



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 $O(n)$
- Suche, Minimum, Maximum, Nachfolger in $O(h)$
- Einfügen, Löschen in $O(h)$

Frage

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

Datenstrukturen

Satz 38

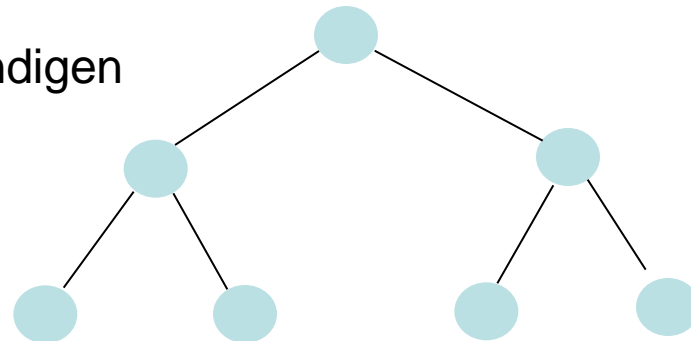
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

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 38

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

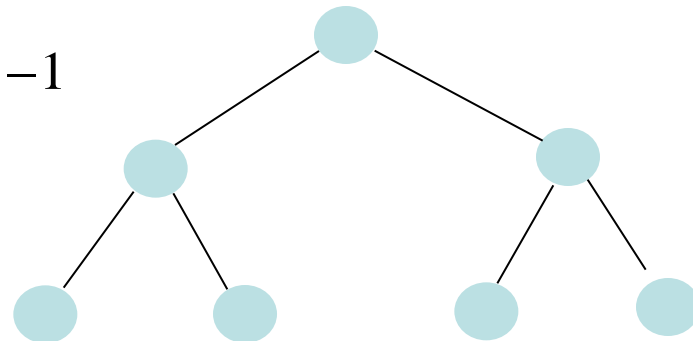
$$\left(\frac{3}{2}\right)^h \leq n \leq 2^{h+1} - 1$$

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 38

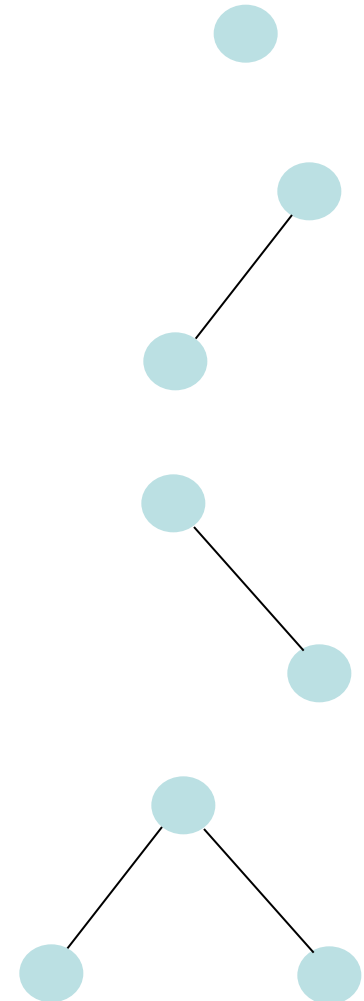
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

Beweis

b) $\left(\frac{3}{2}\right)^h \leq n$:

- 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 $\left(\frac{3}{2}\right)^0 = 1 \leq 1$.
- $h=1$: Der Baum hat 2 oder 3 Knoten. Es gilt $\left(\frac{3}{2}\right)^1 = 1.5 \leq 2 \leq 3$.



Datenstrukturen

Satz 38

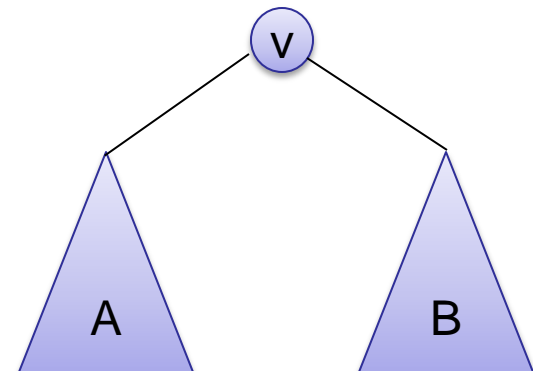
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

Beweis

b) $\left(\frac{3}{2}\right)^h \leq n$:

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



Datenstrukturen

Satz 38

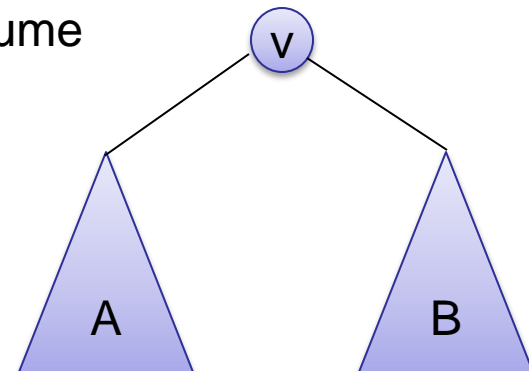
Für jeden AVL-Baum der Höhe $h \geq 0$ mit n Knoten gilt:

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

Beweis

b) $\left(\frac{3}{2}\right)^h \leq n$:

- Wegen AVL-Eigenschaft haben A und B Tiefe mindestens $h-1 \geq 0$.
- Da T ein AVL-Baum ist, sind auch A und B AVL-Bäume.
- Kann also (I.V.) anwenden, da A und B AVL-Bäume der Tiefe ≥ 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 38

