Algorithmen und Datenstrukturen
Balancierte Suchbäume

Matthias Teschner
Graphische Datenverarbeitung
Institut für Informatik
Universität Freiburg

SS 12

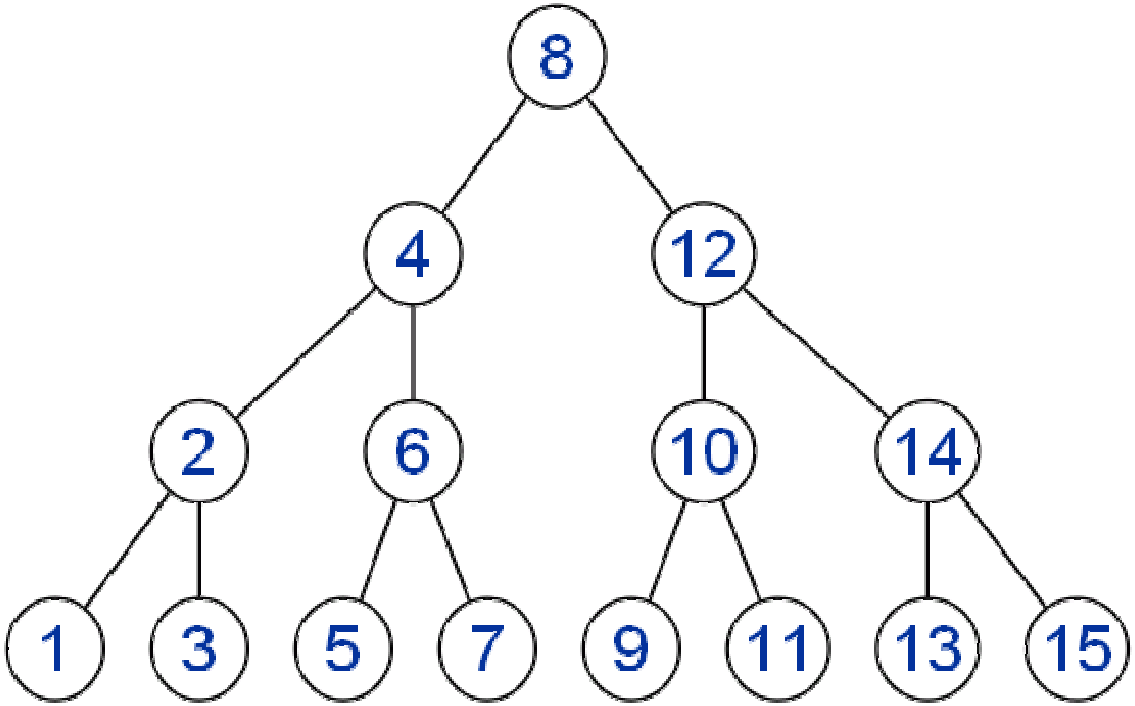
Überblick

- Einführung
- Einfügen und Löschen
- Einfügen
- Löschen

Motivation

- Suchbaum
  - Datenstruktur zur Repräsentation dynamischer Mengen
  - unterstützt `insert, search, delete, minimum, maximum, predecessor, successor`
  - Grundoperationen im mittleren Fall in O (log n), was der Baumhöhe entspricht. Schlechtester Fall O (n), wenn der Baum zur linearen Liste degeneriert ist.
- Balancierter Suchbaum
  - `insert, delete` unterliegen Bedingungen an die Höhendifferenz von linken und rechten Teilbäumen eines Knotens
  - für jeden Knoten soll die Höhendifferenz von linkem und rechtem Teilbaum maximal eins sein
  - verbessert schlechtesten Fall aller Grundoperationen zu O (log n)

Balancierter Suchbaum



balancierter Suchbaum

degenerierter Suchbaum

Balancierte Bäume

- Balancierte Bäume sind von großer Bedeutung, um die Laufzeit der Grundoperationen zu optimieren
- Implementierung von Einfüge- und Löschen-Operation hängt von verschiedenen Parametern ab
- Balance-Bedingung
  - Höhenbedingung
  - Gewichtsbedingung
  - Strukturbedingung
- Art des Baums
  - Suchbaum
  - Bereichsbaum

AVL-Baum

- Suchbaum mit modifizierten Einfüge- und Entferne-Operationen zur Einhaltung einer Höhenbedingung
- verhindert Degenerieren des Suchbaums
- Höhenunterschied von linkem und rechtem Teilbaum aller Knoten des Suchbaums ist maximal eins
- dadurch ist die Höhe des Suchbaums O (log n), womit alle weiteren Grundoperationen in O (log n) ausführbar sind.

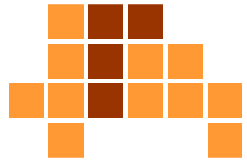
Georgy Maximovich **A**delson-**V**elskii, Yevgeniy Mikhailovich **L**andis,
An algorithm for the organization of information,
Doklady Akademia Nauk SSSR 1962.

AVL-Baum

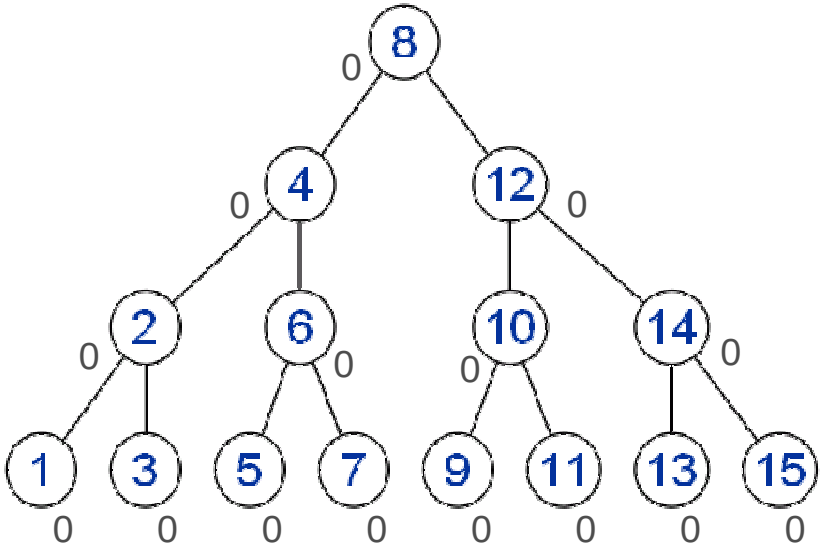
- Definition:
  Ein binärer Suchbaum heißt AVL-Baum oder höhenbalanciert, wenn sich für jeden Knoten die Höhe seines rechten Teilbaums und die Höhe seines linken Teilbaums um maximal eins unterscheiden.

