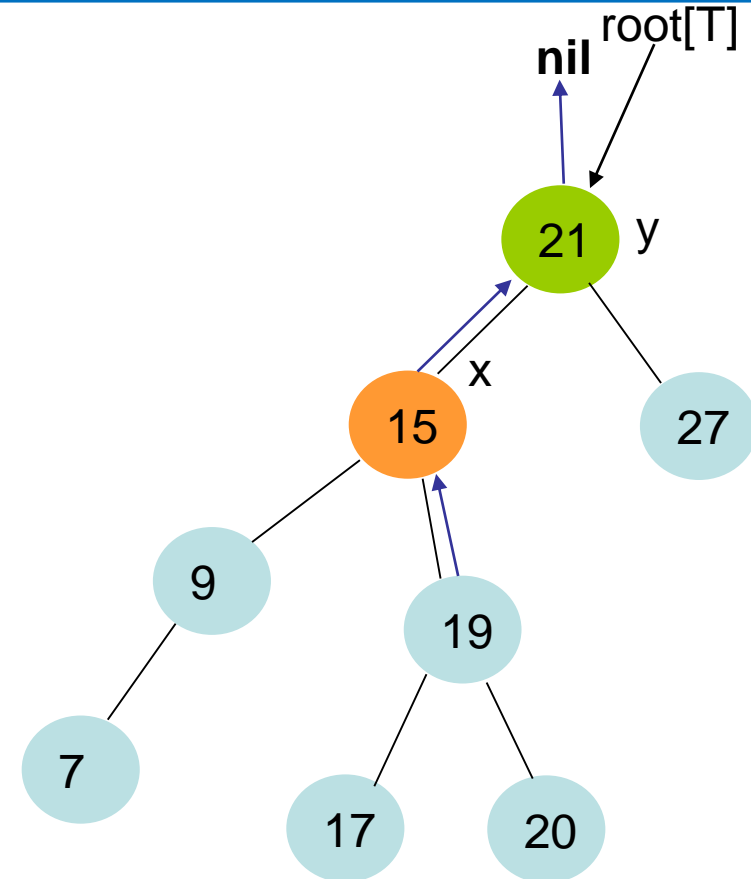




# Linksrotation auf dem Baum T (Rechtsrotation symmetrisch)

Linksrotation(x)

1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \text{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
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.  $p[x] \leftarrow y$



# 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**

## *Dynamische AVL-Bäume: Idee*

- 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  $O(\log n)$

## 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

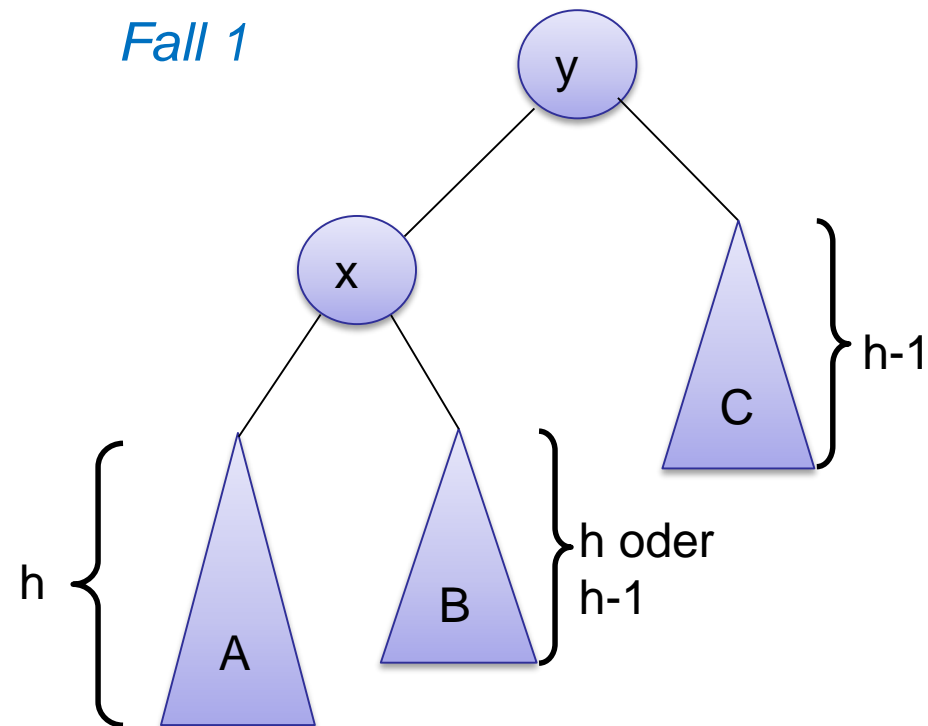
## *Unterproblem*

- Umformen eines beinahe-AVL-Baums in einen AVL-Baum mit Hilfe 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

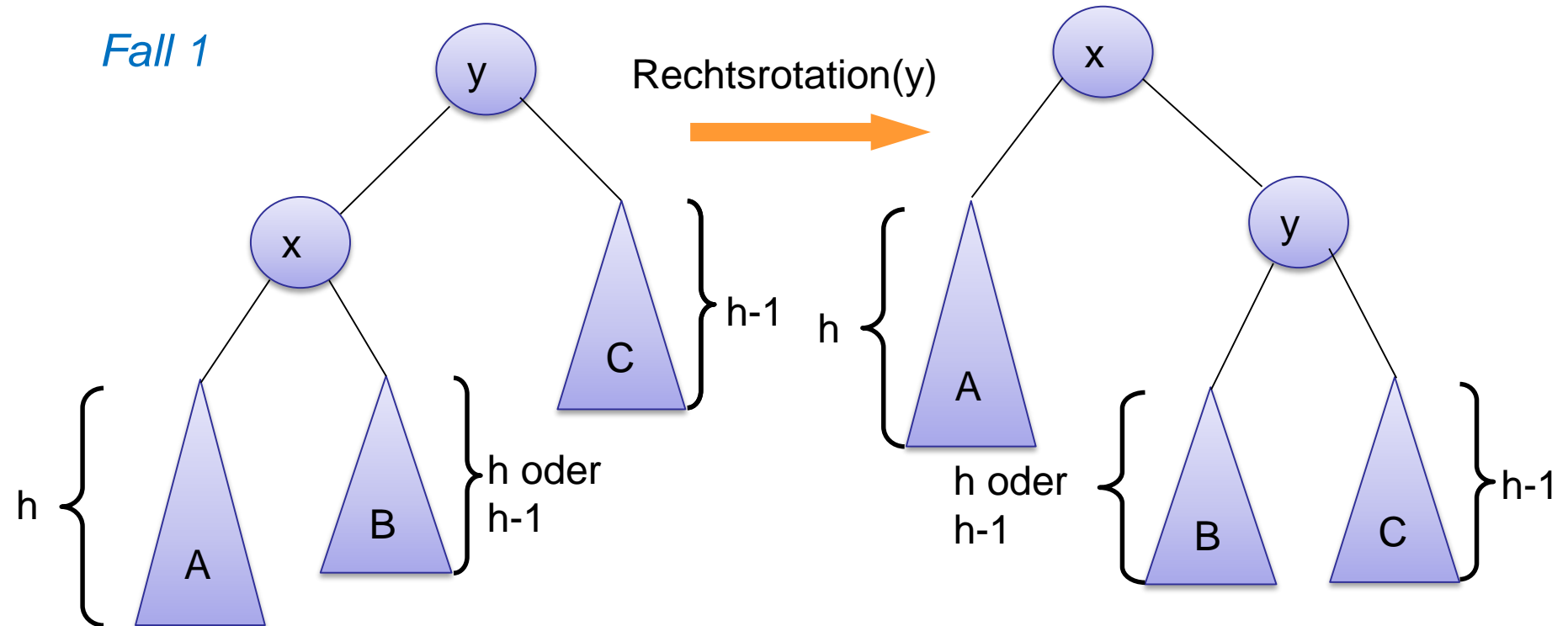
*Fall 1*



# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Fall 1: Einfache Rechtsrotation

