Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Datenstrukturen

Problem

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

Operationen

- $\text{Suche}(x)$; Ausgabe O mit Schlüssel $s(O) = x$;
  $\mathbf{nil}$, falls kein Objekt mit Schlüssel x in Datenbank
- $\text{Einfügen}(O)$; Einfügen von Objekt O in Datenbank
- $\text{Löschen}(O)$; 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 $\mathbf{O}(h)$

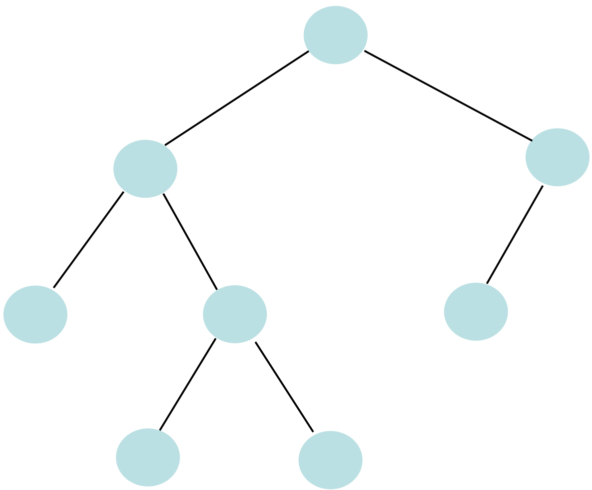
Frage

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

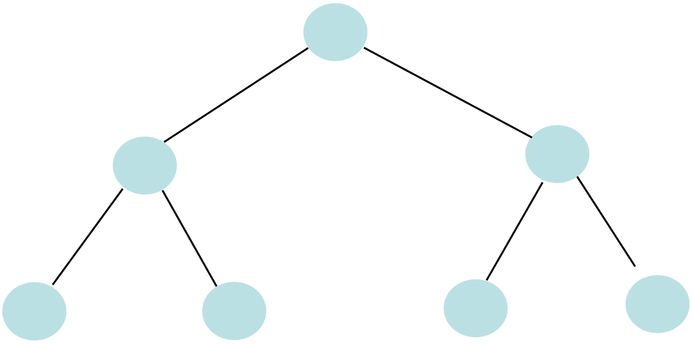
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

Beweis

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

- AVL-Baum ist Binärbaum



Datenstrukturen

Satz 34

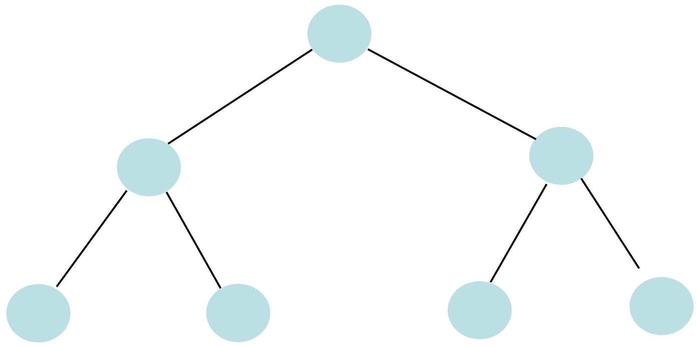
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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

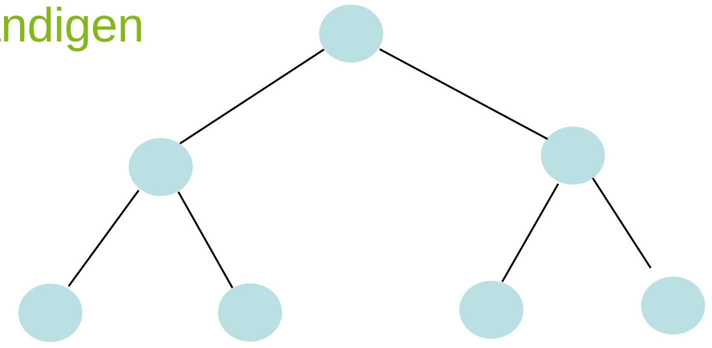
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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

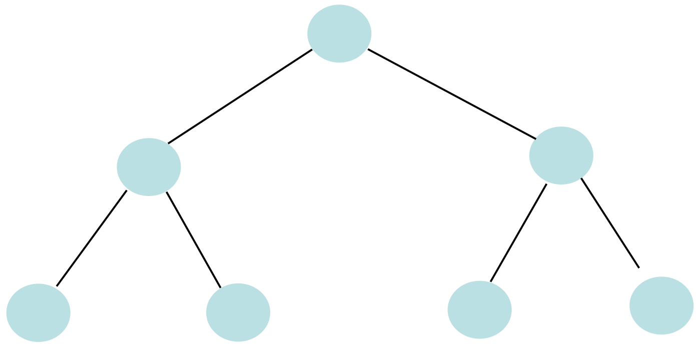
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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:

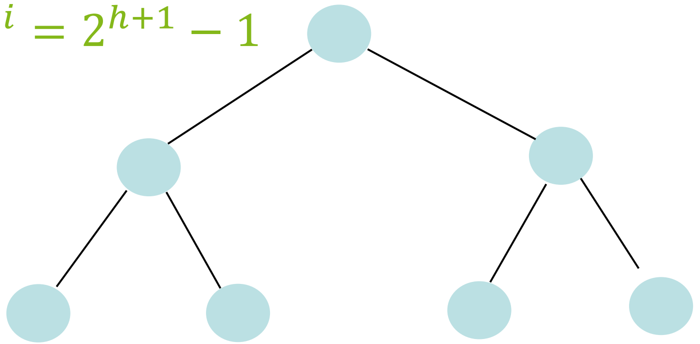
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis

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

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

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



Datenstrukturen

Satz 34

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

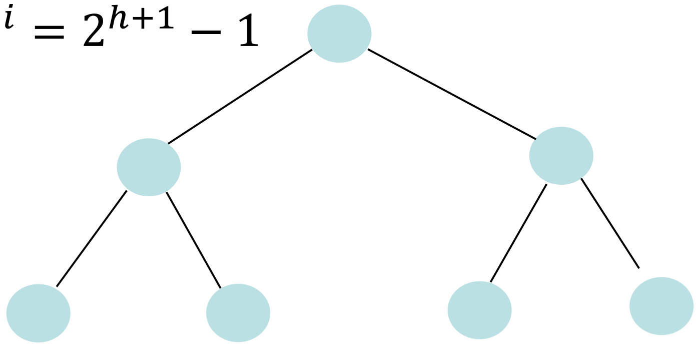
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

Beweis

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

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

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



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:

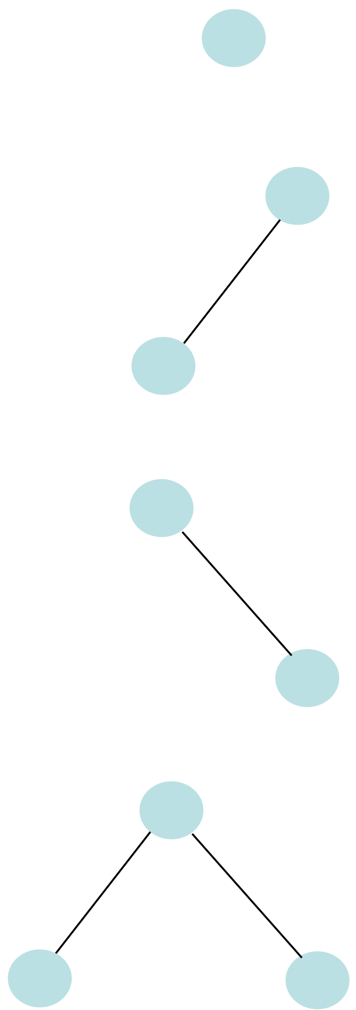
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

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:

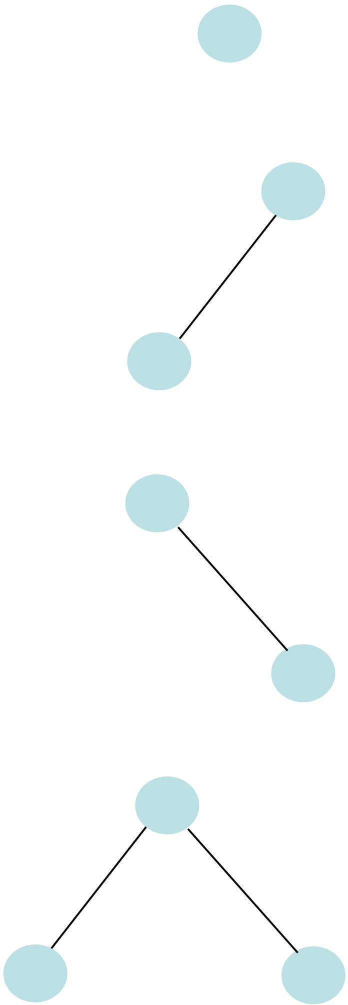
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

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

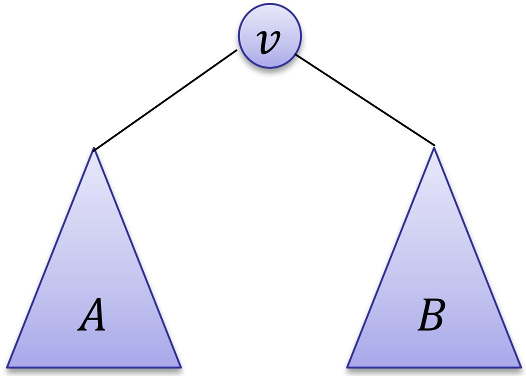
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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

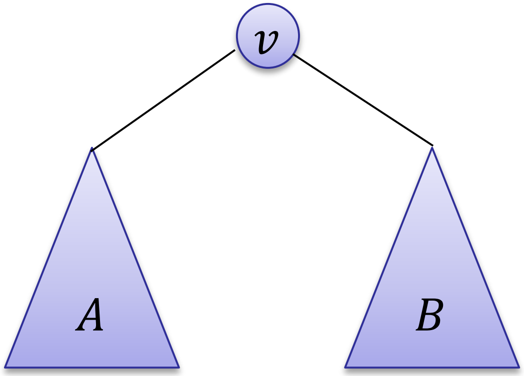
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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 $\mathrm{h} \geq 1$. Betrachte AVL-Baum T der Höhe $h+1$ mit Wurzel v.
- Seien A, B linker bzw. rechter Teilbaum von v.



Datenstrukturen

Satz 34

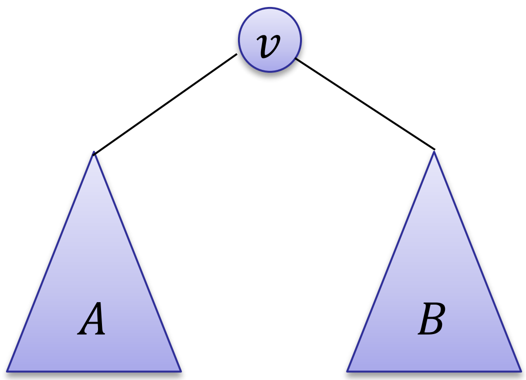
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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 $\mathrm{h} \geq 1$. 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

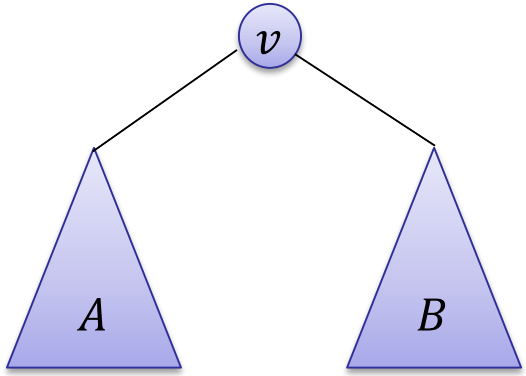
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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 $\mathrm{h} \geq 1$. 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

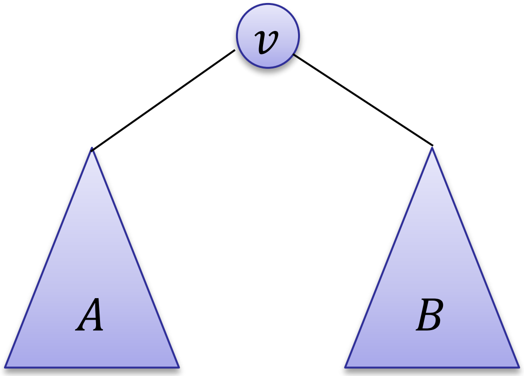
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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 $\mathrm{h} \geq 1$. 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

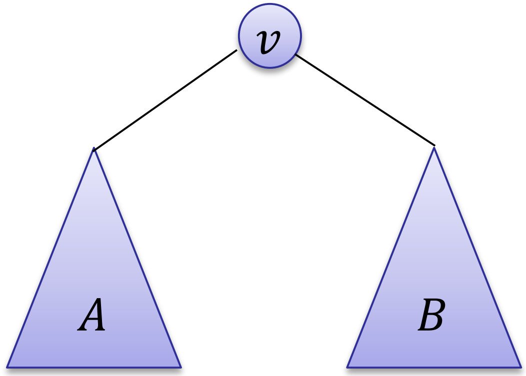
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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