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

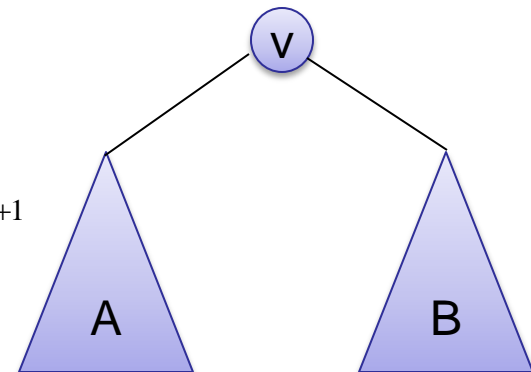
$$\left(\frac{3}{2}\right)^h \leq n \leq 2^{h+1} - 1$$

Beweis

b) $\left(\frac{3}{2}\right)^h \leq n$:

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

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



Datenstrukturen

Satz 38

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

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

Korollar 38

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

Beweis

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

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

Datenstrukturen

Satz 38

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

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

Korollar 38

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

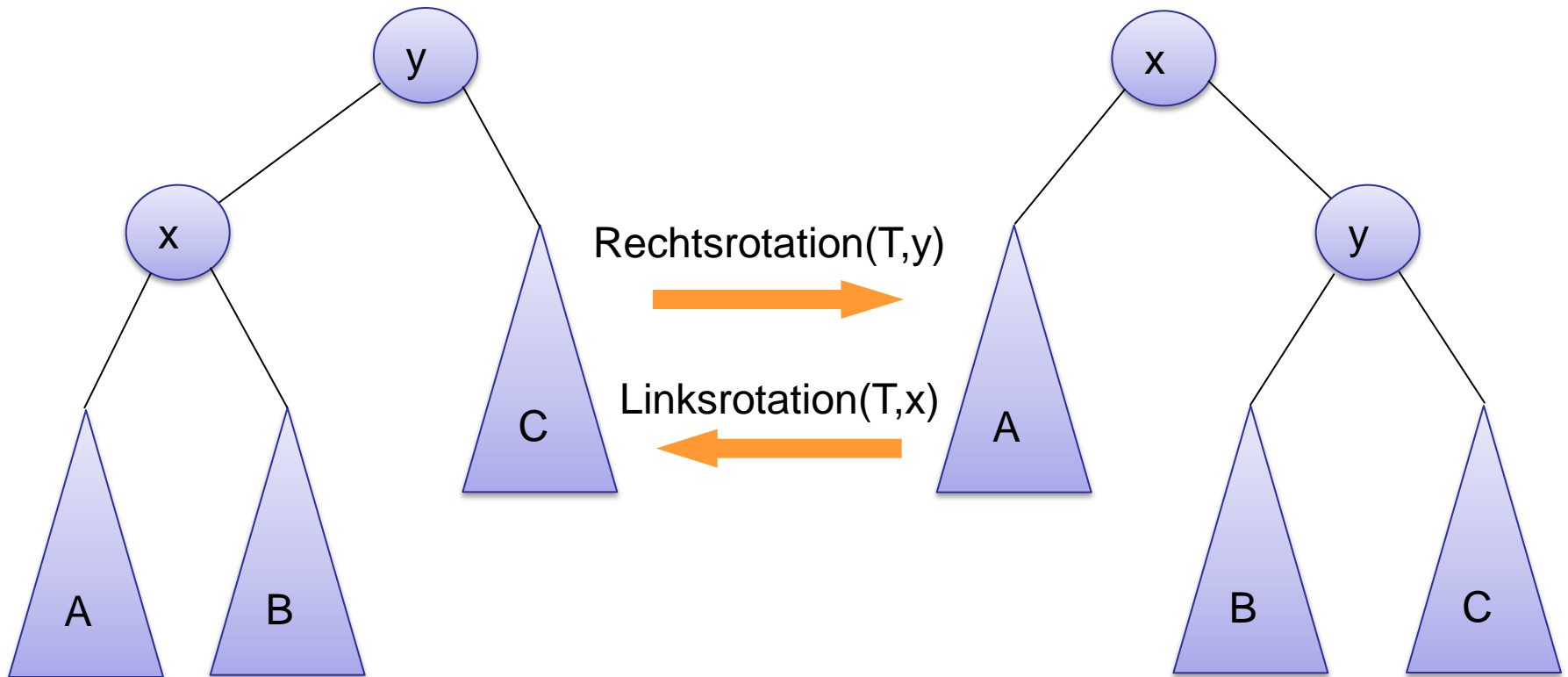
Beweis

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

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

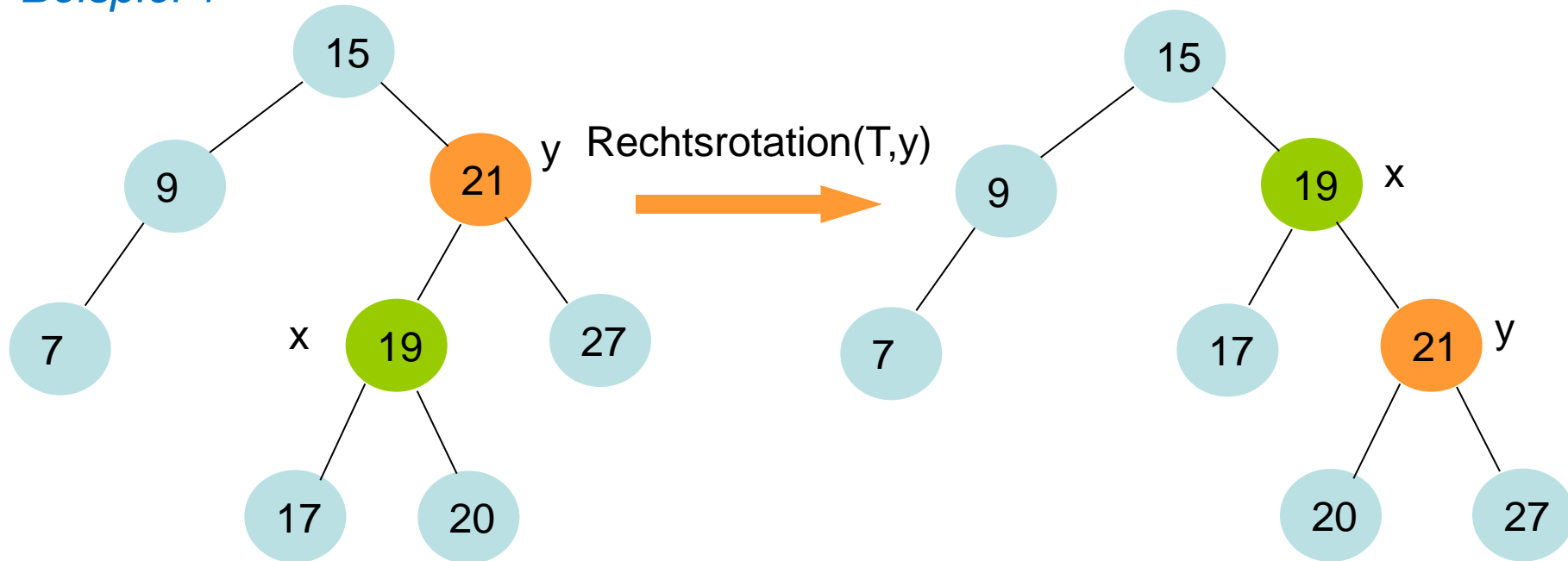
Datenstrukturen

Rotationen



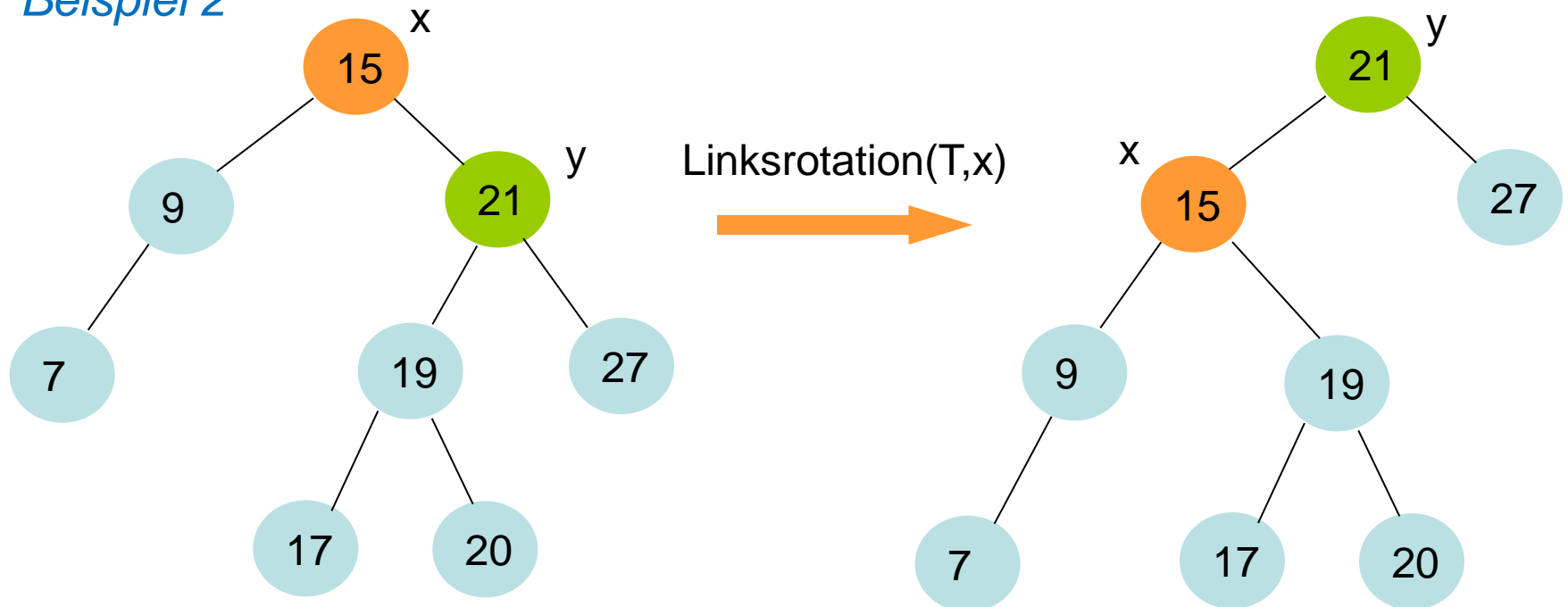
Datenstrukturen

Beispiel 1



Datenstrukturen

Beispiel 2

