



## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

## Datenstrukturen

### *Problem*

- Gegeben sind  $n$  Objekte  $O_1, \dots, O_n$  mit zugehörigen Schlüsseln  $s(O_i)$

### *Operationen*

- **Suche( $x$ )**; Ausgabe  $O$  mit Schlüssel  $s(O) = x$ ;  
**nil**, falls kein Objekt mit Schlüssel  $x$  in Datenbank
- **Einfügen( $O$ )**; Einfügen von Objekt  $O$  in Datenbank
- **Löschen( $O$ )**; Löschen von Objekt  $O$  mit aus der Datenbank

## Datenstrukturen

### *Binäre Suchbäume*

- Ausgabe aller Elemente in  $\mathbf{O}(n)$
- Suche, Minimum, Maximum, Nachfolger in  $\mathbf{O}(h)$
- Einfügen, Löschen in  $\mathbf{O}(h)$

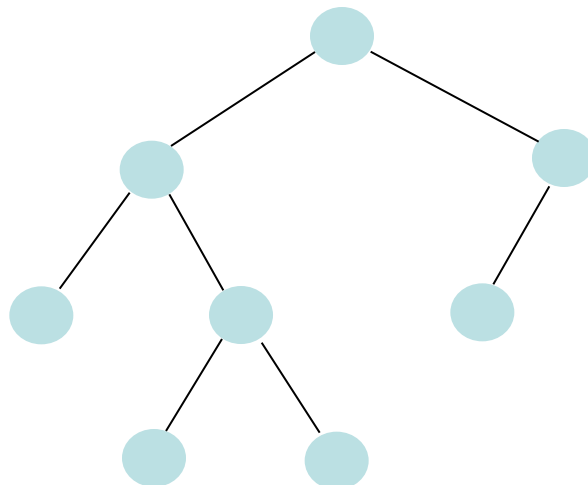
### *Frage*

Wie kann man eine „kleine“ Höhe unter Einfügen und Löschen garantieren?

## Datenstrukturen

### *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.



## Datenstrukturen

### Satz 34

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

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

## Datenstrukturen

### Satz 34

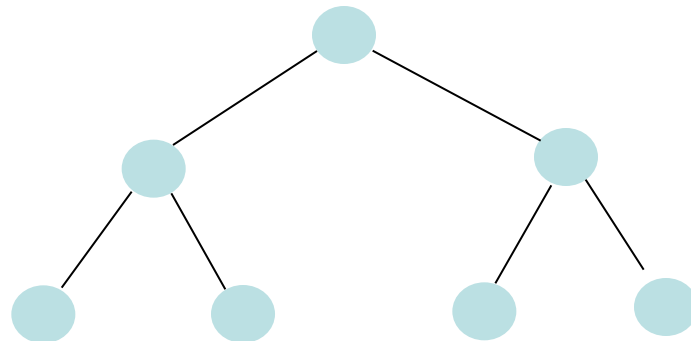
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### Beweis

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

- AVL-Baum ist Binärbaum



## Datenstrukturen

### Satz 34

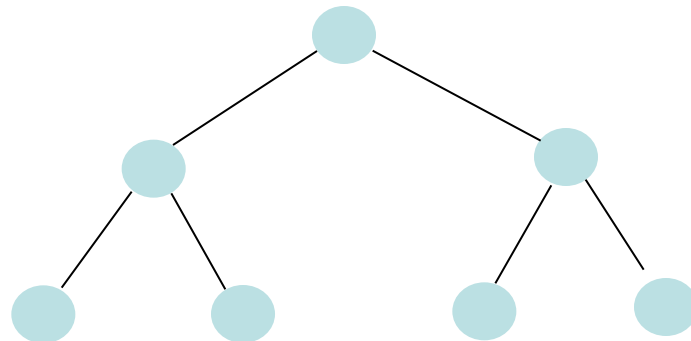
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### 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  $h$



## Datenstrukturen

### Satz 34

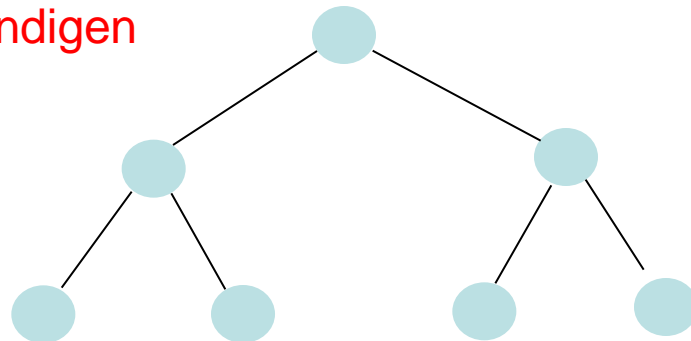
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### 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  $h$





## Datenstrukturen

### Satz 34

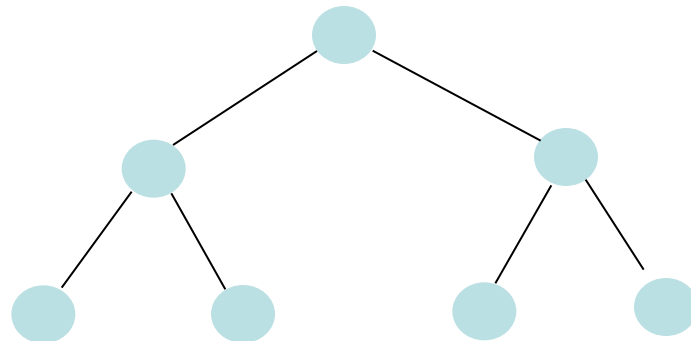
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### Beweis

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

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



## Datenstrukturen

### Satz 34

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

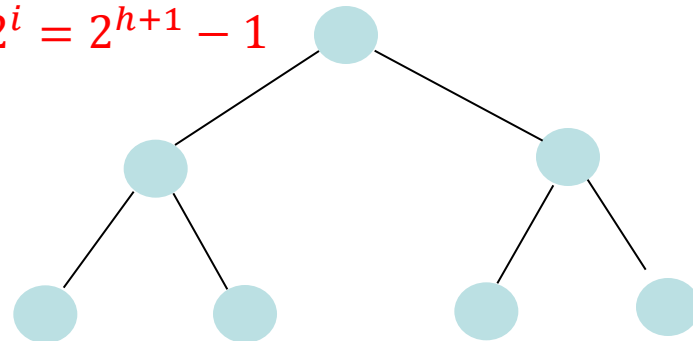
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

### Beweis

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

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

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



## Datenstrukturen

### Satz 34

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

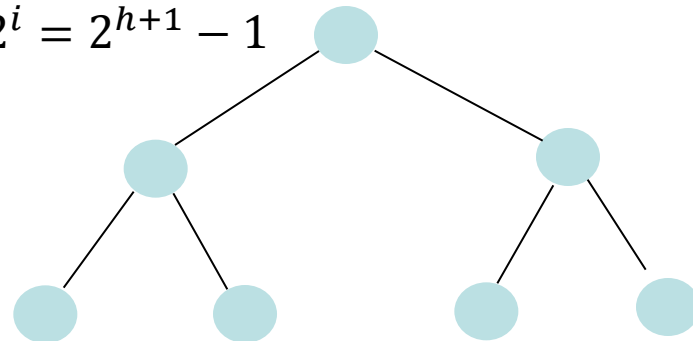
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

### Beweis

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

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

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



## Datenstrukturen

### Satz 34

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

## Datenstrukturen

### Satz 34

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.

## Datenstrukturen



### Satz 34

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  $(3/2)^h = 1 \leq 1$ .

## Datenstrukturen

### Satz 34

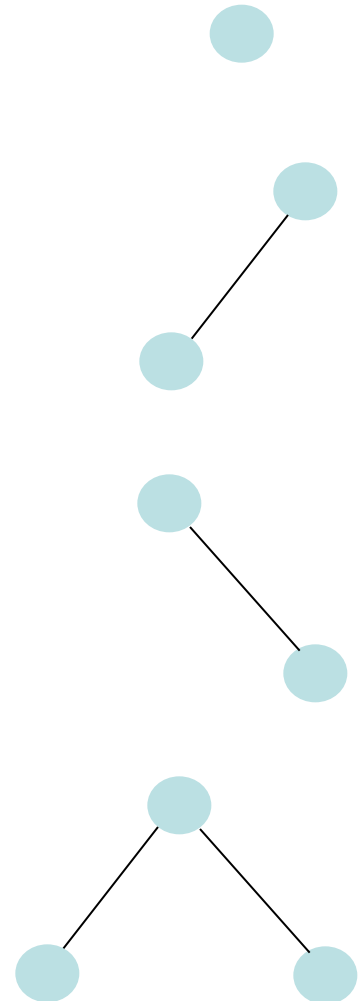
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### 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  $(3/2)^h = 3/2 \leq 2 \leq 3$ .



## Datenstrukturen

### Satz 34

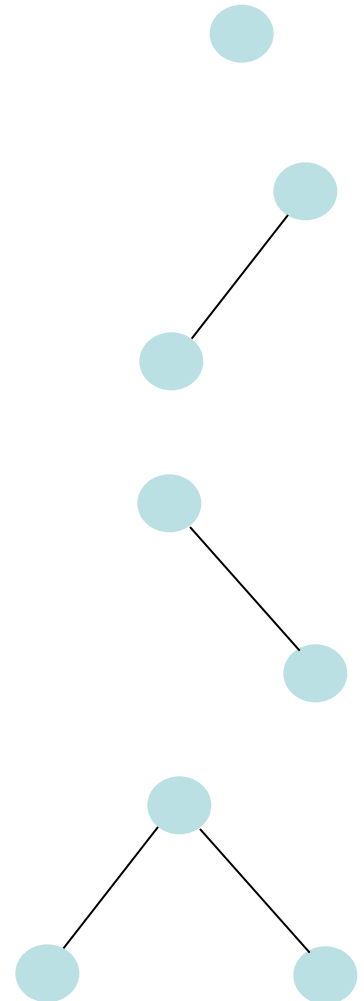
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### 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  $(3/2)^h = 3/2 \leq 2 \leq 3$ .





## Datenstrukturen

### Satz 34

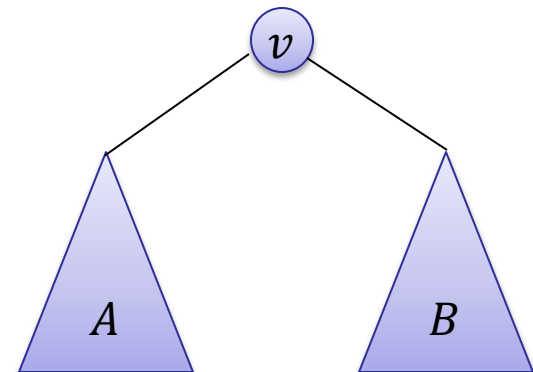
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### Beweis

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

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



## Datenstrukturen

### Satz 34

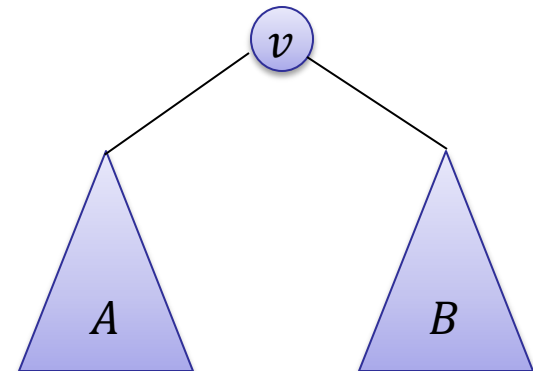
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### 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  $v$ .