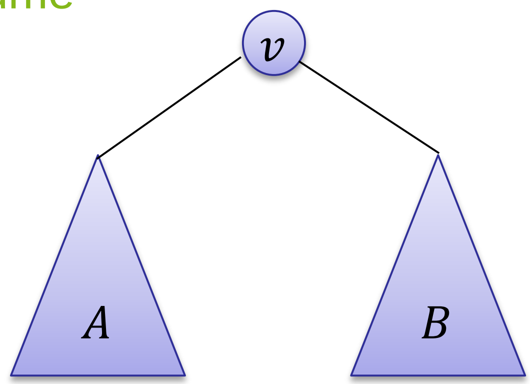
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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 ≥ 0 sind



Datenstrukturen

Satz 34

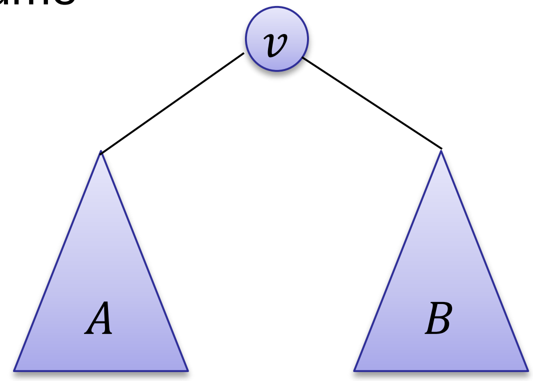
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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 ≥ 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

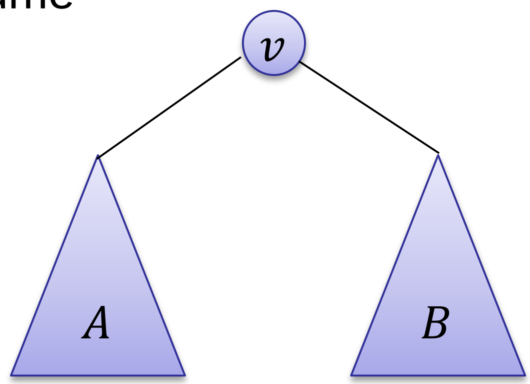
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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 ≥ 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

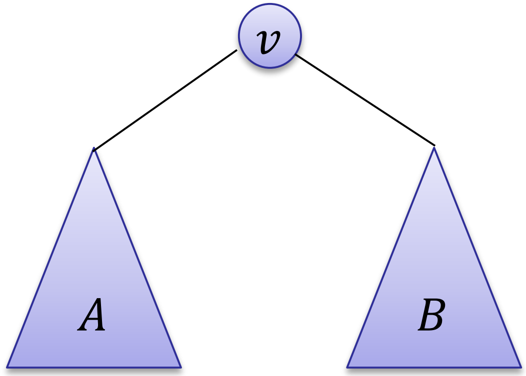
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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:

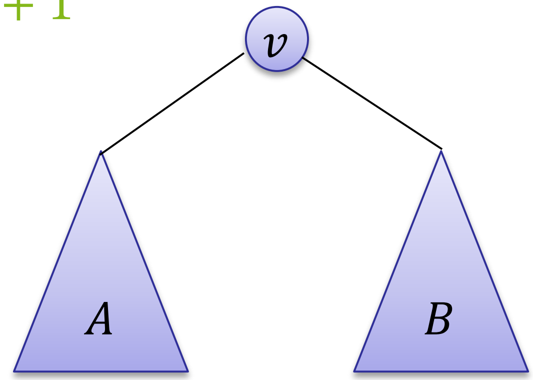
$$(3/2)^h \leq n \leq 2^{h+1} - 1$$

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:

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

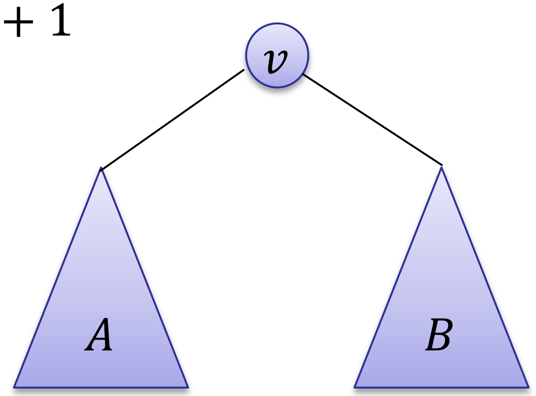
Beweis

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

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

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



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 $\boldsymbol{\Theta}(\log n)$.

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 $\mathbf{\Theta}(\log n)$.

Beweis

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

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

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 $\boldsymbol{\Theta}(\log n)$.

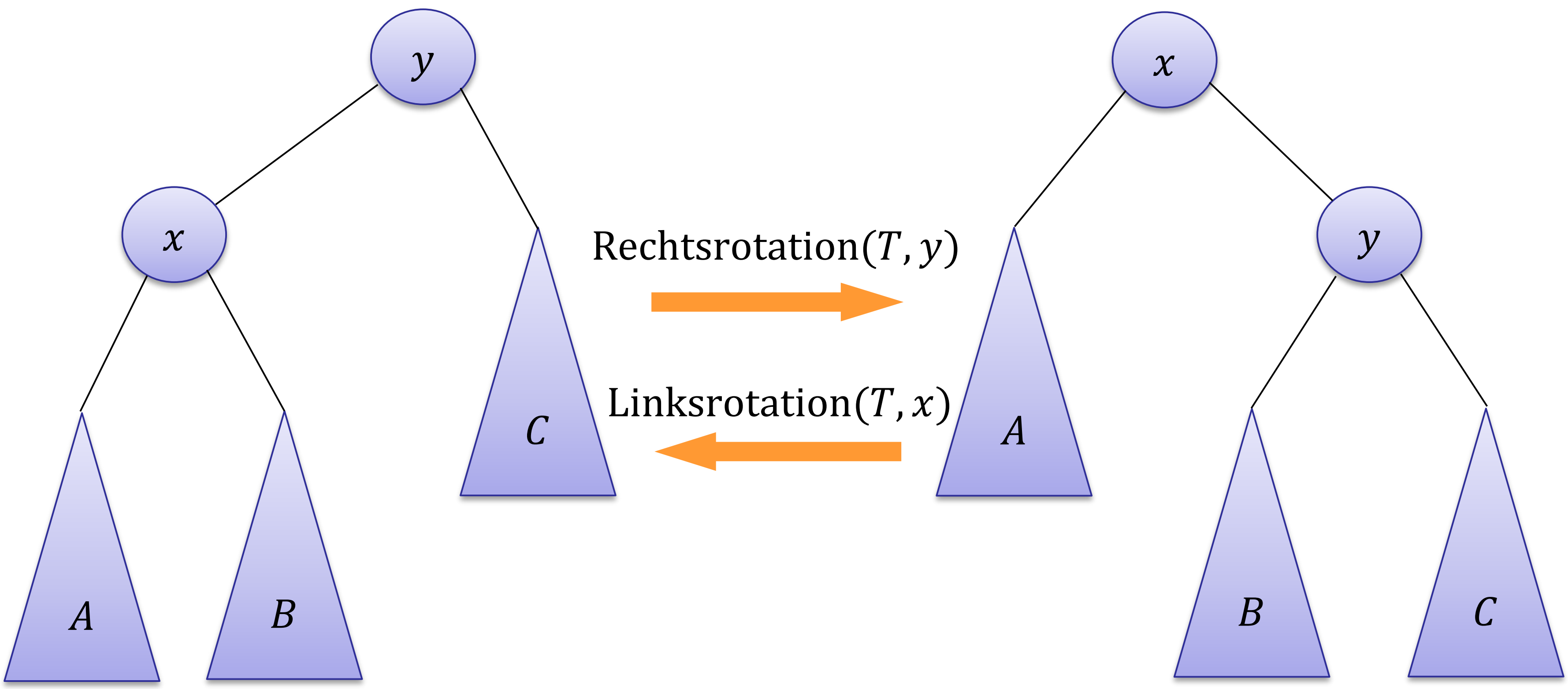
Beweis

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

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

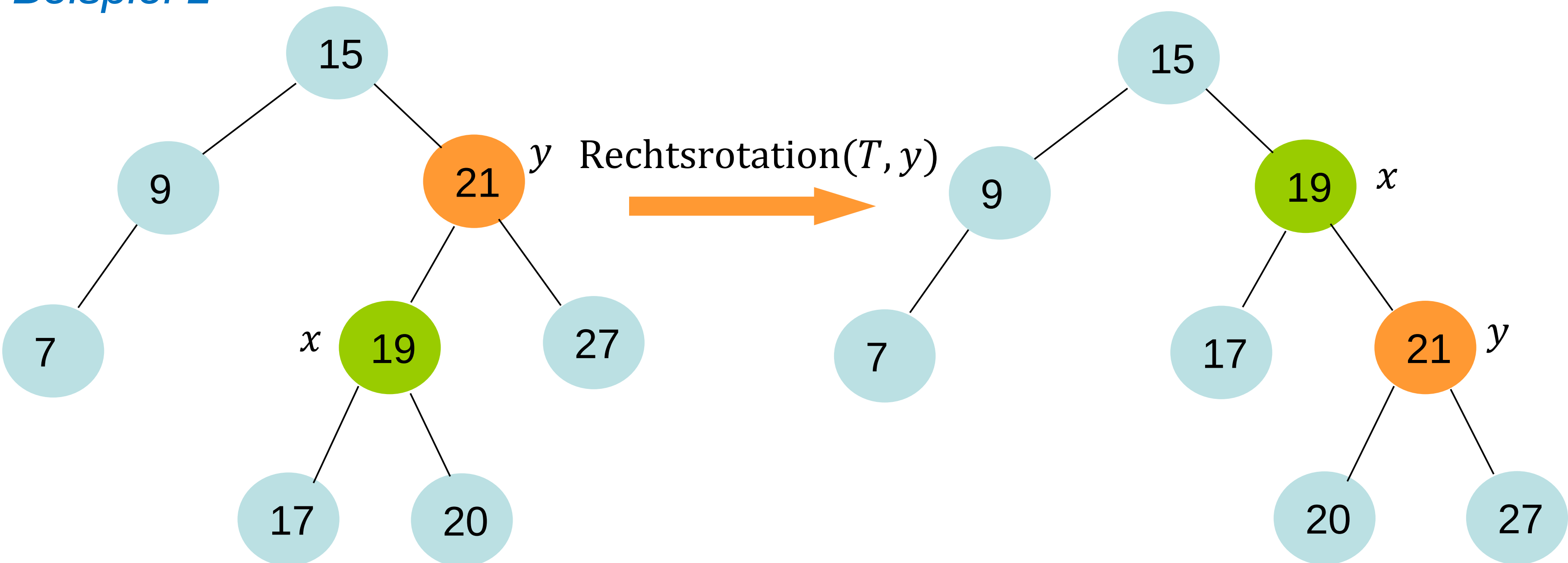
Datenstrukturen

Rotationen



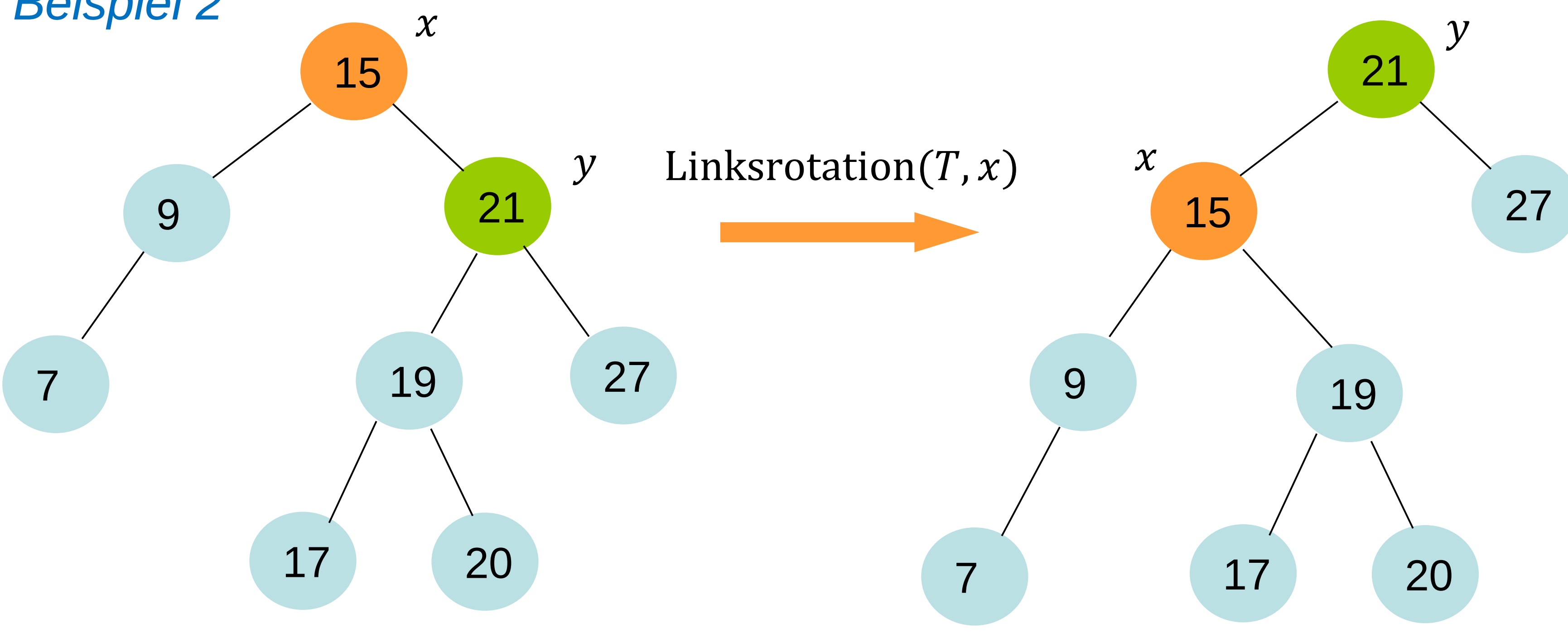
Datenstrukturen

Beispiel 1



Datenstrukturen

Beispiel 2



Datenstrukturen

$\text{Linksrotation}(T, x)$

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

Datenstrukturen

Annahme: x hat rechtes Kind.

