3. CoMa Übung

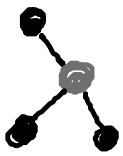
Rot-Schwarz Bäume

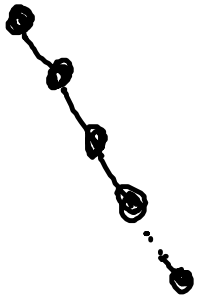
- in Java TreeSet<E> als Suchbaum mit log(n) Operation.
- selbst. balanc. binäre Suchbaumklasse

Def: T ist binärer Suchbaum

1. ein Knoten ist entweder rot oder schwarz
2. die Wurzel ist schwarz
3. alle Blätter sind schwarz
4. beide Kinder eines roten Knotens sind schwarz
5. alle Pfade von einem Knoten zu allen Blätter darunter enthalten dieselbe Anzahl schwarzer Knoten

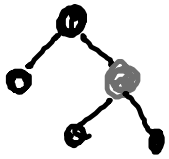
Beispiele:





Problem: Lineare Höhe

Lösung: Forderung nach vollständigen Bäumen (Knoten ist Blatt oder hat 2 Kinder)



Problem: RS-Baum mit 1 Knoten:

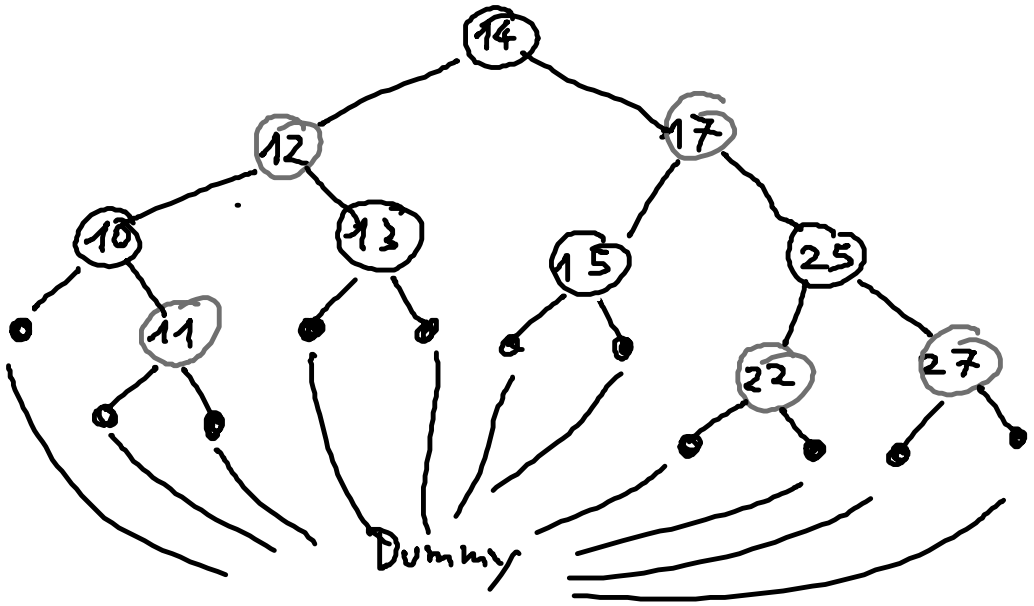
2 Knoten:

nicht möglich

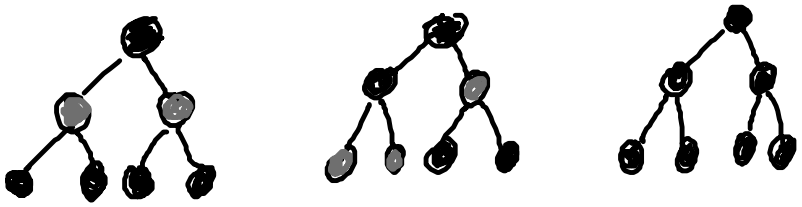
3 Knoten:

4 Knoten:

nicht —"—

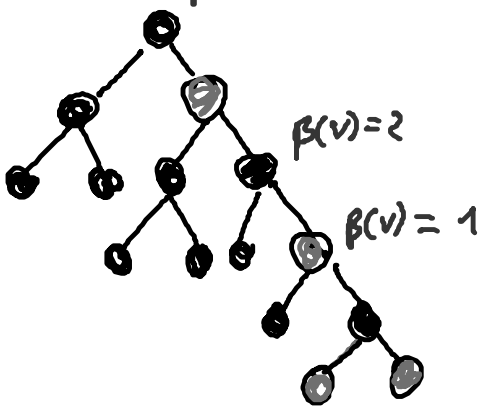


→ Erleichtert Argumentation, ermöglicht ungerade Anzahl an Knoten



gleiche Struktur, andere Färbung

$\beta(v) = 3$ => Kein AVL Baum



Höhenbeschränkung:

Schwarzhöhe = $sh(v) :=$ # schw. Knoten von v zu einem Blatt

wir wissen: Pfad von Wurzel zu Blatt $\geq sh(\text{wurzel})$

$\leq 2 \cdot sh(\text{wurzel})$

Lemma: $h(T) \leq 2 \cdot \log(n+1)$

Beweisidee: - Teilbaum von Knoten v hat min. $2^{h(v)} - 1$ innere Knoten

- $sh(\text{wurzel}) \geq h/2$

$\Rightarrow n \geq 2^{h/2} - 1 \Leftrightarrow \log(n+1) \geq h/2$

$\Leftrightarrow h \leq 2 \log(n+1)$

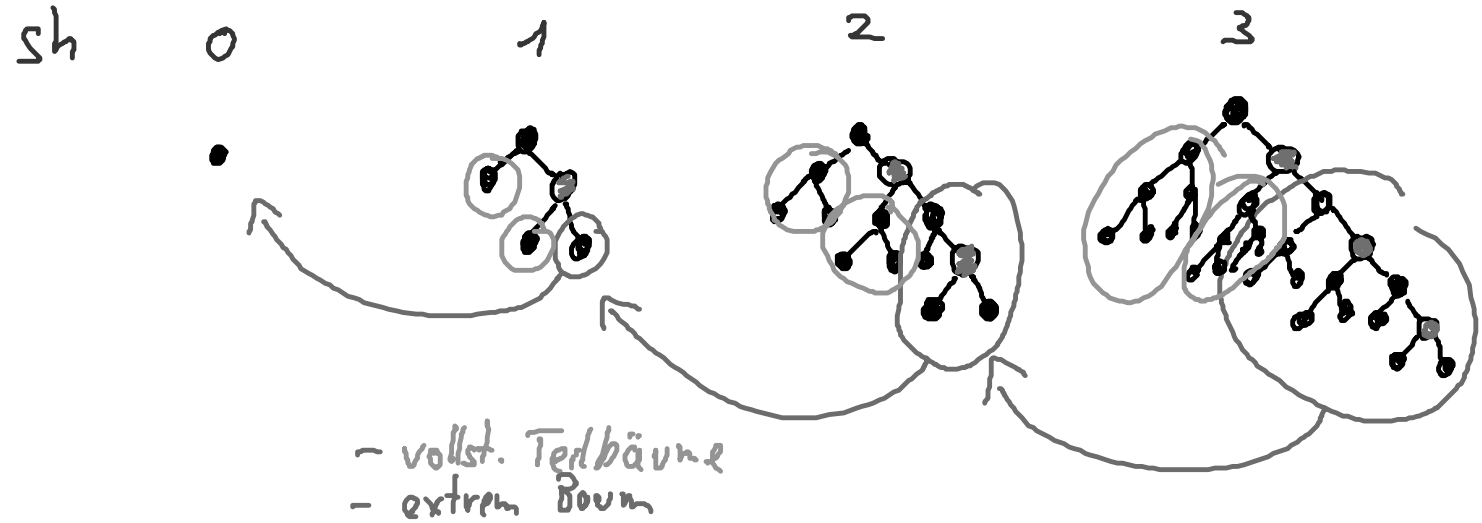
Bemerkungen:

- wenigen Einschränkungen als bei AVL-Bäumen
  ↪ höhere Bäume, 2 zu 1,44
- Praxis: . schneller beim Einfügen/Löschen als AVL
  . langsamer beim Suchen

Extremale Bäume:

fest: sh

gesucht: wenig Knoten (n klein), große Höhe



- vollst. Teilbäume
- extrem Baum

Bäume enthalten jeweils 2 volle Bäume und einen um eins kleineren extremalen Baum