## Datenstrukturen

### Satz 34

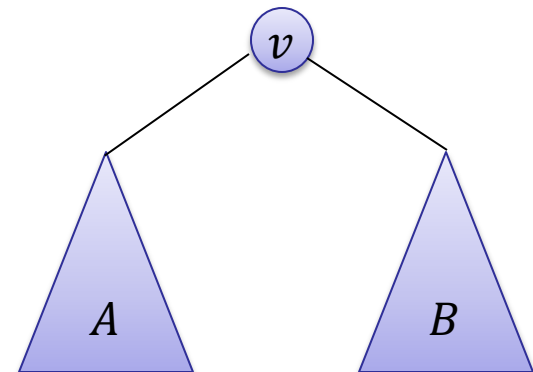
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### 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 Höhe  $h$ .**



## Datenstrukturen

### Satz 34

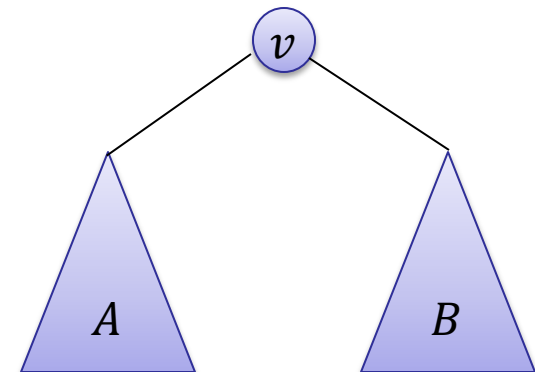
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

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



## Datenstrukturen

### Satz 34

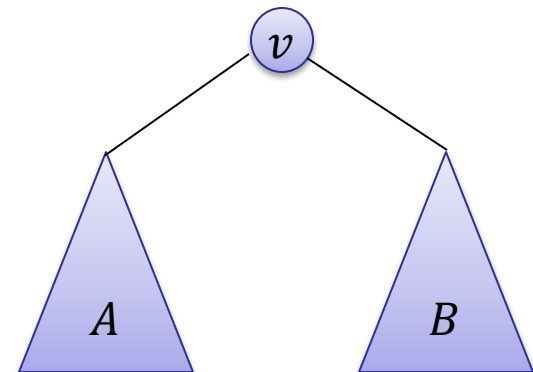
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

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



## Datenstrukturen

### Satz 34

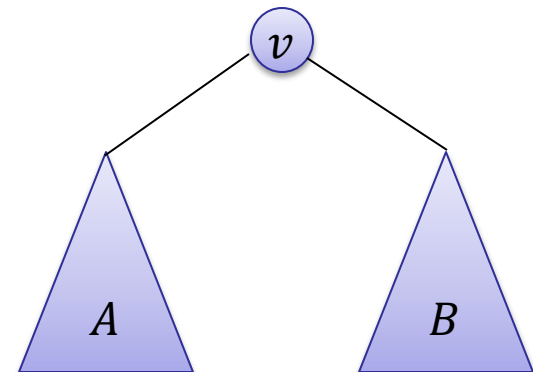
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### Beweis

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

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



## Datenstrukturen

### Satz 34

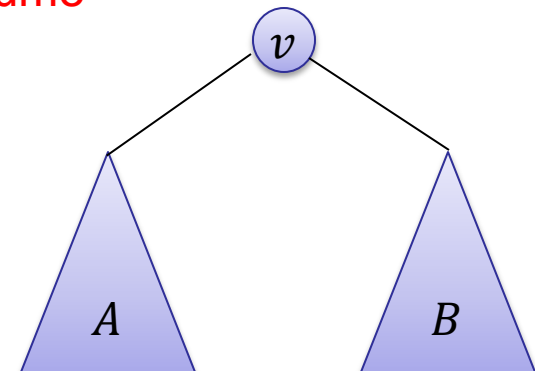
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### Beweis

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

- Wegen AVL-Eigenschaft haben  $A$  und  $B$  Höhe 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 Höhe  $\geq 0$  sind



## Datenstrukturen

### Satz 34

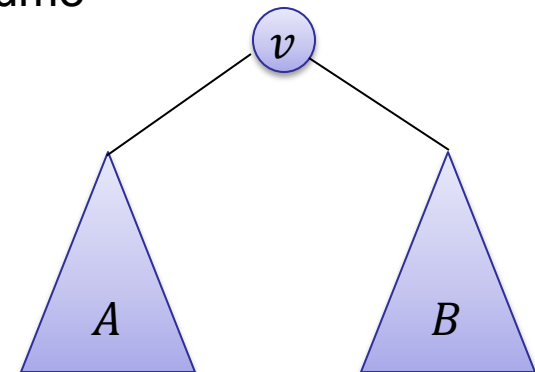
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### Beweis

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

- Wegen AVL-Eigenschaft haben  $A$  und  $B$  Höhe 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 Höhe  $\geq 0$  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  $h$





## Datenstrukturen

### Satz 34

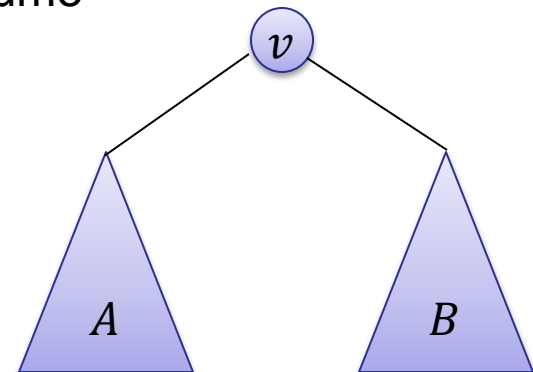
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### Beweis

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

- Wegen AVL-Eigenschaft haben  $A$  und  $B$  Höhe 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 Höhe  $\geq 0$  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  $h$



## Datenstrukturen

### Satz 34

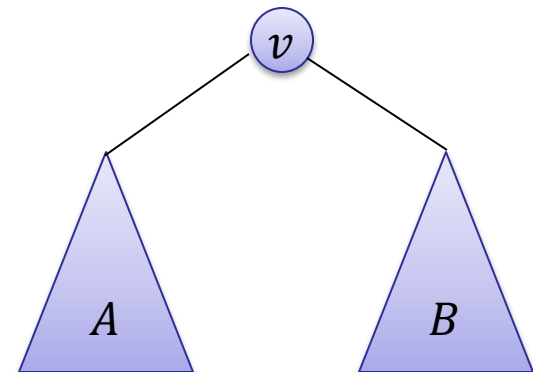
Für jeden AVL-Baum der Höhe  $h \geq 0$  mit  $n$  Knoten gilt:

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

### Beweis

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

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



## Datenstrukturen

### Satz 34

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

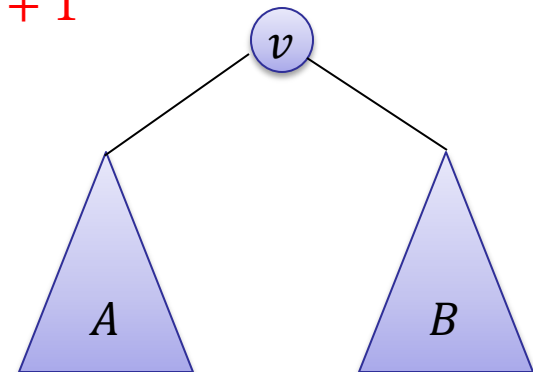
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

### Beweis

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

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

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



## Datenstrukturen

### Satz 34

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

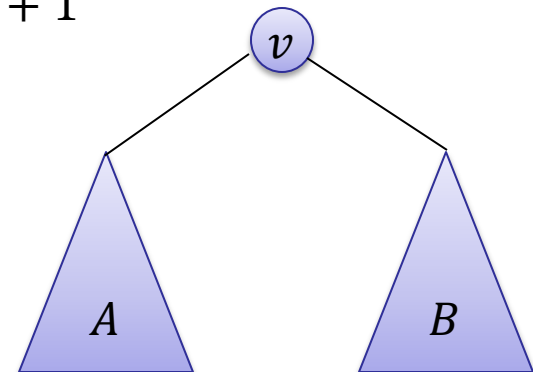
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

### Beweis

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

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

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



## Datenstrukturen

### Satz 34

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

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

### Korollar 35

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

## Datenstrukturen

### Satz 34

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

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

### Korollar 35

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

### Beweis

(1) Zeige  $h = \Theta(\log n)$ : Es gilt  $n \geq \left(\frac{3}{2}\right)^h$  nach Satz 34

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

## Datenstrukturen

### Satz 34

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

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

### Korollar 35

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

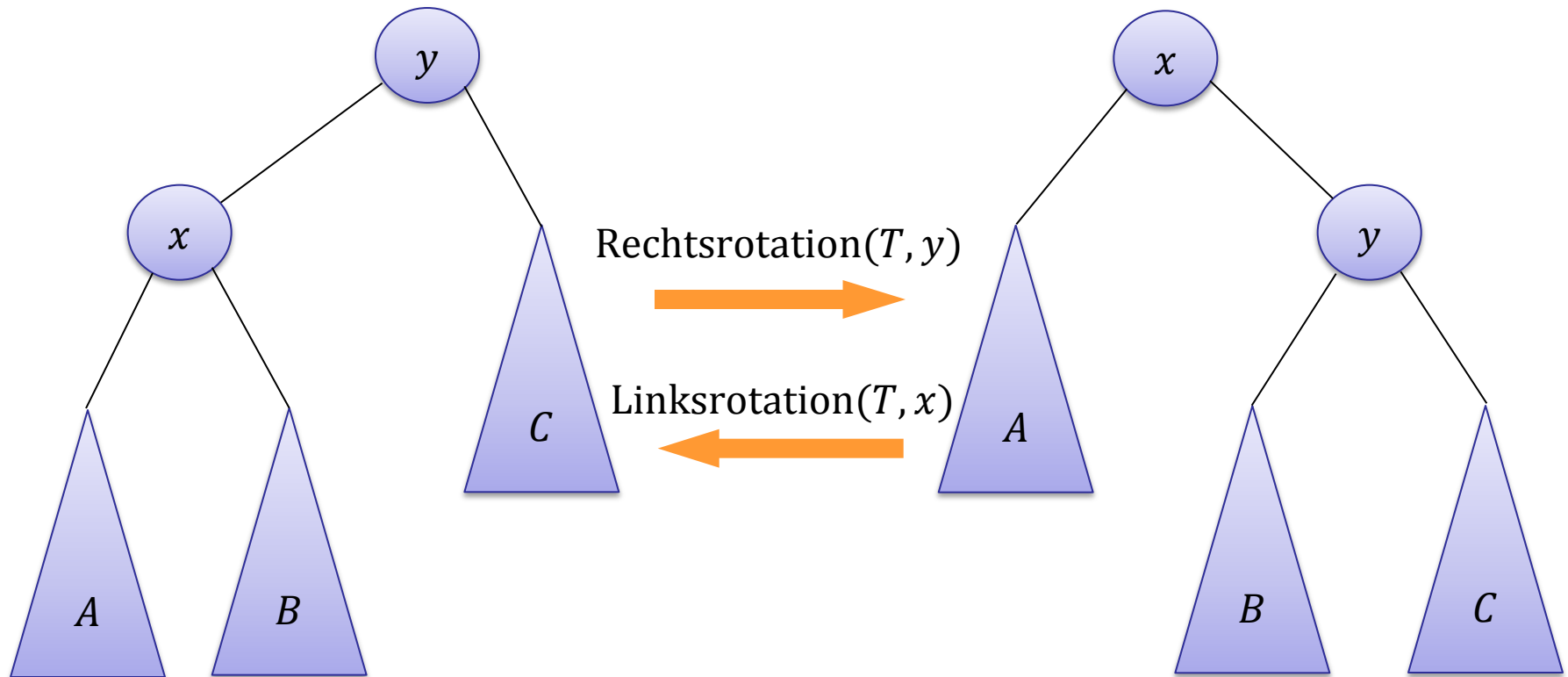
### Beweis

(2) Zeige  $h = \Omega(\log n)$ : Es gilt  $n \leq 2^{h+1} - 1 \leq 2^{h+1}$  nach Satz 34