$\mathrm{Linksrotation}(T, x)$

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

Datenstrukturen

Annahme: x hat rechtes Kind.

$\boxed{\text{Linksrotation}(T, x)}$

$y \leftarrow \text{rc}[x]$

$\text{rc}[x] \leftarrow \text{lc}[y]$

if $\text{lc}[y] \neq$ **nil then** $\text{p}\big[\text{lc}[y]\big] \leftarrow x$

$\text{p}[y] \leftarrow \text{p}[x]$

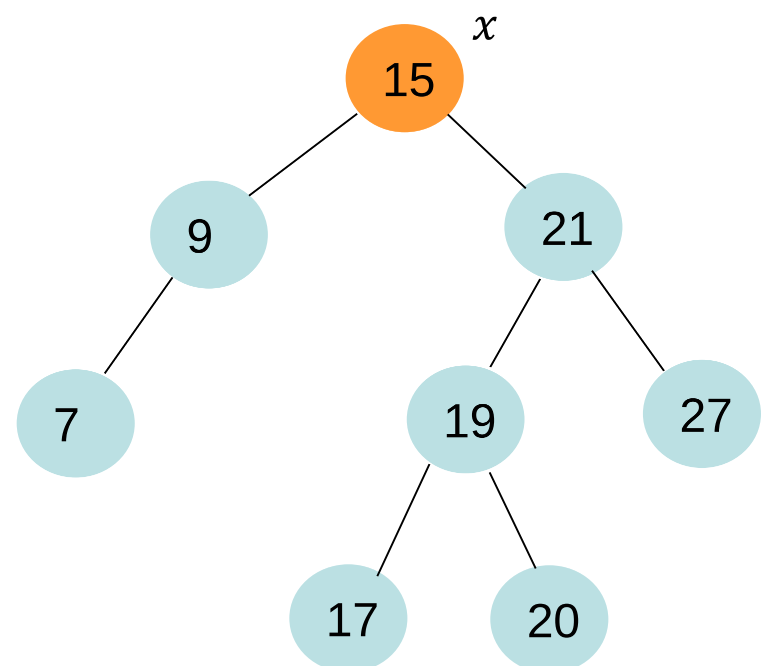
if $\text{p}[x] =$ **nil then** $\text{root}[T] \leftarrow y$

else if $x = \text{lc}\big[\text{p}[x]\big]$ **then** $\text{lc}\big[\text{p}[x]\big] \leftarrow y$

  else $\text{rc}\big[\text{p}[x]\big] \leftarrow y$

$\text{lc}[y] \leftarrow x$

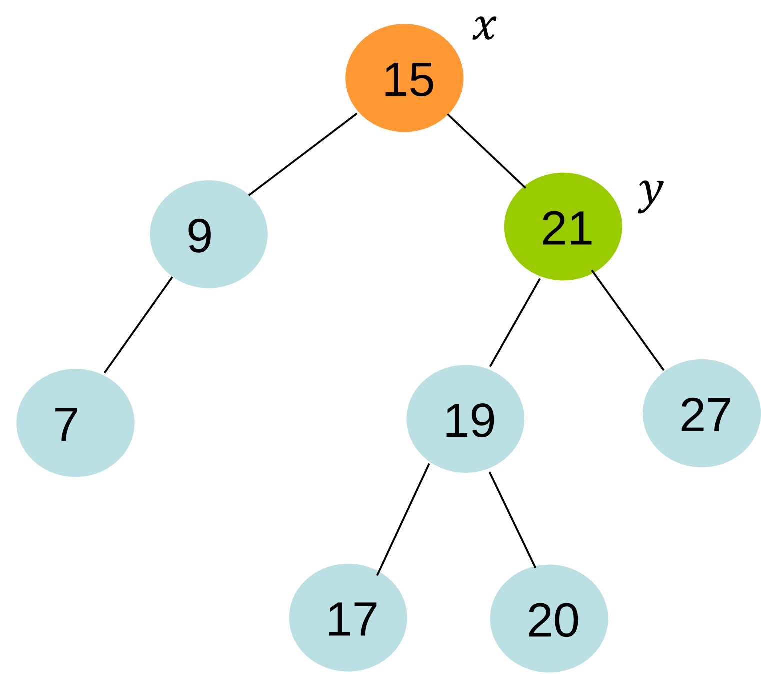
$\text{p}[x] \leftarrow y$



Datenstrukturen

$\text{Linksrotation}(T, x)$

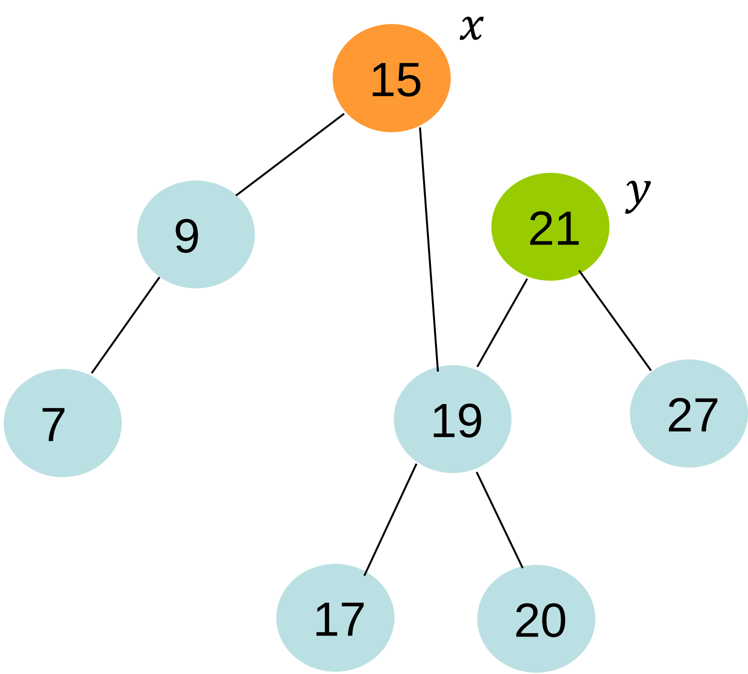
1. $\boxed{y \leftarrow \mathrm{rc}[x]}$
2. $\mathrm{rc}[x] \leftarrow \mathrm{lc}[y]$
3. $\textbf{if } \mathrm{lc}[y] \neq \textbf{nil then } \mathrm{p}\big[\mathrm{lc}[y]\big] \leftarrow x$
4. $\mathrm{p}[y] \leftarrow \mathrm{p}[x]$
5. $\textbf{if } \mathrm{p}[x] = \textbf{nil then } \mathrm{root}[T]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



Datenstrukturen

Linksrotation(T, x)

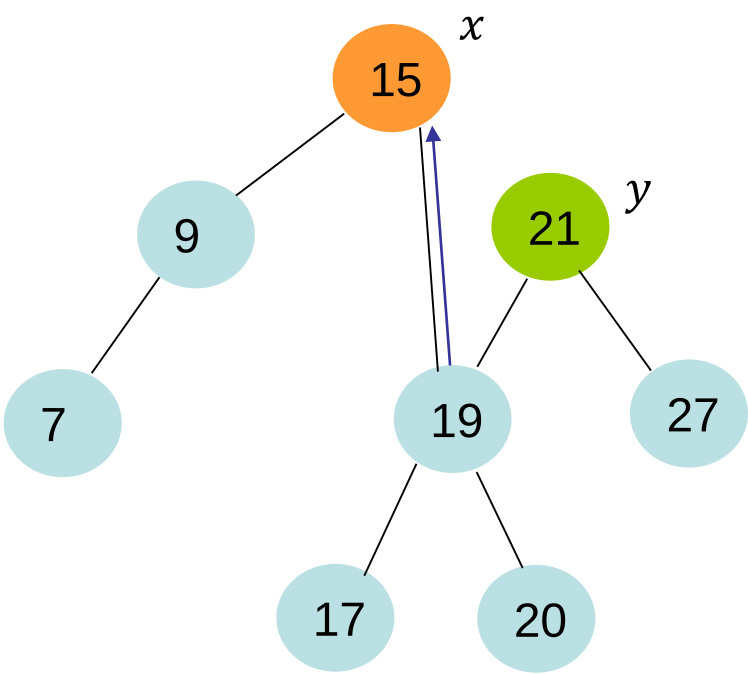
1. $y \leftarrow \mathrm{rc}[x]$
2. $\boxed{\mathrm{rc}[x] \leftarrow \mathrm{lc}[y]}$
3. **if** $\mathrm{lc}[y] \neq$ **nil then** $\mathrm{p}\big[\mathrm{lc}[y]\big] \leftarrow x$
4. $\mathrm{p}[y] \leftarrow \mathrm{p}[x]$
5. **if** $\mathrm{p}[x] =$ **nil then** $\mathrm{root}[T] \leftarrow y$
6.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



Datenstrukturen

Linksrotation(T, x)

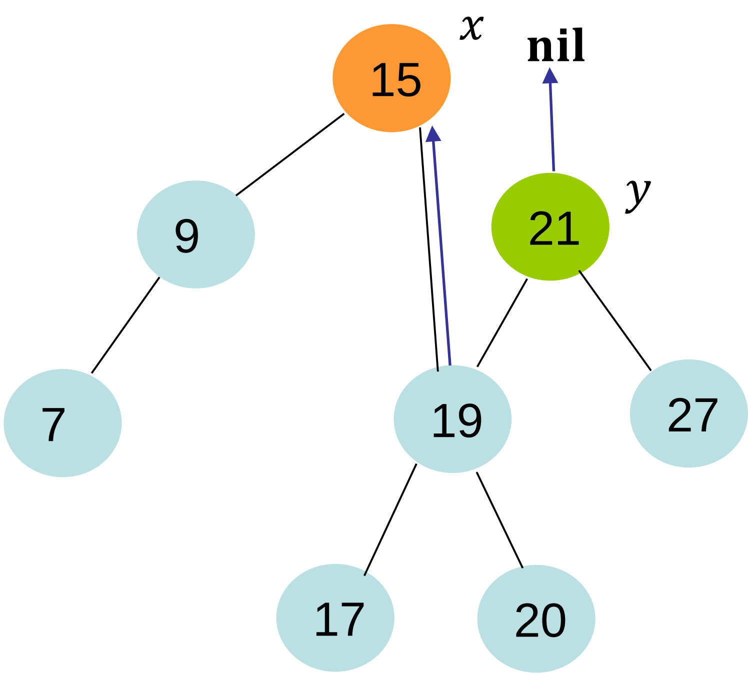
1. $y \leftarrow \text{rc}[x]$
2. $\text{rc}[x] \leftarrow \text{lc}[y]$
3. $\boxed{\textbf{if } \text{lc}[y] \neq \textbf{nil then } \text{p}\big[\text{lc}[y]\big] \leftarrow x}$
4. $\text{p}[y] \leftarrow \text{p}[x]$
5. $\textbf{if } \text{p}[x] = \textbf{nil then } \text{root}[T] \leftarrow y$
6.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



Datenstrukturen

Linksrotation(T, x)

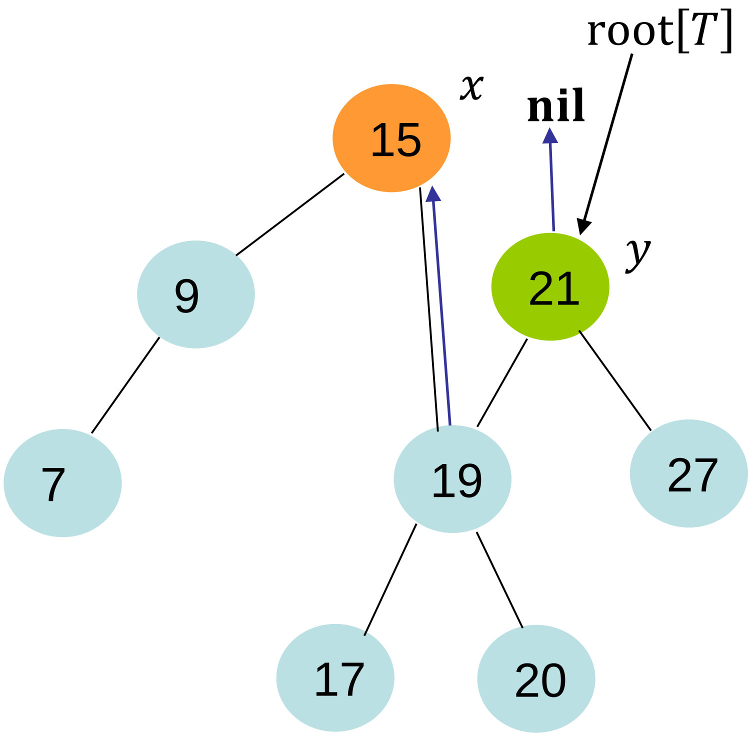
1. $y \leftarrow \mathrm{rc}[x]$
2. $\mathrm{rc}[x] \leftarrow \mathrm{lc}[y]$
3. **if** $\mathrm{lc}[y] \neq$ **nil then** $\mathrm{p}\big[\mathrm{lc}[y]\big] \leftarrow x$
4. $\boxed{\mathrm{p}[y] \leftarrow \mathrm{p}[x]}$
5. **if** $\mathrm{p}[x] =$ **nil then** $\mathrm{root}[T] \leftarrow y$
6.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