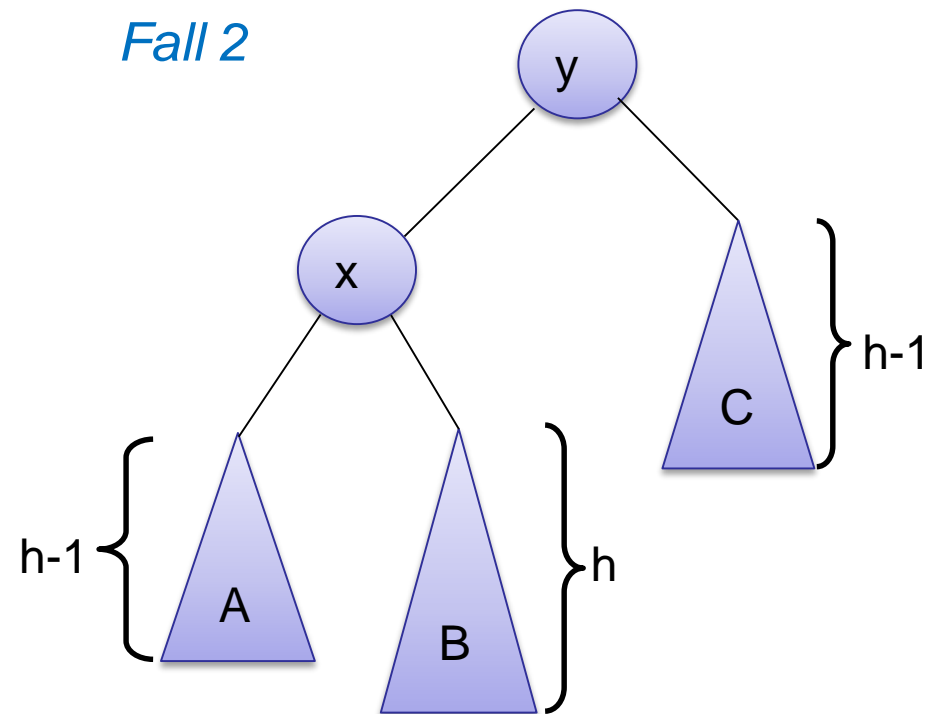
*Fall 1*



# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Fall 2: Doppelrotation

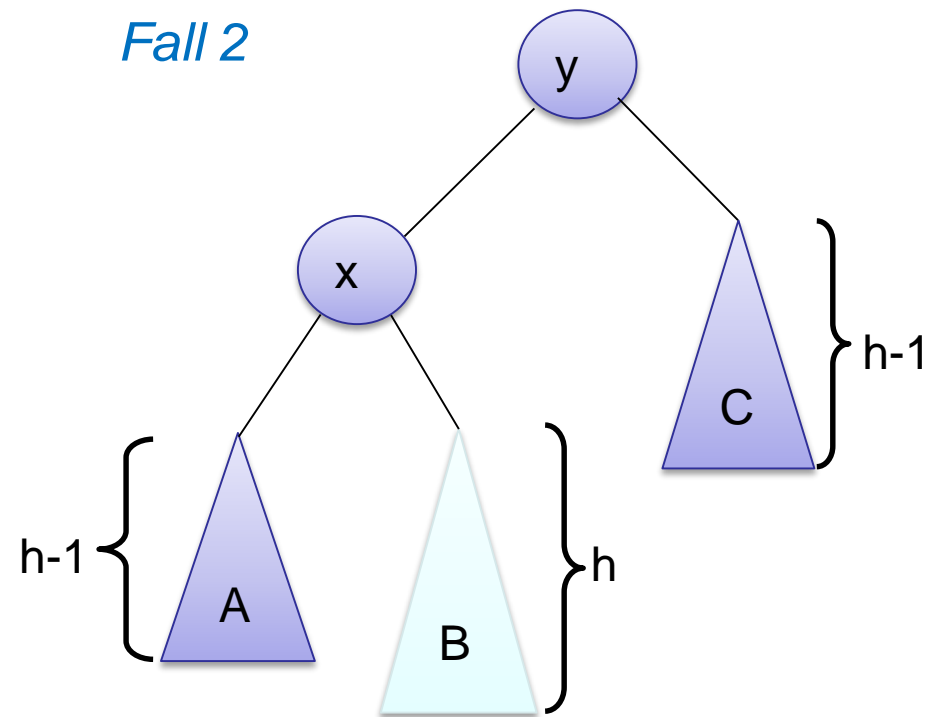
*Fall 2*



# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Fall 2: Doppelrotation

*Fall 2*

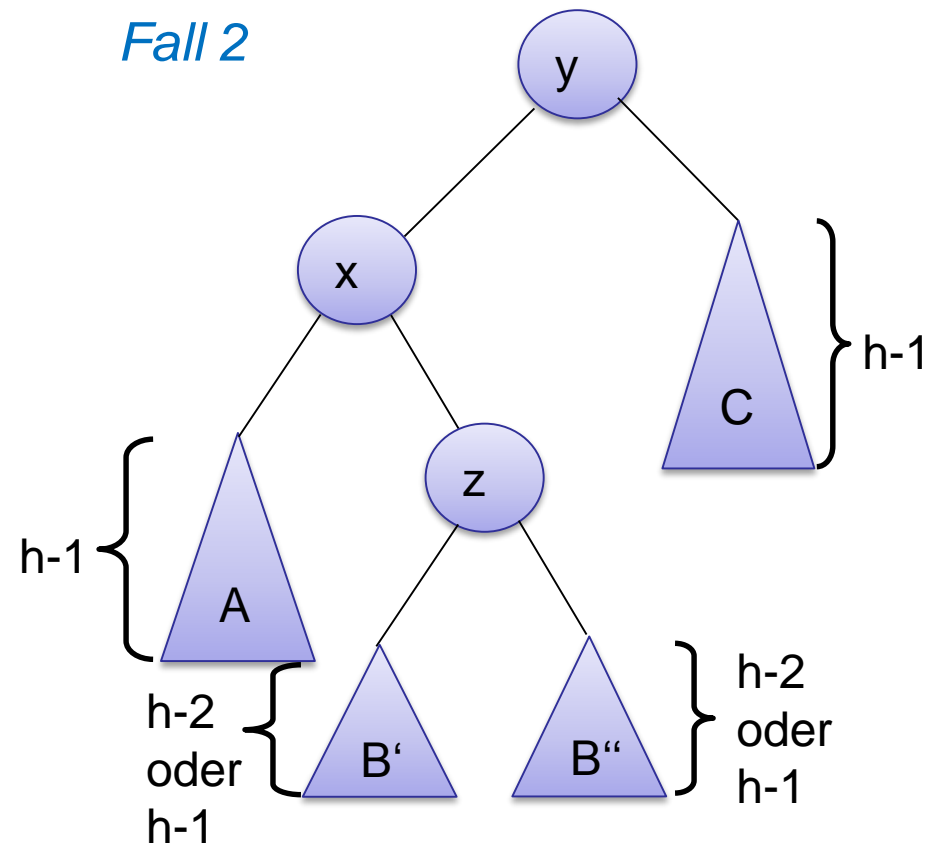




# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Fall 2: Doppelrotation

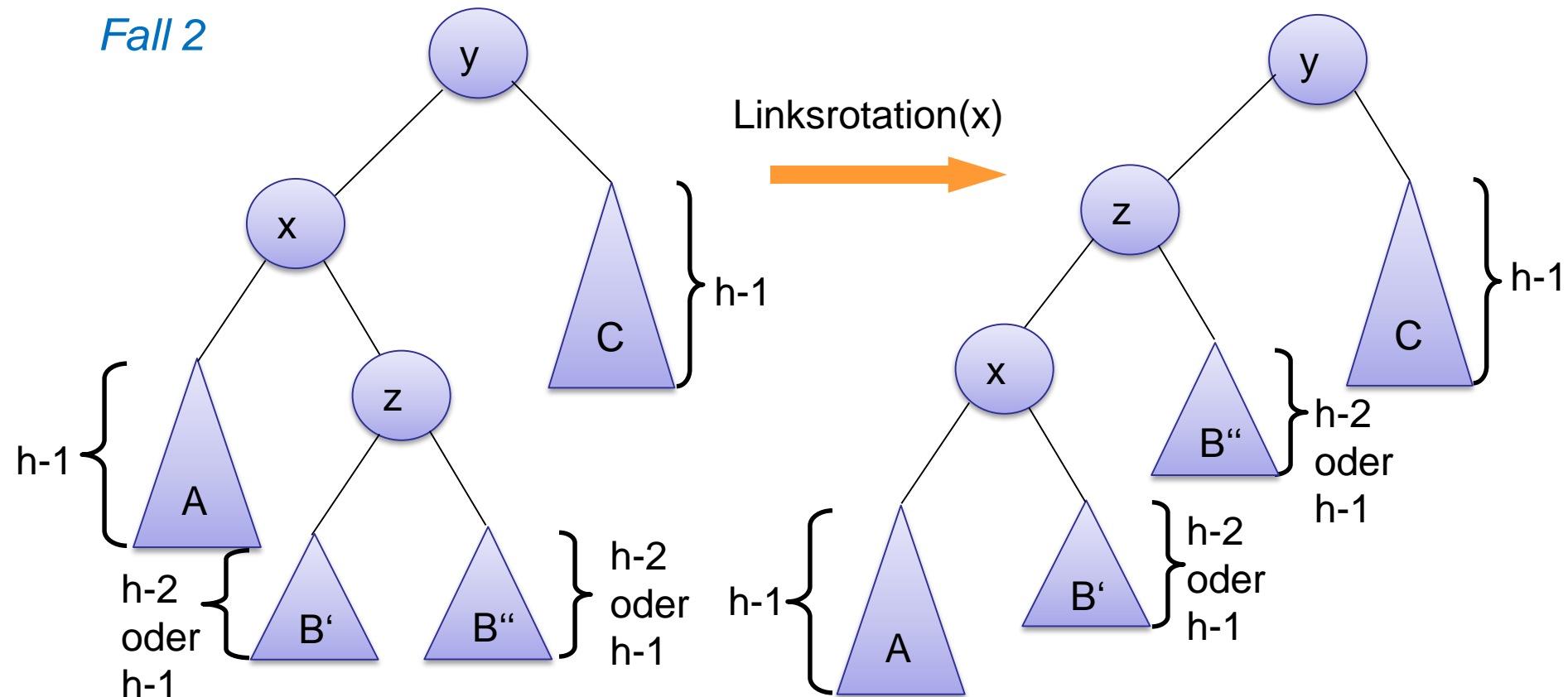
*Fall 2*



# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Fall 2: Doppelrotation