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

## Datenstrukturen

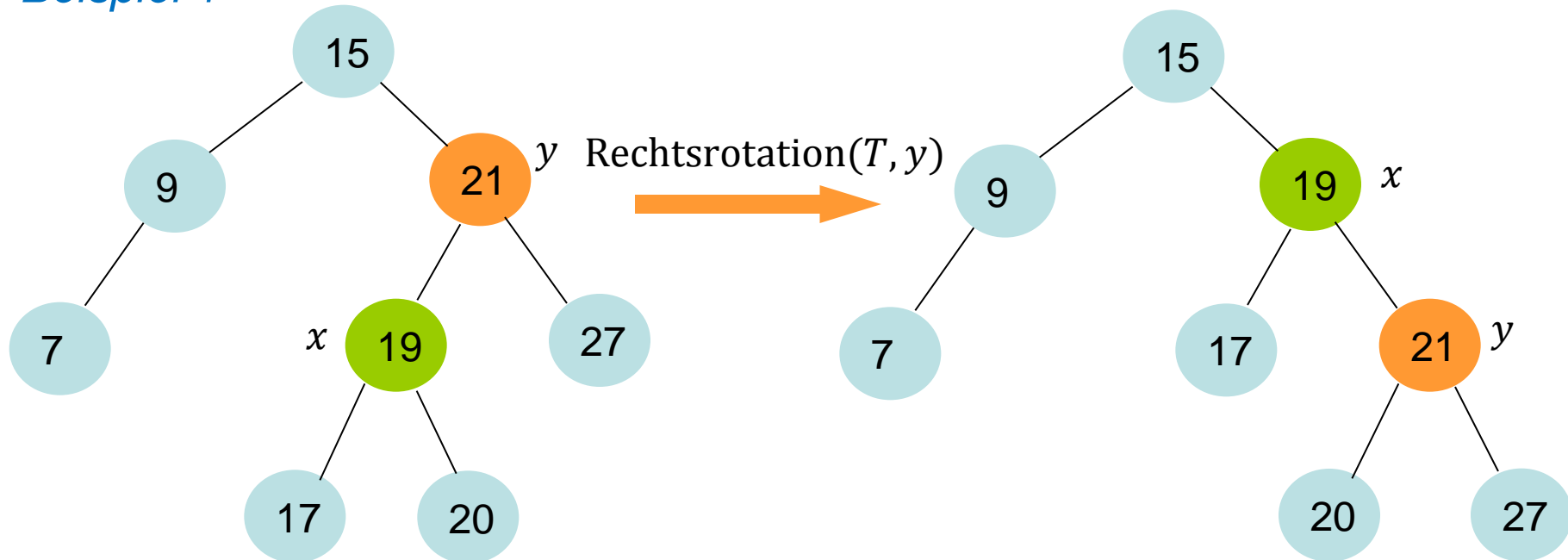
### Rotationen





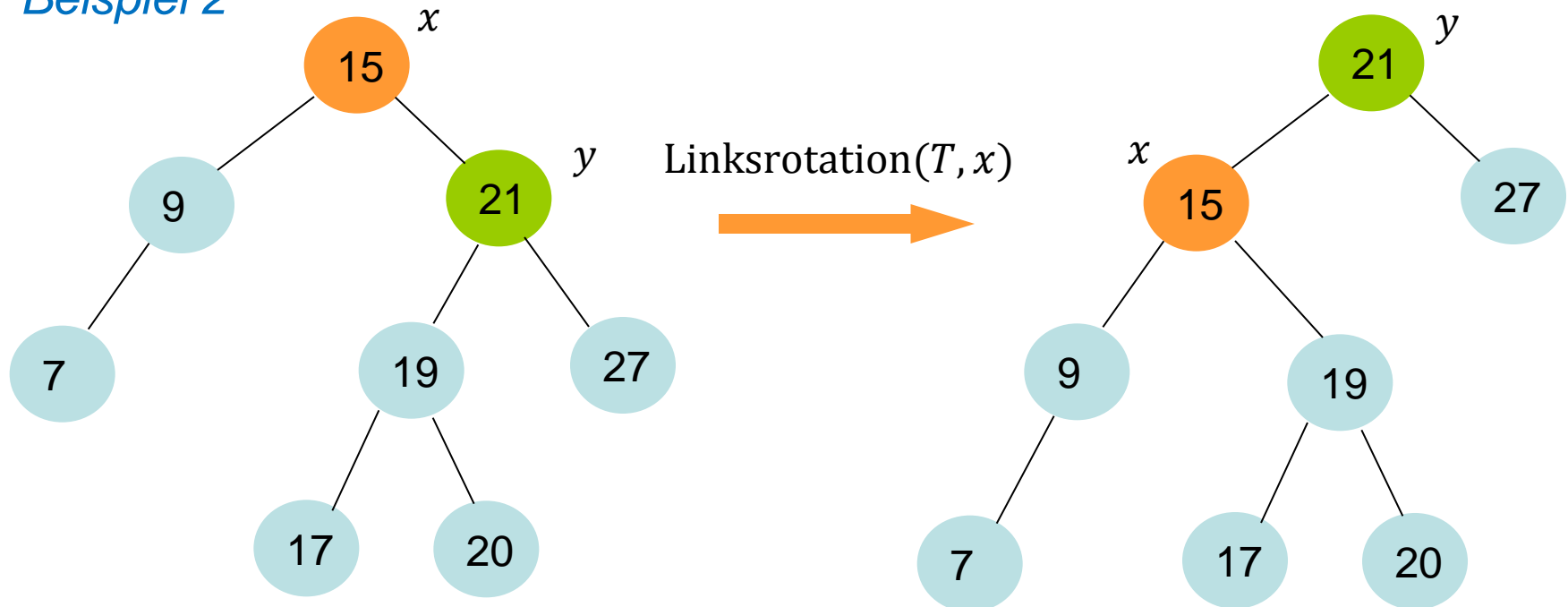
## Datenstrukturen

### Beispiel 1



## Datenstrukturen

### Beispiel 2



## Datenstrukturen

Linksrotation( $T, x$ )

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

## Datenstrukturen

Linksrotation( $T, x$ )

Annahme:  $x$  hat  
rechtes Kind.

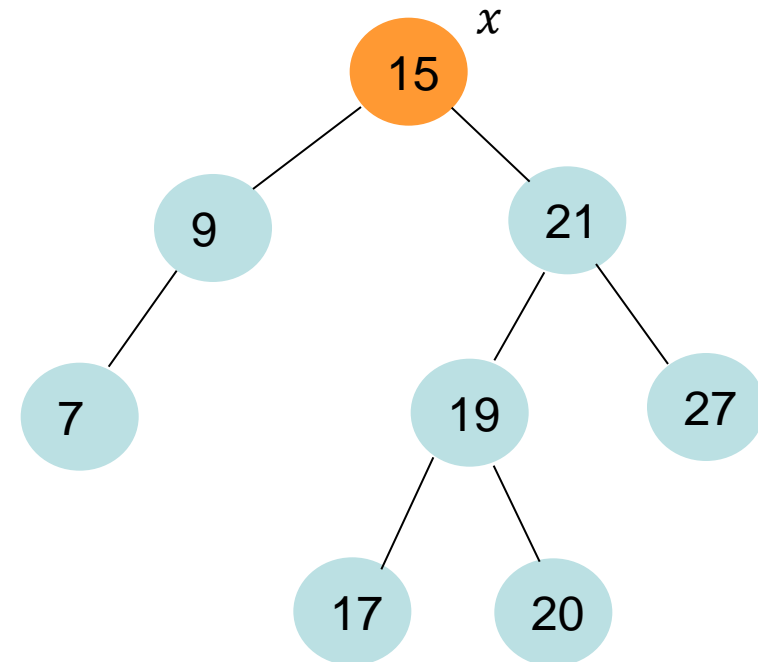
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \mathbf{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \mathbf{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7.     **else**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

## Datenstrukturen

### Linksrotation( $T, x$ )

Annahme:  $x$  hat  
rechtes Kind.

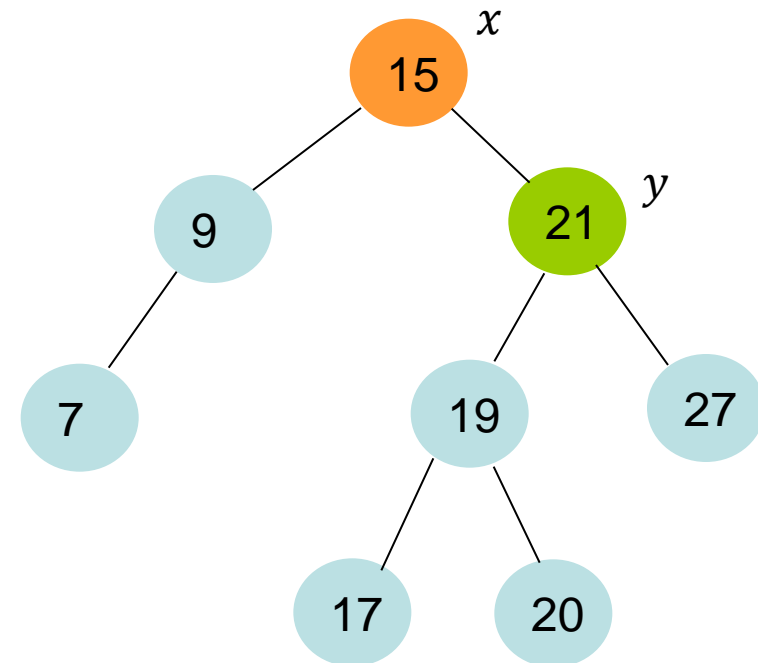
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \mathbf{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \mathbf{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7.     **else**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



## Datenstrukturen

Linksrotation( $T, x$ )

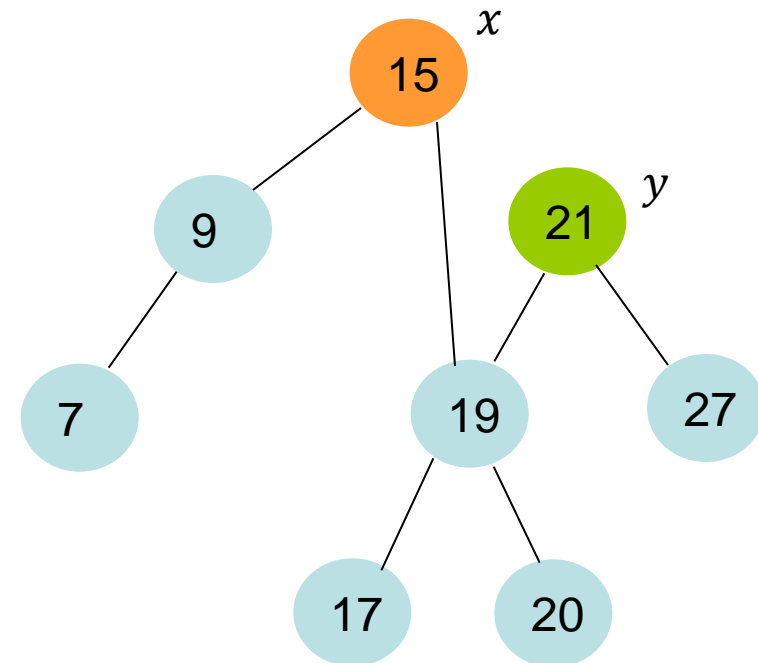
1.  $y \leftarrow \text{rc}[x]$
2.  $\text{rc}[x] \leftarrow \text{lc}[y]$
3. **if**  $\text{lc}[y] \neq \text{nil}$  **then**  $p[\text{lc}[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \text{nil}$  **then**  $\text{root}[T] \leftarrow y$
6. **else if**  $x = \text{lc}[p[x]]$  **then**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



## Datenstrukturen

Linksrotation( $T, x$ )

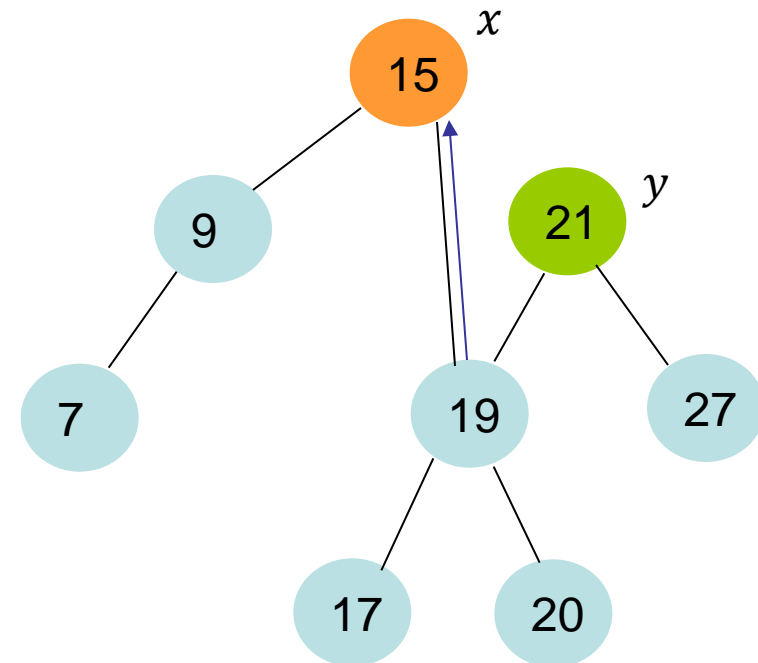
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \mathbf{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \mathbf{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7. **else**  $rc[p[x]]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



## Datenstrukturen

Linksrotation( $T, x$ )

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.  $p[x] \leftarrow y$

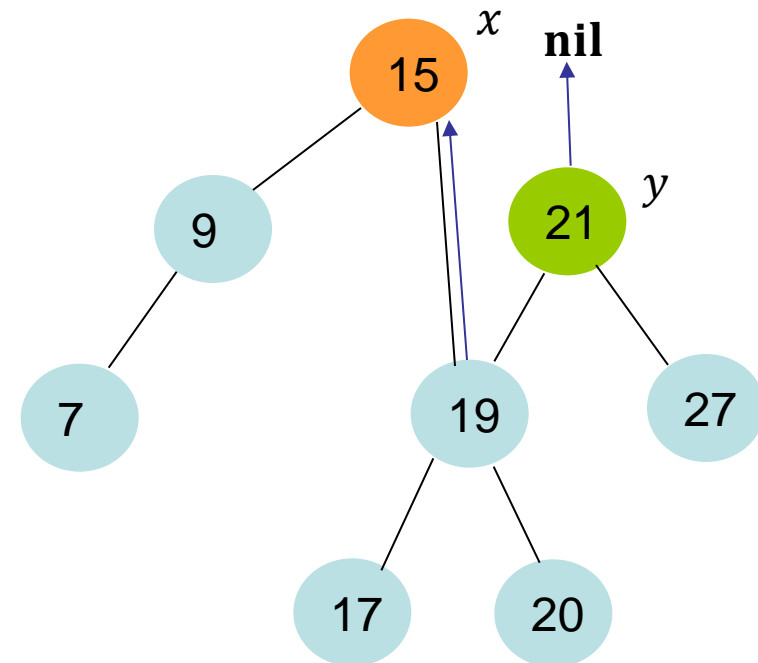




## Datenstrukturen

Linksrotation( $T, x$ )

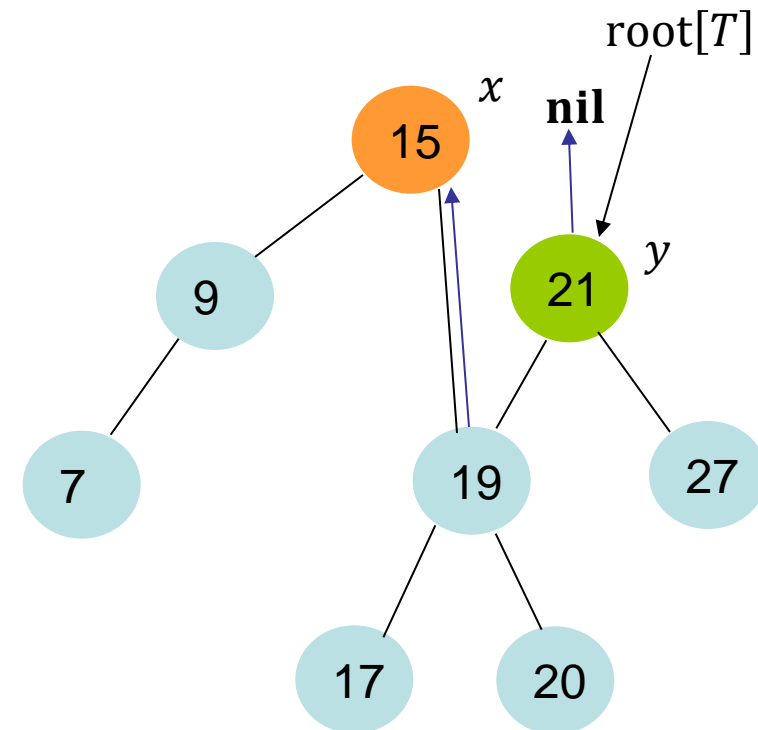
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \mathbf{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \mathbf{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7.     **else**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