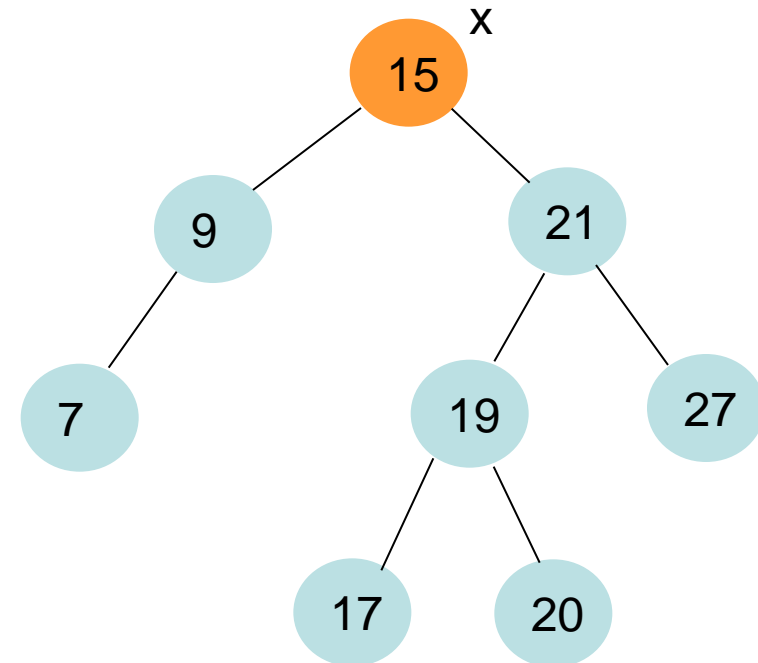


Datenstrukturen (siehe Vollversion)

Linksrotation(T, x)

Annahme: x hat
rechtes Kind.

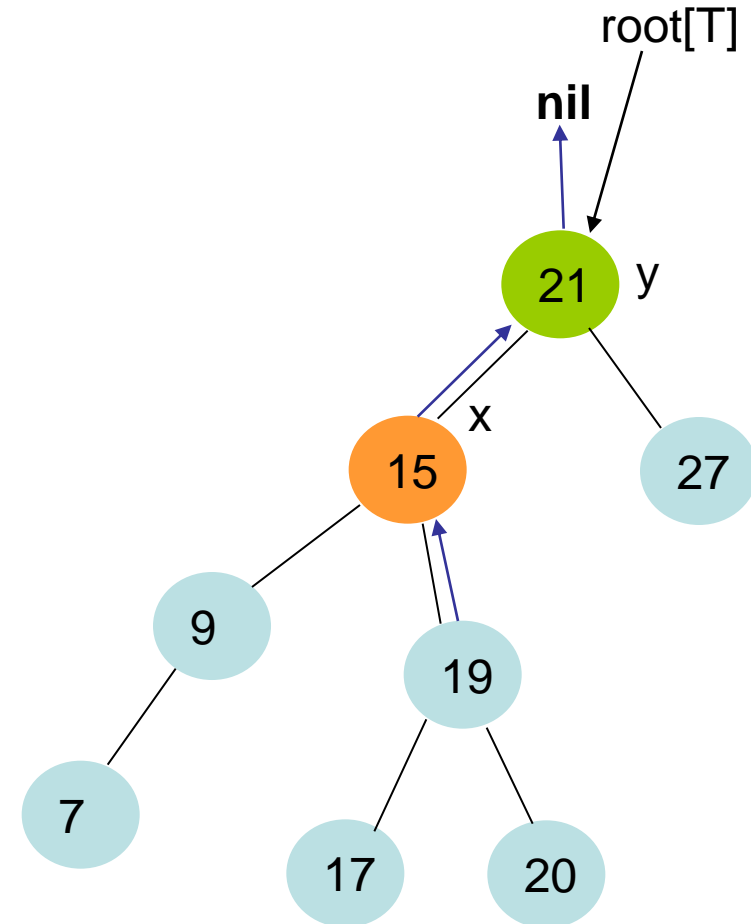
1. $y \leftarrow rc[x]$
2. $rc[x] \leftarrow lc[y]$
3. **if** $lc[y] \neq \text{nil}$ **then** $p[lc[y]] \leftarrow x$
4. $p[y] \leftarrow p[x]$
5. **if** $p[x] = \text{nil}$ **then** $\text{root}[T] \leftarrow y$
6. **else if** $x = lc[p[x]]$ **then** $lc[p[x]] \leftarrow y$
7. **else** $rc[p[x]] \leftarrow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



Datenstrukturen (siehe Vollversion)

Linksrotation(T, x)