Datenstrukturen

Linksrotation(T, x)

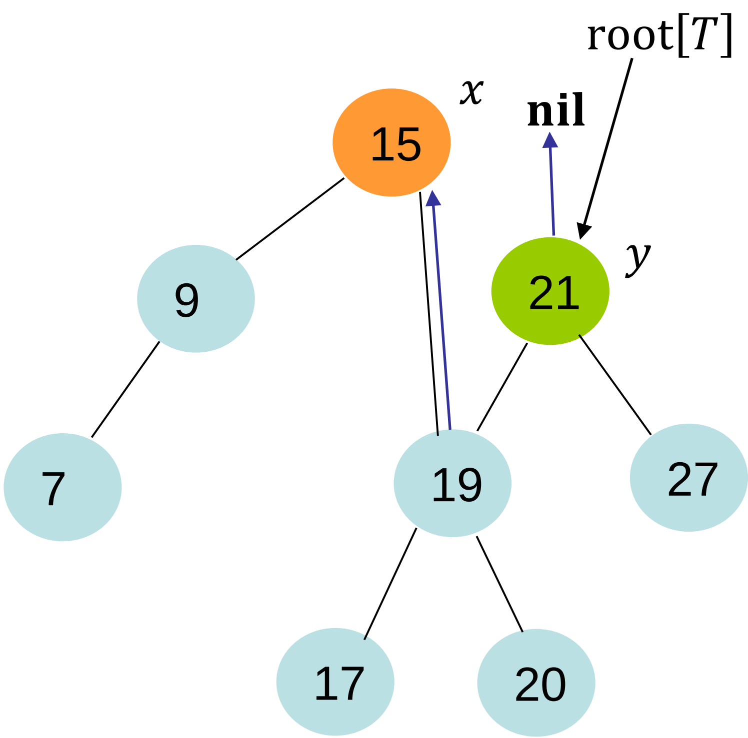
1. $y \leftarrow \text{rc}[x]$
2. $\text{rc}[x] \leftarrow \text{lc}[y]$
3. **if** $\text{lc}[y] \neq$ **nil then** $\text{p}\big[\text{lc}[y]\big] \leftarrow x$
4. $\text{p}[y] \leftarrow \text{p}[x]$
5. $\boxed{\textbf{if } \text{p}[x] = \textbf{nil then } \text{root}[T] \leftarrow y}$
6. **else if**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



Datenstrukturen

Linksrotation(T, x)

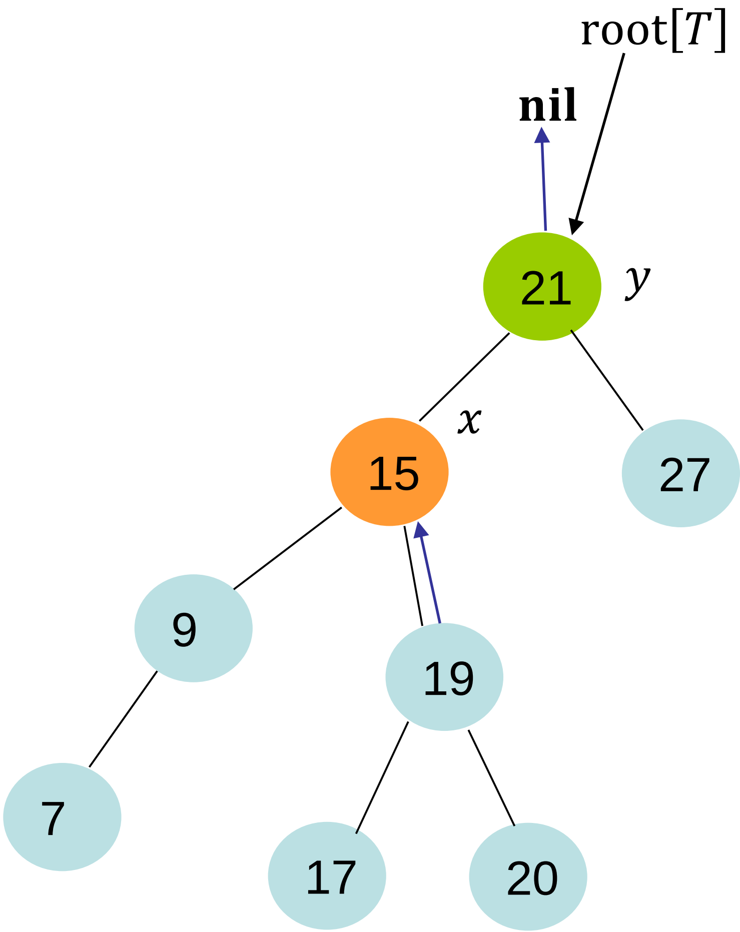
1. $y \leftarrow \mathrm{rc}[x]$
2. $\mathrm{rc}[x] \leftarrow \mathrm{lc}[y]$
3. **if** $\mathrm{lc}[y] \neq$ **nil then** $\mathrm{p}\big[\mathrm{lc}[y]\big] \leftarrow x$
4. $\mathrm{p}[y] \leftarrow \mathrm{p}[x]$
5. **if** $\mathrm{p}[x] =$ **nil then** $\mathrm{root}[T] \leftarrow y$
6. **else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



Datenstrukturen

Linksrotation(T, x)

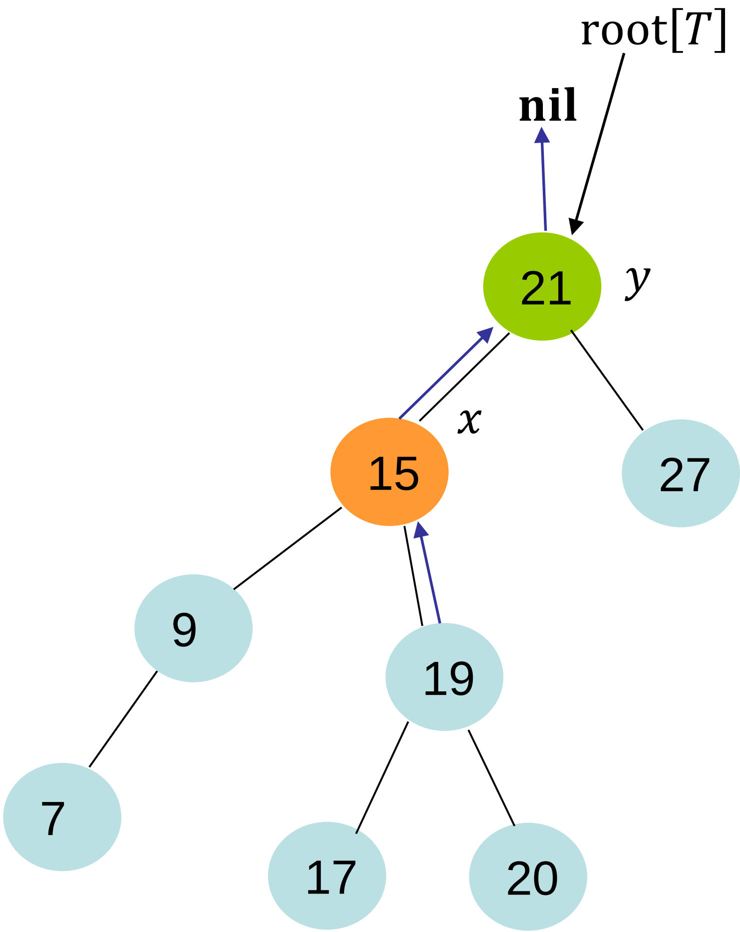
1. $y \leftarrow \mathrm{rc}[x]$
2. $\mathrm{rc}[x] \leftarrow \mathrm{lc}[y]$
3. **if** $\mathrm{lc}[y] \neq$ **nil then** $\mathrm{p}\big[\mathrm{lc}[y]\big] \leftarrow x$
4. $\mathrm{p}[y] \leftarrow \mathrm{p}[x]$
5. **if** $\mathrm{p}[x] =$ **nil then** $\mathrm{root}[T] \leftarrow y$
6. **else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



Datenstrukturen

Linksrotation(T, x)

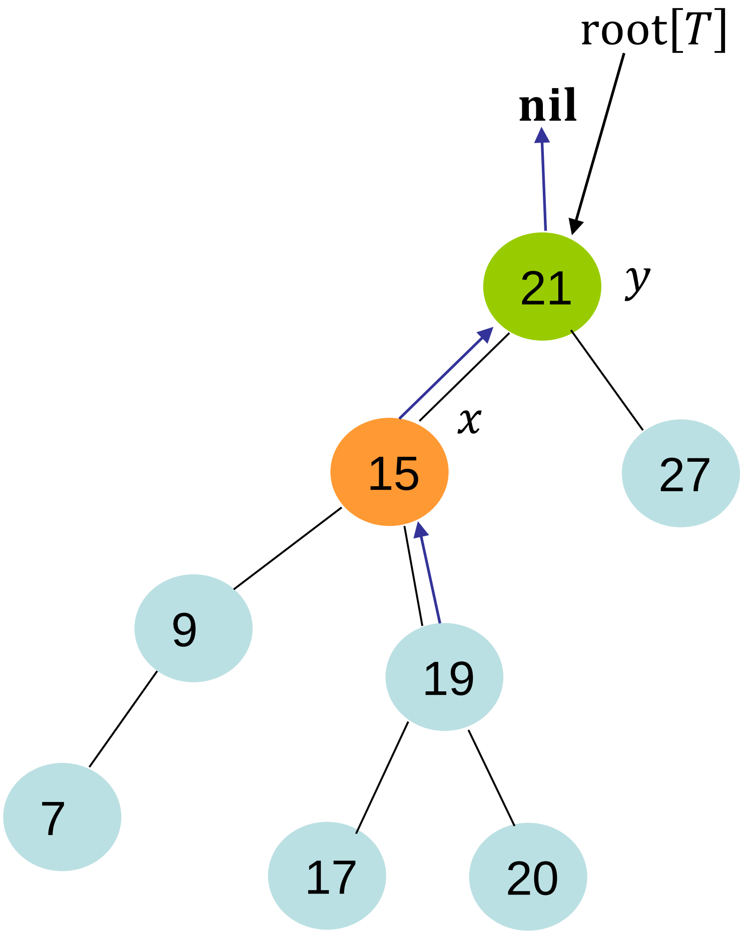
1. $y \leftarrow \mathrm{rc}[x]$
2. $\mathrm{rc}[x] \leftarrow \mathrm{lc}[y]$
3. **if** $\mathrm{lc}[y] \neq$ **nil then** $\mathrm{p}[\mathrm{lc}[y]] \leftarrow x$
4. $\mathrm{p}[y] \leftarrow \mathrm{p}[x]$
5. **if** $\mathrm{p}[x] =$ **nil then** $\mathrm{root}[T] \leftarrow y$
6. **else if** $x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



Datenstrukturen

Linksrotation(T, x)

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



Datenstrukturen

Dynamische AVL-Bäume

- Operationen Suche, Einfügen, Löschen, Min/Max, Vorgänger/Nachfolger,… wie in der letzten Vorlesung
- Laufzeit $\mathbf{O}(h)$ 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-B

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

- Operationen S... en, Min/Max,
Vorgänger/Nac... etzten Vorlesung
- Laufzeit $\mathbf{O}(h)$ für diese Operationen
- Nur Einfügen/Löschen verändern Struktur des Baums

Idee

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

Datenstrukturen

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.

