

Binäre Suchbäume

Find, Insert, Delete in Zeit proportional zur Höhe des Baums.
Wie kann geringe Höhe erreicht werden?

Binäre Suchbäume

Find, Insert, Delete in Zeit proportional zur Höhe des Baums.
Wie kann geringe Höhe erreicht werden?

Sich auf Zufall in der Eingabe verlassen:

Theorem

*Wenn n Schlüssel in zufälliger Reihenfolge (jede Permutation gleich wahrscheinlich) in einen anfangs leeren binären Suchbaum eingefügt werden, dann hat jeder Schlüssel **erwartete** Tiefe kleiner als $1 + 2 \cdot \ln n = O(\log n)$.*

Dieser Satz gilt nicht mehr, wenn Löschooperationen auch vorkommen.

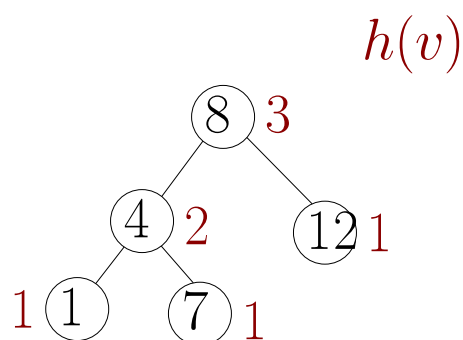
AVL Bäume

- ▶ Name AVL von Erfindern Adelson-Velskii und Landis
- ▶ Erfunden 1962, älteste Datenstruktur für balancierte Bäume

AVL Baum

Def.: ein binärer Suchbaum T heißt AVL Baum falls sich für jeden Knoten v von T die Höhen der Unterbäume um höchstens 1 unterscheiden

- ▶ Wir speichern Höhe $h(v)$ für jeden Knoten v



Haben AVL-Bäume immer logarithmische Tiefe?

Sei \min_h was ist die minimale Knotenzahl eines AVL Baums der Höhe h ?

Es gilt:

$$\min_0 = 0$$

$$\min_1 = 1$$

$$\min_h = \min_{h-1} + \min_{h-2} + \underbrace{1}_{\text{Wurzel}}$$

Grund: AVL Eigenschaft

Lemma

$$\min_h \geq (3/2)^h. \quad \text{Bew. durch Induktion über } h.$$

Corollary

Für einen AVL-Baum mit n Knoten und Höhe h gilt $h = O(\log n)$.

Proof.

$n \geq \min_h \geq (3/2)^h$, also nach Logarithmieren

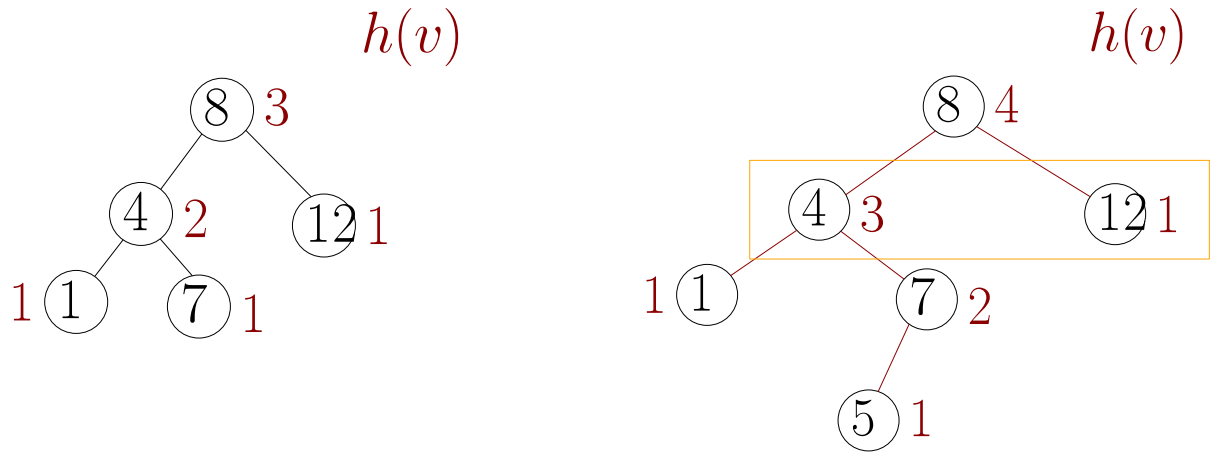
$h \leq \log_{3/2} n = O(\log n)$. □

Operationen

- ▶ Aufzählen - kann direkt von BST übernommen werden
- ▶ Suche - kann direkt von BST übernommen werden
- ▶ Einfügen - BST Methode kann zu Problemen führen (AVL Eigenschaft stimmt evtl. nicht mehr)
- ▶ Löschen - BST Methode kann zu Problemen führen (AVL Eigenschaft stimmt evtl. nicht mehr)

