ADS 1: Algorithmen und Datenstrukturen

Teil IX

Uwe Quasthoff

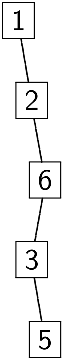
Institut für Informatik
Abteilung Automatische Sprachverarbeitung
Universität Leipzig

12. Dezember 2017

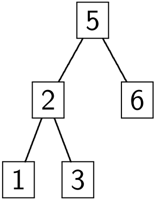
[Letzte Aktualisierung: 04/12/2017, 10:56]

Motivation / Wiederholung: natürliche Suchbäume

im Worst Case: lineare Zugriffskosten (d.h. für Suchen, Einfügen, Löschen)



$\Rightarrow$ beschränke die Höhe:



Z.B. in ausgeglichenem Baum; statisch ok, aber einen Baum bei Einfüge/Lösch-Operationen ausgeglichen zu halten, ist sehr teuer!

Was wäre z.B. nötig, nachdem 4 eingefügt wird?

Wiederholung: ℓ-balancierter Binärbaum

Seien $B_l(x)$ und $B_r(x)$ die linken und rechten Tochterbäume eines Knotens x. Weiterhin sei $\mathrm{h}(B)$ die Höhe eines Baumes B.

Definition: Ein Binärbaum heisst *ℓ-balanciert*, genau dann wenn für jeden seiner Knoten x gilt:

$$|\,\mathrm{h}(B_l(x)) - \mathrm{h}(B_r(x))| \leq \ell\,.$$

(Anmerkung: Nach dieser Definition ist der leere Baum auch ℓ-balanciert.)

d.h. der maximale Höhenunterschied von Tochterbäumen eines Knotens ist durch ℓ beschränkt

ℓ lässt sich als Maß für die Abweichung von einer ausgeglichenen Baumstruktur auffassen.

Offenbar ist jeder ausgeglichene Baum auch 1-balanciert. Aber ist jeder 1-balancierte Baum ausgeglichen?

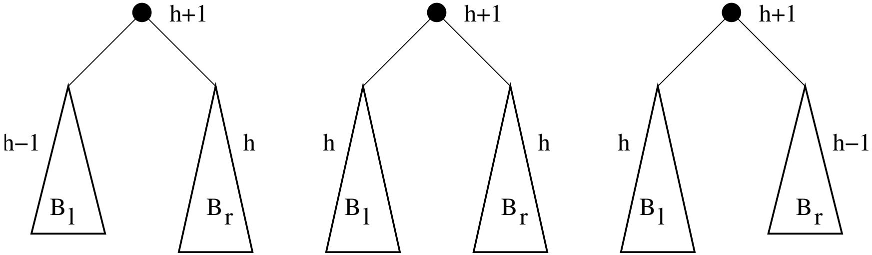
AVL-Baum

- nach russischen Mathematikern **A**delson-**V**elski und **L**andis (1962)
- **Definition:**

  Ein 1-balancierter binärer Suchbaum heißt AVL-Baum.
- also: **AVL-Kriterium**

$$|\,\mathrm{h}(B_l(x)) - \mathrm{h}(B_r(x))| \leq 1 \; .$$

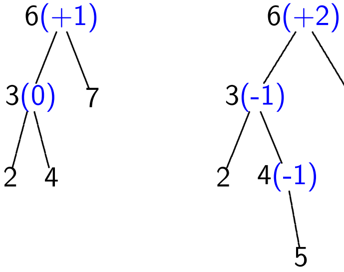
- Konstruktionsprinzip:



Definiere Balancefaktor $\mathrm{BF}(x)$ für Knoten x als

$$\mathrm{BF}(x) = \mathsf{h}(B_l(x)) - \mathsf{h}(B_r(x)) \ .$$

Markieren der Knoten mit dieser Höhendifferenz der Tochterbäume, z.B.

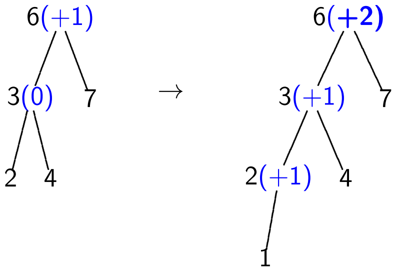


macht AVL-Kriterium deutlich ($|\,\mathrm{BF}(x)| \leq 1$).