- Balance-Grad (Knoten)
  = Höhe (rechter Teilbaum) – Höhe (linker Teilbaum)
  $\in \{-1, 0, 1\}$
- bal (v) = h (T_r) – h (T_l)

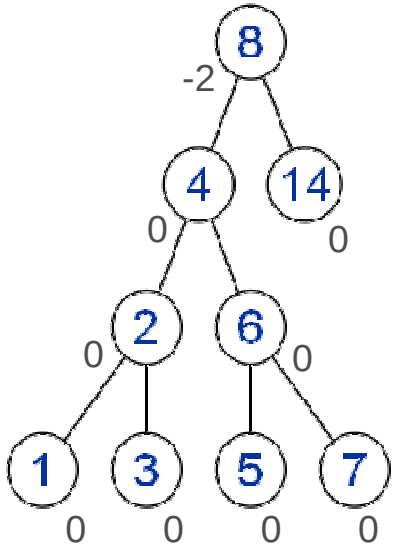
Beispiele



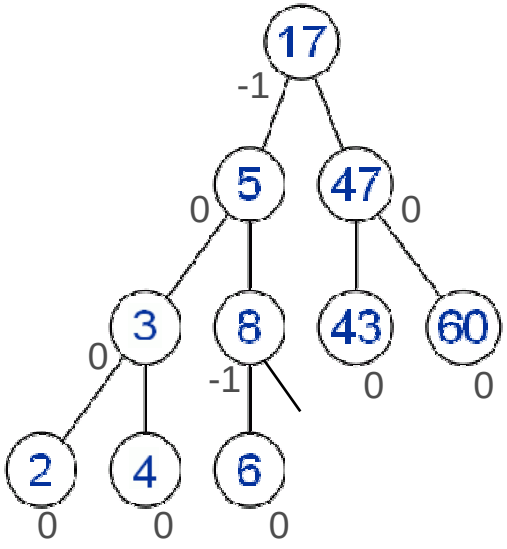
- Suchbäume mit Balance-Grad



AVL-Baum



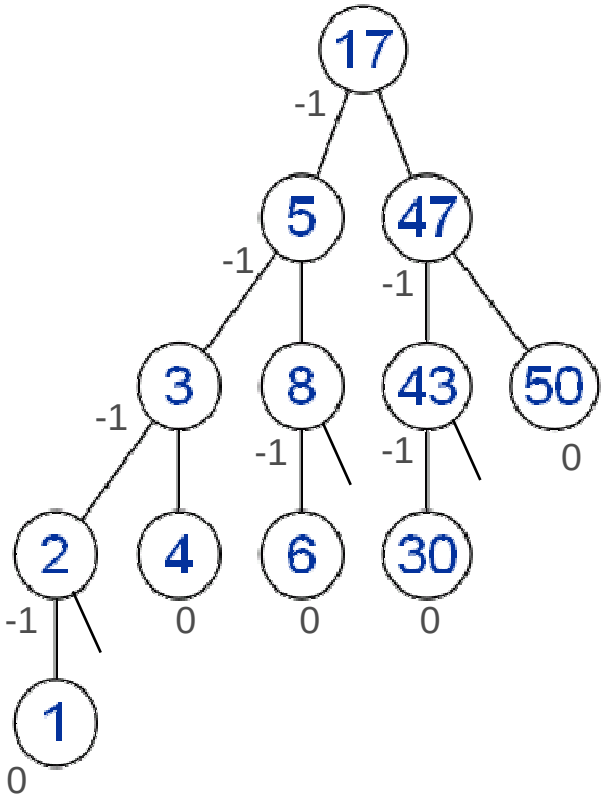
kein AVL-Baum



AVL-Baum

Höhe eines AVL-Baums

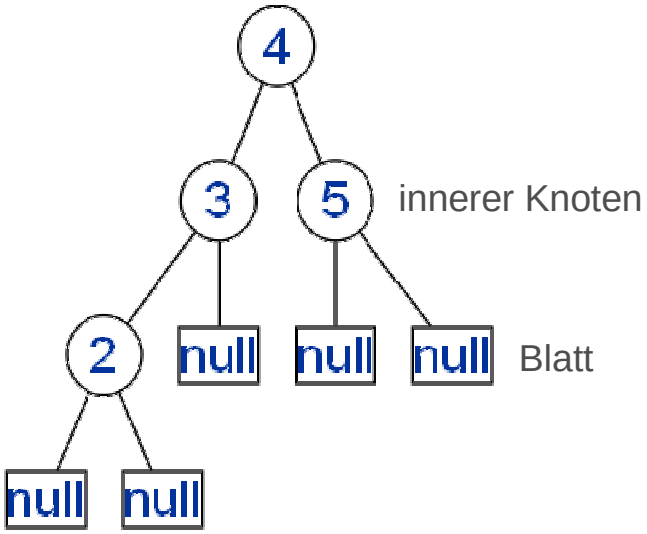
- O (log n) oder O (n) ?
- Kann der Baum degenerieren ?



AVL-Baum,
obwohl mehrere Ebenen nicht vollständig sind

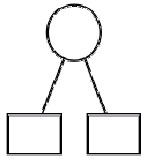
Zusammenhang Knoten / Höhe

- höchstmöglicher Baum für gegebene Zahl von Knoten ?
- minimale Zahl von Knoten bei gegebener Höhe ?
- minimale Zahl von Blättern bei gegebener Höhe ?
  - Binärbaum: Zahl der Blätter = Zahl der inneren Knoten + 1