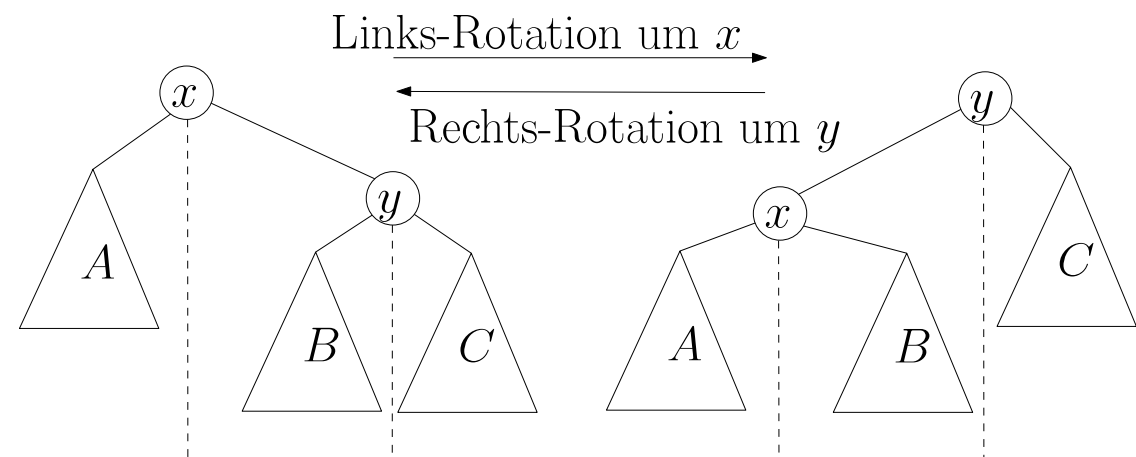
Problem

Durch Operationen Einfügen (im Bsp. von Knoten mit Schlüssel 5) und Löschen wird AVL Eigenschaft verletzt



Rotation

Um Baum wieder zu balancieren, wird er lokal umstrukturiert, so dass die Schlüsselordnung erhalten bleibt:



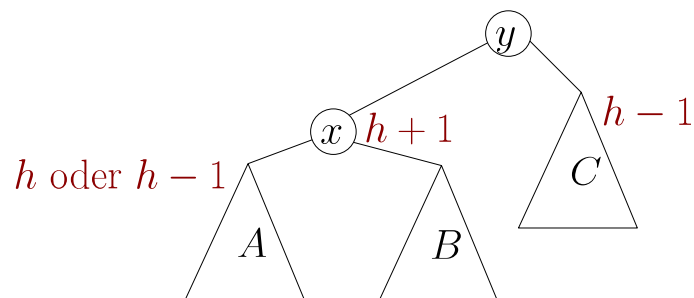
Links-Rotation kann nur auf x angewendet werden, wenn x ein rechtes Kind hat (symmetrisch für Rechts-Rotation)

Balancieren

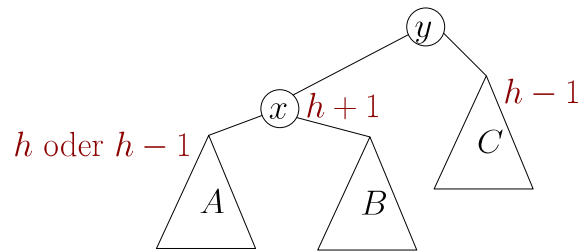
- ▶ **Beinahe-AVL-Baum:** AVL-Kriterium bei allen Knoten erfüllt außer bei der Wurzel. Höhenunterschied zwischen linkem und rechtem Teilbaum der Wurzel ist genau 2.
- ▶ Rotationen können genutzt werden, um einen beinahe-AVL-Baum zu balancieren.

Balancieren

Annahme: linker Teilbaum größer als rechter Teilbaum (anderer Fall symmetrisch)



Balancieren



Fall 1: A hat Höhe h

⇒ B hat Höhe h oder $h-1$

⇒ Rechts-Rotation um y
ergibt AVL-Baum

Fall 2: A hat Höhe $h-1$

⇒ B hat Höhe h

⇒ 1. Links-Rotation um x
2. Rechts-Rotation um y
(auch Doppelrotation
genannt) ergibt
AVL-Baum

Balancieren

- ▶ Ein beinahe-AVL-Baum kann mit maximal zwei Rotationen balanciert werden
- ▶ Die Funktion, die dies erzielt, nennen wir im Folgenden `BALANCE(KNOTEN X)`

Einfügen

Algorithm 1: AVL_Einfügen(Knoten x von Baum T , Knoten z)

```
1 if ( $x = TNULL$ ) then
2   | /*Füge den Knoten als Blatt in den Baum ein*/
3   |  $h(z) = 1$ 
4 end
5 else if ( $Schlüssel(z) < Schlüssel(x)$ ) then
6   | AVL_Einfügen(Left( $x$ ),  $z$ )
7 end
8 else if ( $Schlüssel(z) > Schlüssel(x)$ ) then
9   | AVL_Einfügen(Right( $x$ ),  $z$ )
10 end
11 else
12   | /*Schlüssel schon in  $T$ */
13 end
14  $h(x) = 1 + \max(h(Left(x)), h(Right(x)))$ 
15 Balance( $x$ )
```

Einfügen

- ▶ Einfügen führt zu maximal $O(h)$ Aufrufen von BALANCE
- ▶ Aber: Einfügen führt zu maximal einem beinahe-AVL-Baum (d.h. BALANCE hat maximal ein Mal etwas zu tun)
- ▶ Damit: maximal zwei Rotationen werden durchgeführt
- ▶ Gesamtzeit Einfügen: $O(h)$

Löschen

- ▶ Lösche Element wie in BST
- ▶ Balanciere Baum von der untersten veränderten Stelle entlang des Pfads zur Wurzel neu
- ▶ Maximal h Doppelrotationen in BALANCE möglich
- ▶ Gesamtzeit Löschen: $O(h)$

Zusammenfassung AVL-Bäume

Laufzeit der Operationen in einem AVL-Baum mit n Knoten:

Aufzählen	$O(n)$
Suchen	$O(\log(n))$
Einfügen	$O(\log(n))$
Löschen	$O(\log(n))$