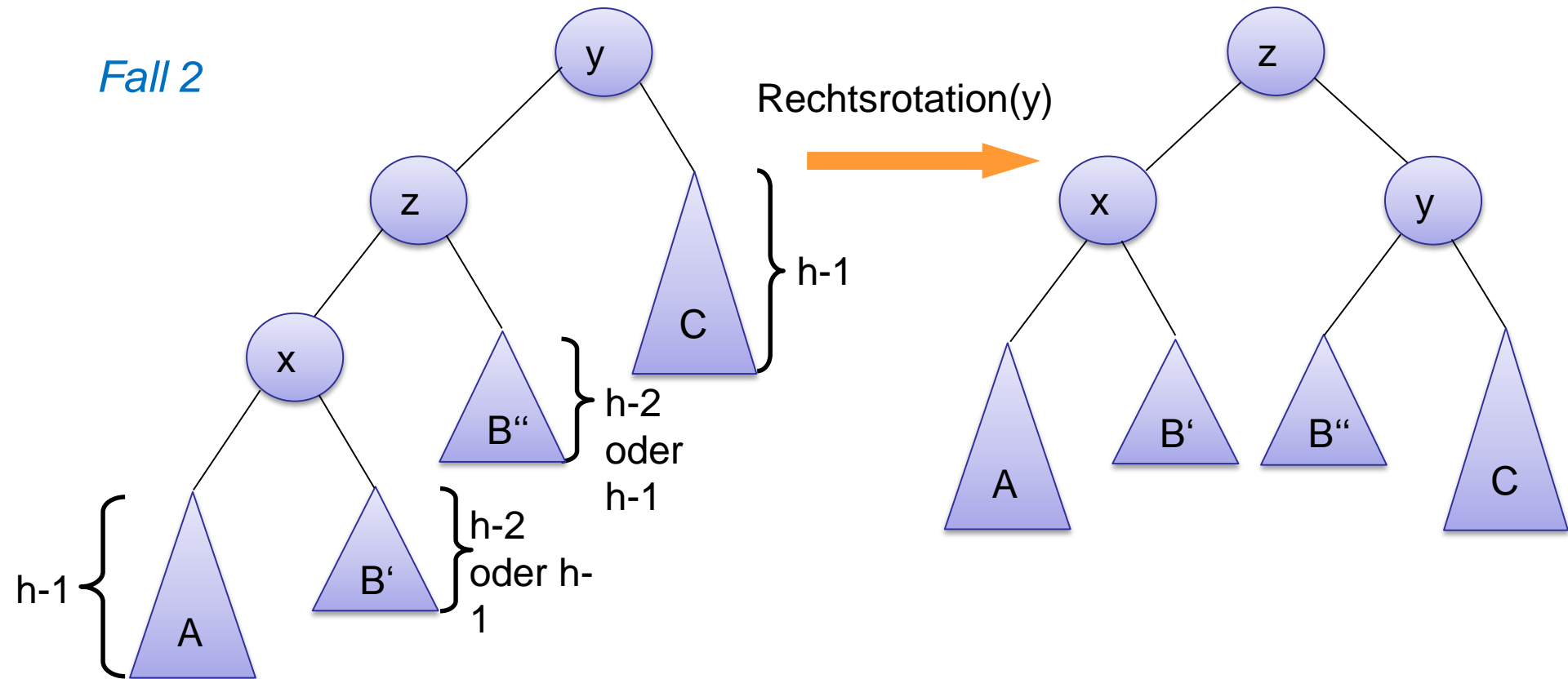
Fall 2



# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Fall 2: Doppelrotation

*Fall 2*

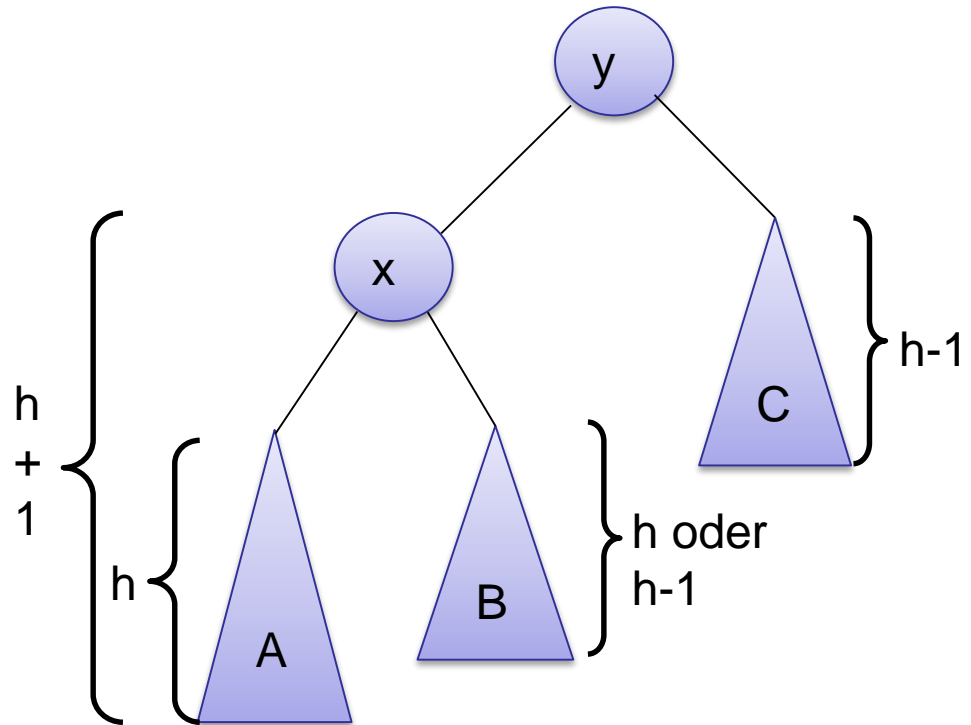


# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Operation: 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)



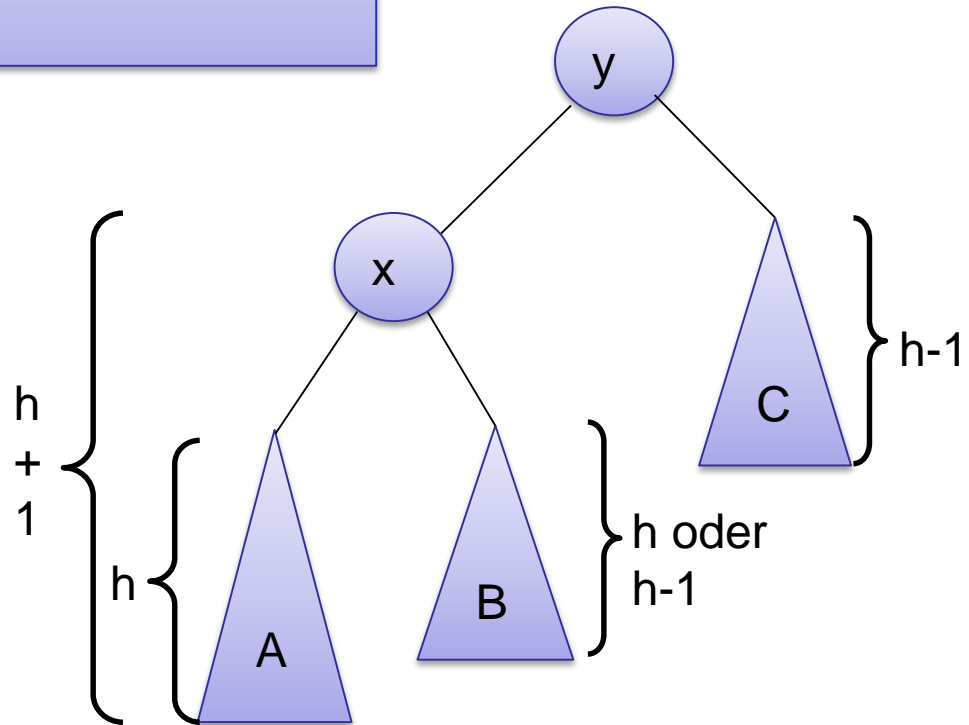
# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Operation: Balance

$h$  gibt Höhe des Teilbaums an.  
 Dies müssen wir zusätzlich in  
 unserer Datenstruktur aufrecht  
 erhalten.