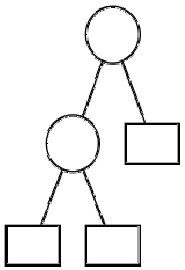


AVL-Baum mit Höhe h und minimaler Blattzahl

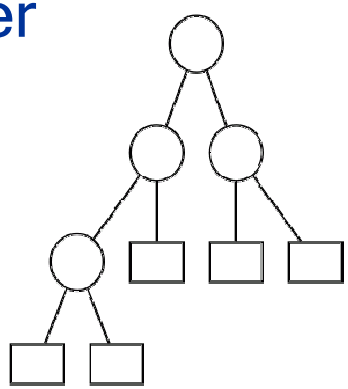
- Höhe 1, 2 Blätter



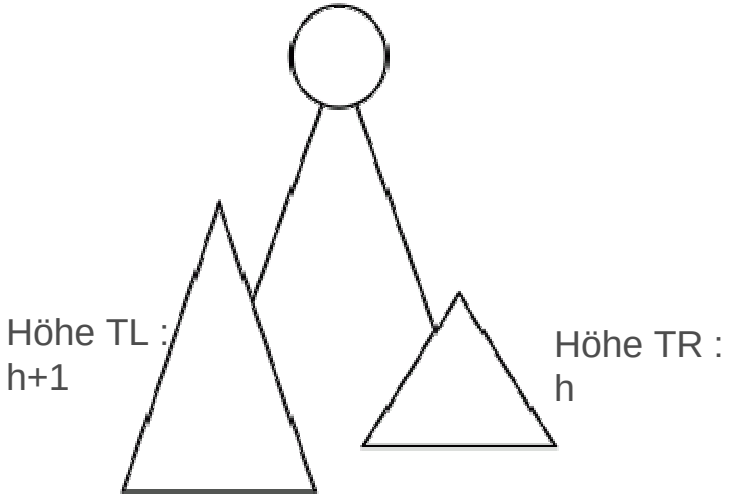
- Höhe 2, 3 Blätter



- Höhe 3, 5 Blätter



Höhe h+2



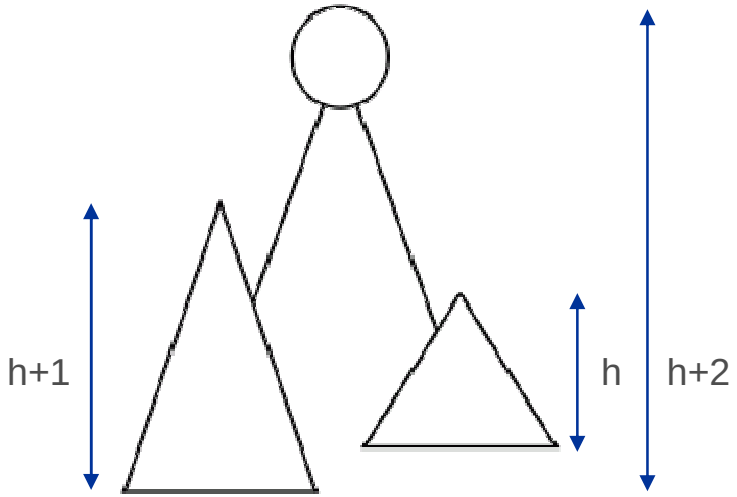
Blätter (h+2) = Blätter (h) + Blätter (h+1)

Ist minimal:
- h(TL) = h → Gesamthöhe ≠ h+2
- h(TL) = h+2 → Gesamthöhe ≠ h+2
- h(TR) = h-1 → kein AVL-Baum
- h(TR) = h+1 → mehr Blätter als bei h(TR) = h

Minimale Blattzahl

- Ein AVL-Baum der Höhe h hat mindestens F_{h+2} Blätter mit

  $F_0 = 0$
  $F_1 = 1$
  $F_{h+2} = F_{h+1} + F_h$



- F_h ist die h-te Fibonacci-Zahl

- Äquivalent zu: Ein AVL-Baum der Höhe h hat mindestens F_{h+2} -1 innere Knoten.

Höhe eines AVL-Baums

- Die Höhe h eines AVL-Baums mit n Blättern (n-1 inneren Knoten) ist in O (log n) bzw. $h \leq c \cdot \log n$, c – konstant .

- Begründung:

$$n \geq F_{h+2} = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{h+2} - \left(\frac{1-\sqrt{5}}{2} \right)^{h+2} \right)$$

$$n \geq F_{h+2} \approx 1.2 \cdot 1.6^{h}$$

$$c \cdot \log n \geq h$$

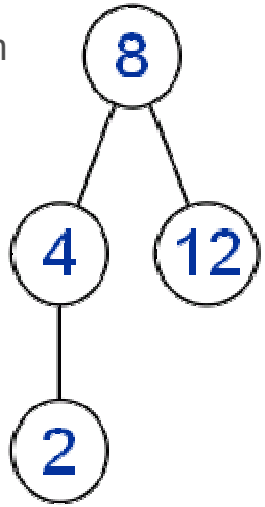
Überblick

- Einführung
- Einfügen und Löschen
- Einfügen
- Löschen

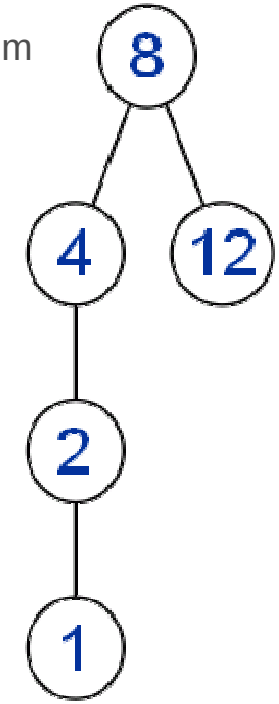
Problem

- einfaches Einfügen kann AVL-Bedingung verletzen

AVL-Baum



kein AVL-Baum nach dem Einfügen von 1



- Baum muss eventuell umstrukturiert werden

Eigenschaften / Schwierigkeiten

- modifiziertes Einfügen und Löschen
  - kann Suchbaumeigenschaft ausnutzen
  - muss Suchbaumeigenschaft erhalten
  - muss die Höhendifferenz aller Knoten auf maximal eins begrenzen
  - sollte weiterhin in O (log n) ausführbar sein
- Suchen, Maximum, Minimum, Vorgänger, Nachfolger
  - sind von veränderten Einfüge- und Löschen-Operationen nicht betroffen, wenn Suchbaumeigenschaft erhalten bleibt

Implementierung

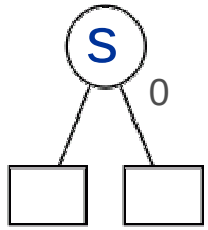
- Knoten müssen Balance-Grad mitführen,
  um AVL-Bedingung bei Operationen zu garantieren
- bal (p) = h (p.right) – h (p.left) $\in$ {-1, 0, 1}
- Absolute Höhe eines Knotens interessiert nicht. AVL-Bedingung sagt lediglich etwas über den Balance-Grad aus.
- Einfüge- und Löschen-Operationen
  - aktualisieren den Balance-Grad für alle Knoten
  - nehmen Strukturveränderungen am Baum vor,
    wenn aktualisierter Balance-Grad $\notin$ {-1, 0, 1}