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



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

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

Idee

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

Datenstrukturen

Definition

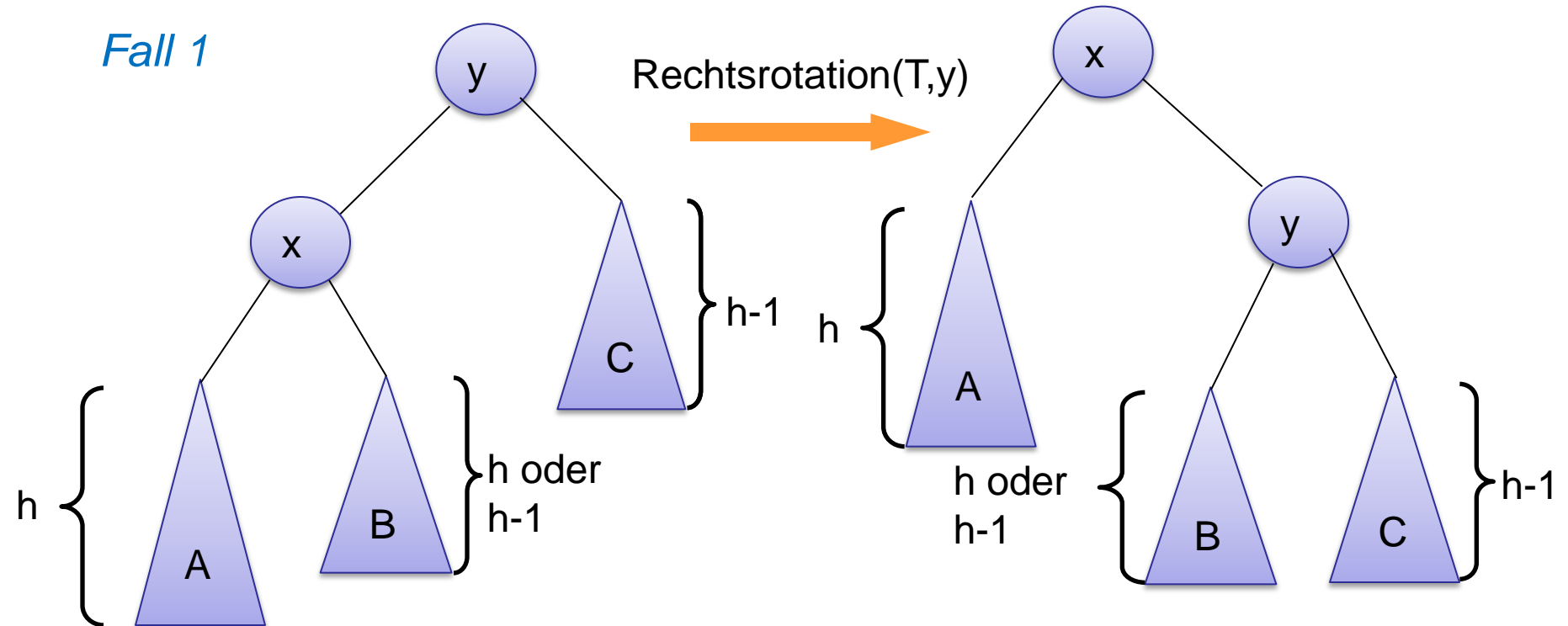
- Ein Baum heißt **beinahe-AVL-Baum**, wenn die AVL-Eigenschaft in jedem Knoten außer der Wurzel erfüllt ist und sich die Höhe der Unterbäume der Wurzel um höchstens 2 unterscheidet.