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( $y$ )

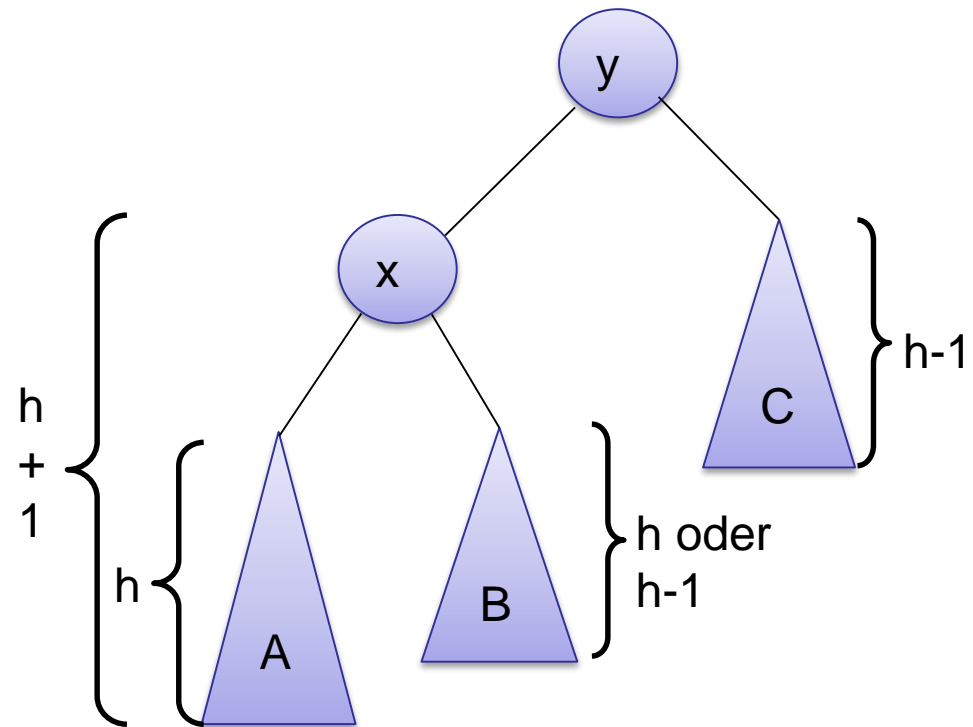


# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Operation: 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)



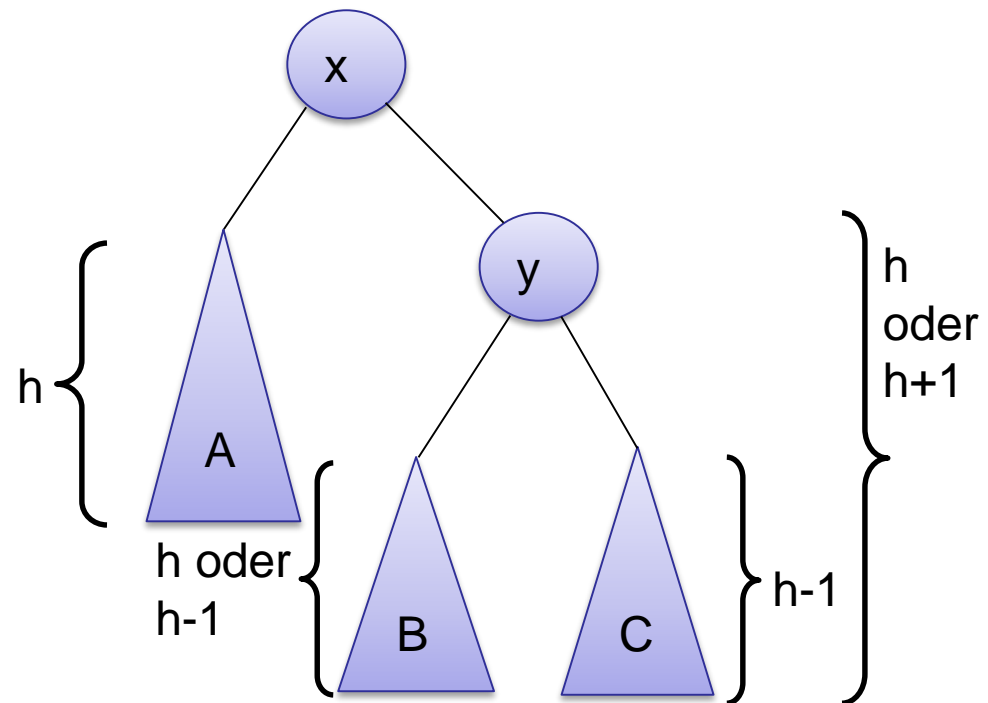
# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Operation: 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)

■ Laufzeit:  $O(1)$

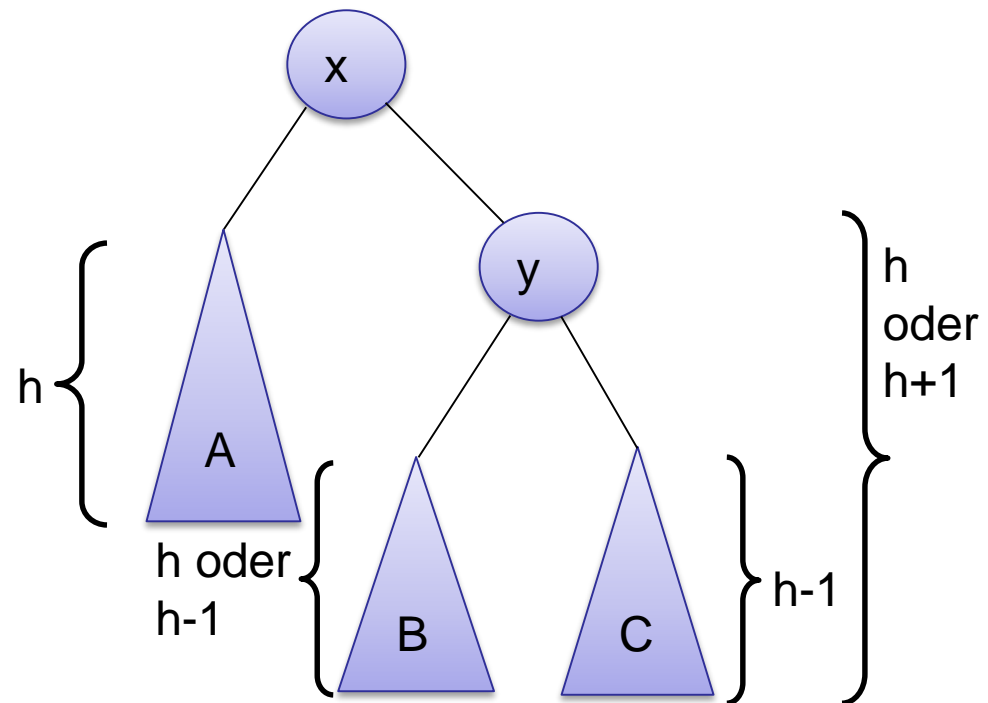


# Umformung von Beinahe-AVL-Baum zu AVL-Baum

## Operation: 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)



### Wichtiger Hinweis

- 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 mit Hilfe von maximal 2 Rotationen einen AVL-Baum machen
- Dabei erhöht sich die Höhe des Baums nicht

## *Einfügen*

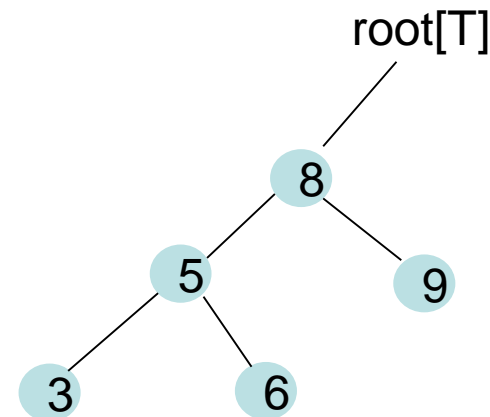
- Wir fügen ein wie früher
- Dann laufen wir den Pfad zur Wurzel zurück (z.B. als Teil der Rekursion)
- An jedem Knoten balancieren wir, falls der Unterbaum ein beinahe-AVL-Baum ist

# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

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

Wird aufgerufen mit  
AVL-Einfügen( $\text{root}[T], 2$ )

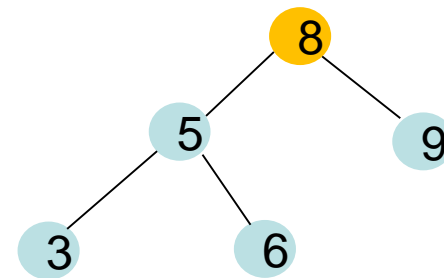


Einfügen 2

# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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( $t$ )

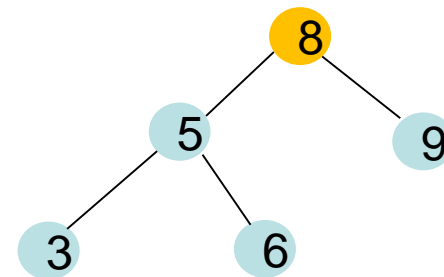


Einfügen 2

# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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( $t$ )

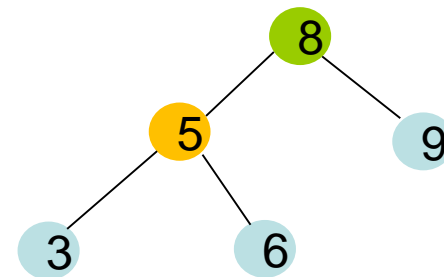


Einfügen 2

# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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( $t$ )