Überblick

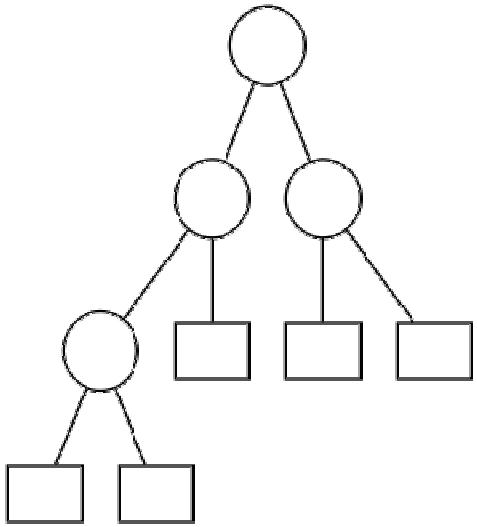
- Einführung
- Einfügen und Löschen
- Einfügen
- Löschen

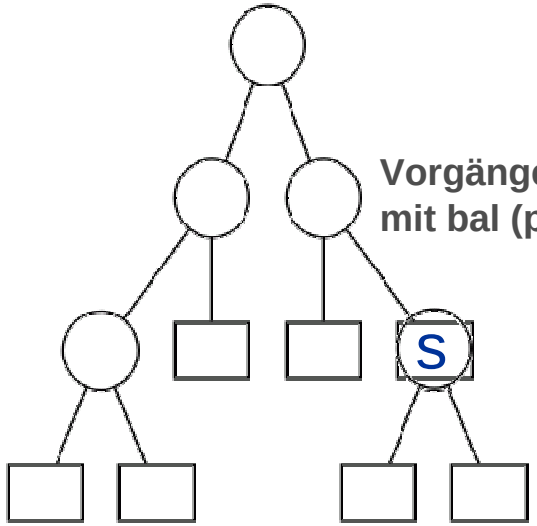
Fallunterscheidung

- Baum leer
  - generiere Knoten mit zwei Blättern
  - Balance-Grad ist 0



- Baum ist nicht leer
  - generiere Knoten mit zwei Blättern an einem Blatt mit Vorgänger-Knoten p

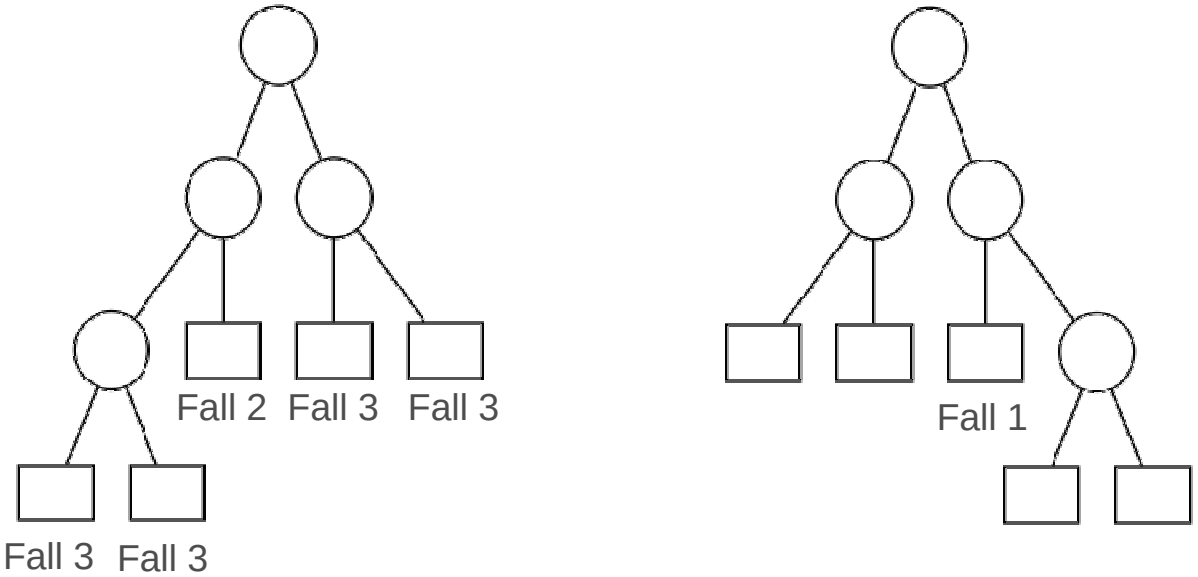




Vorgänger-Knoten p
mit bal (p) ∈ {-1, 0, 1}

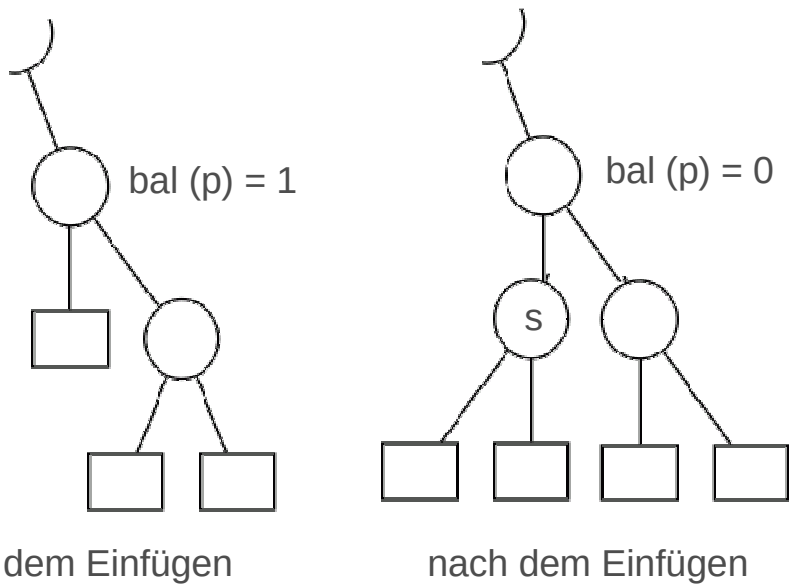
Fallunterscheidung

- Zustand vor dem Einfügen
- Da p Vorgänger von einem Blatt ist und bal (p) $\in$ {-1, 0, 1},
  - linker Nachfolger von p ist Blatt, der rechte nicht (Fall 1: bal (p) = 1)
  - rechter Nachfolger von p ist Blatt, der linke nicht (Fall 2: bal (p) = -1)
  - beide Nachfolger von p sind Blätter (Fall 3: bal (p) = 0)



Fall 1

- bal (p) = 1
  - füge inneren Knoten mit Schlüssel s und Balance-Grad 0 ein
  - aktualisiere Balance-Grad des Vorgänger-Knotens p auf bal (p) = 0