## Datenstrukturen

Linksrotation( $T, x$ )

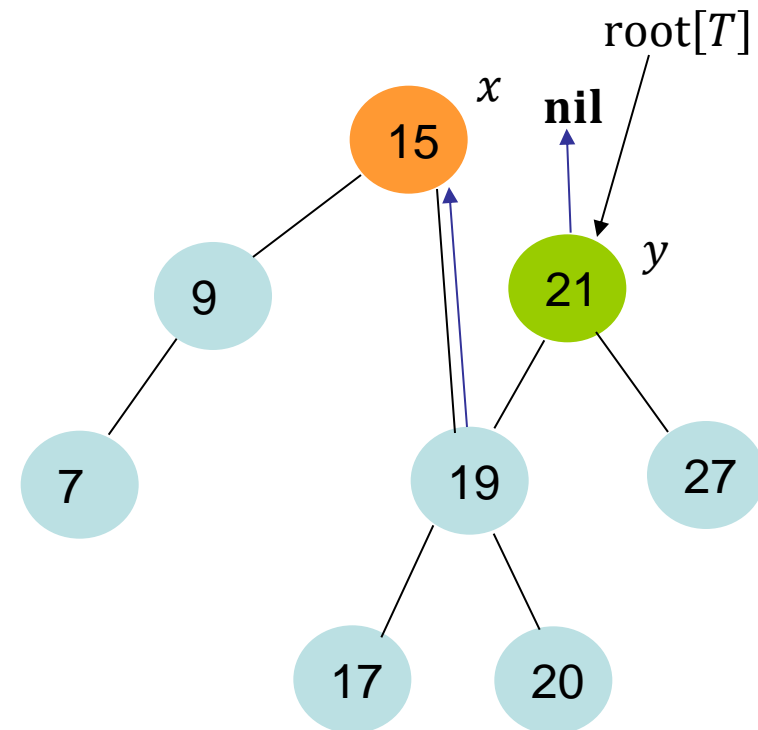
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \mathbf{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \mathbf{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7.     **else**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



## Datenstrukturen

Linksrotation( $T, x$ )

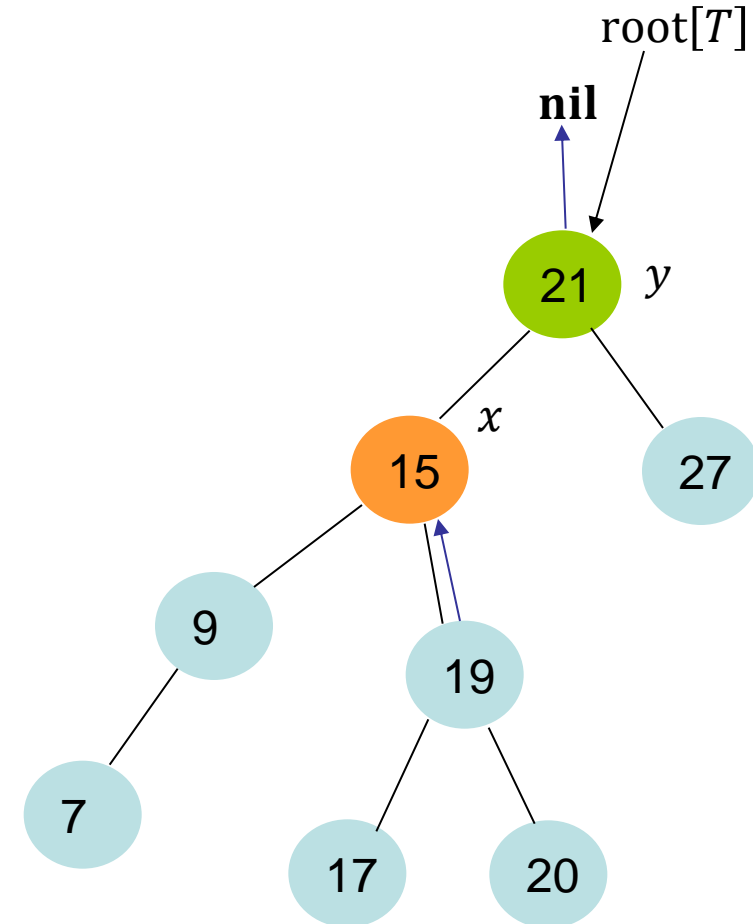
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \mathbf{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \mathbf{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7. **else**  $rc[p[x]]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



## Datenstrukturen

Linksrotation( $T, x$ )

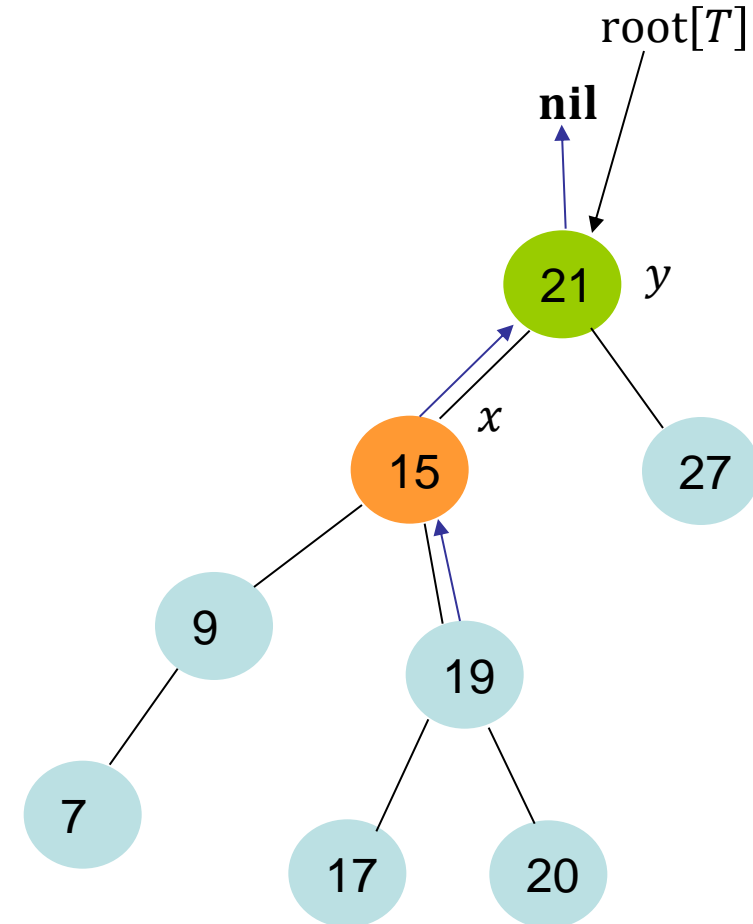
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \mathbf{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \mathbf{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7.     **else**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



## Datenstrukturen

Linksrotation( $T, x$ )

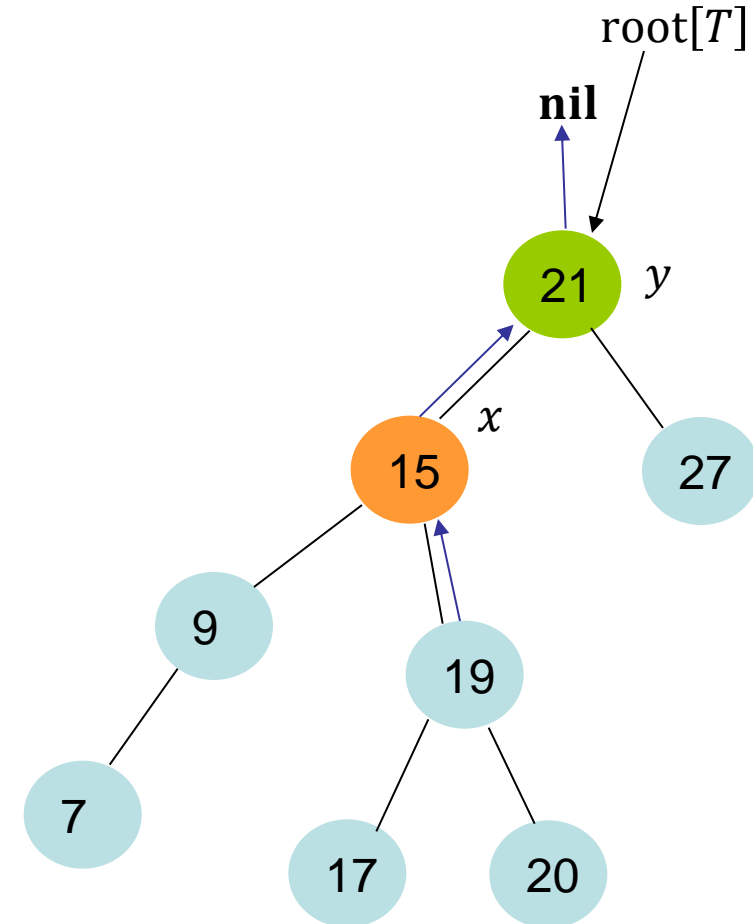
1.  $y \leftarrow rc[x]$
2.  $rc[x] \leftarrow lc[y]$
3. **if**  $lc[y] \neq \mathbf{nil}$  **then**  $p[lc[y]] \leftarrow x$
4.  $p[y] \leftarrow p[x]$
5. **if**  $p[x] = \mathbf{nil}$  **then**  $root[T] \leftarrow y$
6. **else if**  $x = lc[p[x]]$  **then**  $lc[p[x]] \leftarrow y$
7.     **else**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



## Datenstrukturen

Linksrotation( $T, x$ )

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



## Datenstrukturen

### *Dynamische AVL-Bäume*

- Operationen Suche, Einfügen, Löschen, Min/Max, Vorgänger/Nachfolger,... wie in der letzten Vorlesung
- Laufzeit  $O(h)$  für diese Operationen
- Nur Einfügen/Löschen verändern Struktur des Baums

### *Idee*

Wir brauchen eine Prozedur, um die AVL-Eigenschaft nach Einfügen/Löschen wiederherzustellen.

## Datenstrukturen

### Dynamische AVL

- Operationen Suchen, Einfügen, Löschen, Min/Max, Vorgänger/Nachfolger, etc. (siehe letzten Vorlesung)
- Laufzeit  $O(h)$  für diese Operationen
- Nur Einfügen/Löschen verändern Struktur des Baums

Nach Korollar 35  
gilt  $h = \Theta(\log n)$

### Idee

Wir brauchen eine Prozedur, um die AVL-Eigenschaft nach Einfügen/Löschen wiederherzustellen.