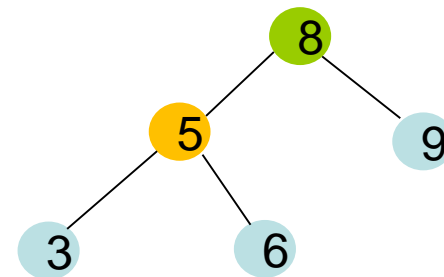


Einfügen 2

# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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

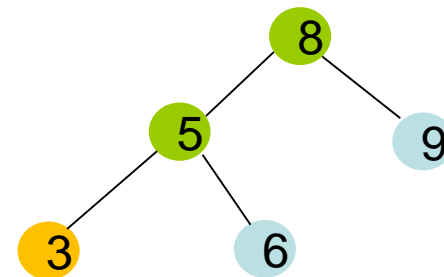
# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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( $t$ )

## Hinweis

- Eventuell muss die Wurzel des Baums ( $\text{root}[T]$ ) angepasst werden

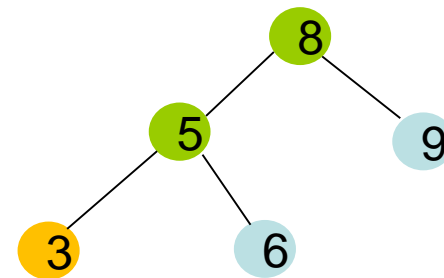


Einfügen 2

# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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( $t$ )



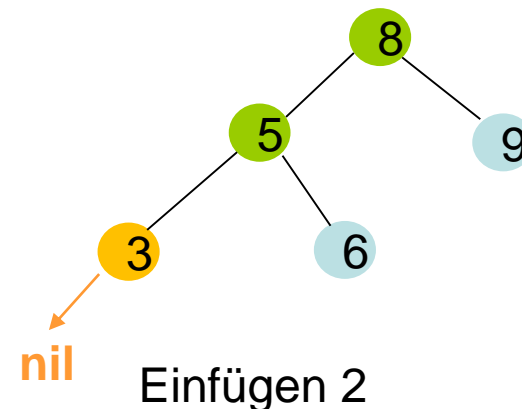
Einfügen 2



# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.      $t \leftarrow \text{new node}(x); h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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( $t$ )

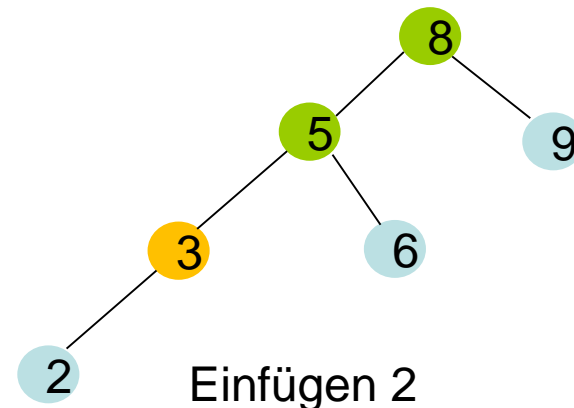
Neuen Knoten erzeugen.  
Zusätzlich noch Zeiger  
 $lc[t]$  und  $rc[t]$  auf **nil**  
setzen, sowie  $p[t]$  und  
den Zeiger von  $p[t]$   
setzen.



# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

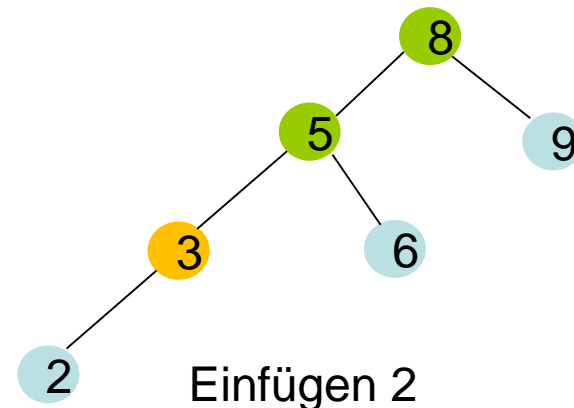
1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Einfügen( $rc[t], x$ )
5. **else return**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



# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

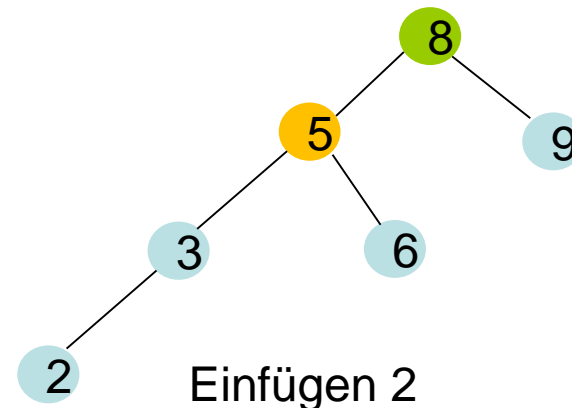
1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Einfügen( $rc[t], x$ )
5. **else return**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



# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

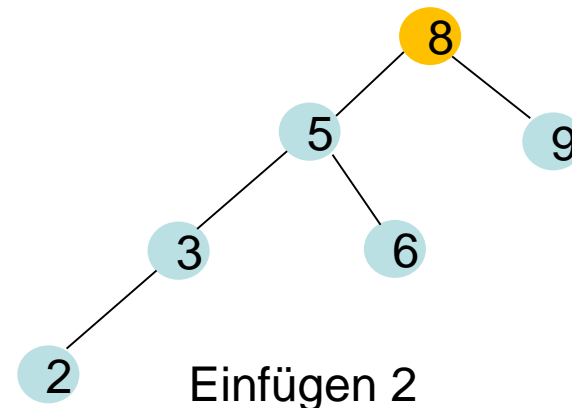
1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Einfügen( $rc[t], x$ )
5. **else return**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



# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

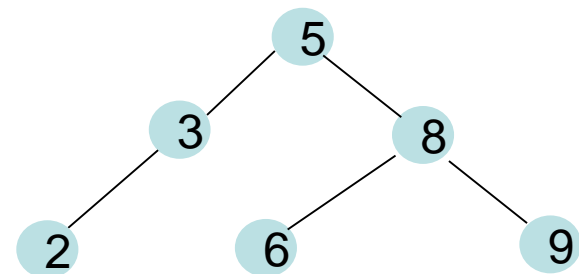
1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Einfügen( $rc[t], x$ )
5. **else return**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



# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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.  $\text{Balance}(t)$



Einfügen 2

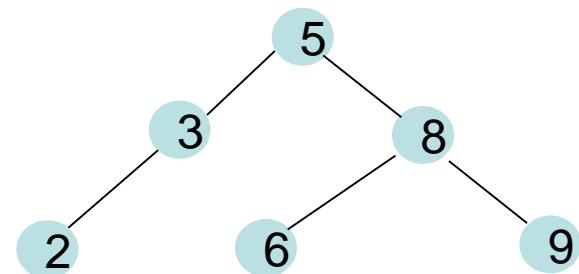
# AVL-Bäume: Einfügen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$  ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $lc[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **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( $t$ )

*Laufzeit*

- $O(h) = O(\log n)$



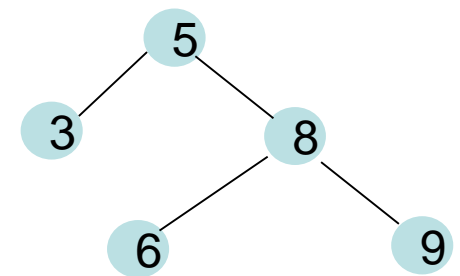
Einfügen 2



# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return** ➤  $x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere Informationen von  $u$  nach  $t$
8.     AVL-Löschen( $\text{key}[u], \text{lc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $h[t] = 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )



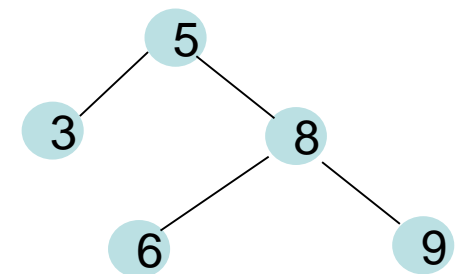
## AVL-Bäume: Löschen

x bezeichnet Schlüssel des zu löschenden Elements und t ist der Teilbaum.  
Wird aufgerufen mit AVL-Löschen(root[T], 3)

### AVL-Löschen(t,x)

1. **if** key[x]<key[t] **then** AVL-Löschen(lc[t],x)
2. **else if** key[x]>key[t] **then** AVL-Löschen(rc[t],x)
3. **else if** t=nil **then return** ➤ x nicht im Baum
4. **else if** lc[t]=nil **then** ersetze t durch rc[t]
5. **else if** rc[t]=nil **then** ersetze t durch lc[t]
6. **else** u=MaximumSuche(lc[t])
7.     Kopiere Informationen von u nach t
8.     AVL-Löschen(key[u],lc[t])
9. **if** t≠nil **then** h[t] = 1 + max{h[lc[t]], h[rc[t]]}
10. Balance(t)

Löschen(3)