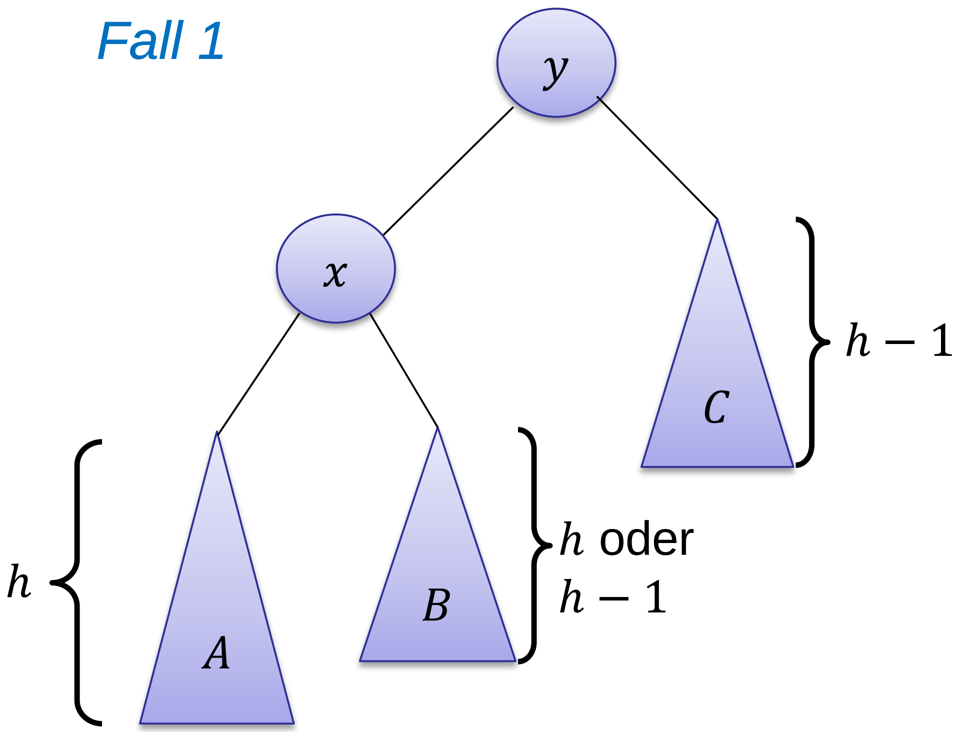
Datenstrukturen

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

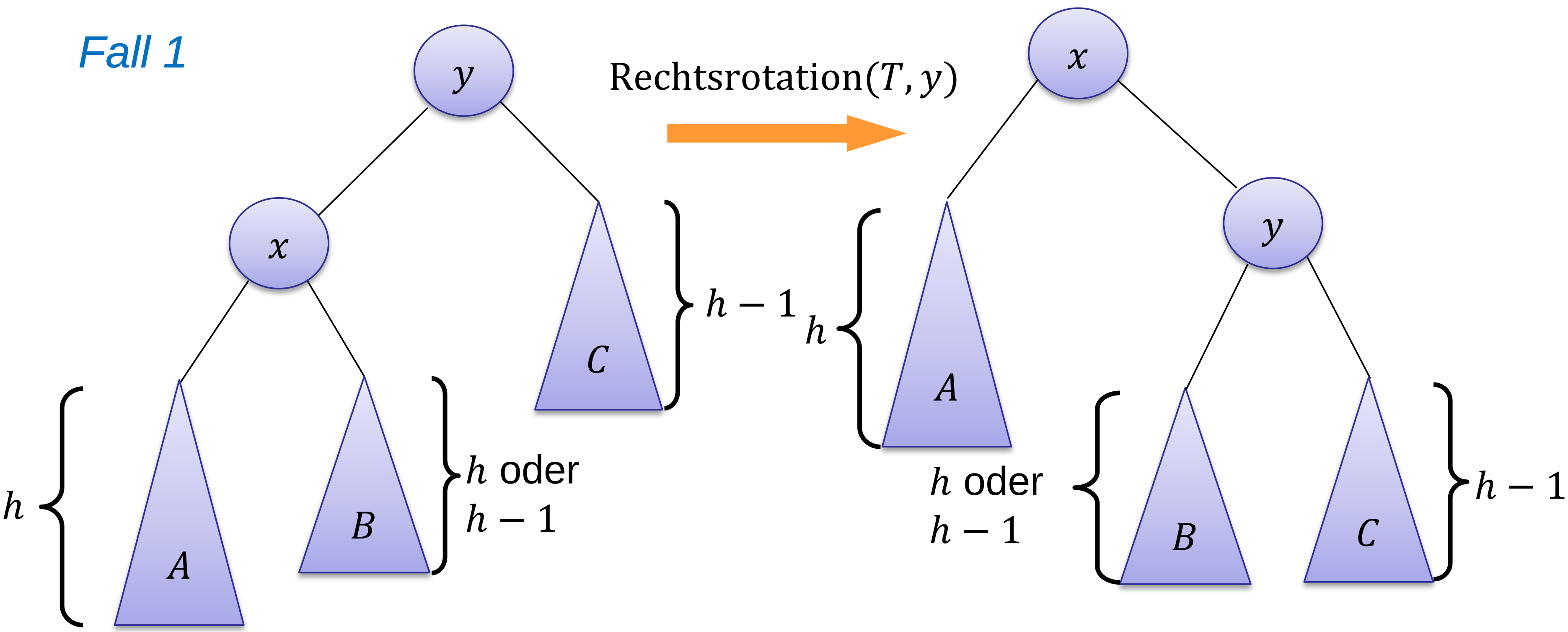
Datenstrukturen

Fall 1



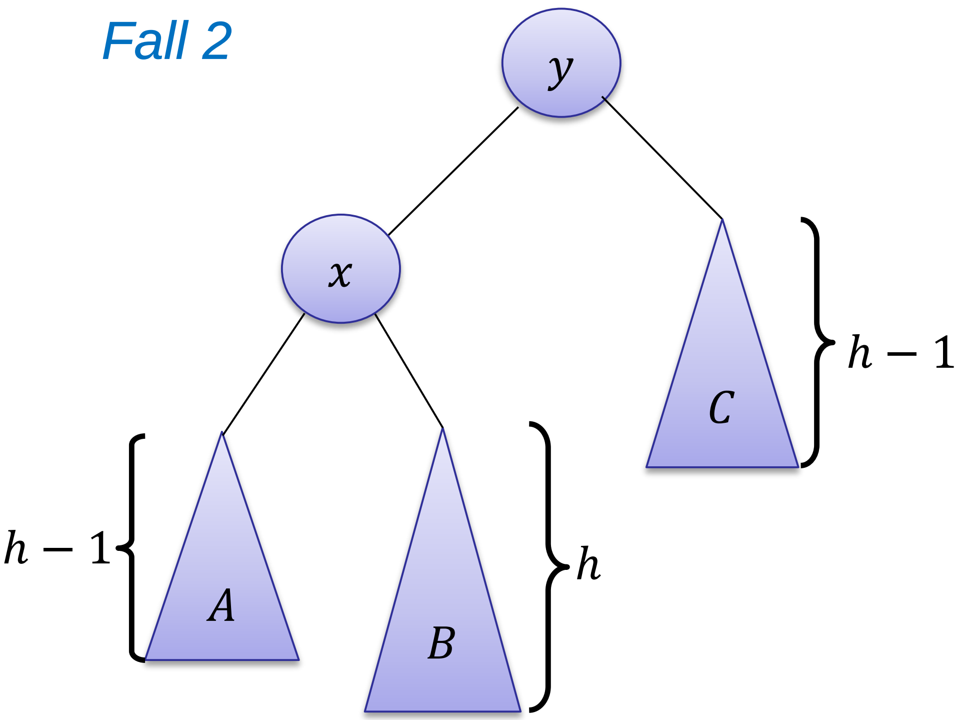
Datenstrukturen

Fall 1



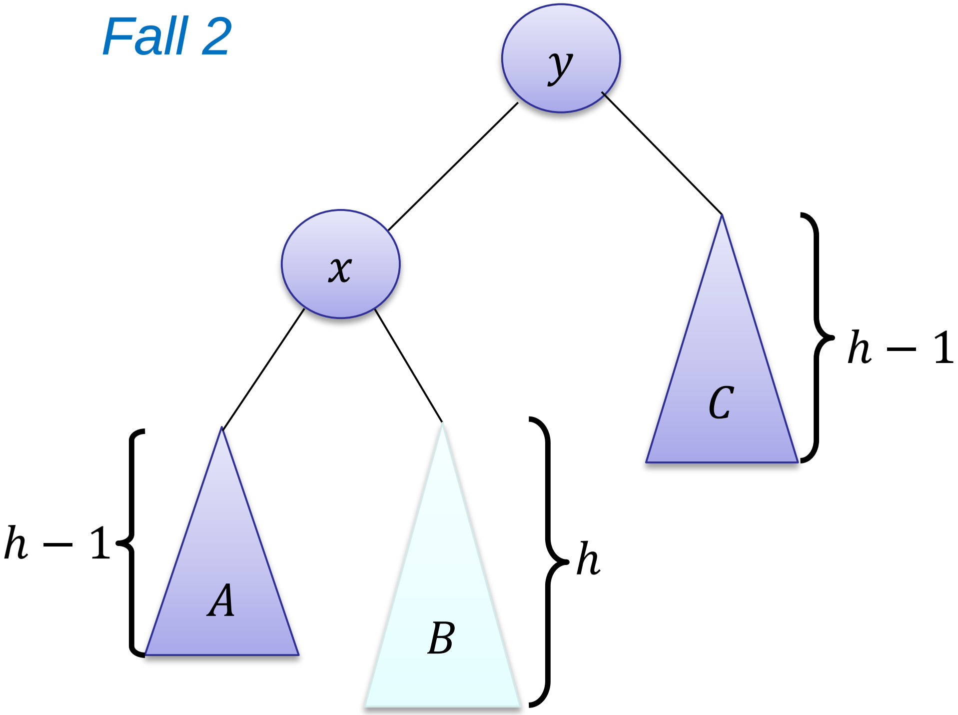
Datenstrukturen

Fall 2



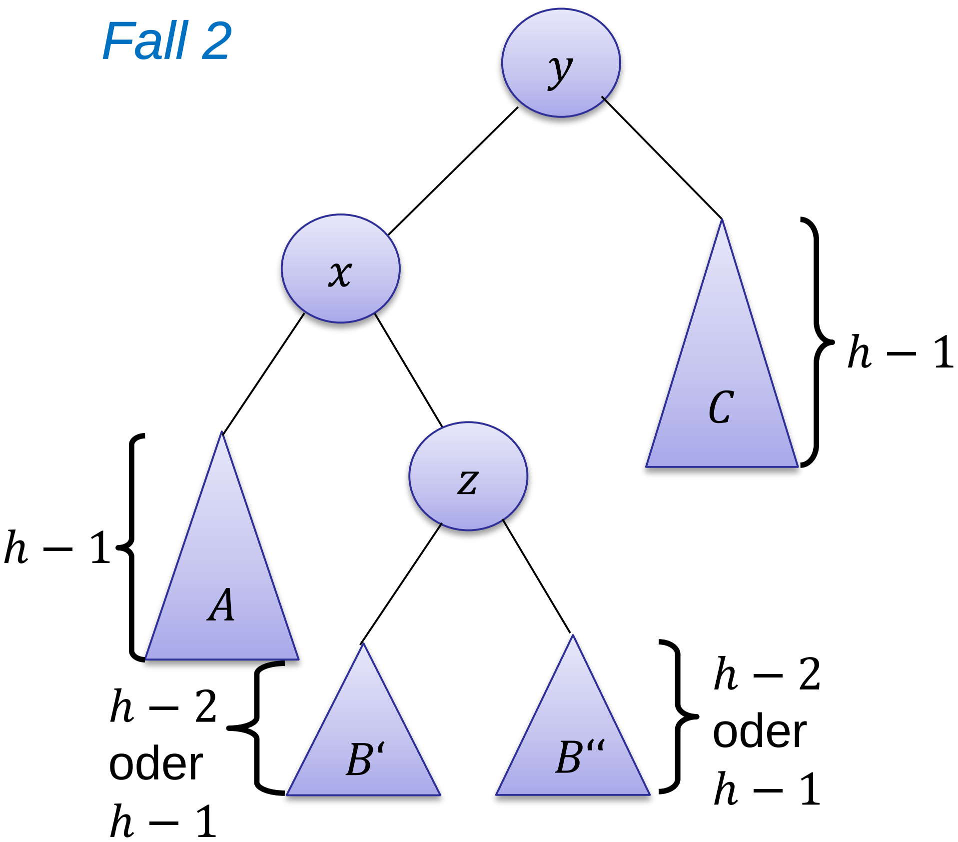
Datenstrukturen

Fall 2



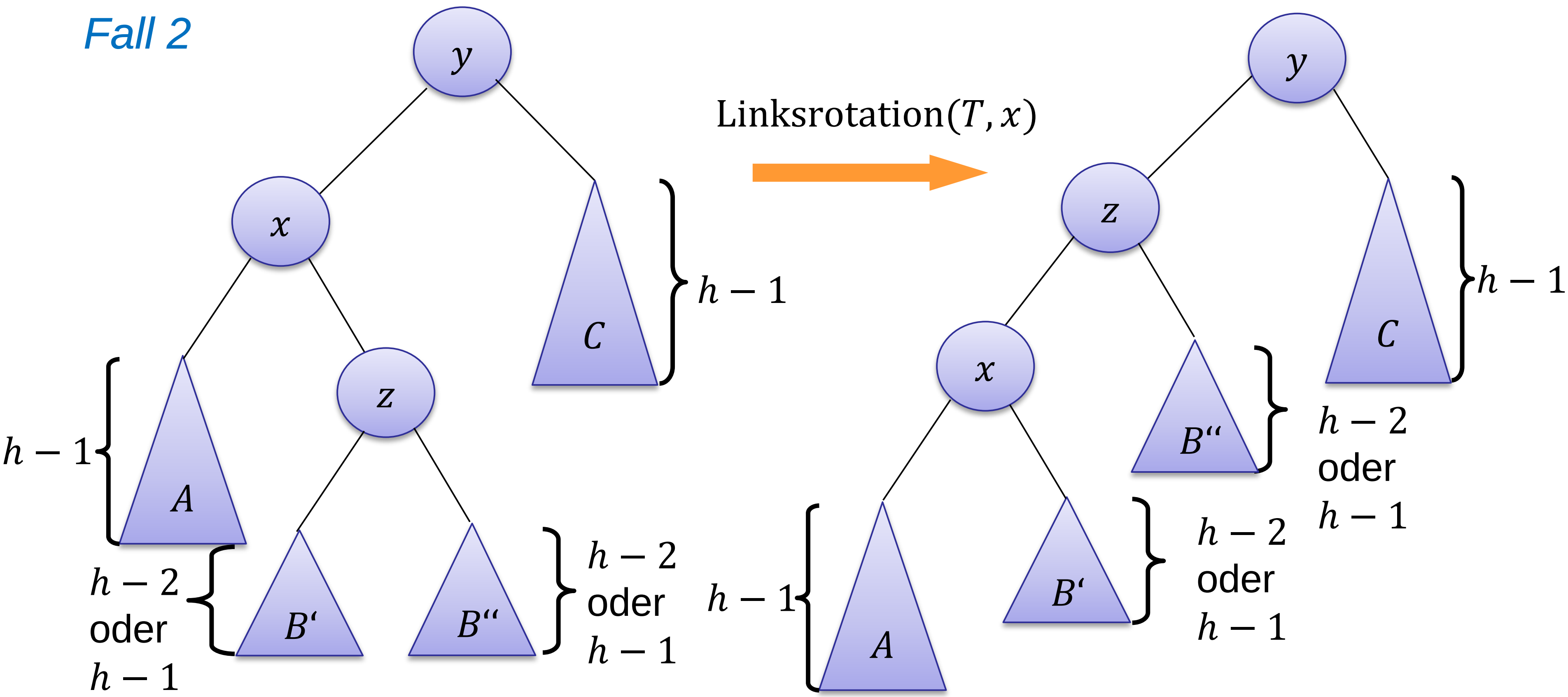
Datenstrukturen

Fall 2



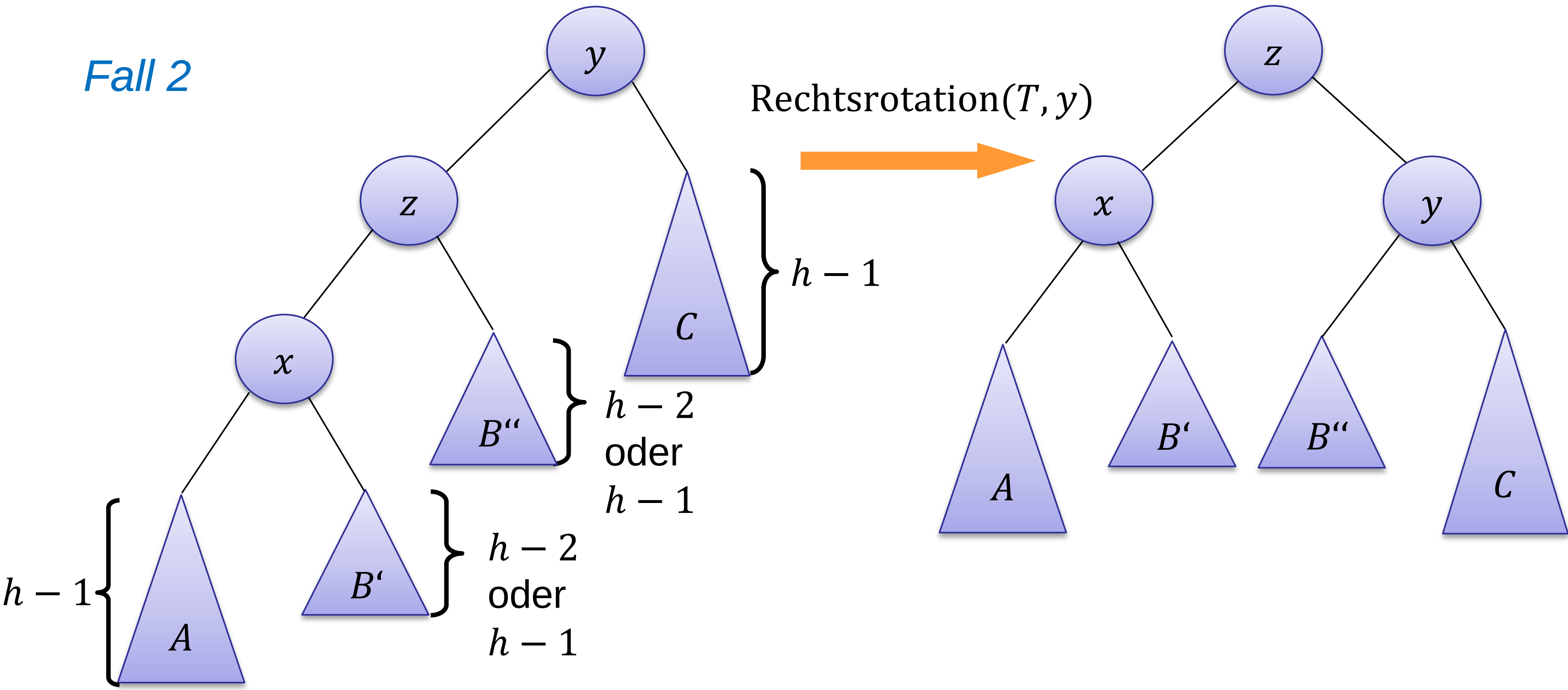
Datenstrukturen

Fall 2



Datenstrukturen

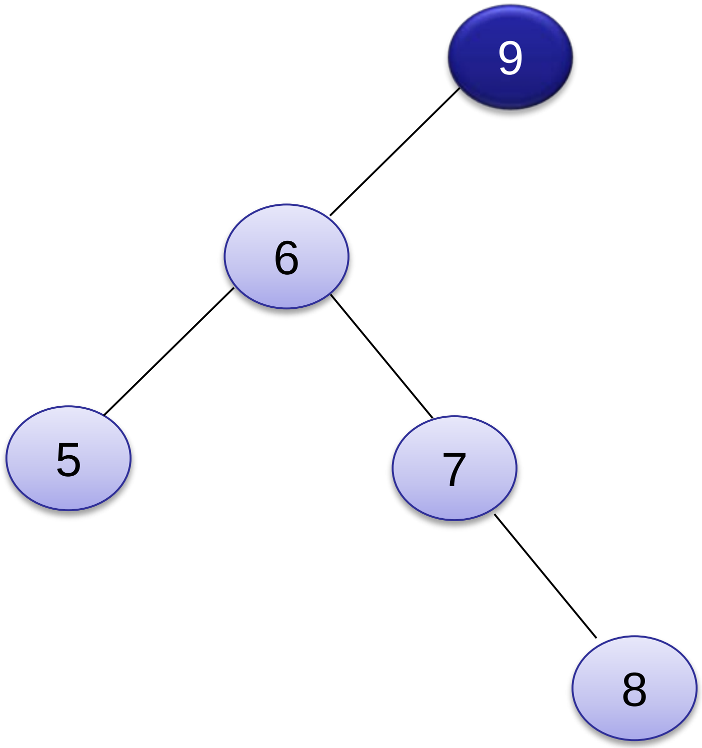
Fall 2



Datenstrukturen

Aufgabe

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



Datenstrukturen

$\text{Balance}(T)$

$t \leftarrow \text{root}[T]$

$\textbf{if } h[\text{lc}[t]] > h[\text{rc}[t]] + 1 \textbf{ then}$

$\quad \textbf{if } h[\text{lc}[\text{lc}[t]]] < h[\text{rc}[\text{lc}[t]]] \textbf{ then}$