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

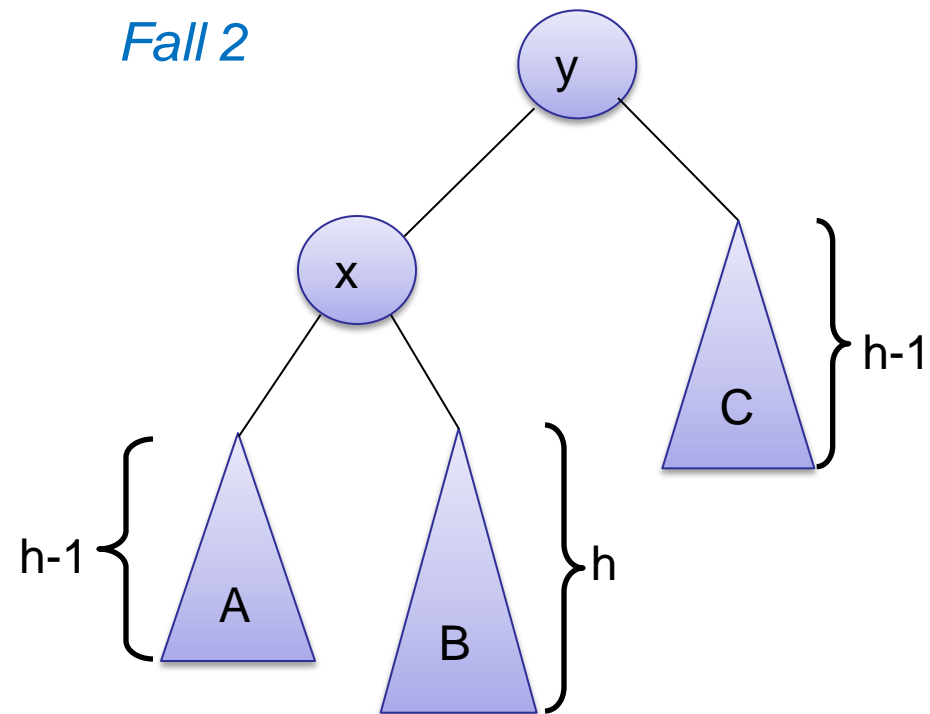
Datenstrukturen

Fall 1



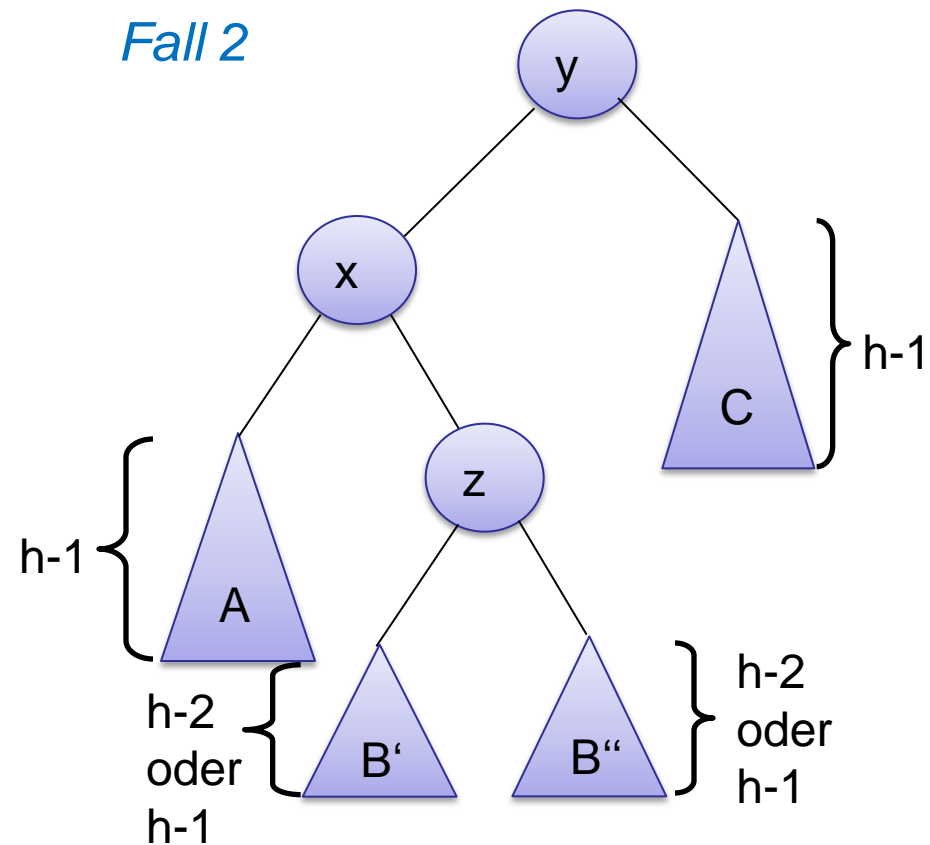
Datenstrukturen

Fall 2



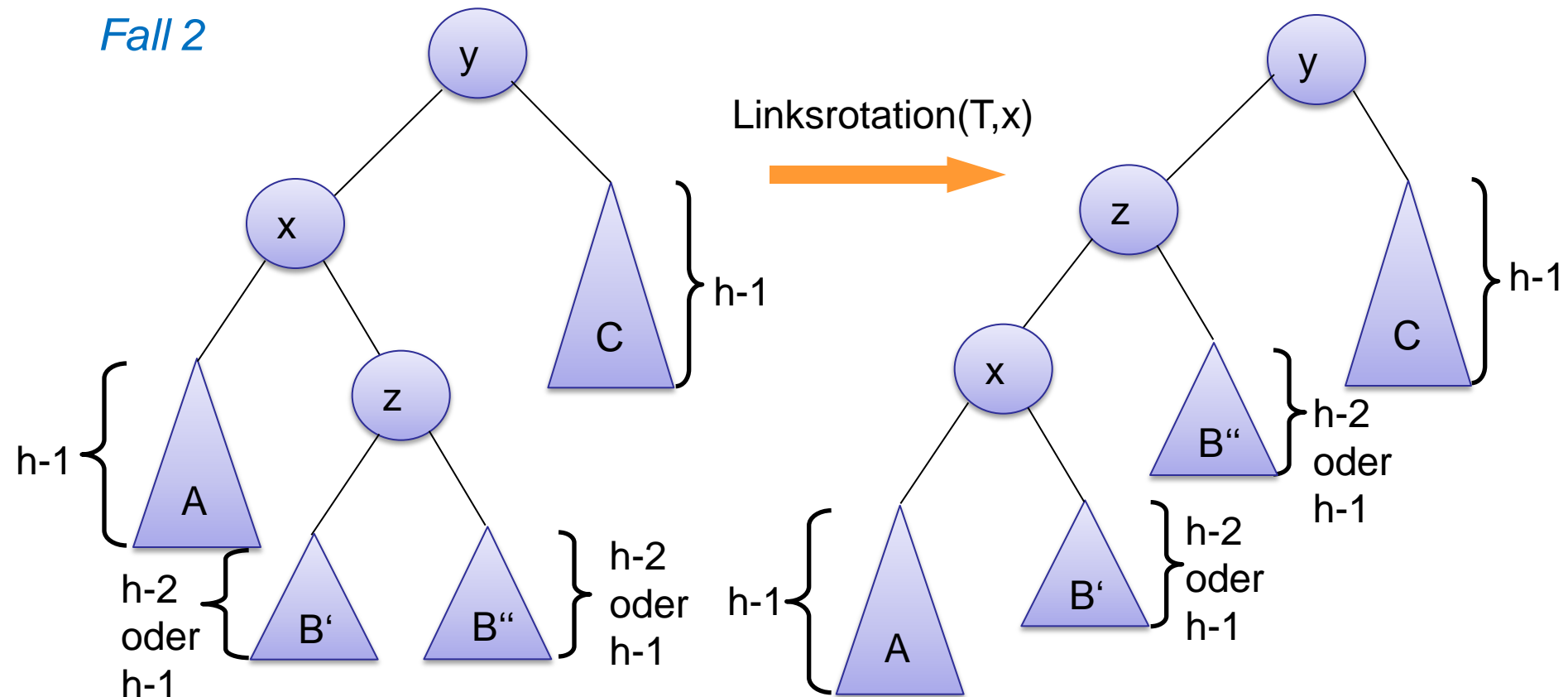
Datenstrukturen

Fall 2



Datenstrukturen

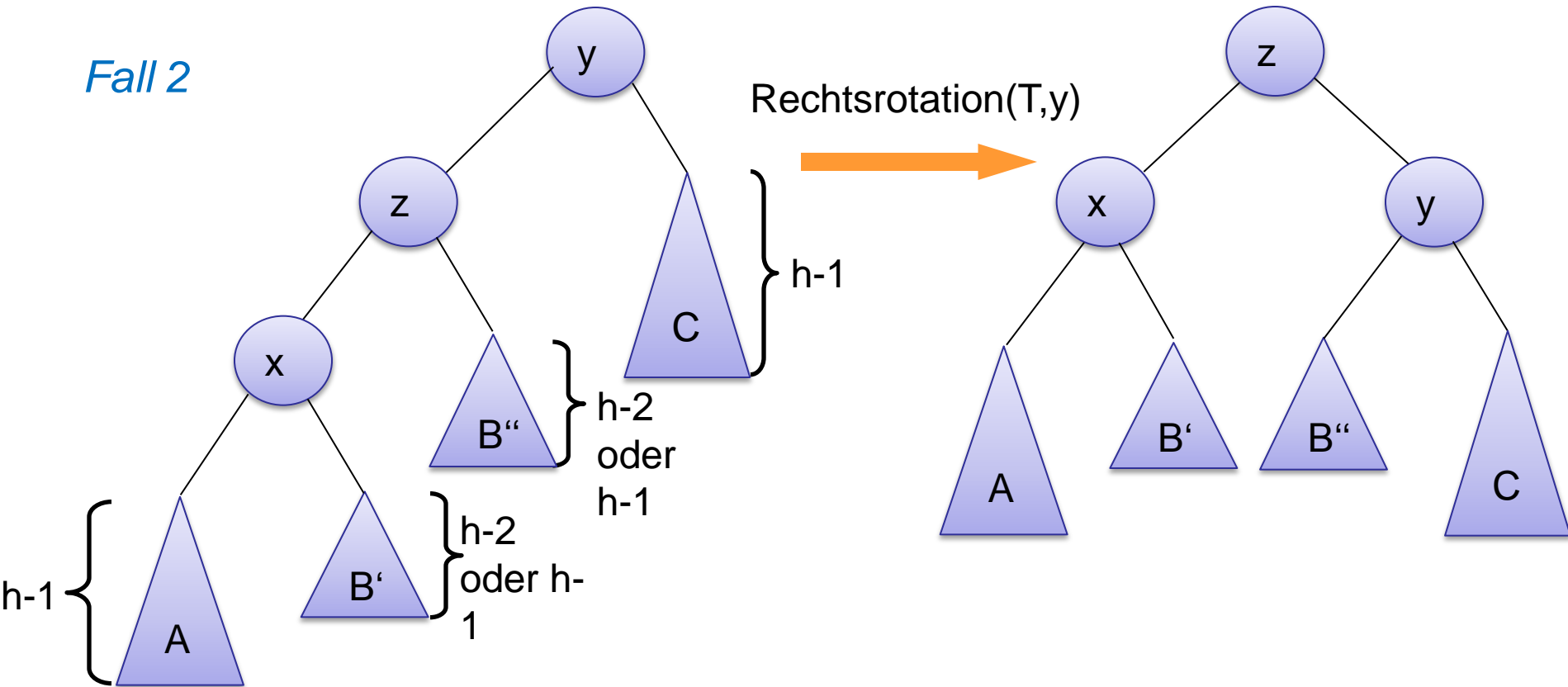