Anzahl Knoten:

$n(sh)$ = # Knoten die RS-Baum mit schwarzhöhe sh mindestens hat, wenn er max. Höhe besitzt

$$n(sh) = \begin{cases} 1 & \text{falls } sh = 0 \\ \underbrace{2}_{\text{Wurzel + roten Kn.}} + \underbrace{2 \cdot (2^{sh-1+1} - 1)}_{\text{2 volle Bäume}} + \underbrace{n(sh-1)}_{\text{extrem. Baum}} = n(sh-1) + 2^{sh+1} \end{cases}$$

Test: $n(0) = 1$

$n(1) = 1 + 2^2 = 5$

$n(2) = 5 + 2^3 = 13$

$n(3) = 13 + 2^4 = 29$

geschlossene form:

$$n(sh) = 2^{sh+2} - 3$$ (Quelle: Wolfram Alpha)

Bew. per Induktion über sh:

IA: $sh = 0$ $\quad n(sh) = 1 = 2^{0+2} - 3$

IV: für bel. aber festes sh gilt $n(sh) = 2^{sh+2} - 3$

IS: $sh \to sh+1$

$$n(sh+1) = n(sh) + 2^{sh+2} \overset{IV}{=} 2^{sh+2} + 2^{sh+2} - 3$$

$$= 2^{(sh+1)+2} - 3 \qquad \square$$

Höhe abschätzung:

$$2^{sh+2} - 3 \leq n \iff sh \leq \log\left(\frac{n+3}{4}\right) = \log(n+3) - 2$$

$$sh \leq \log(n+3) - 2 \Rightarrow \underline{h \leq 2 \cdot \log(n+3) - 4}$$

$\Rightarrow$ Höhe ist in $\Theta(\log(n))$

Operationen:

Einfügen: wie bei sBäumen → Position suchen, als neues Blatt anhängen,
Farbe ist initial rot
(verändert Schwarzhöhe nicht)

evenll. verletzt: (4): zwei rote Knoten untereinander
→ repariere diese Knoten

RS-Baum-Einfüge-Reparatur(T,v)

Input: Rot-Schwarz-Baum T, Knoten v
Output: gültiger Rot-Schwarz-Baum

```
1:  WHILE color(v.parent) = rot DO
2:    IF v.parent = v.parent.parent.left THEN
3:      u := v.parent.parent.right {u is the uncle of v}
4:      IF color(u) = rot THEN
5:        color(v.parent) := schwarz
6:        color(u) := schwarz
7:        color(v.parent.parent) := rot
8:        v := v.parent.parent
9:      ELSE
10:       IF u = v.parent.right THEN
11:         v := v.parent
12:         rotateLeft(T, v)
13:       ENDIF
14:       color(v.parent) := schwarz
15:       color(v.parent.parent) := rot
16:       rotateRight(T, v.parent.parent)
17:     ENDIF
18:   ELSE
19:     same as THEN clause with left and right exchanged
20:   ENDIF
21: ENDWHILE
22: color(T.root) := schwarz
```

Beispiele: v wurde eingefügt, u ist Onkel

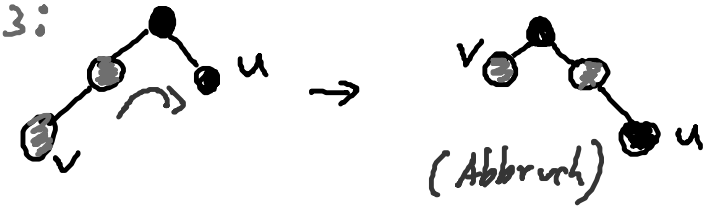
Fall 1:

(über prüfe w)

Fall 2:

(weiter mit Fall 3)

Fall 3:



(Abbruch)

Fall 4-6: links & rechts vertauscht zu Fall 1-3

Beobachtung: - Erhöhung der Schwarzhöhe nur beim Umfärben der Wurzel (Fall 1)

- max. 2 Rotationen nötig (nach Fall 3 term. der Algorithmus)
- Aufwand $O(\log n)$

Beispiel: 1, 4, 2, 3, 5, 6