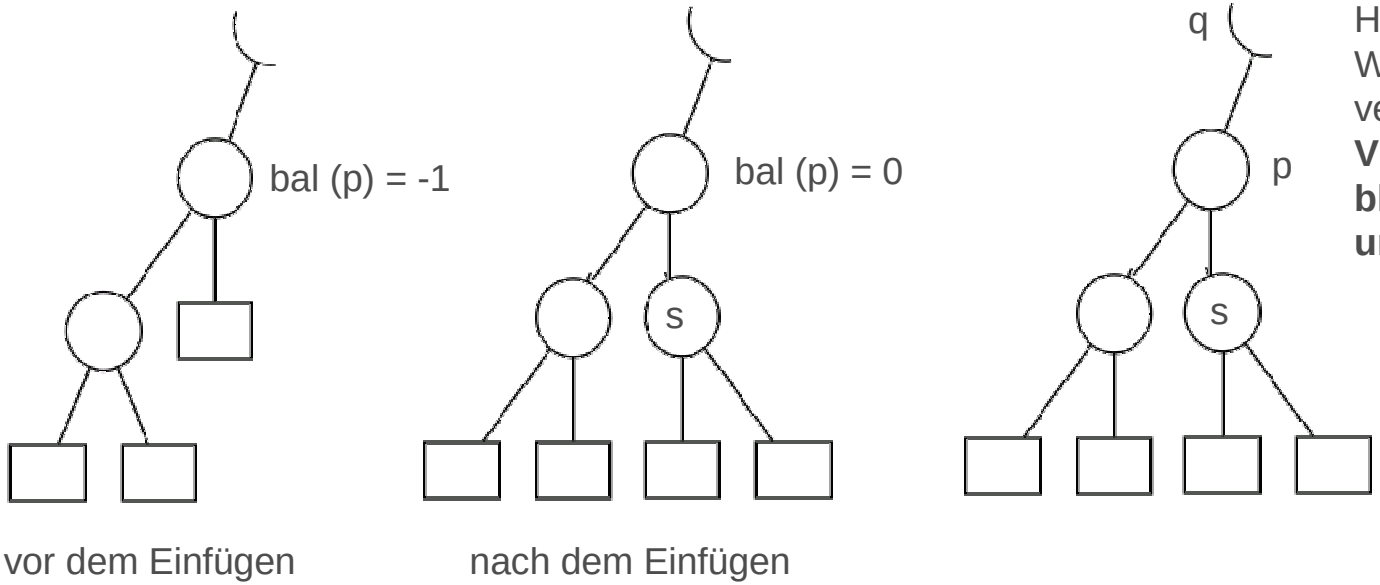
vor dem Einfügen

nach dem Einfügen

Höhe des Teilbaums mit Wurzel p hat sich nicht verändert. **Für alle Vorgänger q von p bleibt bal (q) unverändert.**

Fall 2

- bal (p) = -1
  - füge inneren Knoten mit Schlüssel s und Balance-Grad 0 ein
  - aktualisiere Balance-Grad des Vorgänger-Knotens p auf bal (p) = 0



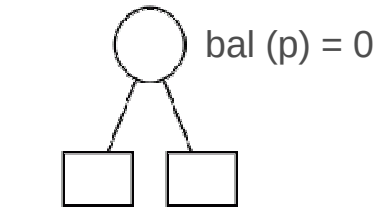
vor dem Einfügen

nach dem Einfügen

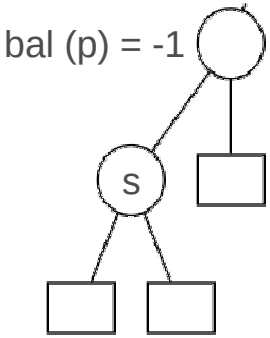
Höhe des Teilbaums mit Wurzel p hat sich nicht verändert. **Für alle Vorgänger q von p bleibt bal (q) unverändert.**

Fall 3

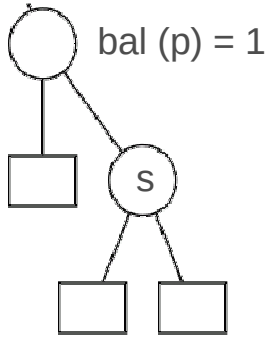
- bal (p) = 0
  - füge inneren Knoten mit Schlüssel s und Balance-Grad 0 ein
  - aktualisiere Balance-Grad des Vorgänger-Knotens p auf bal (p) = 1 oder bal (p) = -1



vor dem Einfügen



nach dem Einfügen,
Fall a: s < p.key



nach dem Einfügen,
Fall b: s > s.key

  - die Höhe des Teilbaums mit Wurzel p wächst um 1. Balance-Grad für Vorgänger-Knoten von p wird ungültig.

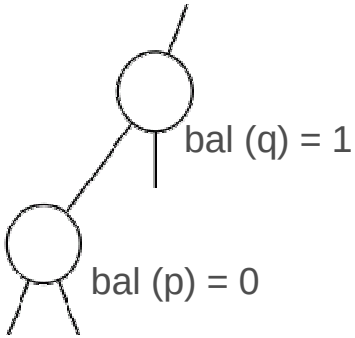
Aktualisierung der Vorgänger von p für Fall 3

- Funktion upin (p)
  - läuft von p zur Wurzel
  - aktualisiert die Balance-Grade der Vorgänger-Knoten von p
  - nimmt ggf. Umstrukturierungen am Baum vor (sogenannte Rotationen), um AVL-Baum-Eigenschaft wiederherzustellen
  - weiss, dass bal (p) $\in$ {-1, 1}
    (vor dem Einfügen war bal (p) = 0)
  - weiss, dass Höhe des Teilbaums mit Wurzel p um eins gewachsen ist
  - weiss, dass Teilbaum mit Wurzel p ein AVL-Baum ist

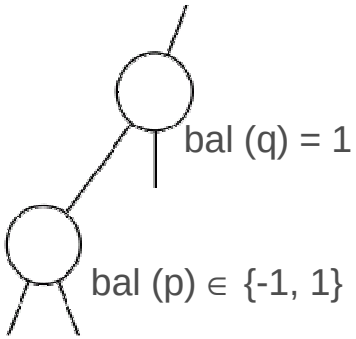
p kann Wurzel des linken oder des rechten Teilbaums des Vorgängers von p sein.
Im Folgenden wird nur der Fall betrachtet, dass p die Wurzel des linken Teilbaums ist.