$\quad\quad \text{Linksrotation}(T, \text{lc}[t])$

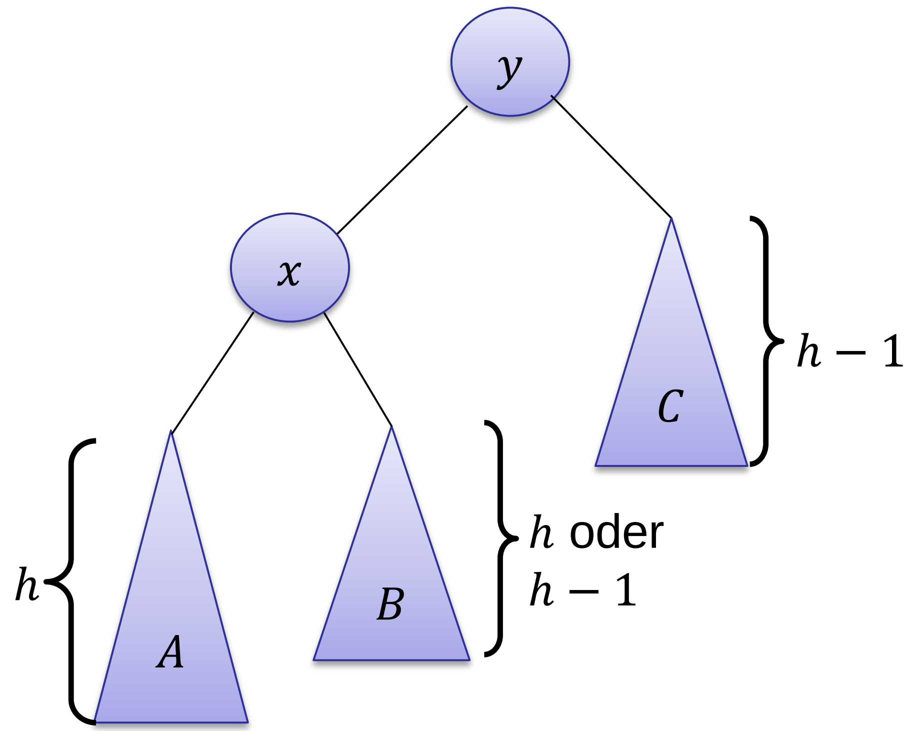
$\quad \text{Rechtsrotation}(T, t)$

$\textbf{else if } h[\text{rc}[t]] > h[\text{lc}[t]] + 1 \textbf{ then}$

$\quad \textbf{if } h[\text{rc}[\text{rc}[t]]] < h[\text{lc}[\text{rc}[t]]] \textbf{ then}$

$\quad\quad \text{Rechtsrotation}(T, \text{rc}[t])$

$\quad \text{Linksrotation}(T, t)$



Datenstrukturen

Balance(T)

$\boxed{t \leftarrow \text{root}[T]}$

if $h[\text{lc}[t]] > h[\text{rc}[t]] + 1$ **then**

    if $h[\text{lc}[\text{lc}[t]]] < h[\text{rc}[\text{lc}[t]]]$ **then**

        Linksrotation($T, \text{lc}[t]$)

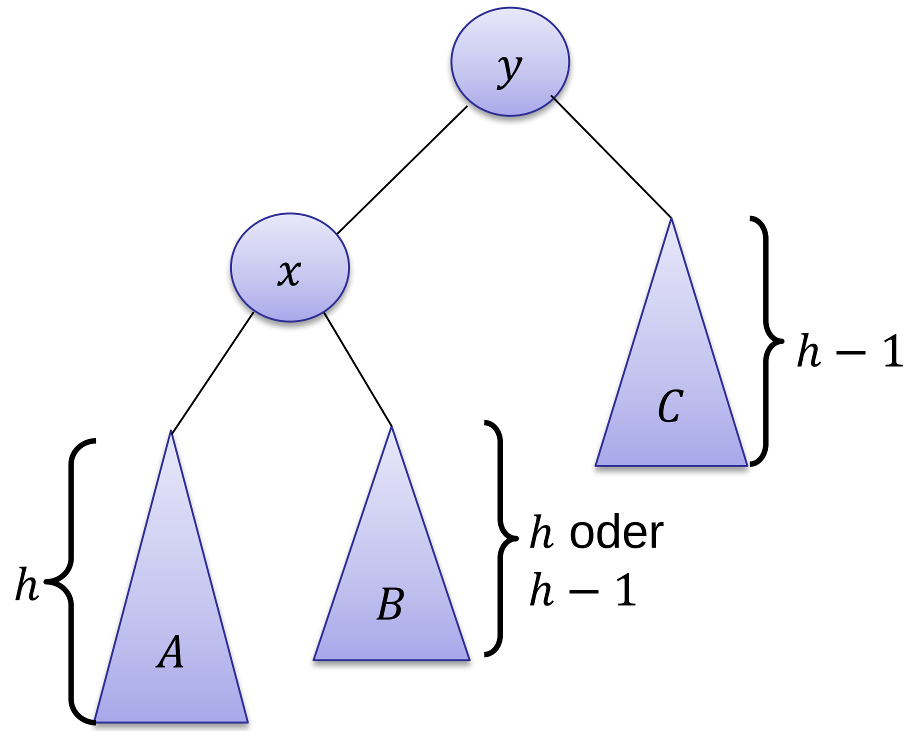
    Rechtsrotation(T, t)

else if $h[\text{rc}[t]] > h[\text{lc}[t]] + 1$ **then**

    if $h[\text{rc}[\text{rc}[t]]] < h[\text{lc}[\text{rc}[t]]]$ **then**

        Rechtsrotation($T, \text{rc}[t]$)