## Datenstrukturen

### *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.

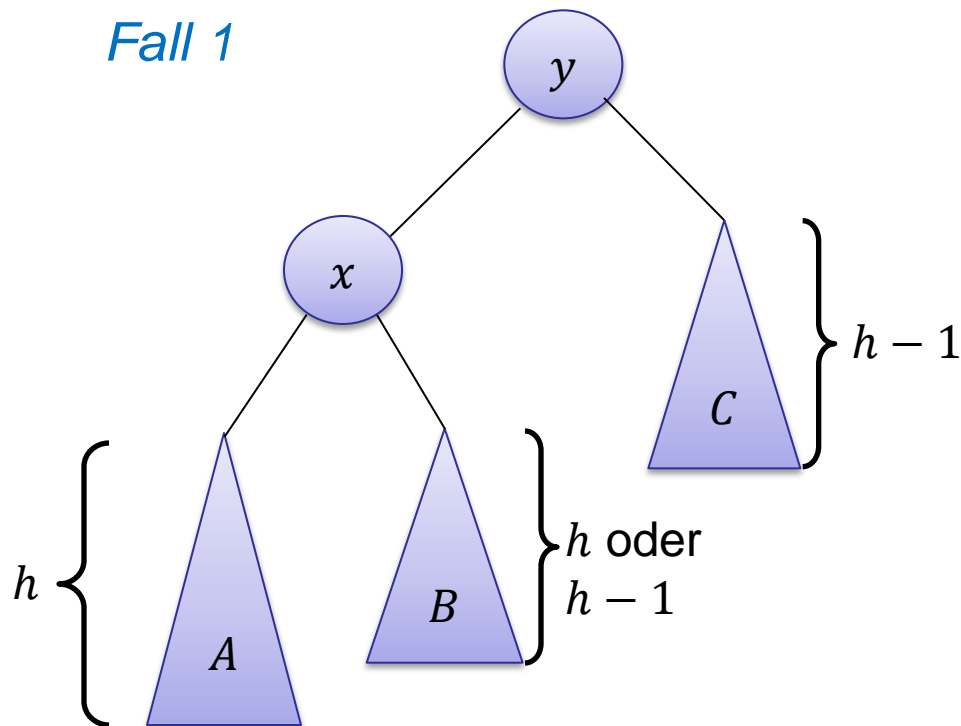
## Datenstrukturen

### *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

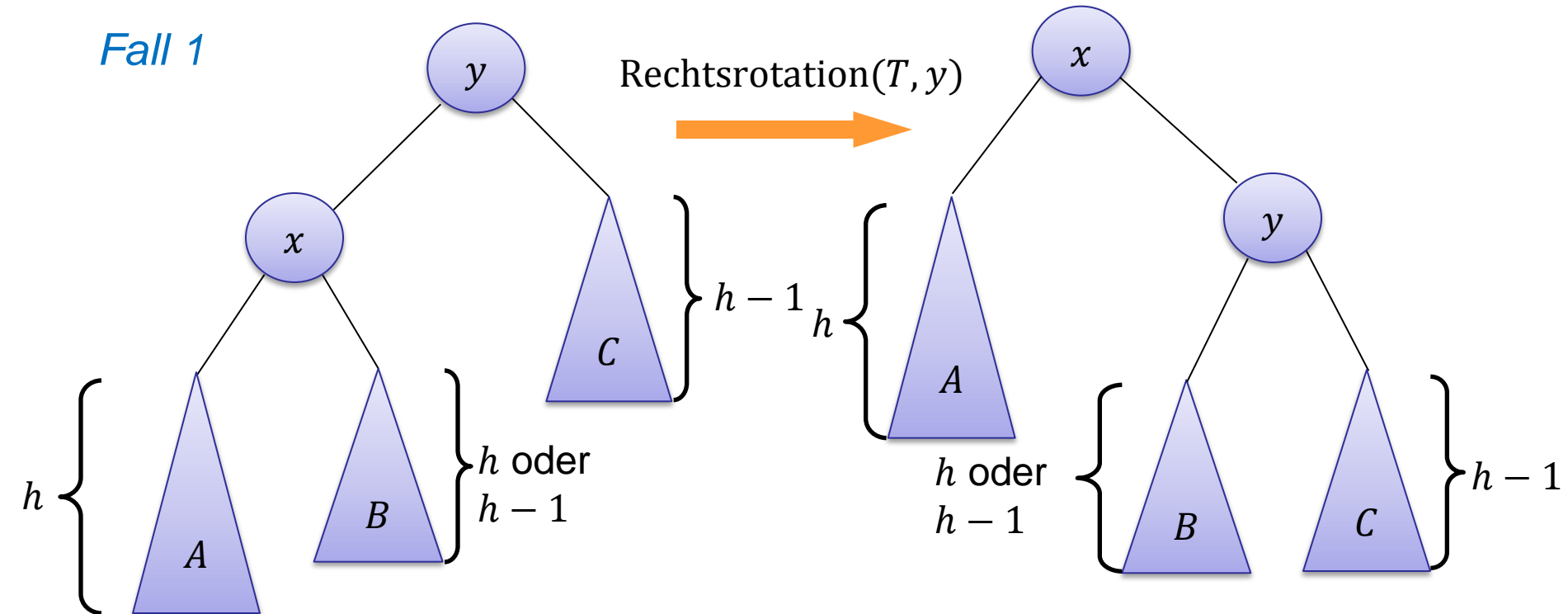
## Datenstrukturen

*Fall 1*



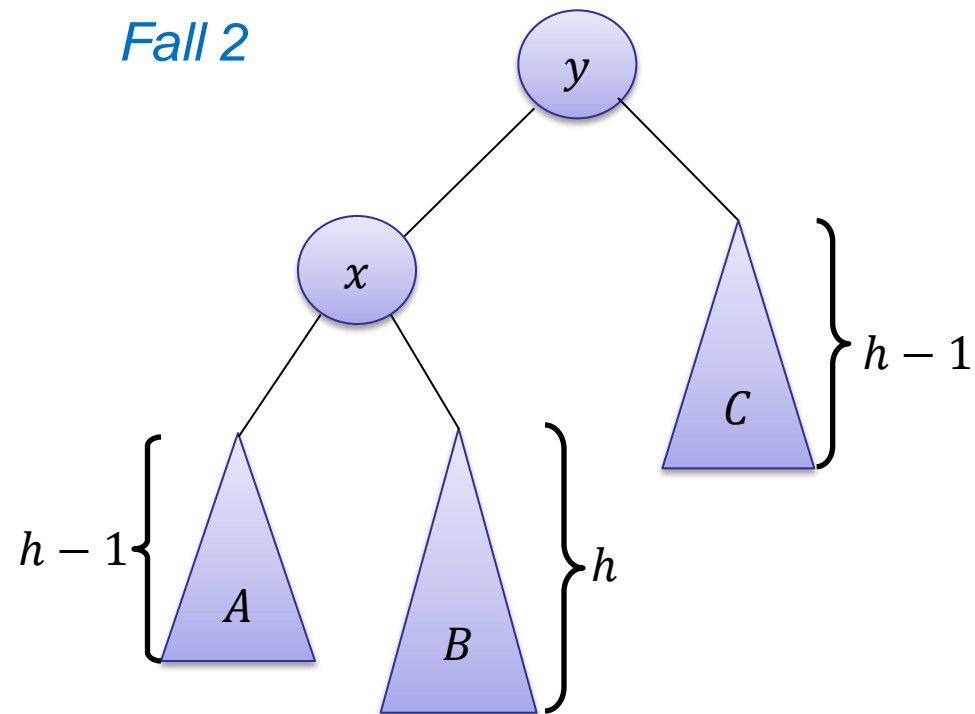
## Datenstrukturen

Fall 1



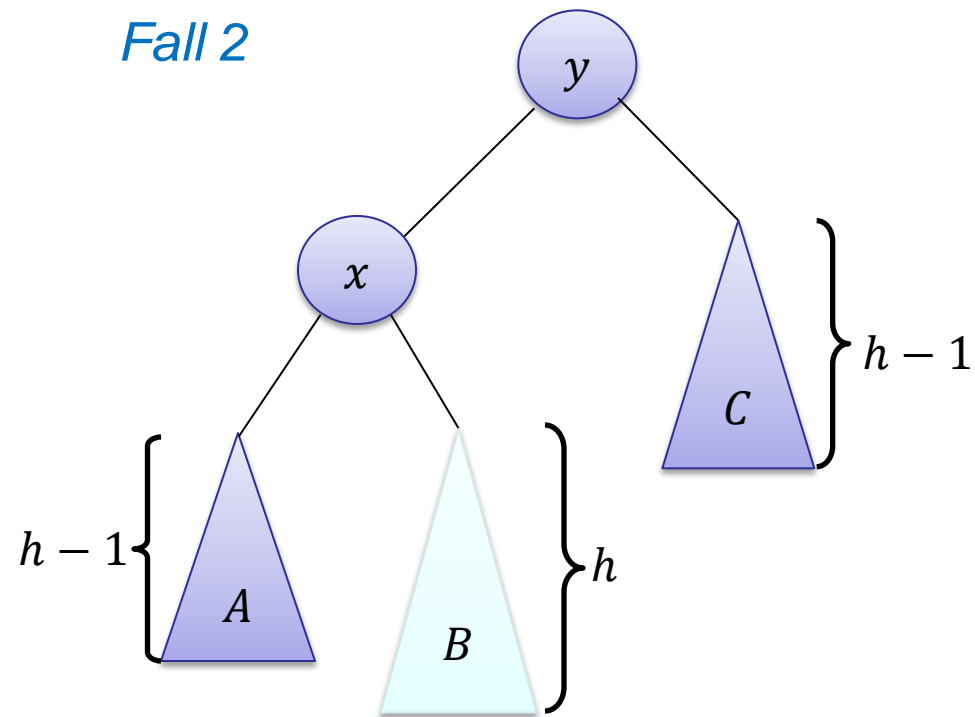
## Datenstrukturen

*Fall 2*



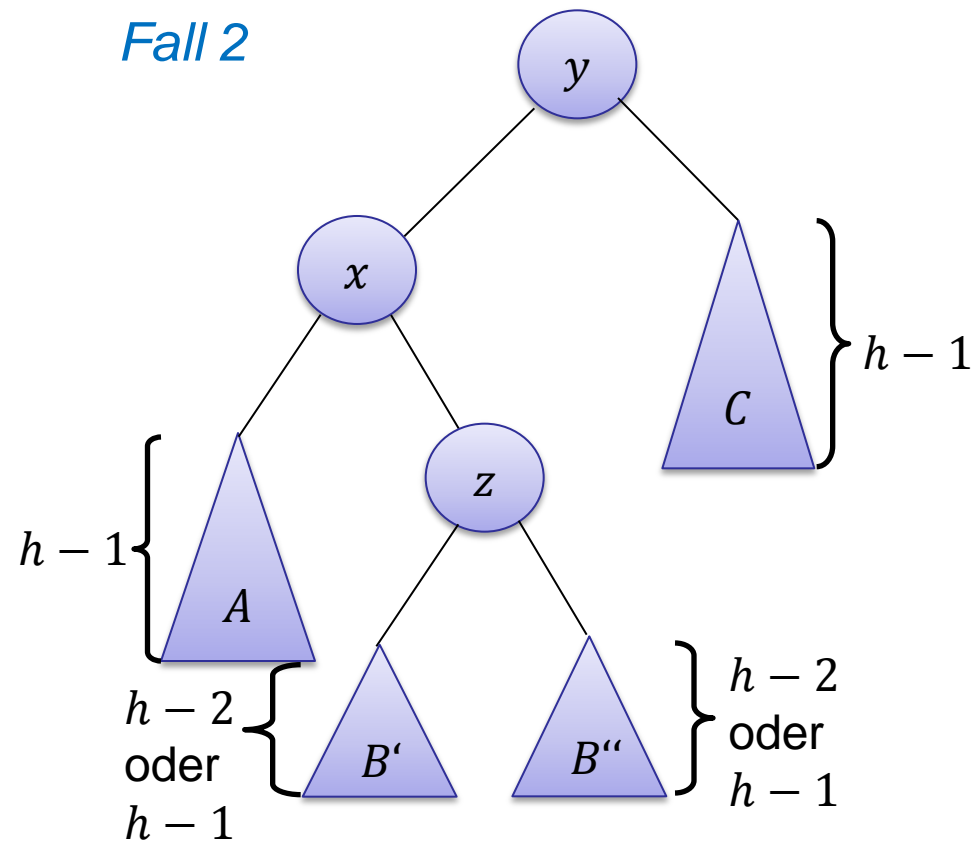
## Datenstrukturen

*Fall 2*



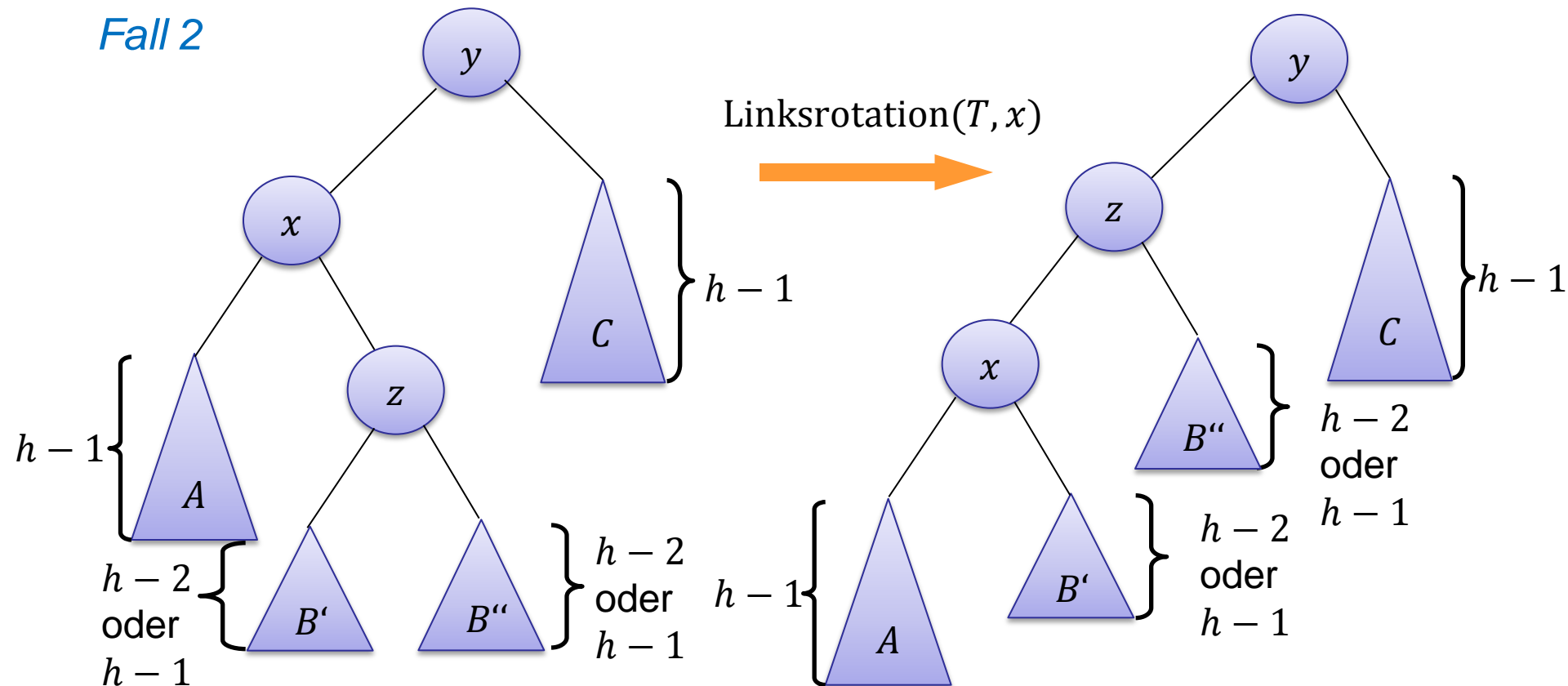
## Datenstrukturen

*Fall 2*



## Datenstrukturen

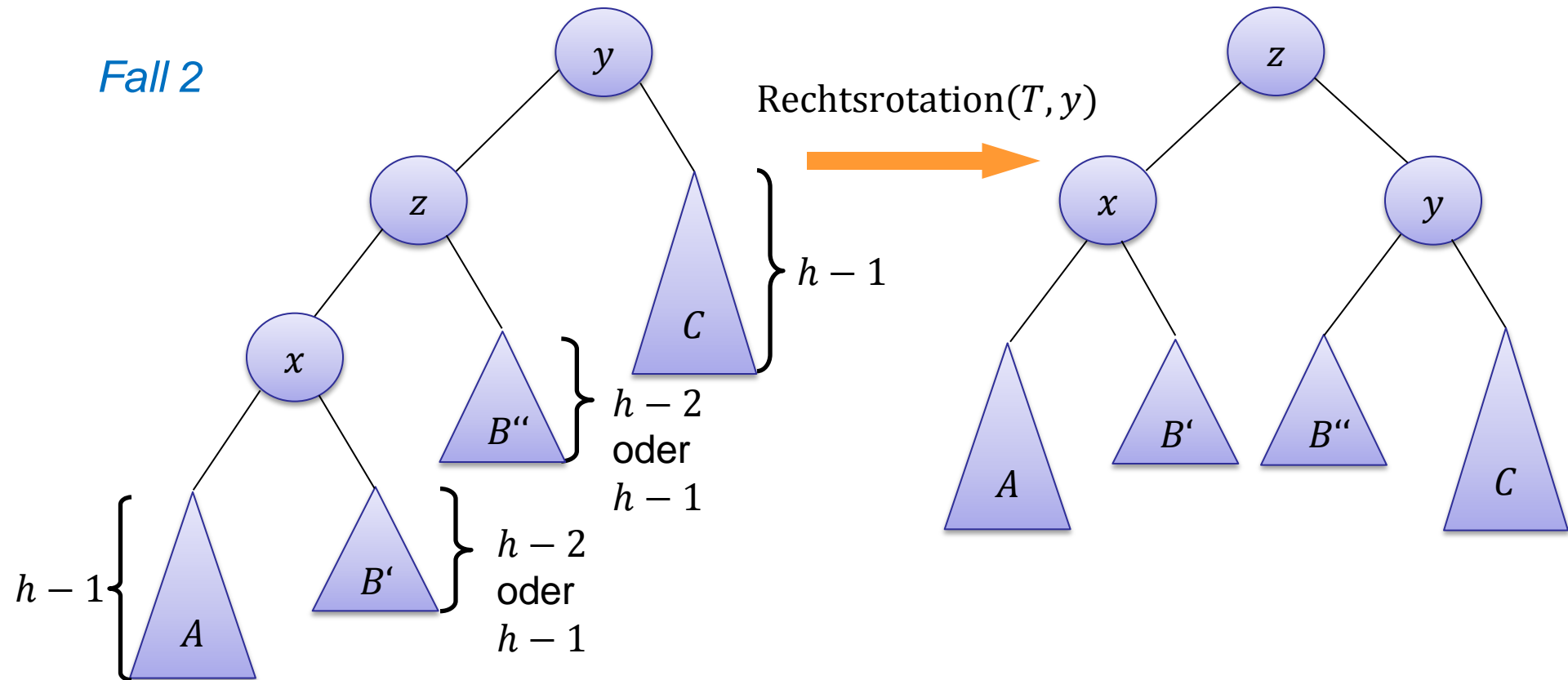
*Fall 2*





## Datenstrukturen

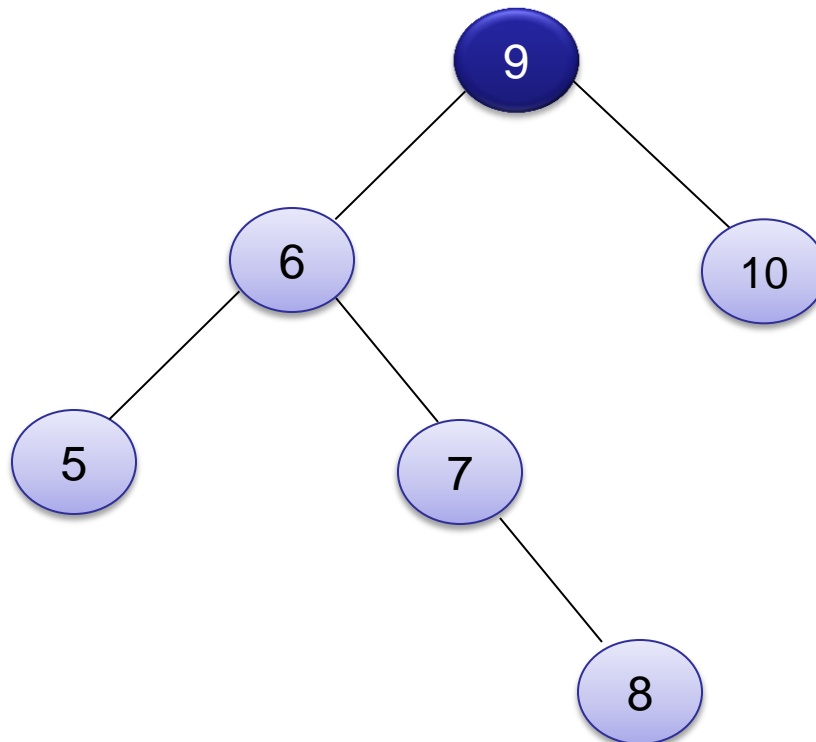
Fall 2



## Datenstrukturen

### Aufgabe

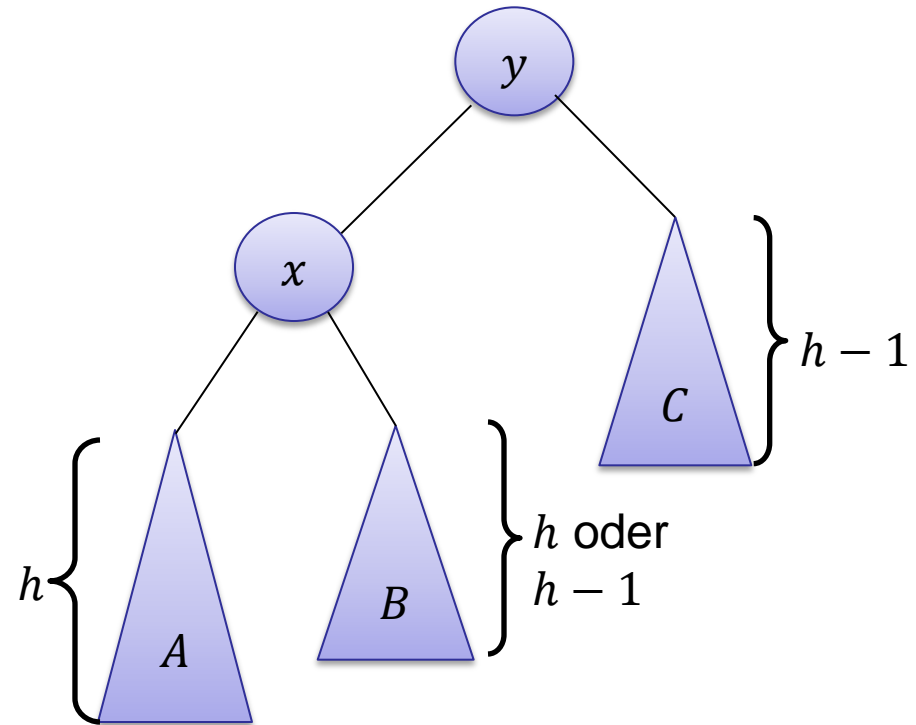
Bestimmen Sie den richtigen Fall und führen Sie die Rotation(en) durch



## Datenstrukturen

Balance( $T$ )

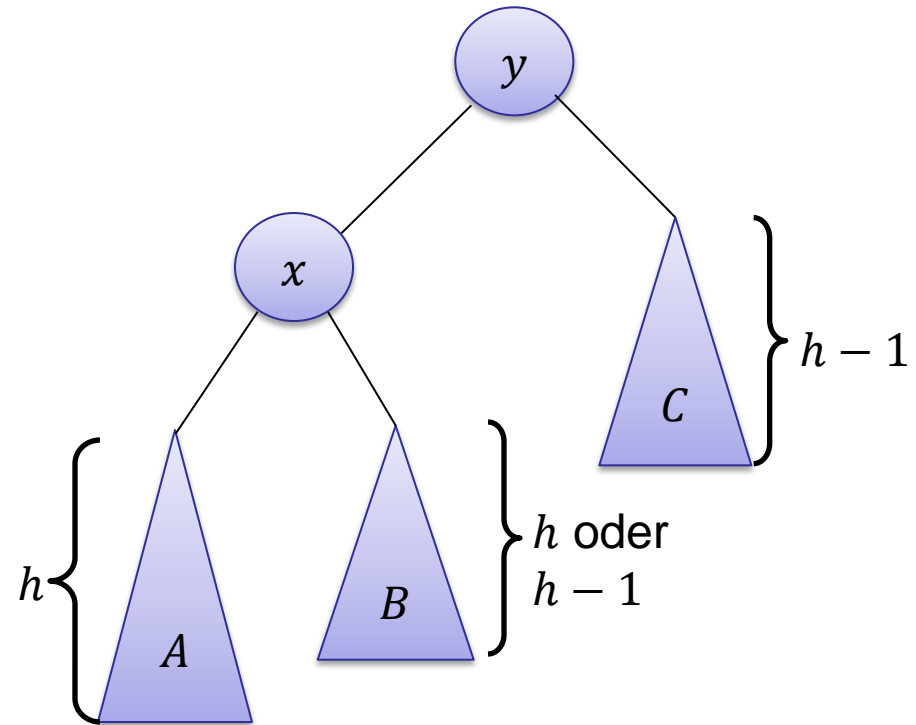
1.  $t \leftarrow \text{root}[T]$
2. **if**  $h[\text{lc}[t]] > h[\text{rc}[t]] + 1$  **then**