Fall 3.1: bal (q) = 1

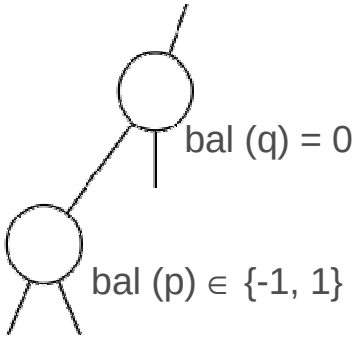
- Balance-Grad des Vorgängers q von p ist vor dem Einfügen 1



vor dem Einfügen



nach dem Einfügen,
vor upin (p)



nach dem Einfügen,
nach upin (p)

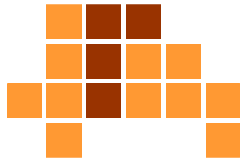
Höhe des Teilbaums mit Wurzel q hat sich nicht verändert.

Fertig.

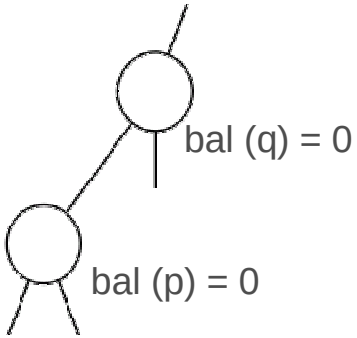
Fall 3.1: bal (q) = 1

- Vor dem Einfügen ist der rechte Teilbaum von q um eins höher als der linke Teilbaum.
- Das Einfügen vergrößert die Höhe des linken Teilbaums um eins.
- Also muss bal (q) von eins auf null korrigiert werden.
- Vorgänger von q sind nicht von Änderungen betroffen, da die Höhe des Teilbaums mit Wurzel q unverändert ist.

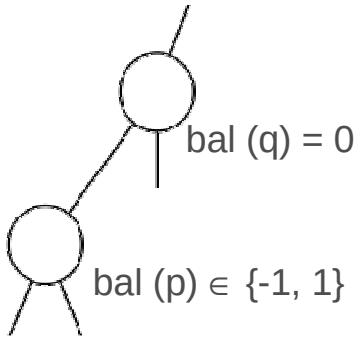
Fall 3.2: bal (q) = 0



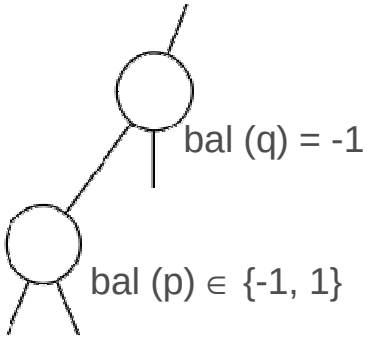
- Balance-Grad des Vorgängers q von p ist vor dem Einfügen 0



vor dem Einfügen



nach dem Einfügen,
vor upin (p)



nach dem Einfügen,
nach upin (p)

Aufruf von upin (q)

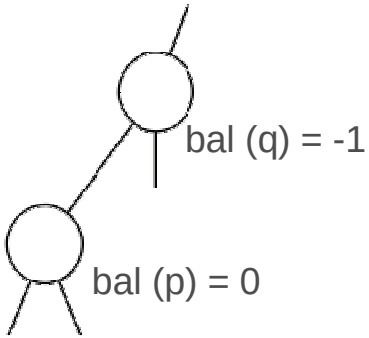
Wenn q die Wurzel ist, **fertig**.

Fall 3.2: bal (q) = 0

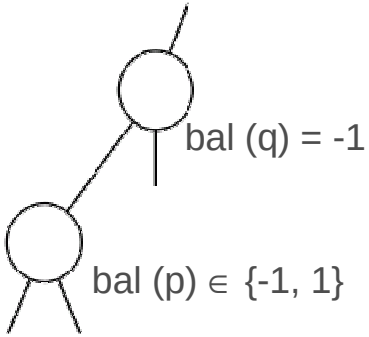
- Vor dem Einfügen sind linker und rechter Teilbaum von q gleich hoch.
- Das Einfügen vergrößert die Höhe des linken Teilbaums um eins.
- Also muss bal (q) von null auf -1 korrigiert werden.
- Vorgänger von q sind von Änderungen betroffen, da sich die Höhe des Teilbaums mit Wurzel q um eins vergrößert hat.
- Rekursiver Aufruf von upin (q) .

Fall 3.3: bal (q) = -1

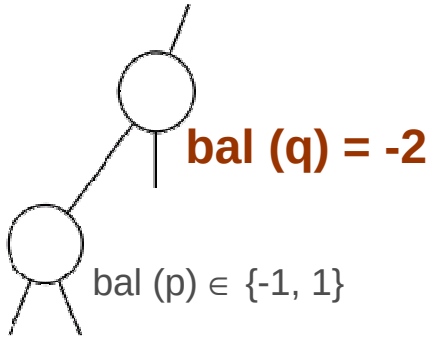
- Balance-Grad des Vorgängers q von p ist vor dem Einfügen -1



vor dem Einfügen



nach dem Einfügen,
vor upin (p)



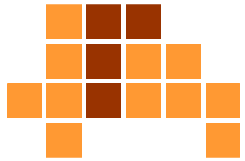
nach dem Einfügen,
nach upin (p)

- Verletzung der AVL-Baum-Eigenschaft erfordert Umstrukturierung des Baums.

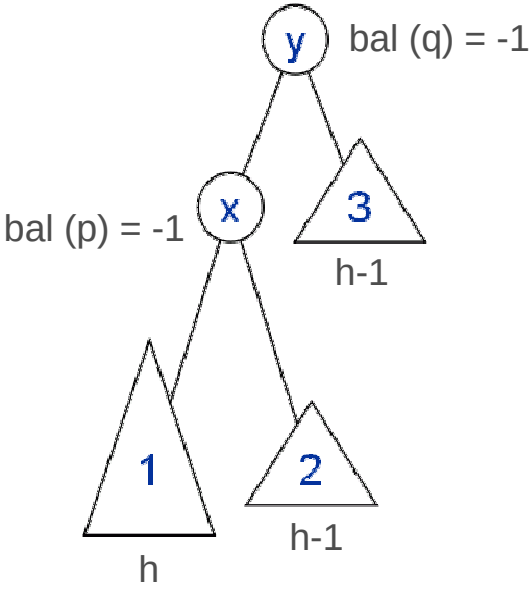
Fall 3.3: bal (q) = -1

- Vor dem Einfügen ist der linke Teilbaum von q um eins höher als der rechte Teilbaum von q.
- Das Einfügen vergrößert die Höhe des linken Teilbaums um eins.
- Also ist die Höhendifferenz von linkem und rechten Teilbaum auf zwei gewachsen, was die AVL-Baum-Eigenschaft verletzt.
- Umstrukturierung des Baums (Rotation)