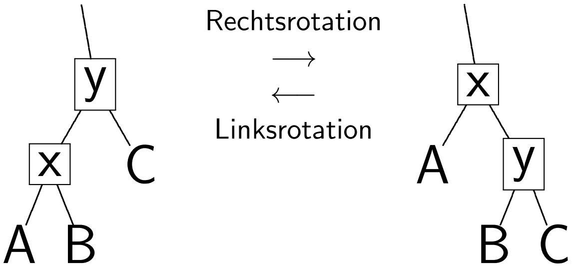
Rebalancierung / Wartungsoperationen nach Einfügen

Wann und wo kann das AVL-Kriterium beim Einfügen verletzt werden?

- Durch Einfügen kann sich nur der Balancefaktor von Knoten auf dem Pfad von der Wurzel des Baumes zum neuen Blatt verändern. Höchstens *h* Knoten sind betroffen.



- Reorganisation lässt sich lokal begrenzen.
- Geeignete Operationen: “Rotationen”.

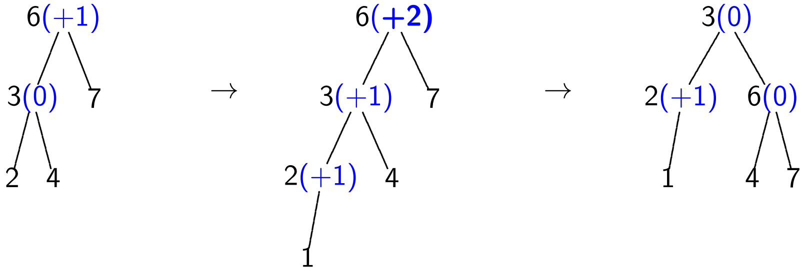


Hier: x und y sind Knoten; A,B und C stehen für Teilbäume.

Warum erhält Rotation die Suchbaumeigenschaft?
Wie verändert Rotation die Balancierung?

Beispiel: Einfügen und Einfachrotation

AVL-Baum → Einfügen von 1 → Rechtsrotation



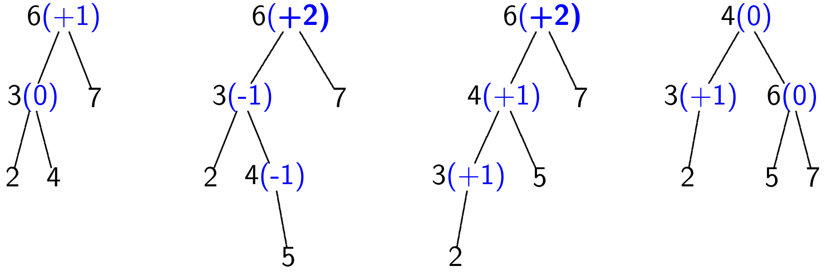
Beispiel: Einfügen und Doppelrotation

AVL-Baum

Einfügen von 5

Linksrotation
$3 \curvearrowleft 4$

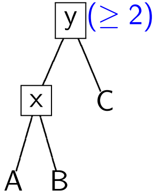
Rechtsrotation
$4 \curvearrowright 6$



Rebalancierung nach Einfügen eines Knotens i

Test $\mathrm{BF}(y)$ für alle Knoten y auf Pfad von i bis zur Wurzel (in dieser Reihenfolge).

- Falls $\mathrm{BF}(y) \geq +2$: Betrachte linken Tochterknoten x von y. Falls $\mathrm{BF}(x) \geq 0$ (d.h $\mathrm{h}(A) \geq \mathrm{h}(B)$), rotiere rechts um y. Sonst Doppelrotation nötig: rotiere zuerst links um x, dann rechts um y.



