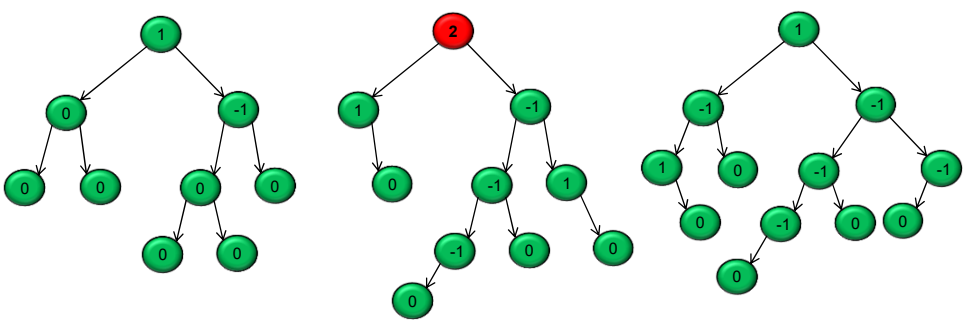
Algorithmen und Datenstrukturen

Aufgabe 3

Aufgabe

- AVLBaum als **höhenbalancierten binären Suchbaum**: Implementieren Sie den ADT **AVL-Baum** gemäß der Definition aus der Vorlesung.
- Implementieren Sie die beiden Operationen `insertBT` und `deleteBT`. Implementieren Sie zudem die benötigten Rotationen LinksRotation, RechtsRotation, DoppeltLinksRotation, DoppeltRechtsRotation.
- Erweitern Sie den ADT so, dass die Anzahl der jeweiligen Rotationen gezählt werden. Wie oft muss jeweils bei insert/delete korrigiert (balanciert) werden?

AVL-Bäume: Beispiele



AVL-Baum | kein AVL-Baum | AVL-Baum

Satz: Für jeden AVL-Baum der Höhe h>0 gilt $fib^2(h+2)-1 \le n \le 2^h-1$

Ein AVL-Baum mit n Knoten hat höchstens die Höhe

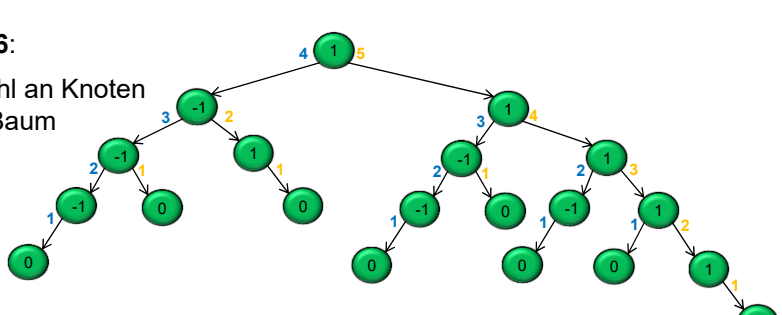
$$h \le 1{,}44... * \log_2(n) + const \in O(\log_2(n))$$

Erinnerung: Für die Höhe jedes binären Baumes mit n Knoten gilt $\log_2(n+1) \le h$

AVL-Bäume: Beispiel

Beispiel für **h=6**:

Minimale Anzahl an Knoten in einem AVL-Baum
h=6, n=20 (63)
h=5, n=12 (31)
h=4, n=7 (15)
h=3, n=4 (7)
h=2, n=2 (3)
h=1, n=1 (1)



Beispiel für **h=6**>0: $fib^2(6+2)-1 = 20 \le n \le 2^6-1 = 63$

Beispiel für **n=20**: $6 \le 6{,}322 \approx 1{,}44... * \log_2(20) + 0$

Fibonacci der Ordnung 2:

$$fib^2(n) = fib^2(n-1) + fib^2(n-2)$$
$$fib^2(1) = fib^2(2) = 1$$

AVL-Bäume

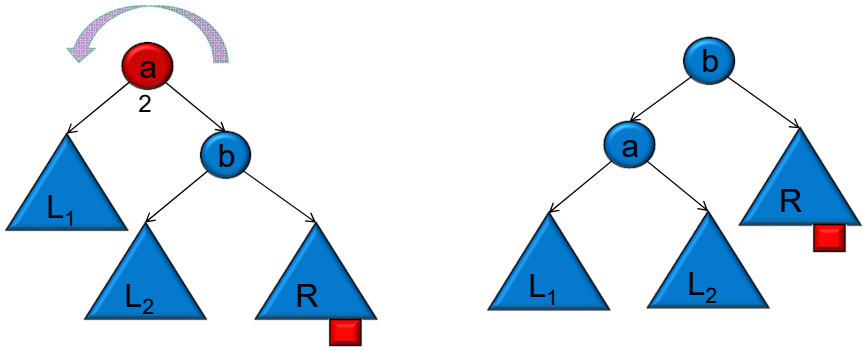
Einfügen

- Füge neuen Knoten gemäß der Sortierung als Blatt in den AVL-Baum ein.
- Überprüfe von diesem Blatt ausgehend bottom-up die AVL-Bedingung.
- Führe ggf. an dem Knoten, an dem die Bedingung verletzt wurde, eine Rebalancierung durch (durch entsprechende Rotation).
- Im Unterschied zu gewöhnlichen binären Suchbäumen ist es bei AVL-Bäumen wegen den Rotationen prinzipiell nicht möglich, gleiche Schlüssel immer im rechten (linken) Teilbaum einzufügen (oder dort zu finden).

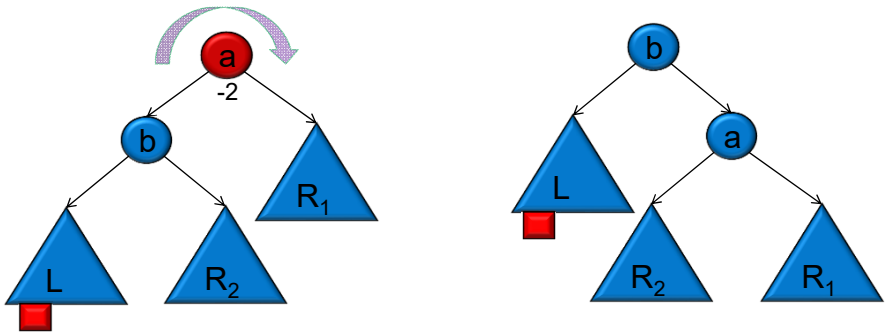
Löschen

- Kopiere den größten Knoten im linken Teilbaum bzw. kleinstem Knoten im rechten Teilbaum auf den zu löschenden Knoten. Lösche nun (rekursiv) diesen Knoten.
- Ist der zu löschende Knoten ein Blatt (Rekursionsende), so wird der Knoten aus dem AVL-Baum entfernt.
- Überprüfe von dem Vorgänger des entfernten Blatts ausgehend bottom-up die AVL-Bedingung.
- Führe ggf. an dem Knoten, an dem die Bedingung verletzt wurde, eine Rebalancierung durch (durch entsprechende Rotation).