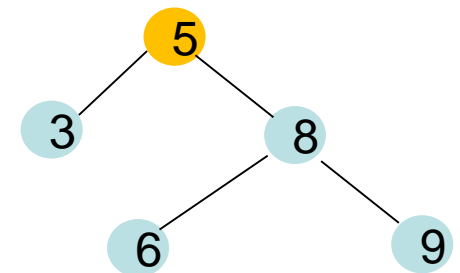


# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return** ➤  $x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere Informationen von  $u$  nach  $t$
8.     AVL-Löschen( $\text{key}[u], \text{lc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $h[t] = 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Löschen(3)

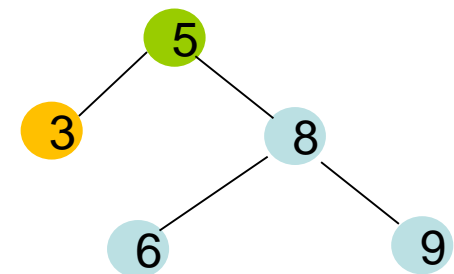


# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return** ➤  $x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere Informationen von  $u$  nach  $t$
8.     AVL-Löschen( $\text{key}[u], \text{lc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $h[t] = 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Löschen(3)



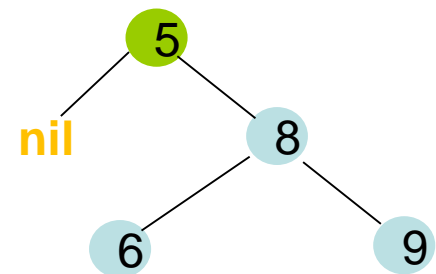
# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return**  $\triangleright x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere Informationen von  $u$  nach  $t$
8.     AVL-Löschen( $\text{key}[u], \text{lc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $h[t] = 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Und die anderen  
Zeiger aktualisieren

Löschen(3)



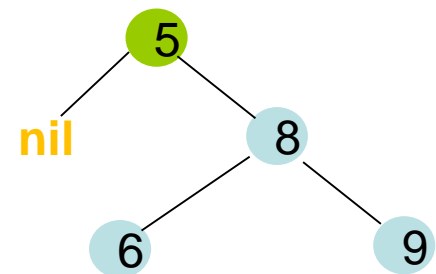
# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return** ➤  $x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere Informationen von  $u$  nach  $t$
8.     AVL-Löschen( $\text{key}[u], \text{lc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $h[t] = 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Anpassen der Höhe.

Löschen(3)

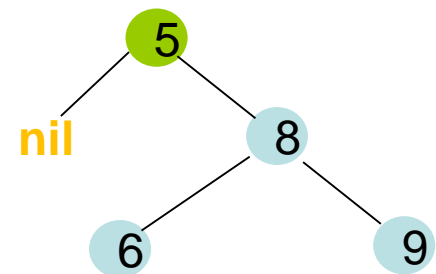


# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return** ➤  $x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere  $u$  nach  $t$ , dann nichts zu tun, dann  $u$  nach  $t$
8.     AVL-Löschen( $\text{lc}[t], x$ )
9. **if**  $t \neq \text{nil}$  **then** Balance( $t$ )
10. Balance( $t$ )

Löschen(3)

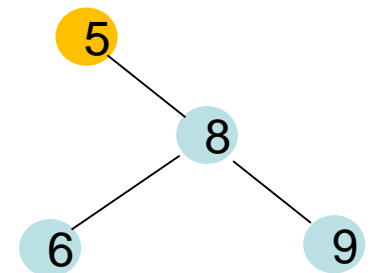


# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return** ➤  $x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere Informationen von  $u$  nach  $t$
8.     AVL-Löschen( $\text{key}[u], \text{lc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $h[t] = 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Löschen(3)



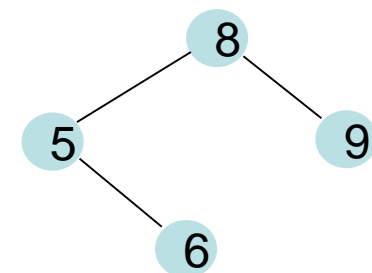


# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return** ➤  $x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere Informationen von  $u$  nach  $t$
8.     AVL-Löschen( $\text{key}[u], \text{lc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $h[t] = 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. **Balance**( $t$ )

Löschen(3)



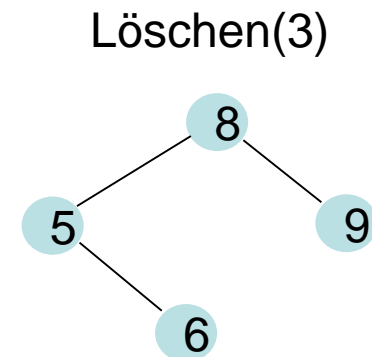
# AVL-Bäume: Löschen

AVL-Löschen( $t, x$ )

1. **if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return** ➤  $x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{rc}[t]$
5. **else if**  $\text{rc}[t] = \text{nil}$  **then** ersetze  $t$  durch  $\text{lc}[t]$
6. **else**  $u = \text{MaximumSuche}(\text{lc}[t])$
7.     Kopiere Informationen von  $u$  nach  $t$
8.     AVL-Löschen( $\text{key}[u], \text{lc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $h[t] = 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

## Hinweis

- Eventuell muss die Wurzel des Baums ( $\text{root}[T]$ ) angepasst werden



# AVL-Bäume: Beweis für Höhe

## Satz

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

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

# AVL-Bäume: Höhe – Obere Schranke

## Satz

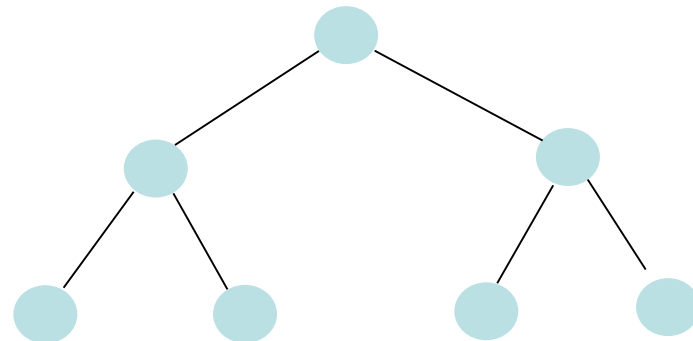
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

a)  $n \leq 2^{h+1} - 1$ :

- AVL-Baum ist Binärbaum



# AVL-Bäume: Höhe – Obere Schranke

## Satz

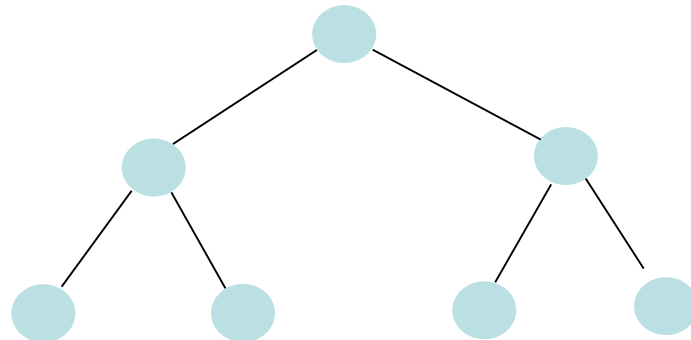
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

a)  $n \leq 2^{h+1} - 1$ :

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



# AVL-Bäume: Höhe – Obere Schranke

## Satz

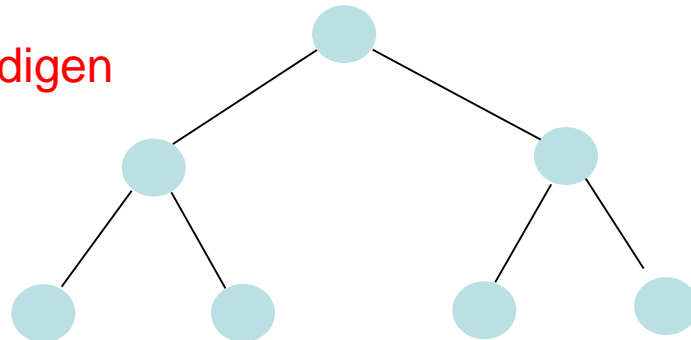
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

a)  $n \leq 2^{h+1} - 1$ :

- 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  $h$



# AVL-Bäume: Höhe – Obere Schranke

## Satz

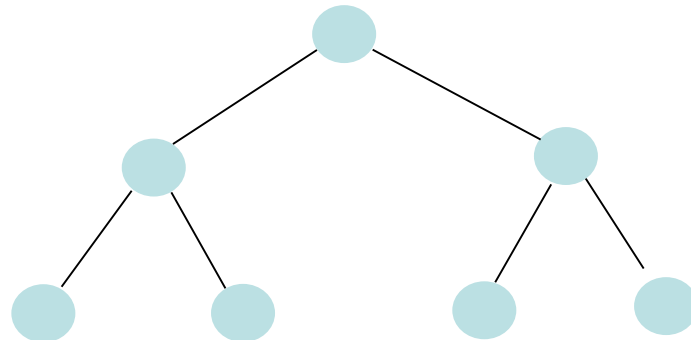
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

a)  $n \leq 2^{h+1} - 1$ :