3.     **if**  $h[\text{lc}[\text{lc}[t]]] < h[\text{rc}[\text{lc}[t]]]$  **then**
4.         Linksrotation( $T, \text{lc}[t]$ )
5.     Rechtsrotation( $T, t$ )
6. **else if**  $h[\text{rc}[t]] > h[\text{lc}[t]] + 1$  **then**
7.     **if**  $h[\text{rc}[\text{rc}[t]]] < h[\text{lc}[\text{rc}[t]]]$  **then**
8.         Rechtsrotation( $T, \text{rc}[t]$ )
9.     Linksrotation( $T, t$ )



## Datenstrukturen

Balance( $T$ )

1.  $t \leftarrow \text{root}[T]$
2. **if**  $h[\text{lc}[t]] > h[\text{rc}[t]] + 1$  **then**
3.     **if**  $h[\text{lc}[\text{lc}[t]]] < h[\text{rc}[\text{lc}[t]]]$  **then**
4.         Linksrotation( $T, \text{lc}[t]$ )
5.     Rechtsrotation( $T, t$ )
6. **else if**  $h[\text{rc}[t]] > h[\text{lc}[t]] + 1$  **then**
7.     **if**  $h[\text{rc}[\text{rc}[t]]] < h[\text{lc}[\text{rc}[t]]]$  **then**
8.         Rechtsrotation( $T, \text{rc}[t]$ )
9.     Linksrotation( $T, t$ )

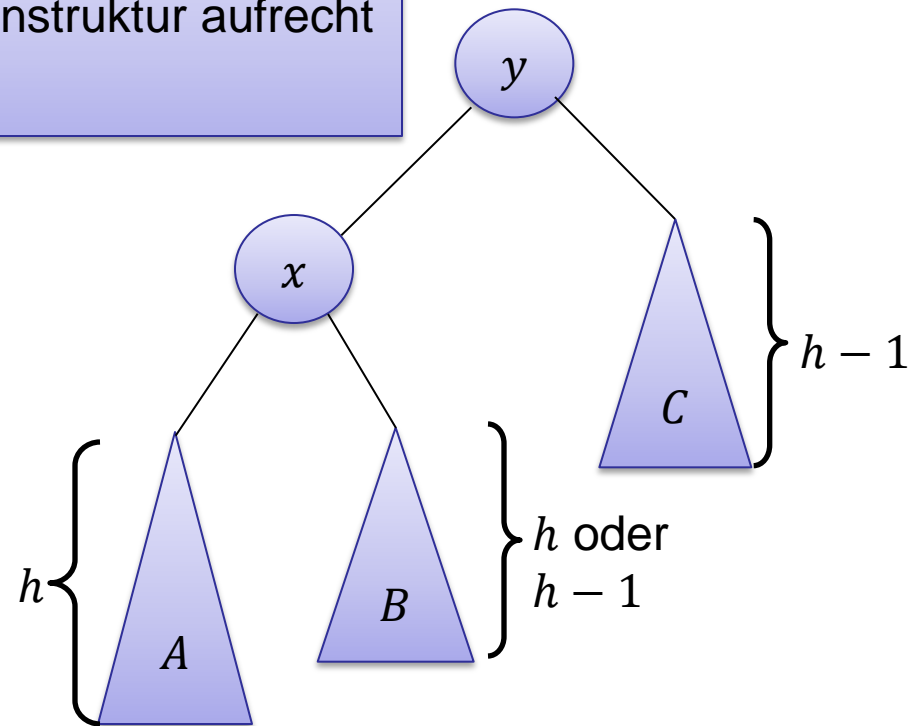


## Datenstrukturen

Balance( $T$ )