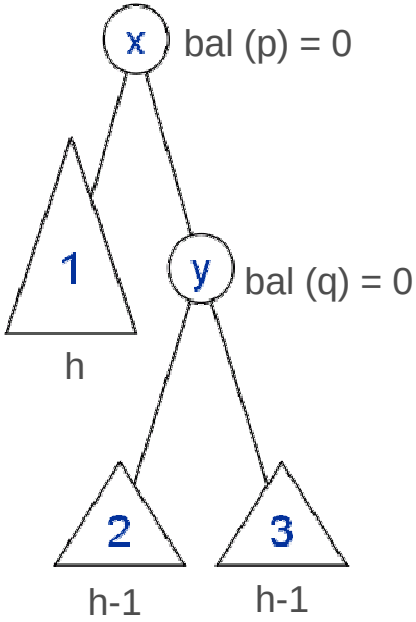
Fall 3.3.1: bal (q)=-1, bal (p)=-1



- Rotation nach rechts



vor der Rotation



nach der Rotation

Durch die Rotation wurde die Höhe des Teilbaums wieder auf die Höhe vor dem Einfügen reduziert.

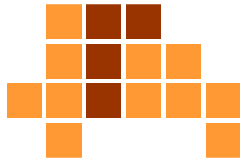
Dadurch ist keine weitere Behandlung von Vorgänger-Knoten durch upin notwendig.

Fertig.

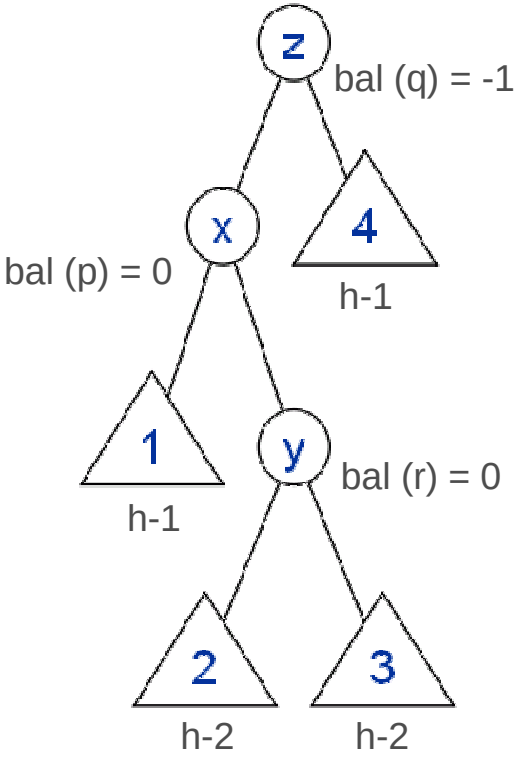
Fall 3.3.1: bal (q)=-1, bal (p)=-1

- Wert von h muss nicht bekannt sein.
- Aus h(1) = h folgt aber h(2)=h(3)=h-1
  wegen der Balance-Grade.
- AVL-Eigenschaft ist nach der Rotation erfüllt.
- Suchbaum-Eigenschaft bleibt erhalten.
  - Schlüssel in 1 sind kleiner als x und y
  - Schlüssel in 2 sind kleiner als y und größer als x
  - Schlüssel in 3 sind größer als x und y
  - x ist kleiner als y
- Nach der Rotation ist die Höhe des betrachteten Teilbaums wieder auf Größe vor dem Einfügen reduziert.
- Keine weitere Behandlung von Vorgängern notwendig.

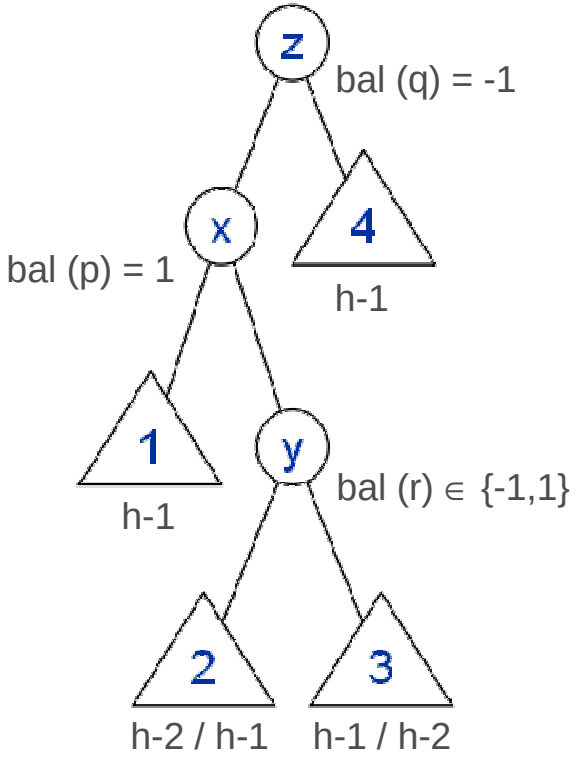
Fall 3.3.2: bal (q)=-1, bal (p)=1



- Situation vor der Umstrukturierung



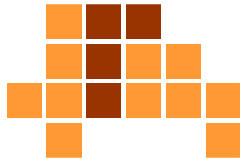
vor dem Einfügen



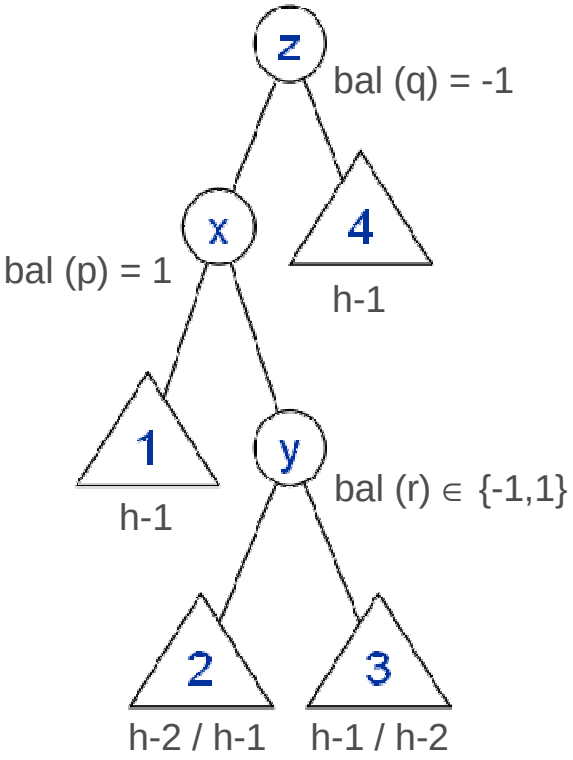
nach dem Einfügen,
vor der Umstrukturierung

Einfügen vergrößert Höhe von Teilbaum mit Wurzel q um eins.

