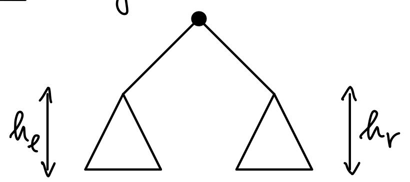
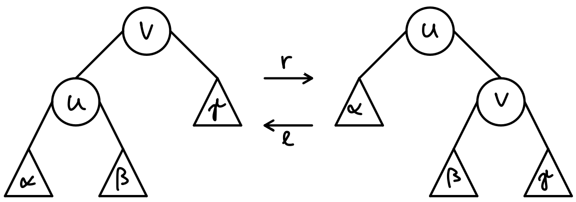
Letztes Mal

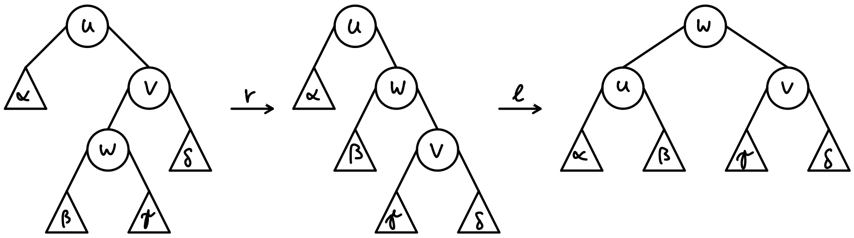
AVL-Bäume in jedem Knoten: $|h_\ell - h_r| \le 1$



- AVL-Bäume haben Höhe $O(\log n)$.
- Rotation:



rl:



lr symmetrisch

- Sei n niedrigster unausgeglichener Knoten ($|h_\ell - h_r| = 2$), dann existiert immer eine Rotation (l, r, lr, rl), welche den Teilbaum mit Wurzel n ausgleicht
- Einfügen / Löschen:
  1. Gehe vor wie bei normalen BSBs
  2. Betrachte Suchpfad
     Gehe den Suchpfad rückwärts und führe nötige Rotationen durch.

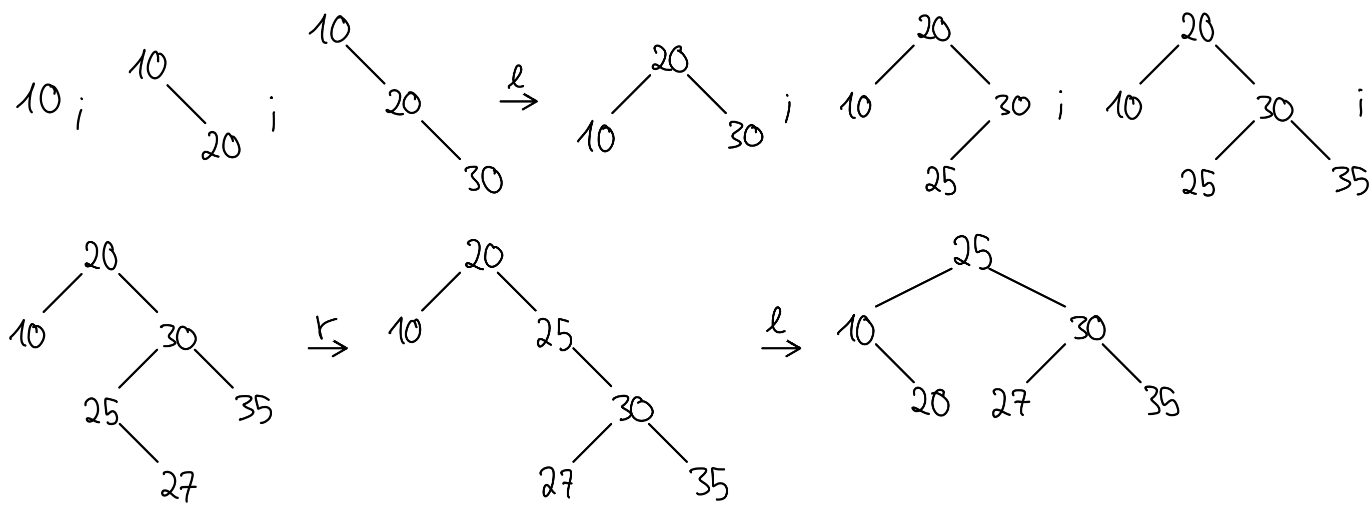
  Fakt Beim Einfügen muss man höchstens einmal rotieren
  Beim Löschen ggf. mehrmals.

