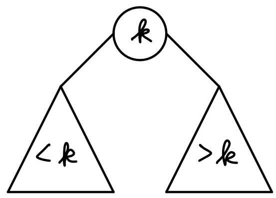
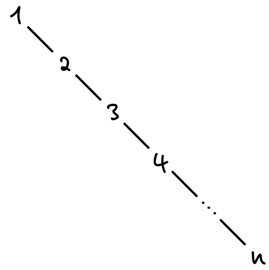
<u>Wiederholung</u>

- Binäre Suchbäume: gewurzelter geordneter binärer Baum,
  BSB Eigenschaft



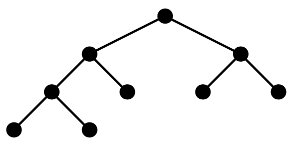
- Alle Operationen: O(Höhe des Baumes)

<u>Problem</u> BSB kann Höhe $\Omega(n)$ haben → schlechte Laufzeit



<u>Aber</u> Ein perfekter Suchbaum hat Höhe $O(\log n)$

alle Ebenen bis auf die letzte voll besetzt



<u>Idee</u> Modifiziere put / remove, so dass Baum jederzeit perfekt ist

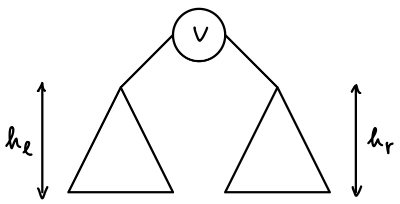
<u>Nur</u>: Das kann zu großen Umbauaktionen führen
Wir können nicht effizient sicherstellen, dass der BSB jederzeit perfekt ist.

<u>Daher</u>: Müssen Anforderungen lockern, so dass Höhe immer $O(\log n)$ ist, aber die Struktur flexibel genug ist, um put & remove effizient zu implementieren.

Dazu gibt es viele Möglichkeiten:

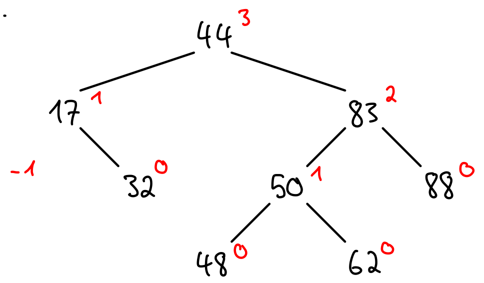
Heute: AVL-Bäume (Adelson-Velsky & Landis)

Ein AVL-Baum T ist ein höhen-balancierter BSB, d.h., für jeden inneren Knoten v in T gilt: die Höhe der Unterbäume unterscheiden sich um höchstens 1 (Höhe des leeren Baums $= -1$).



$|h_\ell - h_r| \le 1$

z.B.



ist AVL-Baum

AVL-Bäume sind fast perfekt.

<u>Satz</u> Die Höhe eines AVL-Baums mit n Knoten ist $O(\log n)$.

<u>Beweis</u> Sei n_h = minimale # Knoten, die ein AVL-Baum der Höhe h enthalten kann.

<u>Es gilt:</u> $n_0 = 1$, $n_1 = 2$, $n_h = 1 + n_{h-1} + n_{h-2}$ für $h \ge 2$

<u>Beh</u> $n_h \ge 2^{\lceil \frac{h}{2} \rceil}$

<u>Beweis durch Induktion:</u>

$n_0 = 1 \ge 2^0$, $n_1 = 2 \ge 2^1$ ✓

$$n_h = 1 + n_{h-1} + n_{h-2} \overset{\text{I.A.}}{\ge} 2^{\lceil \frac{h-1}{2} \rceil} + 2^{\lceil \frac{h-2}{2} \rceil} \ge 2 \cdot 2^{\lceil \frac{h}{2} \rceil - 1} = 2^{\lceil \frac{h}{2} \rceil}$$

<u>Also</u> Wenn Baum Höhe h hat und n Elemente enthält, ist

$$n \ge 2^{\lceil \frac{h}{2} \rceil} \rightarrow \log n \ge \frac{h}{2} \Rightarrow h \le 2 \log n \in O(\log n) \quad \square$$

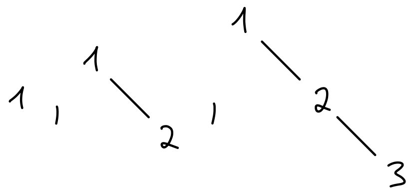
<u>Bem</u> Tatsächlich ist h sogar $\le \underbrace{1,44\ldots}_{\log_\Phi 2} \log n$

Φ = goldener Schnitt

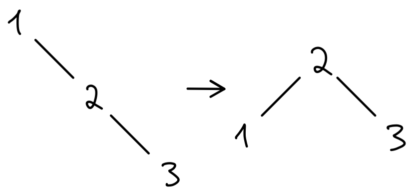
Wie kann man die AVL-Eigenschaft sicherstellen?

Speichere in jeden Knoten die Höhe des Teilbaums (lässt sich in put / remove leicht aktualisieren). Wenn ein Knoten unausgeglichen wird (d.h. $|h_\ell - h_r| = 2$), müssen wir handeln.