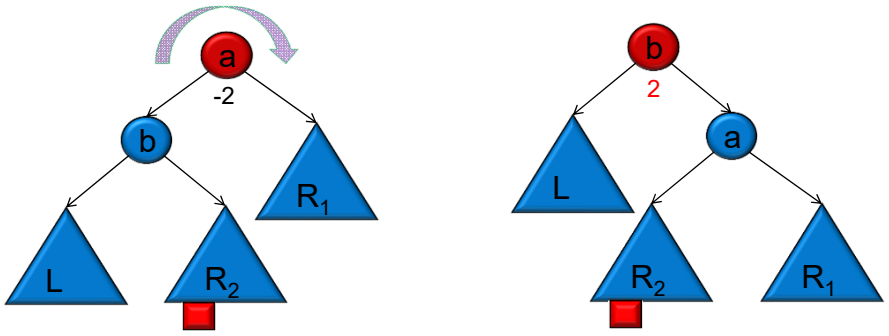
AVL-Bäume: Linksrotation



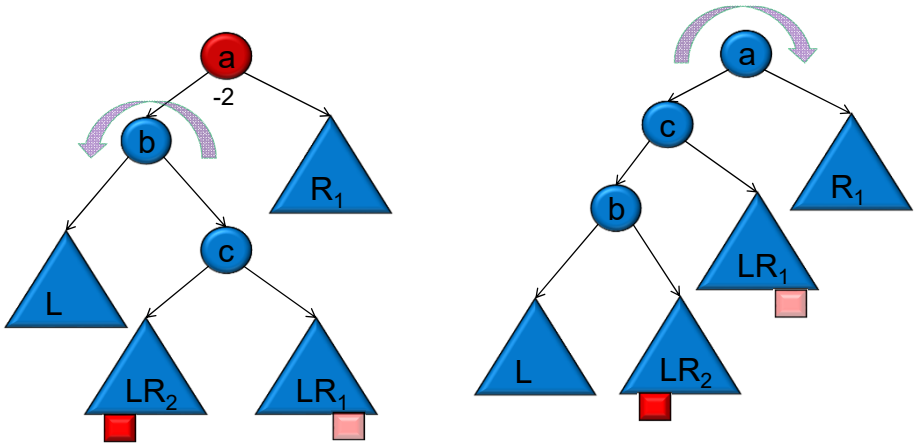
AVL-Bäume: Rechtsrotation



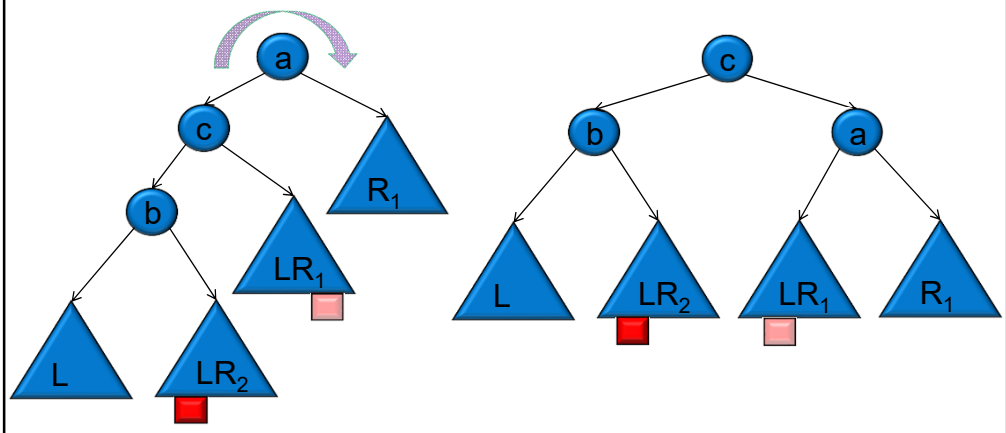
AVL-Bäume: Problemfall



AVL-Bäume: Doppelrotation rechts



AVL-Bäume: Doppelrotation rechts



ADT AVLTree

Objektmengen: btree, elem

Operationen:

- **initBT:** ∅ → btree
- **isBT:** btree → bool
- **insertBT:** btree × elem → btree
- **deleteBT**: btree × elem → btree
- **printBT**: btree × filename → dot
- **isEmptyBT:** btree → bool
- **equalBT:** btree × btree → bool

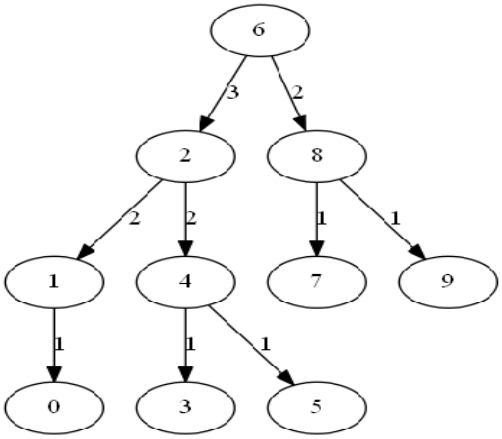
Vorgabe:
Funktional (nach außen)
Definition wie in der Vorlesung vorgestellt;
Die Elemente sind vom Typ „ganze Zahl".
Duplikate von Elementen sind nicht zulässig.
Alle für die ADT BTree bestehenden Funktionen müssen auch auf dem AVL Baum laufen und umgekehrt.
Technisch (nach innen)
Die ADT AVLTree ist mittels ADT BTree zu realisieren.
Die zugehörige Datei heißt avltree.erl

Fehlerbehandlung: analog bei der ADT Liste.

Aufgabe

```
digraph avltree
  {
  6 -> 2 [label = 3];
  6 -> 8 [label = 2];
  2 -> 1 [label = 2];
  2 -> 4 [label = 2];
  1 -> 0 [label = 1];
  4 -> 3 [label = 1];
  4 -> 5 [label = 1];
  8 -> 7 [label = 1];
  8 -> 9 [label = 1];
  }
```

dot -Tpng graph.dot > graph.png



Aufgabe

1) Test 1: Laufzeitmessung mit möglichst vielen Zahlen, erzeugt mittels `util:randomliste/1`.
2) Test 2:
   255 Zufallszahlen erstellen und mittels printBT ausgeben und *.png Bild erzeugen.
   Von diesen 255 Zufallszahlen 42 zufällig ausgewählte Zahlen löschen , mittels printBT ausgeben und *.png Bild erzeugen.
3) Test 1) und Test 2) mit einer `avltree.beam` eines anderen Teams durchführen.
4) Alle Ergebnisse dokumentieren und interpretieren. Dies in **einer** *.pdf nachvollziehbar zusammenfassen.