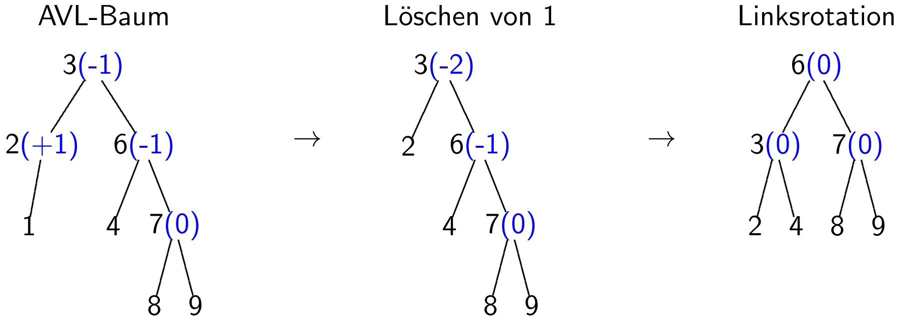
- Falls $\mathrm{BF}(y) \leq -2$: Betrachte rechten Tochterknoten x von y. Falls $\mathrm{BF}(x) \leq 0$, rotiere links um y. Sonst $(\mathrm{BF}(x) > 0)$, rotiere zuerst rechts um x, dann links um y.

Nach Einfügen eines Knotens ist maximal eine Einfach- oder Doppelrotation nötig. Der reorganisierte Teilbaum hat danach dieselbe Höhe wie vorher, so dass das AVL-Kriterium für alle Knoten oberhalb wieder erfüllt ist.

Löschen in AVL-Bäumen I

Einfacher Fall: Löschen eines Blattes bzw. eines Randknotens
(also Löschen von Knoten mit max. 1 Tochter)

- kann Balancierungsfaktoren von Vorgängern ändern - **Rebalancierung** an Vorgängerknoten mit BF = +/- 2 **wie bei Einfügen**
- **gegebenenfalls wiederholte Rebalancierung** (nur auf dem Pfad zur Wurzel). Dies ist ein **Unterschied zu Einfügen**, denn Rebalancierung nach Löschen kann Höhe eines Teilbaums erniedrigen.



Löschen in AVL-Bäumen II

Löschen eines Knotens x mit 2 Töchtern wird auf Löschen für Blatt/Randknoten zurückgeführt:

- ersetze x durch kleinsten Schlüssel y im rechten Tochterbaum von x (oder wahlweise größten Schlüssel im linken);
  Beachte: die Suche nach y benötigt $O(\log n)$ Schritte.

  Warum hat der Knoten mit Schlüssel y höchstens eine Tochter?

- Lösche ursprünglichen Knoten von y und rebalanciere **wie zuvor**

Höhe von AVL-Bäumen I

Zur Erinnerung: Fibonacci-Zahlen $F_0 = 0$; $F_1 = 1$; $F_i = F_{i-1} + F_{i-2}$ $(i > 1)$

Behauptung: Für jeden AVL-Baum T der Höhe h gilt, T hat mindestens F_h Knoten $(|T| \geq F_h)$.

Beweis durch vollständige Induktion (über Baumhöhe)

- **Induktionsstart**: $h = 0$ (leerer Baum), $h = 1$ (nur Wurzel) ok.
- **Induktionsschritt** $(h \geq 2)$. Sei T ein AVL-Baum der Höhe h; sei ausserdem $|T|$ minimal.
    - Einer der Tochterbäume von T muss die Höhe $h - 1$ haben.
    - Wegen Minimalität von T hat der andere Tochterbaum die kleinstmögliche Höhe; wegen AVL-Eigenschaft, also $h - 2$.
    - Nach *Induktionshypothese* gilt, dass die Tochterbäume mindestens F_{h-1} und F_{h-2} Knoten enthalten, daher gilt $|T| \geq F_{h-1} + F_{h-2} + 1 \geq F_h$. □

Höhe von AVL-Bäumen II

Um die minimale Knotenzahl eines AVL-Baums der Höhe h abzuschätzen, schätzen wir also F_h weiter ab.

Nach Binetscher Formel (vgl. 1. Vorlesung) gilt

$$F_h = (A^h - B^h)/\sqrt{5}\ ,$$

wobei $A = (1 + \sqrt{5})/2$ und $B = (1 - \sqrt{5})/2$.

Da $|B| < 1$, ist $B^h < 1$. Wir können deshalb weiter abschätzen

$$F_h \geq (A^h - 1)/\sqrt{5}\ .$$

Sei T ein AVL-Baum der Höhe h mit n Knoten. Nun gilt $n \geq F_h \geq (A^h - 1)/\sqrt{5}$, also (nach h aufgelöst)

$$h \leq \frac{\log_2(\sqrt{5}n + 1)}{\log_2 A}.$$

Höhe von AVL-Bäumen III

Aus

$$h \leq \frac{\log_2(\sqrt{5}n + 1)}{\log_2 A} \approx \frac{1}{\log_2 A} \cdot (\log_2 \sqrt{5} + \log_2 n)$$

ergibt sich

$$h < 1.44 \log_2 n \qquad (+\text{kleine Konstante})$$

Vergleich mit vollständigen Bäumen:

$$h = \log_2(n + 1)$$

Resultat: Selbst im ungünstigsten Fall sind AVL-Bäume nur um Faktor 1.44 höher als vollständige Bäume mit derselben Knotenzahl!