  Fakt Durch die Rotation kann sich die Höhe des Teilbaums verringern, aber es kann keine „überunausgeglichenen" Knoten geben

- Beim Einfügen wird ein unausgeglichener Knoten ausgeglichen.
- Beim Löschen kann höchstens ein ausgeglichener Knoten unausgeglichen werden.

$O(\log n)$ Schritte (Höhe ist $O(\log n)$, jede Rotation braucht $O(1)$ Zeit).

<u>Bsp</u> 10, 20, 30, 25, 35, 27



<u>Vorteile:</u>

- $O(\log n)$ worst case
- kompakte Darstellung

Nachteile:

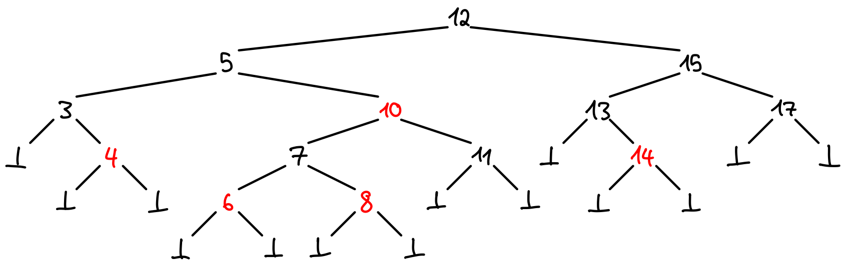
- komplisierte Implementierung (lange Fallunterscheidung)
- beim Löschen: viele Rotationen

<u>Bem</u> Rot-Schwarz-Bäume:

lockern Struktur noch weiter, um die # der Rotationen zu reduzieren

Knoten sind <u>rot</u> oder <u>schwarz</u>, nach folgenden Regeln:

1. Wurzel ist schwarz
2. $\perp$-Knoten (leere Kinder) sind schwarz
3. Kinder eines roten Knoten sind schwarz
4. Alle Pfade von der Wurzel zu einem $\perp$-Knoten enthalten gleich viele schwarze Knoten.



- Hat Höhe $O(\log n)$. Jeder AVL-Baum ist ein RB-Baum.
- Beim Einfügen / Löschen lässt sich die Struktur durch umfärben / rotieren wieder herstellen.
  Es genügen jeweils höchstens 2 Rotationen (anders als beim AVL-Baum).
  Sehr verbreitet (Java, Linux-Kernel, etc.)

(a,b)-Bäume

Erinnerung: Mussten Struktur des perfekten Baums lockern, um effizient zu bleiben

AVL-/RB-Baum: variiere Höhe

(a,b)-Baum: variiere Grad

$a, b \in \mathbb{N}$, $b \geq 2a-1$.

Der Grad aller Knoten ist $\leq b$.

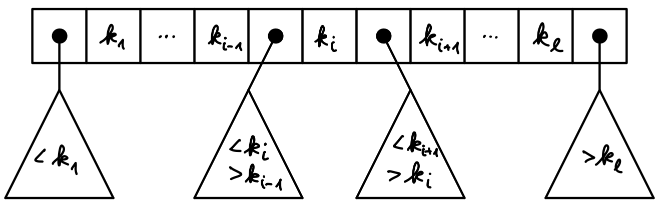
Der Grad der inneren Knoten ist $\geq a$. (hier: Grad $\hat{=}$ # Kinder)

Jeder Knoten speichert $\geq a-1$ und $\leq b-1$ Einträge.