Fall 2



Datenstrukturen

Fall 2

Rechtsrotation(T,y)

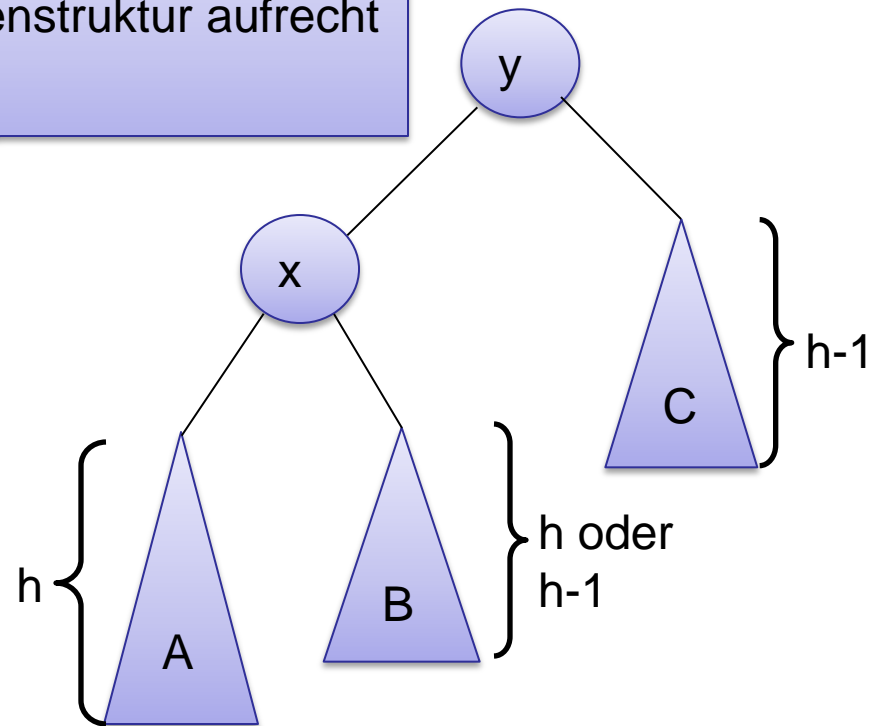


Datenstrukturen (siehe Vollversion)

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($\text{lc}[t]$)
5. Rechtsrotation(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($\text{rc}[t]$)
9. Linksrotation(t)