- $N(h)$  = Anzahl Knoten eines vollständigen Binärbaums der Höhe  $h$



# AVL-Bäume: Höhe – Obere Schranke

## Satz

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

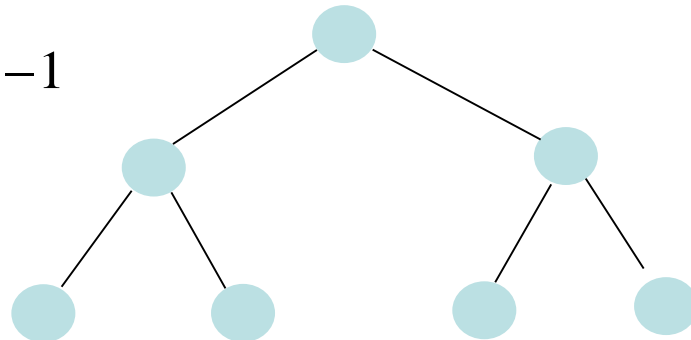
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

## Beweis

a)  $n \leq 2^{h+1} - 1$ :

- $N(h)$  = Anzahl Knoten eines vollständigen Binärbaums der Höhe  $h$

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





# AVL-Bäume: 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)  $(3/2)^h \leq n$ :

- Beweis per Induktion über die Struktur von AVL-Bäumen

# AVL-Bäume: 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)  $(3/2)^h \leq n$ :

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

# AVL-Bäume: 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)  $(3/2)^h \leq n$ :

- 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: Höhe – Untere Schranke

## Satz

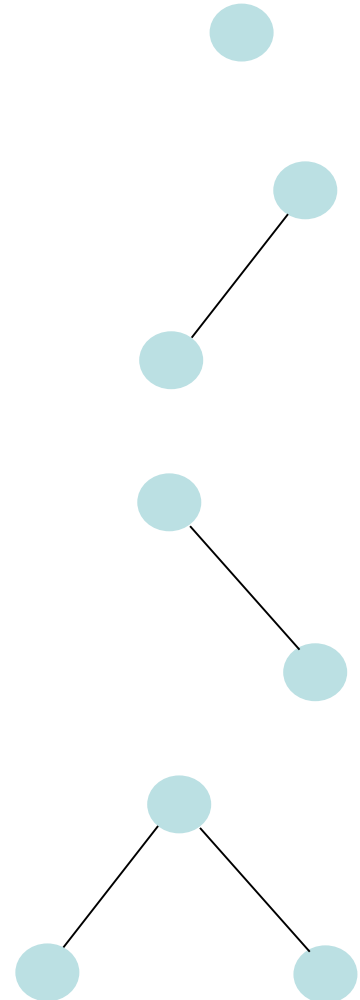
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- 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: Höhe – Untere Schranke

## Satz

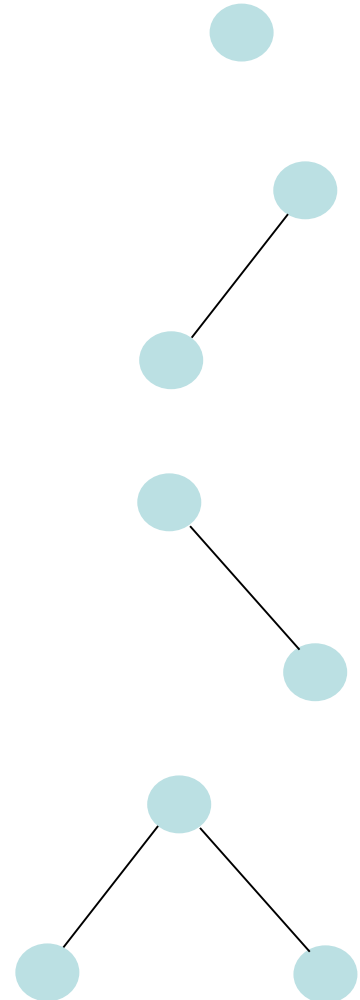
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- 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: Höhe – Untere Schranke

## Satz

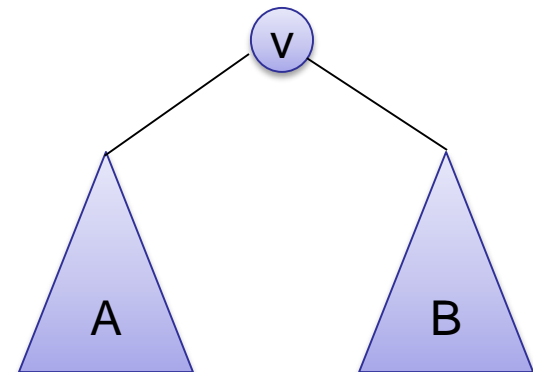
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- (I.V.) Für jeden AVL-Baum der Höhe  $j$ ,  $0 \leq j \leq h$ , gilt der Satz.



# AVL-Bäume: Höhe – Untere Schranke

## Satz

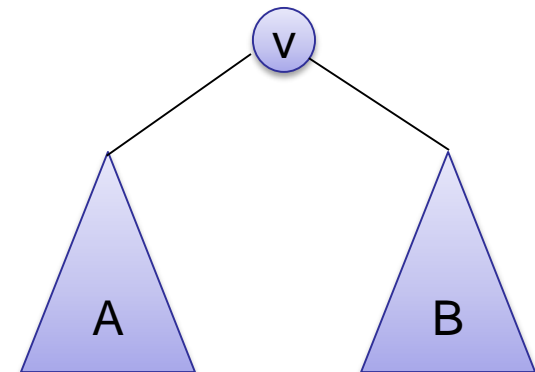
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- (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: Höhe – Untere Schranke

## Satz

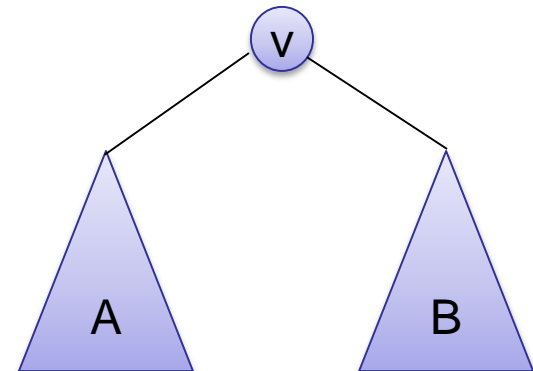
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- (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: Höhe – Untere Schranke

## Satz

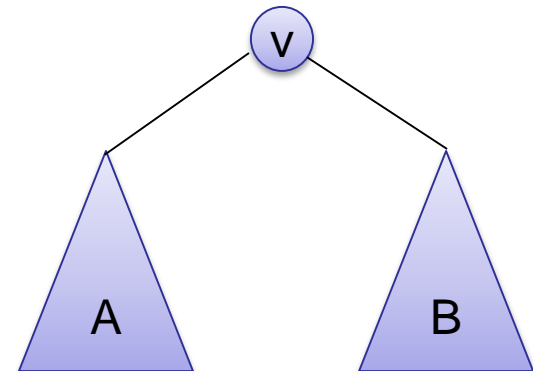
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- (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: Höhe – Untere Schranke

## Satz

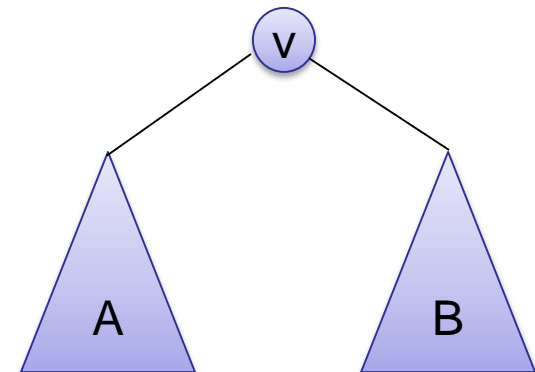
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- (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: Höhe – Untere Schranke

## Satz

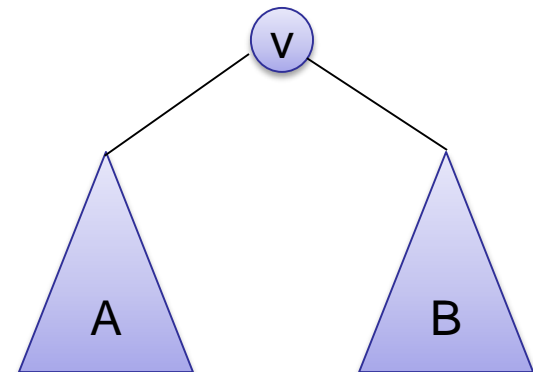
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- 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: Höhe – Untere Schranke

## Satz

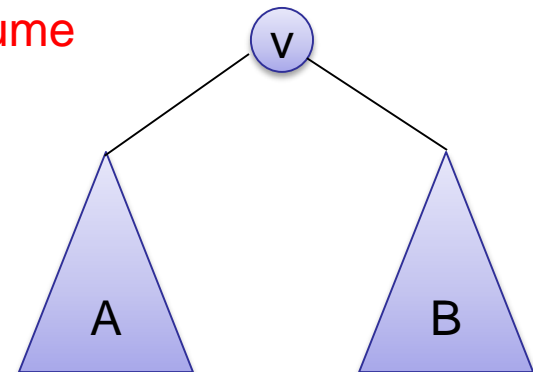
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- 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.
- Kann also (I.V.) anwenden, da A und B AVL-Bäume der Tiefe  $\geq 0$  sind