1.  $t \leftarrow \text{root}[T]$
2. **if**  $h[\text{lc}[t]] > h[\text{rc}[t]] + 1$  **then**
3.   **if**  $h[\text{lc}[\text{lc}[t]]] < h[\text{rc}[\text{lc}[t]]]$  **then**
4.     Linksrotation( $T, \text{lc}[t]$ )
5.     Rechtsrotation( $T, t$ )
6. **else if**  $h[\text{rc}[t]] > h[\text{lc}[t]] + 1$  **then**
7.   **if**  $h[\text{rc}[\text{rc}[t]]] < h[\text{lc}[\text{rc}[t]]]$  **then**
8.     Rechtsrotation( $T, \text{rc}[t]$ )
9.     Linksrotation( $T, t$ )

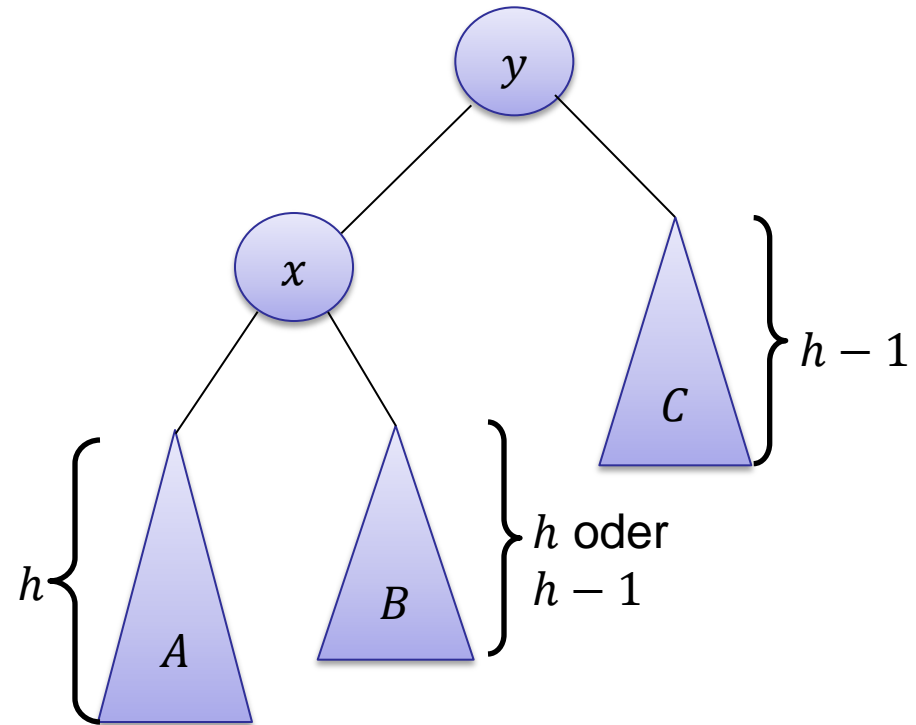
$h$  gibt Höhe des Teilbaums an.  
Dies müssen wir zusätzlich in  
unserer Datenstruktur aufrecht  
erhalten.



## Datenstrukturen

Balance( $T$ )

1.  $t \leftarrow \text{root}[T]$
2. **if**  $h[\text{lc}[t]] > h[\text{rc}[t]] + 1$  **then**
3.     **if**  $h[\text{lc}[\text{lc}[t]]] < h[\text{rc}[\text{lc}[t]]]$  **then**
4.         Linksrotation( $T, \text{lc}[t]$ )
5.     Rechtsrotation( $T, t$ )
6. **else if**  $h[\text{rc}[t]] > h[\text{lc}[t]] + 1$  **then**
7.     **if**  $h[\text{rc}[\text{rc}[t]]] < h[\text{lc}[\text{rc}[t]]]$  **then**
8.         Rechtsrotation( $T, \text{rc}[t]$ )
9.     Linksrotation( $T, t$ )

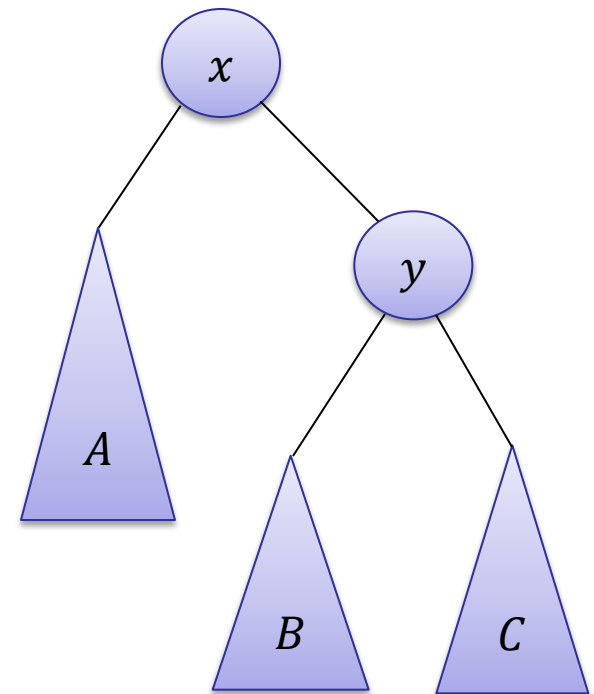


## Datenstrukturen

Balance( $T$ )

1.  $t \leftarrow \text{root}[T]$
2. **if**  $h[\text{lc}[t]] > h[\text{rc}[t]] + 1$  **then**
3.   **if**  $h[\text{lc}[\text{lc}[t]]] < h[\text{rc}[\text{lc}[t]]]$  **then**
4.     Linksrotation( $T, \text{lc}[t]$ )
5.     Rechtsrotation( $T, t$ )
6. **else if**  $h[\text{rc}[t]] > h[\text{lc}[t]] + 1$  **then**
7.   **if**  $h[\text{rc}[\text{rc}[t]]] < h[\text{lc}[\text{rc}[t]]]$  **then**
8.     Rechtsrotation( $T, \text{rc}[t]$ )
9.     Linksrotation( $T, t$ )

- Laufzeit:  $\mathcal{O}(1)$



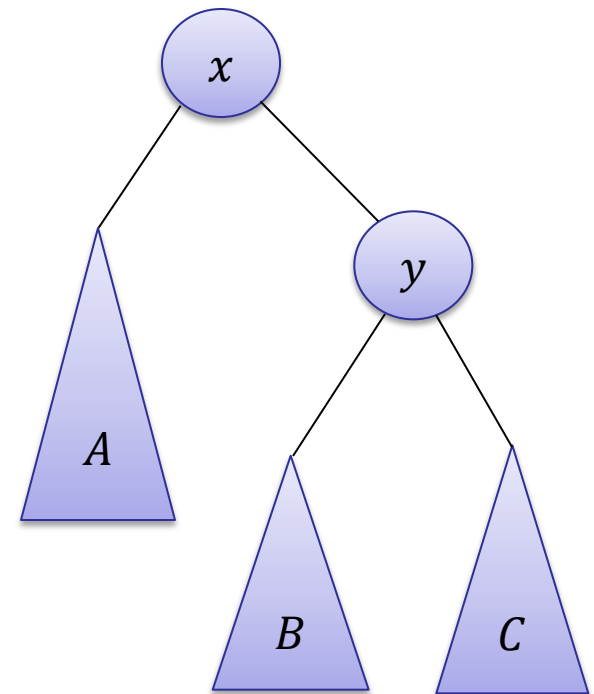
## Datenstrukturen

Balance( $T$ )

1.  $t \leftarrow \text{root}[T]$
2. **if**  $h[\text{lc}[t]] > h[\text{rc}[t]] + 1$  **then**
3.     **if**  $h[\text{lc}[\text{lc}[t]]] < h[\text{rc}[\text{lc}[t]]]$  **then**
4.         Linksrotation( $T, \text{lc}[t]$ )
5.     Rechtsrotation( $T, t$ )
6. **else if**  $h[\text{rc}[t]] > h[\text{lc}[t]] + 1$  **then**
7.     **if**  $h[\text{rc}[\text{rc}[t]]] < h[\text{lc}[\text{rc}[t]]]$  **then**
8.         Rechtsrotation( $T, \text{rc}[t]$ )
9.     Linksrotation( $T, t$ )

### Wichtiger Hinweis

Nach allen Rotationen muss zusätzlich noch die Höhe der Knoten  $x$  und  $y$  angepasst werden





## Datenstrukturen

### *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

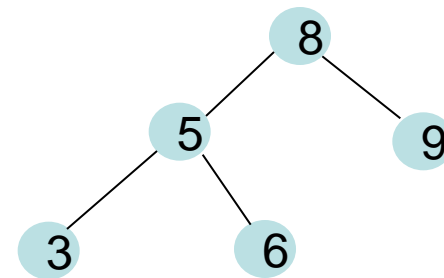
### *Einfügen*

- Wir fügen ein wie früher
- Dann laufen wir den Pfad zur Wurzel zurück
- An jedem Knoten balancieren wir, falls der Unterbaum ein beinahe-AVL-Baum ist

## Datenstrukturen

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( $t$ )

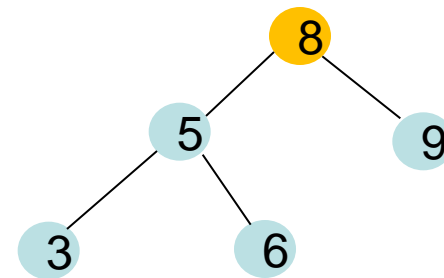


Einfügen 2

## Datenstrukturen

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( $t$ )

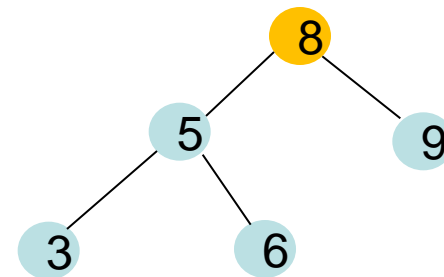


Einfügen 2

## Datenstrukturen

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( $t$ )

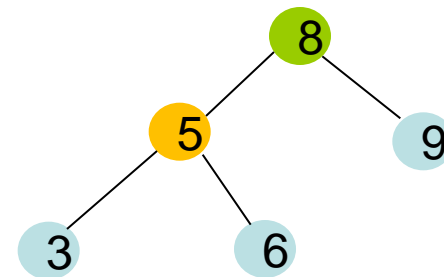


Einfügen 2

## Datenstrukturen

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( $t$ )

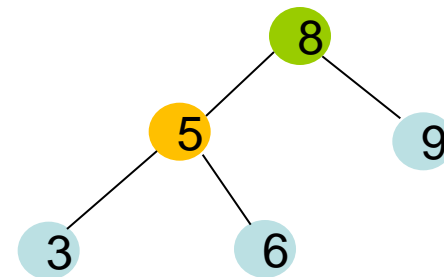


Einfügen 2

## Datenstrukturen

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( $t$ )

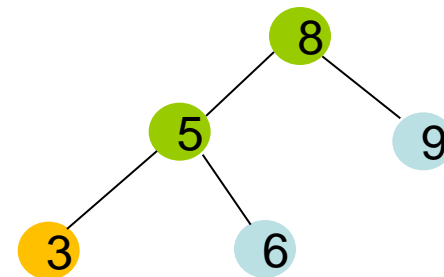


Einfügen 2

## Datenstrukturen

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( $t$ )

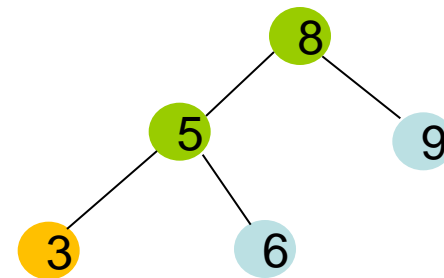


Einfügen 2

## Datenstrukturen

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( $t$ )



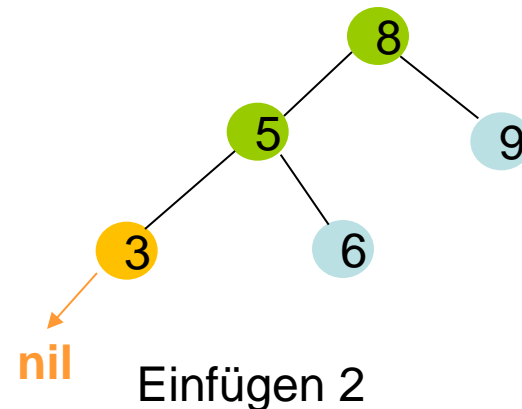
Einfügen 2



## Datenstrukturen

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( $t$ )



## Datenstrukturen

AVL-Einfügen( $t, x$ )

1. **if**  $t = \text{nil}$  **then**
2.  $t \leftarrow \text{new node}(x); h[t] \leftarrow 0; \text{return}$
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( $t$ )

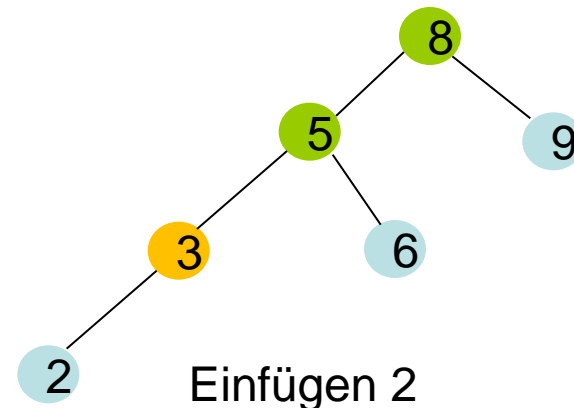
Neuen Knoten erzeugen.  
Zusätzlich noch Zeiger  
 $\text{lc}[t]$  und  $\text{rc}[t]$  auf **nil**  
setzen, sowie  $p[t]$  und  
den Zeiger von  $p[t]$   
setzen.