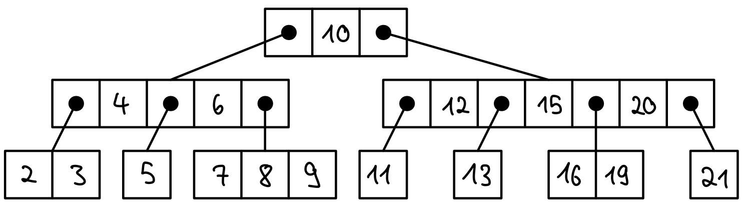
Ausnahme: Für Wurzel gelten untere Schranken nicht

Die Tiefe aller Blätter ist gleich.

Die Schlüssel in den Knoten sind sortiert und die Schlüssel in den Unterbäumen folgen der Suchbaumeigenschaft.



z.B.

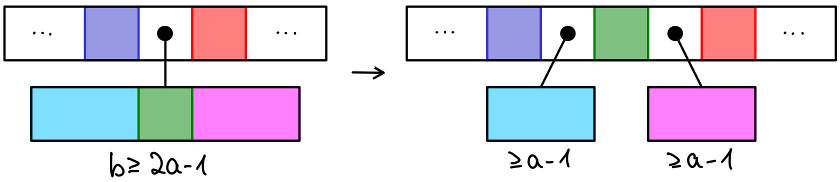


ist (2,4)-Baum

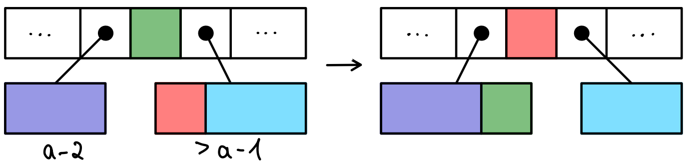
Höhe: zwischen $\Theta(\log_b n)$ und $\Theta(\log_a n)$.

Suche: wie in BSB, müssen aber ggf. mehrere Schlüssel pro Knoten vergleichen: $O(b \log_a n)$ bzw. $O(\log b \cdot \log_a n)$ mit Binärsuche.

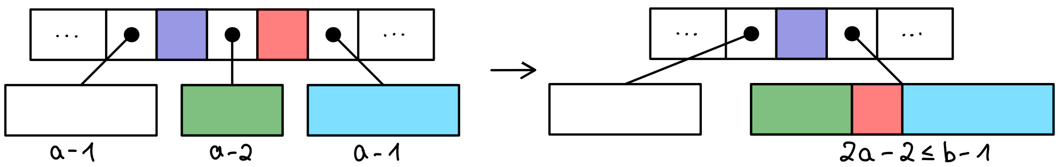
Einfügen Suche bis zum Blatt, füge ins Blatt ein. Wenn der Knoten überläuft (b Einträge), spalte ihn und füge in den Elternknoten ein, wiederhole, bis keine Überläufe mehr auftreten (oder Wurzel)



Löschen
- Wenn Schlüssel in innerem Knoten, suche Nachfolger/Vorgänger in (ist immer in einem Blatt); ersetze durch Nachfolger/Vorgänger, lösche Nachfolger/Vorgänger aus Blatt
- Löschen aus Blatt: Wenn Blatt unterläuft ($a-2$ Einträge), versuche von Geschwister zu stehlen



Wenn das nicht geht, verschmelze mit einem Geschwisterknoten und wiederhole



Anwendung
- (2,3), (2,4)-Bäume als Alternative zu AVL-/RB-Bäumen
- Mit sehr großen a, b für Suchbäume auf der Festplatte (geringe Höhe → wenig Plattenzugriffe)