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

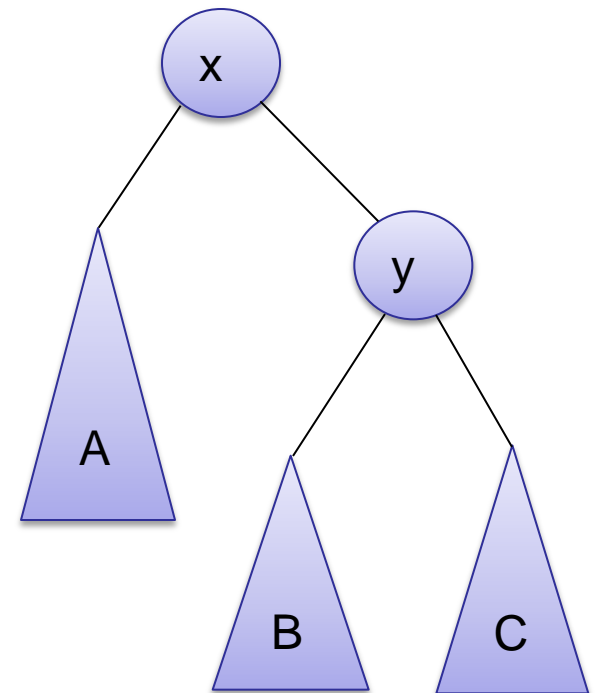


Datenstrukturen (siehe Vollversion)

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($\text{lc}[t]$)
5. Rechtsrotation(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($\text{rc}[t]$)
9. Linksrotation(t)

- Laufzeit: $O(1)$



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

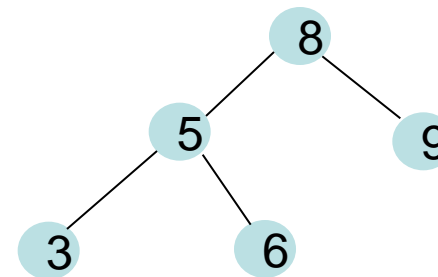
- Wir fügen ein wie früher
- Dann laufen wir den Pfad zur Wurzel zurück
- An jedem Knoten balancieren wir, falls der Unterbaum ein beinahe-AVL-Baum ist

Datenstrukturen (siehe Vollversion)

AVL-Einfügen(t,x)

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

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



Einfügen 2

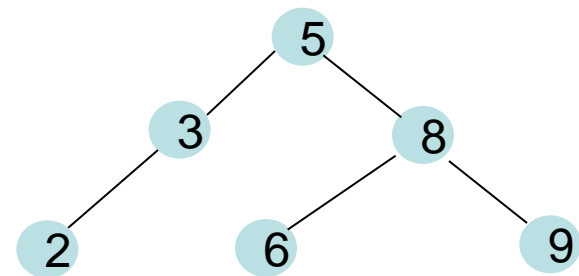
Datenstrukturen (siehe Vollversion)

AVL-Einfügen(t,x)

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

Laufzeit

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



Einfügen 2

Datenstrukturen

Satz 39

- 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 39

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

Beweis

- Induktion über die Höhe des Baumes.
- (I.A.): Für Bäume der Höhe -1 und 0 ist die Aussage korrekt.
- (I.V.): Der Satz gilt für Bäume der Höhe j , $-1 \leq j \leq h$.
- (I.S.): Betrachte den Aufruf von AVL-Einfügen in einem AVL-Baum der Höhe $h+1 \geq 0$.
- Sei o.b.d.A. $\text{key}[x] < \text{key}[t]$ (der andere Fall ist symmetrisch).
- Da $h+1 \geq 0$ ist, wird Zeile (3) ausgeführt.

AVL-Einfügen(t,x)

- if** $t = \text{nil}$ **then**
- $t \leftarrow \text{new node}(x)$; $h[t] \leftarrow 0$; **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** \triangleright Schlüssel schon vorhanden
- $h[t] \leftarrow 1 + \max\{h[\text{lc}[t]], h[\text{rc}[t]]\}$
- Balance(t)

Datenstrukturen

Satz 39

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

Beweis

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

AVL-Einfügen(t,x)

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

Datenstrukture

x bezeichnet Schlüssel des zu löschenden Elements.

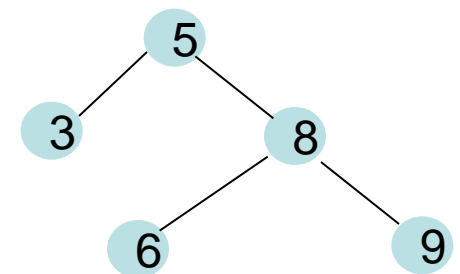
(siehe Vollversion)

AVL-Löschen(t,x)

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

Und die anderen Zeiger aktualisieren

Löschen(3)

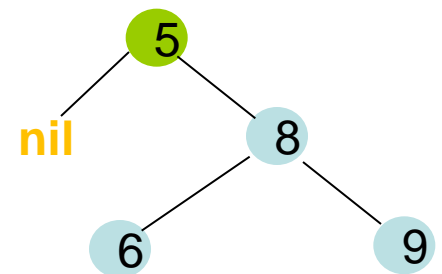


Datenstrukturen (siehe Vollversion)

AVL-Löschen(t,x)

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

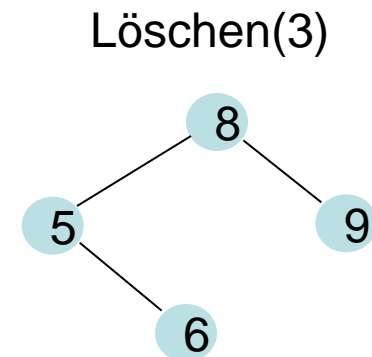
Löschen(3)



Datenstrukturen (siehe Vollversion)

AVL-Löschen(t,x)

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



Datenstrukturen

Korrektheit:

- Ähnlich wie beim Einfügen

Satz 40:

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?