    Linksrotation(T, t)

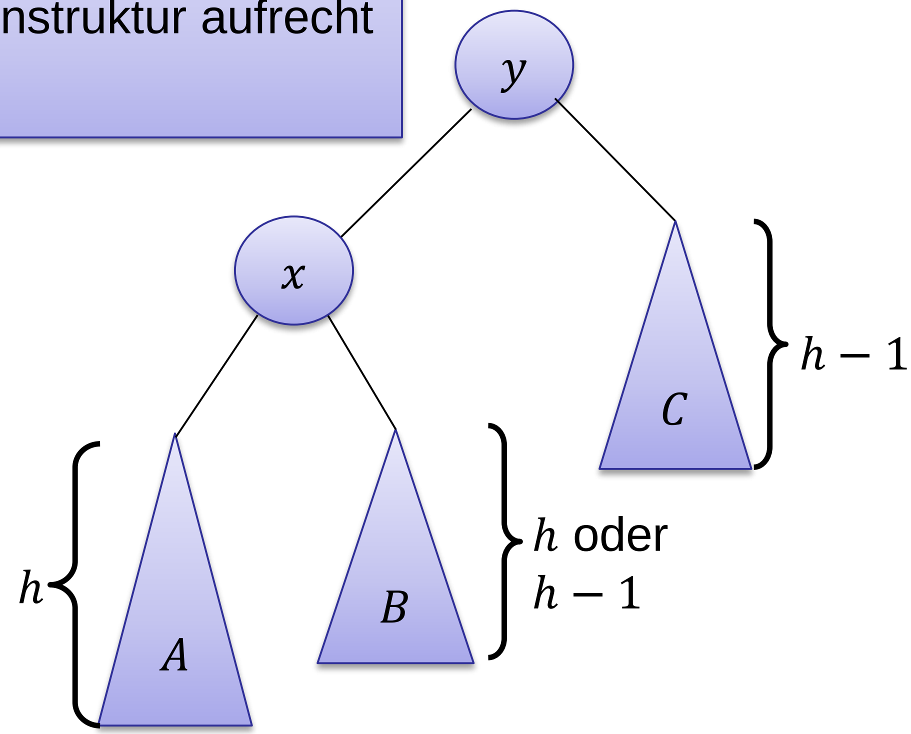


Datenstrukturen

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

Balance(T)

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



Datenstrukturen

Balance(T)

$t \leftarrow \text{root}[T]$

if $h[\text{lc}[t]] > h[\text{rc}[t]] + 1$ **then**

  if $h\left[\text{lc}[\text{lc}[t]]\right] < h\left[\text{rc}[\text{lc}[t]]\right]$ **then**

    Linksrotation($T, \text{lc}[t]$)

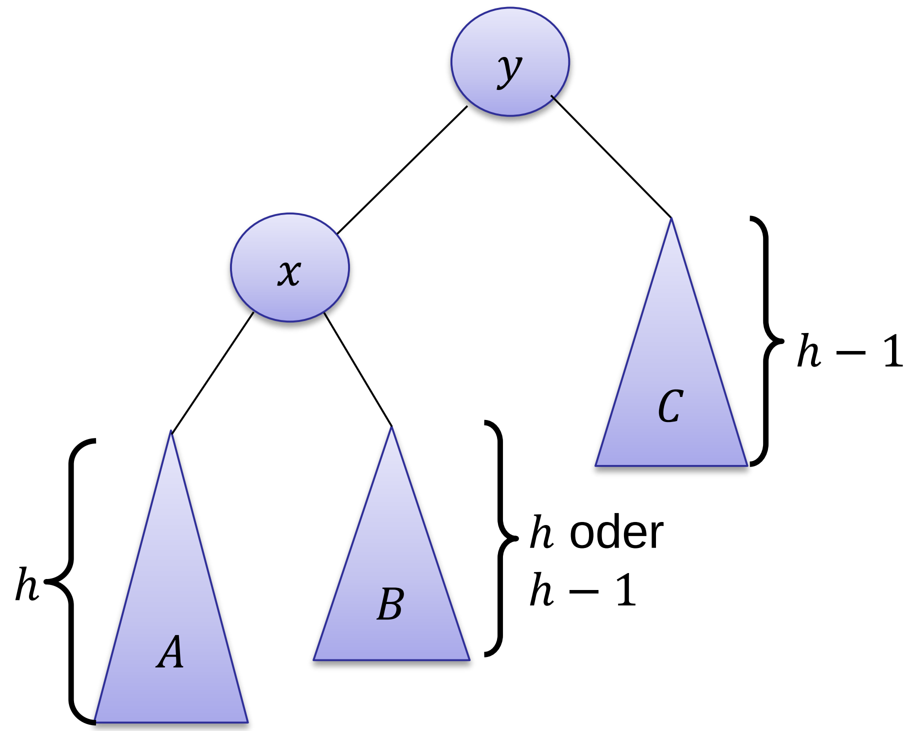
  Rechtsrotation(T, t)

else if $h[\text{rc}[t]] > h[\text{lc}[t]] + 1$ **then**

  if $h\left[\text{rc}[\text{rc}[t]]\right] < h[\text{lc}[\text{rc}[t]]]$ **then**

    Rechtsrotation($T, \text{rc}[t]$)

  Linksrotation(T, t)



Datenstrukturen

Balance(T)

$t \leftarrow \mathrm{root}[T]$

if $h[\mathrm{lc}[t]] > h[\mathrm{rc}[t]] + 1$ **then**

    if $h\left[\mathrm{lc}[\mathrm{lc}[t]]\right] < h\left[\mathrm{rc}[\mathrm{lc}[t]]\right]$ **then**

        Linksrotation($T, \mathrm{lc}[t]$)

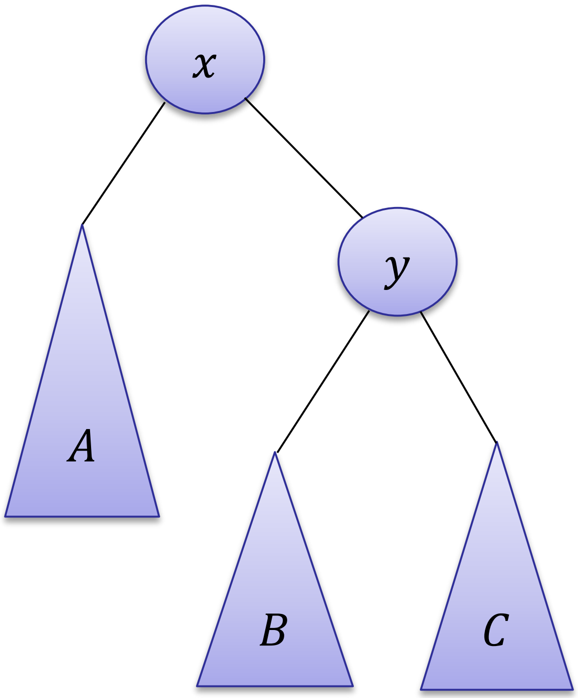
    $\boxed{\text{Rechtsrotation}(T, t)}$

else if $h[\mathrm{rc}[t]] > h[\mathrm{lc}[t]] + 1$ **then**

    if $h\left[\mathrm{rc}[\mathrm{rc}[t]]\right] < h[\mathrm{lc}[\mathrm{rc}[t]]]$ **then**

        Rechtsrotation($T, \mathrm{rc}[t]$)

    Linksrotation(T, t)

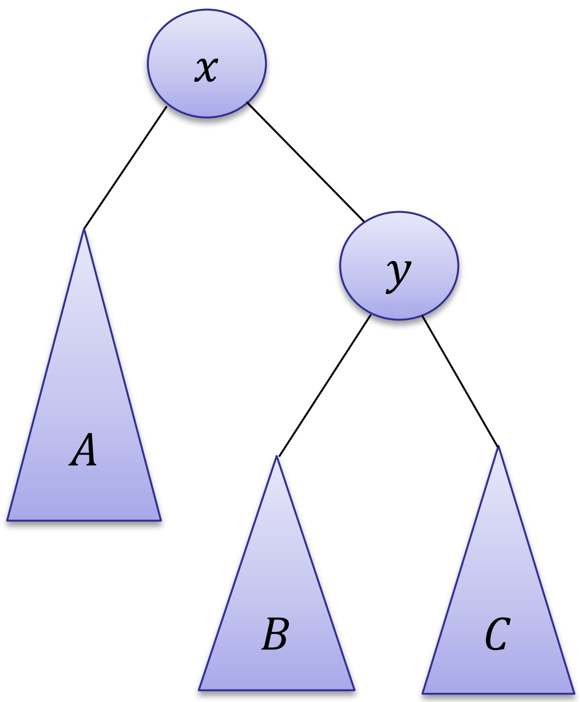


- Laufzeit: $\mathbf{O}(1)$

Datenstrukturen

Balance(T)

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

if $t = $ **nil then**

  $t \leftarrow$ **new** node(x); $h[t] \leftarrow 0$; **return**

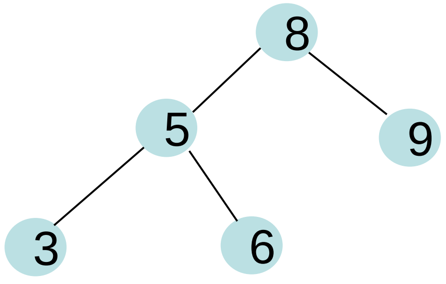
else if $\mathrm{key}[x] < \mathrm{key}[t]$ **then** AVL-Einfügen$(\mathrm{lc}[t], x)$

else if $\mathrm{key}[x] > \mathrm{key}[t]$ **then** AVL-Einfügen$(\mathrm{rc}[t], x)$

else return ➢ Schlüssel schon vorhanden

$h[t] \leftarrow 1 + \max\{h[\mathrm{lc}[t]], h[\mathrm{rc}[t]]\}$

Balance(t)



Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

if $t = $ **nil then**

    $t \leftarrow$ **new** node(x); $h[t] \leftarrow 0$; **return**

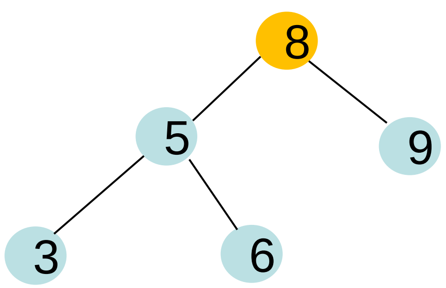
else if key$[x] <$ key$[t]$ **then** AVL-Einfügen$(\mathrm{lc}[t], x)$

else if key$[x] >$ key$[t]$ **then** AVL-Einfügen$(\mathrm{rc}[t], x)$

else return ➤ Schlüssel schon vorhanden

$h[t] \leftarrow 1 + \max\{h[\mathrm{lc}[t]], h[\mathrm{rc}[t]]\}$

Balance(t)

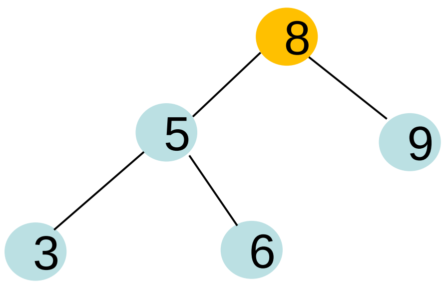


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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

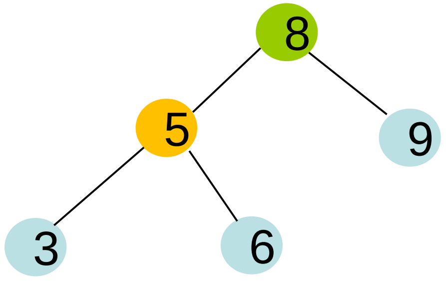


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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

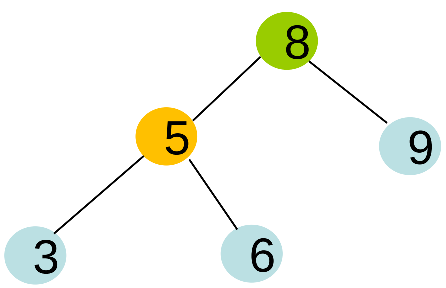


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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

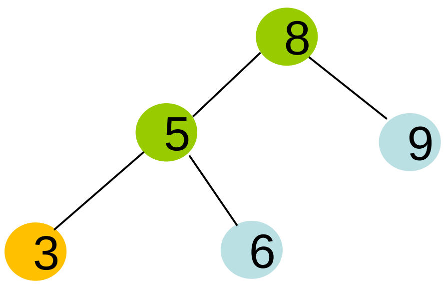


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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

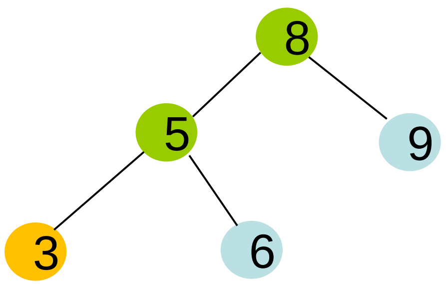


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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

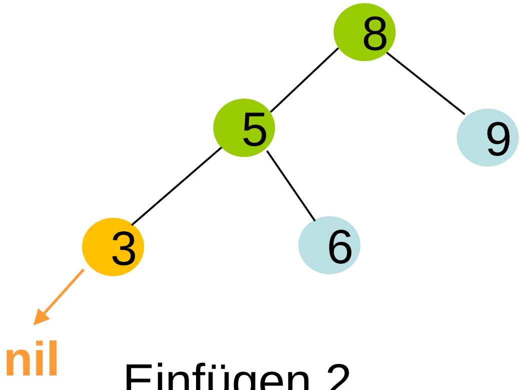


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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



Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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

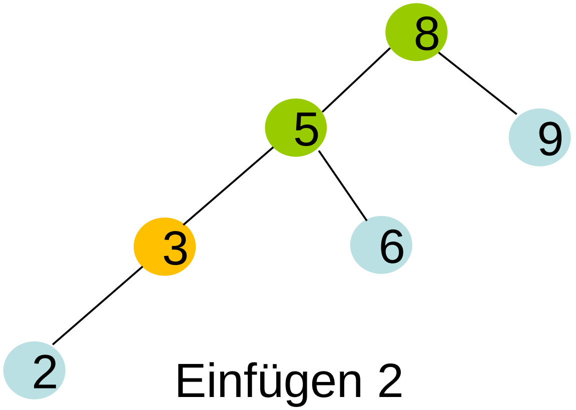
Neuen Knoten erzeugen. Zusätzlich noch Zeiger $\mathrm{lc}[t]$ und $\mathrm{rc}[t]$ auf **nil** setzen, sowie $p[t]$ und den Zeiger von $p[t]$ setzen.

Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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

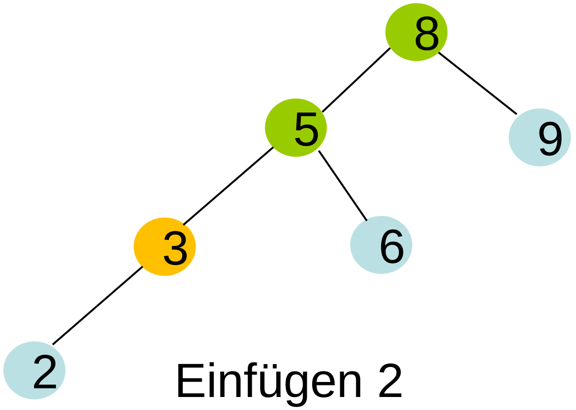


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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



Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

$\textbf{if } t = \textbf{nil then}$

$\quad t \leftarrow \textbf{new } \mathrm{node}(x);\ h[t] \leftarrow 0;\ \textbf{return}$

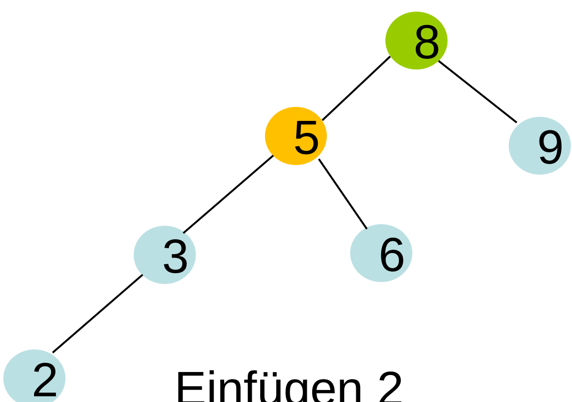
$\textbf{else if } \mathrm{key}[x] < \mathrm{key}[t] \textbf{ then } \text{AVL-Einfügen}(\mathrm{lc}[t], x)$

$\textbf{else if } \mathrm{key}[x] > \mathrm{key}[t] \textbf{ then } \text{AVL-Einfügen}(\mathrm{rc}[t], x)$

$\textbf{else return}$ ➢ Schlüssel schon vorhanden

$h[t] \leftarrow 1 + \max\{h[\mathrm{lc}[t]], h[\mathrm{rc}[t]]\}$

$\boxed{\mathrm{Balance}(t)}$

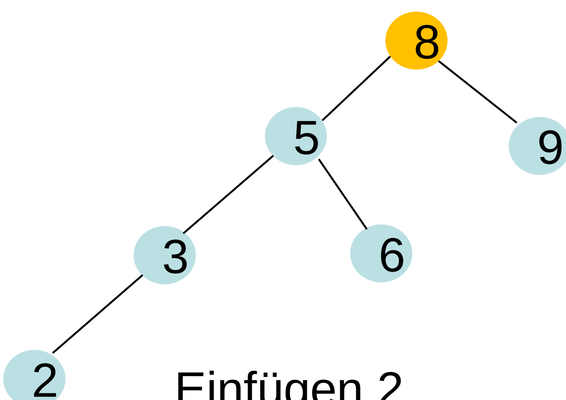


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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

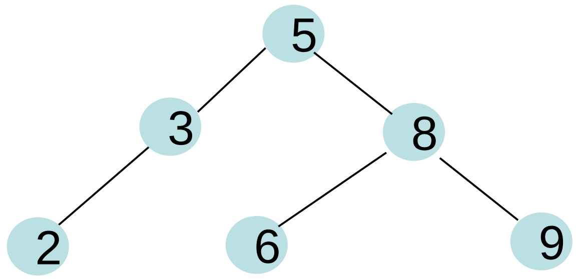


Einfügen 2

Datenstrukturen

AVL-Einfügen(t, x)

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



Einfügen 2

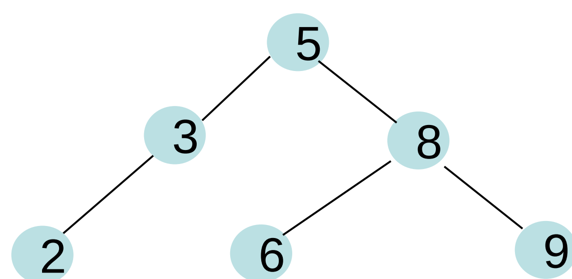
Datenstrukturen

AVL-Einfügen(t, x)

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

Laufzeit

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



Einfügen 2

Datenstrukturen

Satz 36

Wird mit $\mathrm{AVL\text{-}Einfügen}$ 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 Ele
so ist der resultierende Baum

AVL-Einfügen(t, x)

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

Beweis

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

Datenstrukturen

$\mathrm{AVL\text{-}Einfügen}(t, x)$

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

Satz 36

Wird mit $\mathrm{AVL\text{-}Einfügen}$ ein Ele
so ist der resultierende Baum

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

AVL-Einfügen(t, x)

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

Satz 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

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 Ele
so ist der resultierende Baum

AVL-Einfügen(t, x)

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

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

AVL-Einfügen(t, x)

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

Satz 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

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. $\mathrm{key}[x] < \mathrm{key}[t]$ (der andere Fall ist symmetrisch).
- Da $h+1 \geq 0$ ist, wird Zeile (3) ausgeführt.

Datenstrukturen

AVL-Einfügen(t, x)
1. **if** $t = $ **nil then**
2. $\quad t \leftarrow$ **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 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

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

$\text{AVL-Einfügen}(t, x)$

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

Satz 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

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

AVL-Einfügen(t, x)

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

Satz 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

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

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

Satz 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

Beweis

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

Datenstrukturen

Satz 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

$\text{AVL-Einfügen}(t, x)$

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

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.

technische universität
dortmund

Datenstrukturen

AVL-Einfügen(t, x)

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

Satz 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

Beweis

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

AVL-Einfügen(t, x)
1. **if** t = **nil then**
2. $\quad t \leftarrow$ **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 36

Wird mit AVL-Einfügen ein Ele
so ist der resultierende Baum

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 Ele
so ist der resultierende Baum

AVL-Einfügen(t, x)

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

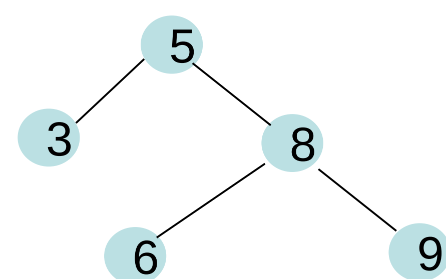
Löschen

- Wir löschen 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-Löschen(t, x)

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



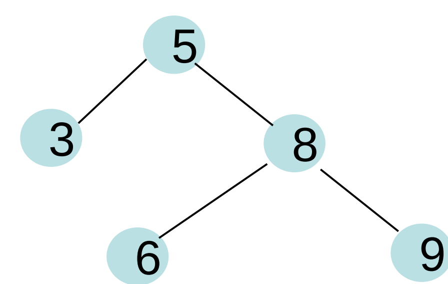
Datenstrukture

x bezeichnet Schlüssel des zu löschenden Elements.

$\text{AVL-Löschen}(t, x)$

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

$\text{Löschen}(3)$

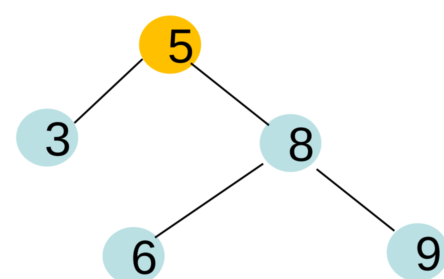


Datenstrukturen

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** $t =$ **nil then return** ➢ x nicht im Baum
4. **else if** $\text{lc}[t] =$ **nil then** ersetze t durch $\text{rc}[t]$
5. **else if** $\text{rc}[t] =$ **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$(\text{lc}[t], \text{key}[u])$
9. **if** $t \neq$ **nil then** $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
10. Balance(t)

Löschen(3)

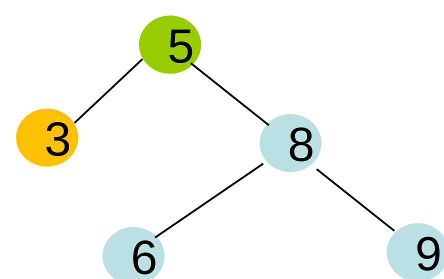


Datenstrukturen

AVL-Löschen(t, x)

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

Löschen(3)



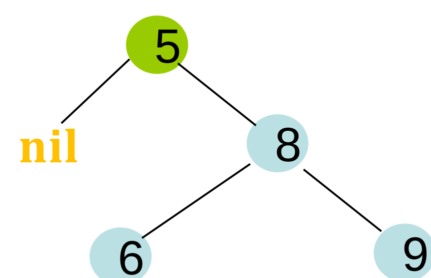
Datenstrukturen

AVL-Löschen(t, x)

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

Und die anderen Zeiger aktualisieren

Löschen(3)

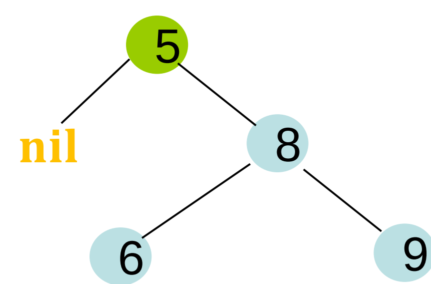


Datenstrukturen

AVL-Löschen(t, x)

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

Löschen(3)



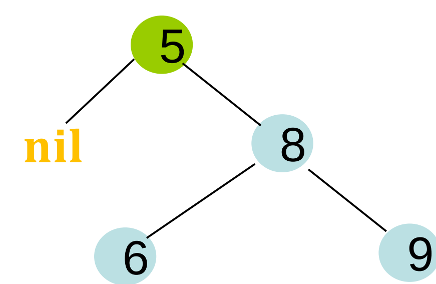
Datenstrukturen

AVL-Löschen(t, x)

```
if x < key[t] then AVL-Löschen(lc[t], x)
else if x > key[t] then AVL-Löschen(rc[t], x)
else if t = nil then return ➢ x nicht im Baum
else if lc[t] = nil then ersetze t durch rc[t]
else if rc[t] = nil then ersetze t durch lc[t]
else u = MaximumSuche(lc[t])
        Kopier… …on u nach t
        AVL-Lö… …])
if t ≠ nil th… …ax{h[lc[t]], h[rc[t]]}
Balance(t)
```

Nichts zu tun, da Baum leer.

Löschen(3)

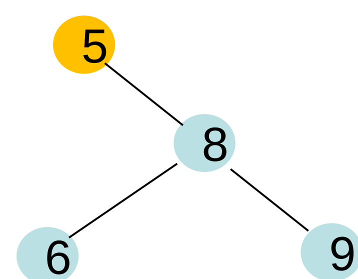


Datenstrukturen

AVL-Löschen(t, x)

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

Löschen(3)

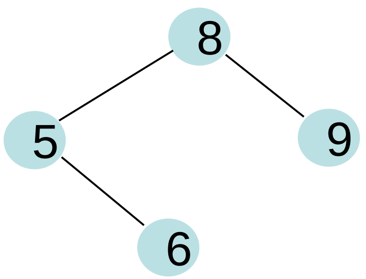


Datenstrukturen

AVL-Löschen(t, x)

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

Löschen(3)

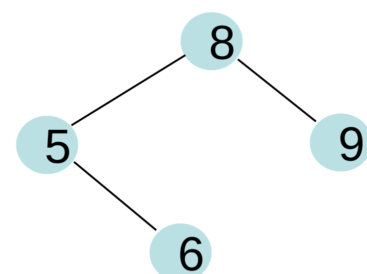


Datenstrukturen

AVL-Löschen(t, x)

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

Löschen(3)



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 $\mathbf{\Theta}(\log n)$ 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?