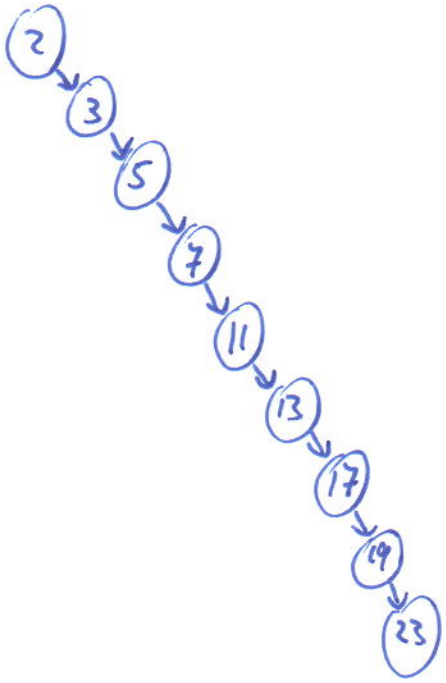
4.6 AVL-Bäume

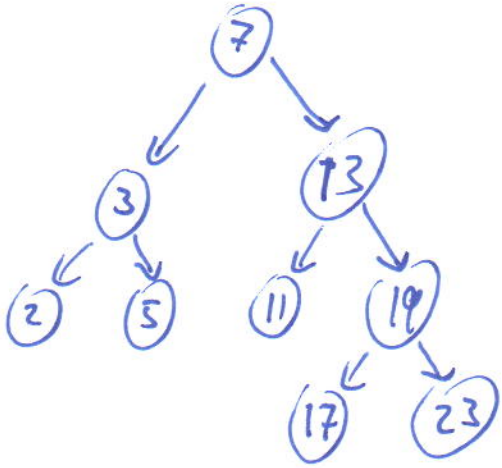
Entscheidend bei binären Suchbäumen: Höhe!

(Einfügen und Entfernen jeweils in $O(h)$)

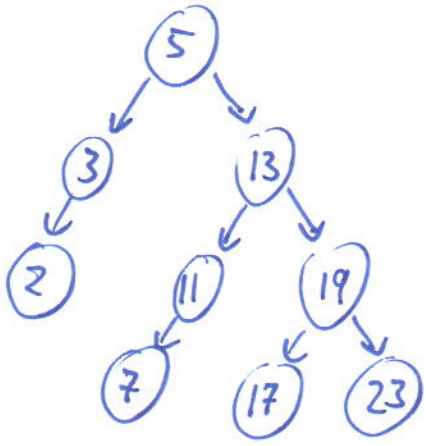
Schlecht:



Gut:



Auch gut:



Idee 1: Gestalte Baum „balanciert", d.h. linken und rechten Teilbaum gleich groß.

Problem 1: Nicht immer möglich!

Idee 2: Betrachte „annähernd balanciert".

Problem 2: Gewicht ist keine lokale Eigenschaft bei lokalen Eigenschaften - eher global...
(Wie setzt man das schnell und lokal um?!)

Idee 3: Wichtig ist eigentlich gar nicht das Gewicht, sondern die Tiefe/Höhe!

Das verfolgt

DEFINITION 4.7 (AVL-Baum, nach Adel'son-Velskij und Landis, 1962)

(1) Ein binärer Suchbaum ist höhenbalanciert, wenn sich für jeden inneren Knoten v die Höhe der beiden Kinder von v um höchstens 1 unterscheidet.

(2) Ein höhenbalancierter Suchbaum heißt auch AVL-Baum.

Beispiele „gut" auf vorheriger Seite sind beide höhenbalanciert.

Problem 3: Reicht „höhenbalanciert", um logarithmische Höhe zu garantieren?

Problem 4: Wie sorgt man dafür, dass die Eigenschaft „höhenbalanciert" bei INSERT und DELETE erhalten bleibt?

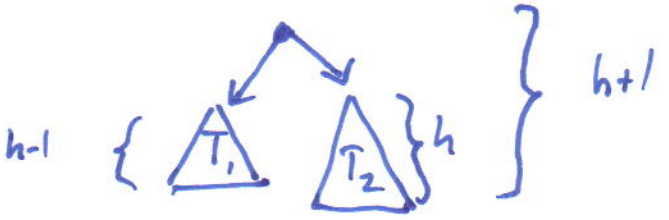
Beobachtung:
Wenn ein Baum höhenbalanciert ist, sind es auch alle seine Teilbäume, d.h. Höhenbalanciertheit ist eine rekursive Eigenschaft!

Idee 3: Wieviele Knoten braucht man mindestens für einen AVL-Baum der Höhe h?

(D.h. wir untersuchen nicht
Höhe in Abhängigkeit von der Knotenzahl
sondern
Knotenzahl in Abhängigkeit von der Höhe!)

wenn n mindestens exponentiell in h wächst,
dann wächst h höchstens logarithmisch in n.

Betrachte allgemein:



Mit $n(1) = 1$ und $n(2) = 2$ • bzw. ↧

und $n(h+1) = 1 + n(h) + n(h-1)$

erhält man

h	$n(h)$
1	1
2	2
3	4
4	7
5	12
6	20
7	33
8	54

(Sehr ähnlich:
$f(0) = 1$, $f(1) = 1$, $f(h+1) = f(h) + f(h-1)$
liefert 1, 1, 2, 3, 5, 8, 13, 21, 34
→ Fibonacci-Zahlen!)