<u>Bsp</u> insert 1, 2, 3

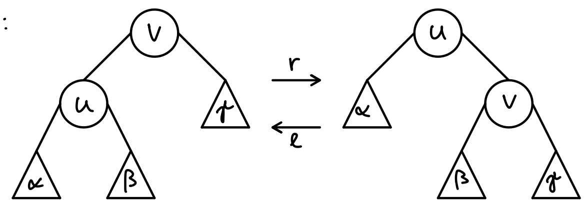


wir müssen ① ausgleichen:



Diese Operation heißt Linksrotation (ℓ-Rotation)

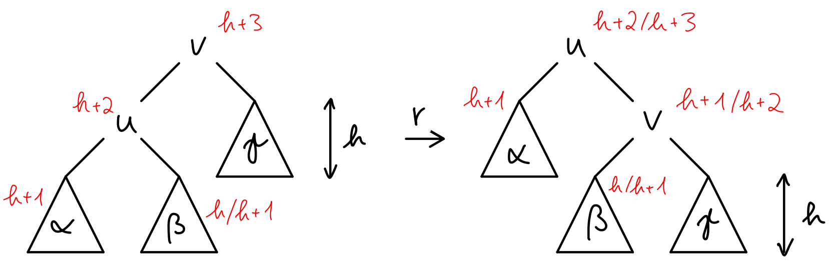
Allgemein:



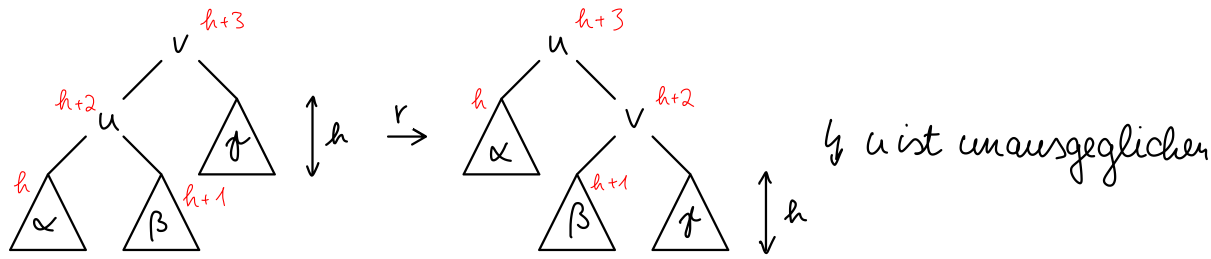
ℓ-/r-Rotationen sind <u>lokal</u>: ändern nur konstant viele Zeiger. Sie erhalten die BSB Eigenschaft.

Was macht eine Rotation mit den Höhen?

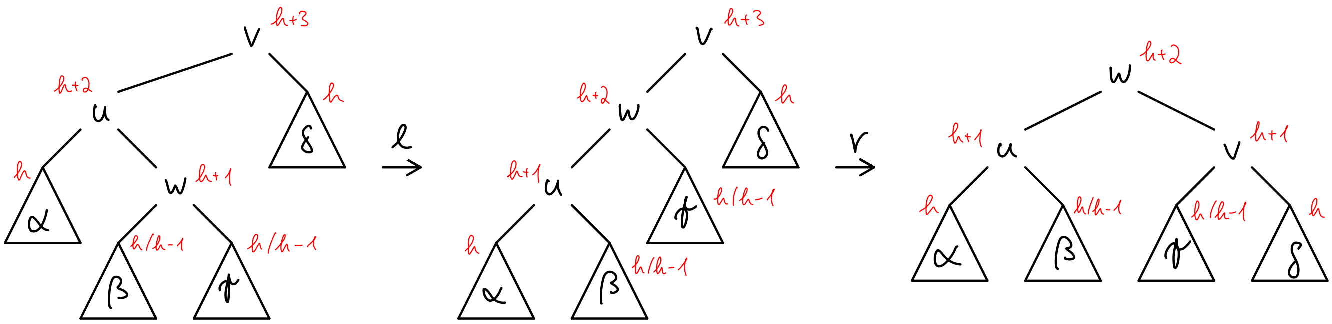
- Wenn v links unausgeglichen ist, und bei u gilt: $h_\ell - h_r \geq 0$, so ist v nach r-Rotation ausgeglichen



Problematische Situation:



Jetzt müssen wir <u>doppelt</u> rotieren (lr-Rotation)



Wenn v unausgeglichen ist ($|h_\ell - h_r| = 2$) und alle Nachbarn ausgeglichen, so existiert eine $(\ell, r, \ell r, r\ell)$-Rotation bei v, die v ausgleicht.

Dann sind v und alle Nachbarn ausgeglichen

Können lokal entscheiden, welche Rotation nötig ist

<u>Also</u> put/remove in AVL-Baum:

Füge ein/lösche wie in ein BSB, aktualisiere Höhe des Pfades zur Wurzel und führe nötige Rotationen durch

$\Rightarrow O(\log n)$ Zeit pro Operation, da jede Rotation $O(1)$ Schritte braucht.

Bsp 10, 20, 30, 25, 35, 27