## Datenstrukturen

AVL-Einfügen( $t, x$ )

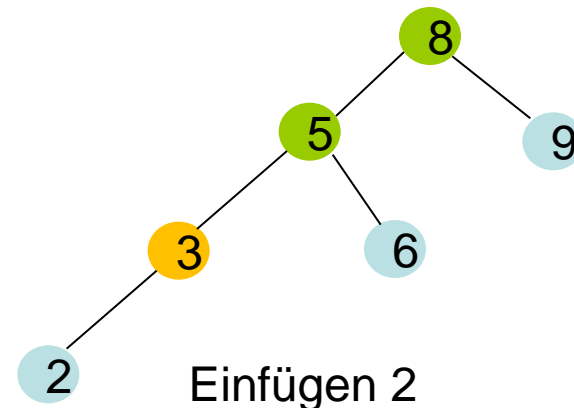
1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$ ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $\text{lc}[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Einfügen( $\text{rc}[t], x$ )
5.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



## Datenstrukturen

AVL-Einfügen( $t, x$ )

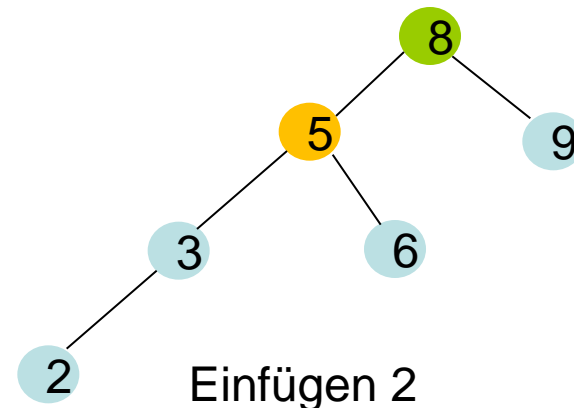
1. **if**  $t = \text{nil}$  **then**
2.    $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$ ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $\text{lc}[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Einfügen( $\text{rc}[t], x$ )
5.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



## Datenstrukturen

AVL-Einfügen( $t, x$ )

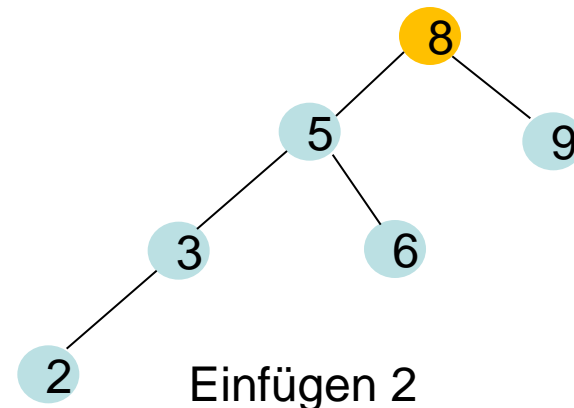
1. **if**  $t = \text{nil}$  **then**
2.      $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$ ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $\text{lc}[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Einfügen( $\text{rc}[t], x$ )
5.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



## Datenstrukturen

AVL-Einfügen( $t, x$ )

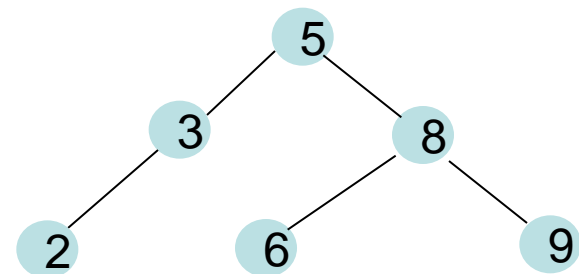
1. **if**  $t = \text{nil}$  **then**
2.      $t \leftarrow \text{new node}(x)$ ;  $h[t] \leftarrow 0$ ; **return**
3. **else if**  $\text{key}[x] < \text{key}[t]$  **then** AVL-Einfügen( $\text{lc}[t], x$ )
4. **else if**  $\text{key}[x] > \text{key}[t]$  **then** AVL-Einfügen( $\text{rc}[t], x$ )
5.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



## Datenstrukturen

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**( $t$ )



Einfügen 2

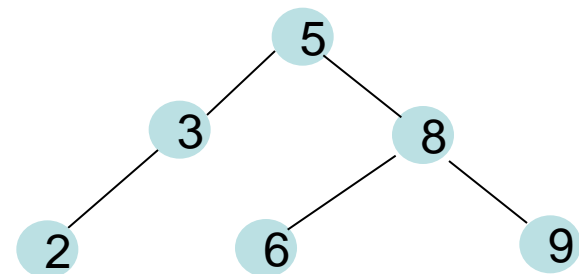
## Datenstrukturen

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( $t$ )

### Laufzeit

- $\mathbf{O}(h) = \mathbf{O}(\log n)$



Einfügen 2



## Datenstrukturen

### Satz 36

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  $h + 1$ .

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element in einen AVL-Baum eingefügt, so ist der resultierende 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( $t$ )

### Beweis

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

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element in einen AVL-Baum eingefügt, so ist der resultierende 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( $t$ )

### 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  $j$ ,  $-1 \leq j \leq h$ .

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element in einen AVL-Baum eingefügt, so ist der resultierende 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( $t$ )

### 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  $h + 1 \geq 0$ .

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende 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( $t$ )

### 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).

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende 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( $t$ )

### 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.

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende 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( $t$ )

### 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.

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element in einen AVL-Baum eingefügt, so ist der resultierende 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( $t$ )

### 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.



## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element eingefügt,  
so ist der resultierende 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( $t$ )

### 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.

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element  $x$  in einen AVL-Baum eingefügt, so ist der resultierende Baum 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( $t$ )

### 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.

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende 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( $t$ )

### 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.

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende 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( $t$ )

### 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.

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende 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( $t$ )

### 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  $h + 2$ .

## Datenstrukturen

### Satz 36

Wird mit AVL-Einfügen ein Element eingefügt, so ist der resultierende 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( $t$ )

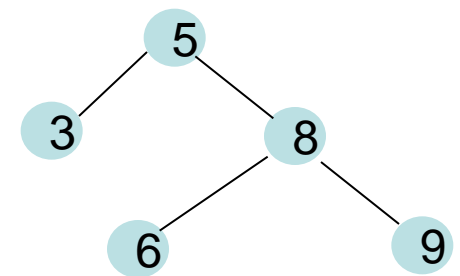
### Beweis

- 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  $h + 2$ .

## Datenstrukturen

AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )



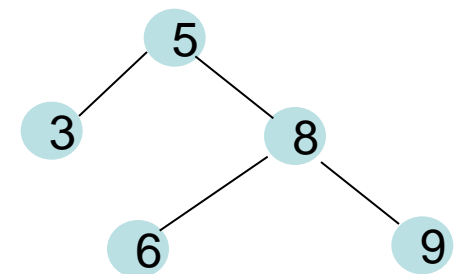
## Datenstrukturen

$x$  bezeichnet Schlüssel  
des zu löschenden  
Elements.

### AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Löschen(3)



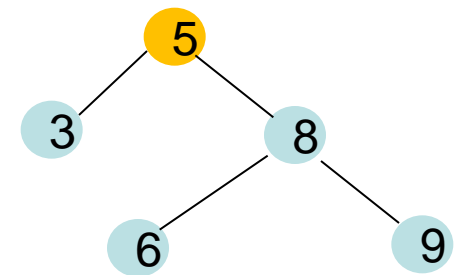


## Datenstrukturen

AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Löschen(3)

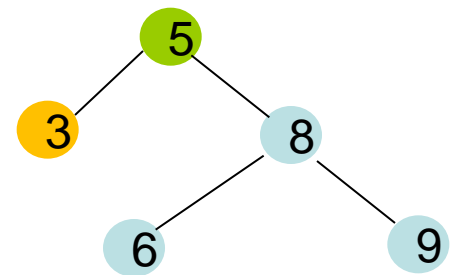


## Datenstrukturen

AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Löschen(3)



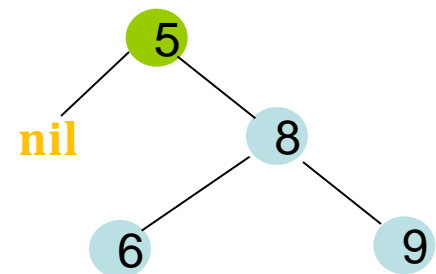
## Datenstrukturen

AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Und die anderen  
Zeiger aktualisieren

Löschen(3)

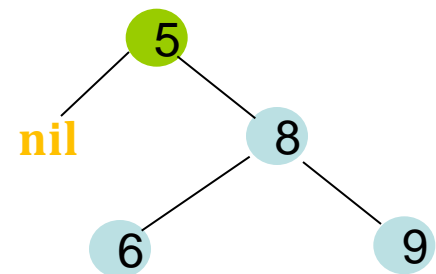


## Datenstrukturen

AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )

Löschen(3)

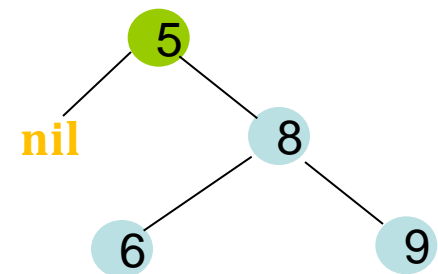


## Datenstrukturen

AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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  $u$  nach  $t$
8.     AVL-Lösche  $u$  da Baum leer. ( $\text{rc}[t]$ )
9. **if**  $t \neq \text{nil}$  **then**  $\text{Balance}(\text{max}\{h[\text{lc}[t]], h[\text{rc}[t]]\})$
10. **Balance**( $t$ )

Löschen(3)

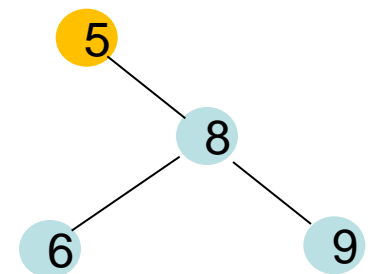


## Datenstrukturen

AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. **Balance**( $t$ )

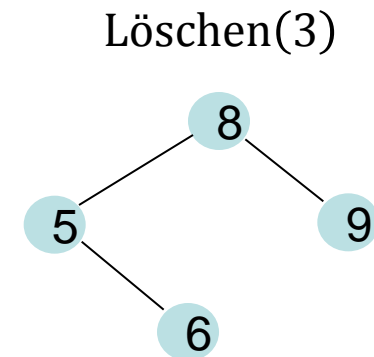
Löschen(3)



## Datenstrukturen

AVL-Löschen( $t, x$ )

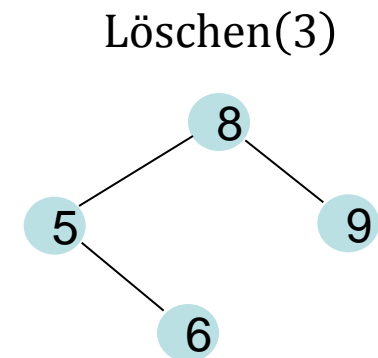
1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **then** AVL-Löschen( $\text{rc}[t], x$ )
3. **else if**  $t = \text{nil}$  **then return**  $\triangleright x$  nicht im Baum
4. **else if**  $\text{lc}[t] = \text{nil}$  **then** ersetze  $t$  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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. **Balance**( $t$ )



## Datenstrukturen

AVL-Löschen( $t, x$ )

1. **if**  $x < \text{key}[t]$  **then** AVL-Löschen( $\text{lc}[t], x$ )
2. **else if**  $x > \text{key}[t]$  **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**  $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance( $t$ )





## Datenstrukturen

### *Korrektheit:*

- Ähnlich wie beim Einfügen

### *Satz 37*

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.

## Datenstrukturen

### *Zusammenfassung und Ausblick*

- Effiziente Datenstruktur für das Datenbank Problem mit Hilfe von Suchbäumen
- Kann man eine bessere Datenstruktur finden?
- Was muss man ggf. anders machen?