# AVL-Bäume: Höhe – Untere Schranke

## Satz

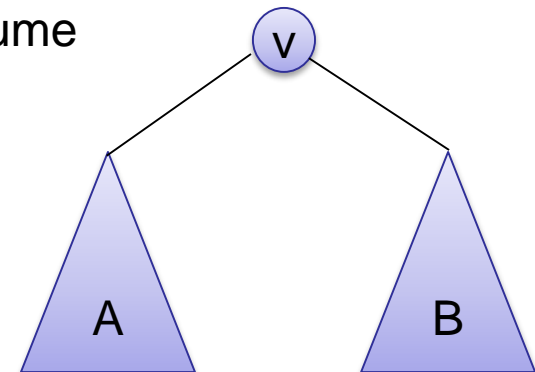
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

- 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.
- Kann also (I.V.) anwenden, da A und B AVL-Bäume der Tiefe  $\geq 0$  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: Höhe – Untere Schranke

## Satz

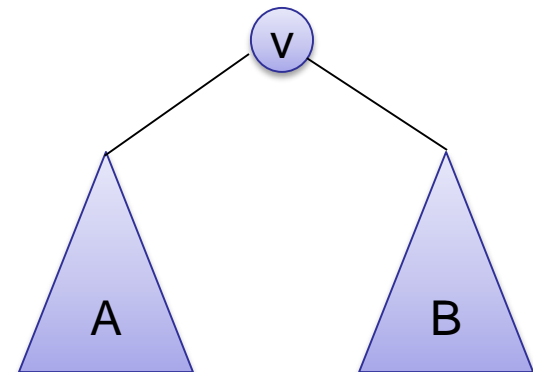
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

## Beweis

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

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



# AVL-Bäume: Höhe – Untere Schranke

## Satz

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

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

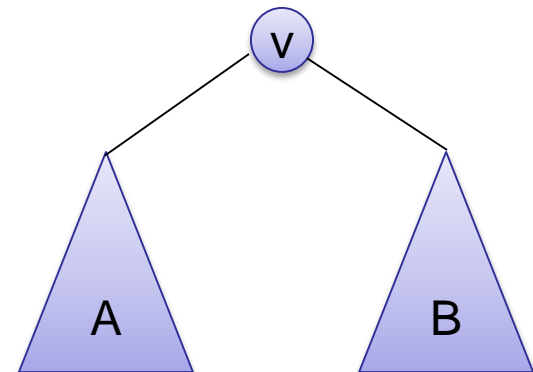
## Beweis

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

- 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: Höhe – Untere Schranke

## Satz

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

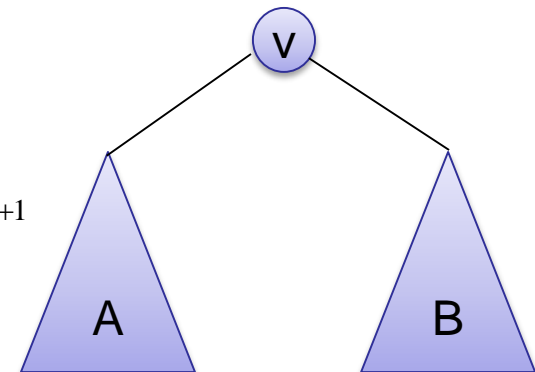
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

## Beweis

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

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

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





## Satz

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

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

## Korollar

Ein AVL-Baum mit  $n$  Knoten hat Höhe  $\Theta(\log n)$ .

# AVL-Bäume Einfügen: Beweis für Höhe

## Satz

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

AV

AVL-Einfügen( $t, x$ )

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum AVL-balanciert und die Höhe um  $h+1$ .

### Beweis

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

## AVL-Einfügen(t,x)

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum AVL-balanciert und die Höhe  $h+1$ .

### Beweis

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

## AVL-Einfügen(t,x)

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe  $h+1$ .

### Beweis

- Induktion über die Höhe des Baumes.
- (I.A.): Für Bäume der Höhe -1 und 0 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  $h+1 \geq 0$ .

## AVL-Einfügen(t,x)

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum AVL-balanciert und die Höhe  $h+1$ .

### Beweis

- Induktion über die Höhe des Baumes.
- (I.A.): Für Bäume der Höhe -1 und 0 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.  $\text{key}[x] < \text{key}[t]$  (der andere Fall ist symmetrisch).

## AVL-Einfügen(t,x)

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum ein AVL-Baum der Höhe  $h+1$ .

### Beweis

- Induktion über die Höhe des Baumes.
- (I.A.): Für Bäume der Höhe -1 und 0 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.  $\text{key}[x] < \text{key}[t]$  (der andere Fall ist symmetrisch).
- Da  $h+1 \geq 0$  ist, wird Zeile (3) ausgeführt.

## AVL-Einfügen(t,x)

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum AVL-balanciert und die Höhe  $h+1$ .

### Beweis

- Induktion über die Höhe des Baumes.
- (I.A.): Für Bäume der Höhe -1 und 0 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.  $\text{key}[x] < \text{key}[t]$  (der andere Fall ist symmetrisch).
- Da  $h+1 \geq 0$  ist, wird Zeile (3) ausgeführt.



AV

AVL-Einfügen( $t, x$ )

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum mit Höhe  $h+1$ .

### Beweis

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum ein AVL-Baum mit Höhe  $h+1$ .

### Beweis

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum ein AVL-Baum mit Höhe  $h+1$ .

### Beweis

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

## AVL-Einfügen(t,x)

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende AVL-Baum ein AVL-Baum mit Höhe  $h+1$ .

### Beweis

- Nach (I.V.) ist der Baum  $\text{lc}[t]$  nach Einfügen ein AVL-Baum mit Höhe  $r$  oder  $r+1$ , wobei  $r$  die Höhe vor dem Einfügen war.
- Hat  $\text{lc}[t]$  nach dem Einfügen Höhe  $h$  oder  $h-1$ , so ist  $t$  ein AVL-Baum.
- Hat  $\text{lc}[t]$  nach dem Einfügen Höhe  $h+1$ , so ist  $t$  u.U. ein beinahe-AVL-Baum.
- Dies wird durch in Zeile 7 durch **Balance** korrigiert.

## AVL-Einfügen(t,x)

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende AVL-Baum ein AVL-Baum mit Höhe  $h+1$ .

### Beweis

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

## AVL-Einfügen( $t, x$ )

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum ein AVL-Baum mit Höhe  $h+1$ .

### Beweis

- Nach (I.V.) ist der Baum  $\text{lc}[t]$  nach Einfügen ein AVL-Baum mit Höhe  $r$  oder  $r+1$ , wobei  $r$  die Höhe vor dem Einfügen war.
- Hat  $\text{lc}[t]$  nach dem Einfügen Höhe  $h$  oder  $h-1$ , so ist  $t$  ein AVL-Baum.
- Hat  $\text{lc}[t]$  nach dem Einfügen Höhe  $h+1$ , so ist  $t$  u.U. ein beinahe-AVL-Baum.
- Dies wird durch 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-Einfügen( $t, x$ )

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

### Satz

- Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende Baum ein AVL-Baum mit Höhe  $h+1$ .

### Beweis

- Nach (I.V.) ist der Baum  $\text{lc}[t]$  nach Einfügen ein AVL-Baum mit Höhe  $r$  oder  $r+1$ , wobei  $r$  die Höhe vor dem Einfügen war.
- Hat  $\text{lc}[t]$  nach dem Einfügen Höhe  $h$  oder  $h-1$ , so ist  $t$  ein AVL-Baum.
- Hat  $\text{lc}[t]$  nach dem Einfügen Höhe  $h+1$ , so ist  $t$  u.U. ein beinahe-AVL-Baum.
- Dies wird durch 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: Aussage für Höhe

## *Korrektheit von AVL Löschen:*

- Ähnlich wie beim Einfügen

## *Satz*

Mit Hilfe 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  $t$  Kindbäumen enthält, welche  $t$ -äre Bäume 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

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

# 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 Variante: Depth first
  - Rekursiv implementierbar
  - Iterative Implementierung: Mittels Stack
  
- Letzte Variante: Breadth first
  - Iterative Implementierung: Mittels Queue (ähnlich zu pre-order)

# Formoptimierung bei Bäumen

- Der Baum verliert seine besonderen Eigenschaften, wenn er degeneriert, d.h. kein angemessenes Breiten-Höhen-Verhältnis mehr besitzt, also z.B. zur linearen Kette wird.
  - Formgebung und –erhaltung sind essenziell für die Effizienz
- Freie Bäume:
  - Hier kann sich der Baum frei entwickeln. Beliebige, auch degenerierte Formen können entstehen

# 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

- Balancierter Baum: Länge der Pfade von der Wurzel zu jedem Blatt unterscheidet sich um maximal 1
- 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)



# 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