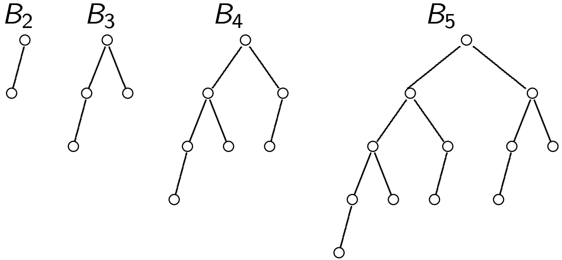
Was haben wir also erreicht?
(Wie sieht der worst-case aus?)

Fibonacci-Bäume

"Maximal unbalancierte" AVL-Bäume.

- Zu jedem $h \geq 0$ gibt es genau einen Fibonacci-Baum B_h mit Höhe h.
- B_0 ist der leere Baum, B_1 ist ein einzelner Knoten
- für $h \geq 2$ ist der Fibonacci-Baum der Höhe h mit Wurzel x.

$$B_h = \mathrm{BUILD}(B_{h-1}, x, B_{h-2}) \ .$$



Gewichtsbalancierte Suchbäume I

Gewichtsbalancierte oder BB-Bäume (bounded balance)

Zulässige Abweichung der Struktur vom ausgeglichenen Binärbaum wird über den Quotient zwischen der Anzahl der Knoten im linken Tochterbaum und dem gesamten Teilbaum beschränkt.

Definition: Sei B ein binärer Suchbaum mit $n > 0$ Knoten, von denen sich n_l Knoten im linkem Tochterbaum B_l befinden.

- $\rho(B) = \frac{n_l+1}{n+1}$ heißt die Wurzelbalance von B.
- Ein Baum B heißt gewichtsbalanciert, BB(α), oder von beschränkter Balance α, wenn für jeden Tochterbaum B' von B gilt:

$$\alpha \leq \rho(B') \leq 1 - \alpha$$

Nievergelt and Reingold (1972), http://portal.acm.org/citation.cfm?id=804906

Gewichtsbalancierte Suchbäume II

Wahl des Parameters α:

- $\alpha = 1/2$: Balancierungskriterium akzeptiert nur vollständige Binärbäume
- $\alpha < 1/2$: Strukturbeschränkung wird zunehmend gelockert
- $\alpha = 0$: keine Beschränkung der Baumstruktur

Wartung

- Einsatz derselben Rotationstypen wie beim AVL-Baum
- Rebalancierung ist gewährleistet durch eine Wahl von

$$\alpha \leq 1 - \sqrt{\frac{1}{2}}$$

Kosten für Suche und Aktualisierung: $O(\log n)$

Positionssuche mit balancierten Bäumen I

Balancierte Suchbäume

- sind linearen Listen in fast allen Grundoperationen überlegen
- Lösung des Auswahlproblems bzw. Positionssuche (Suche nach k-tem Element der Sortierreihenfolge) kann jedoch noch verbessert werden

Definition: Der *Rang* eines Knotens ist die um 1 erhöhte Anzahl der Knoten seines linken Tochterbaums. (Blattknoten haben Rang 1)

Positionssuche mit balancierten Bäumen II

Rangzahlen erlauben Bestimmung eines direkten Suchpfads im Baum für Positionssuche nach dem k-ten Element.

- Position $p := k$; beginne Suche am Wurzelknoten
- Wenn Rang r eines Knotens $= p$, gilt: Element gefunden
- falls $r > p$: Suche im linken Tochterbaum des Knotens weiter
- falls $r < p$: Ersetze p := p - r und setze Suche im rechten Tochterbaum fort.

Die Wartungsoperationen erfordern etwas mehr Aufwand: Nach Einfügen / Löschen eines Knotens müssen die Rangwerte auf dem kompletten Suchpfad angepasst werden.