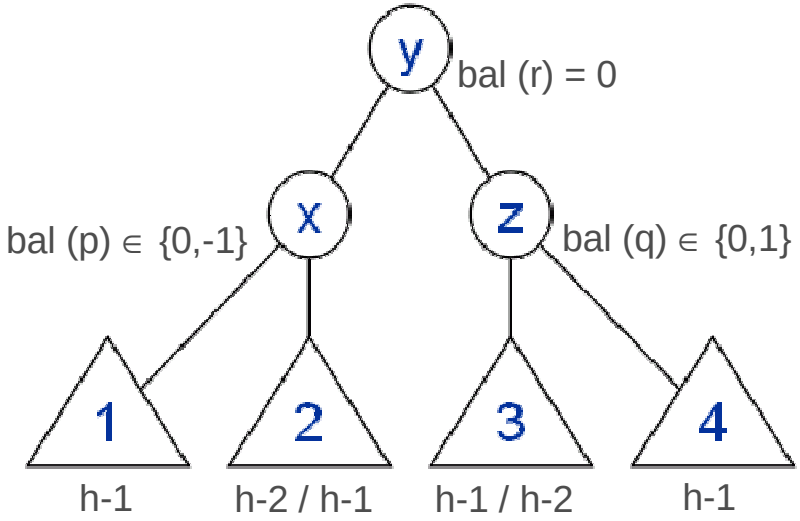
Fall 3.3.2: bal (q)=-1, bal (p)=1



- Links-Rotation (x,y), Rechts-Rotation (y,z)



vor der Umstrukturierung



nach der Umstrukturierung

Umstrukturierung verringert Höhe des Teilbaum um eins.

Fertig.

Fall 3.3.2: bal (q)=-1, bal (p)=1

- Aus h(1) = h-1 und wegen bal (p) folgt:
  - h (2) = h-1 und h (3) = h-2 nach dem Einfügen oder
  - h (2) = h-2 und h (3) = h-1 nach dem Einfügen
- AVL-Eigenschaft ist nach der zweifachen Rotation erfüllt.
- Suchbaum-Eigenschaft bleibt erhalten.
  - Schlüssel in 1 sind kleiner als x, y, z
  - Schlüssel in 2 sind kleiner als y, z und größer als x
  - Schlüssel in 3 sind kleiner als z und größer als x, y
  - Schlüssel in 4 sind größer als x, y, z
  - x ist kleiner als y ist kleiner als z
- Nach der Rotation ist die Höhe des betrachteten Teilbaums wieder auf Größe vor dem Einfügen reduziert.
- Keine weitere Behandlung von Vorgängern notwendig.

Implementierung

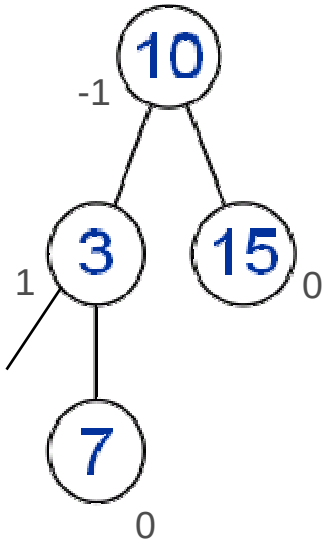
- upin (p) muss für zwei Fälle implementiert werden
  - p ist linker Teilbaum seines Vorgängers (siehe Folien)
  - p ist rechter Teilbaum seines Vorgängers (analog)
- upin (p) wird rekursiv für maximal O (log n) Knoten aufgerufen mit O (1) Operationen pro Knoten, wenn alle für die Umstrukturierung relevanten Zeiger auf Knoten bekannt sind
- Laufzeit für Einfügen
  - O (log n) für Knoten einfügen + O (log n) für Wiederherstellung des AVL-Baums

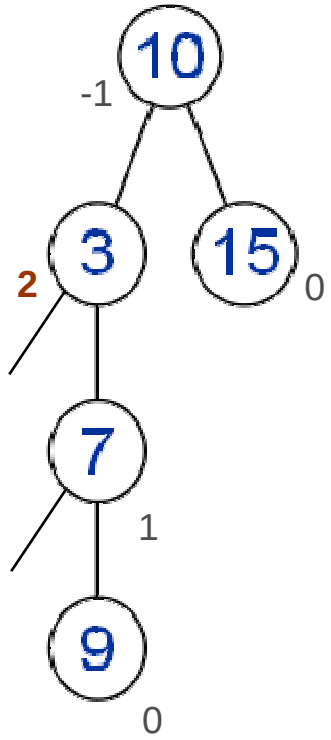
Einfügen - Zusammenfassung

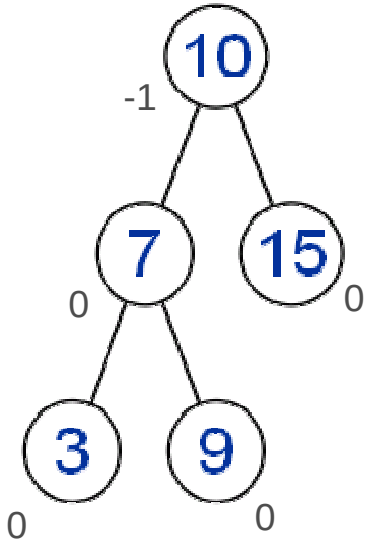
- Einfügen hinter Knoten p
- bal (p) = 1 oder -1: Einfügen und bal (p) aktualisieren,
  - Höhe des Teilbaums mit Wurzel p wurde nicht verändert
  - Vorgänger von p nicht betroffen
- bal (p) = 0: Einfügen und bal (p) aktualisieren
  - Höhe des Teilbaums mit Wurzel p wurde durch das Einfügen um eins erhöht
  - upin (p) aktualisiert bal (q) für alle Vorgängerknoten q von p
  - eventuell Umstrukturierungen in konstanter Laufzeit notwendig

Beispiel

- Links-Rotation für (3, 7)

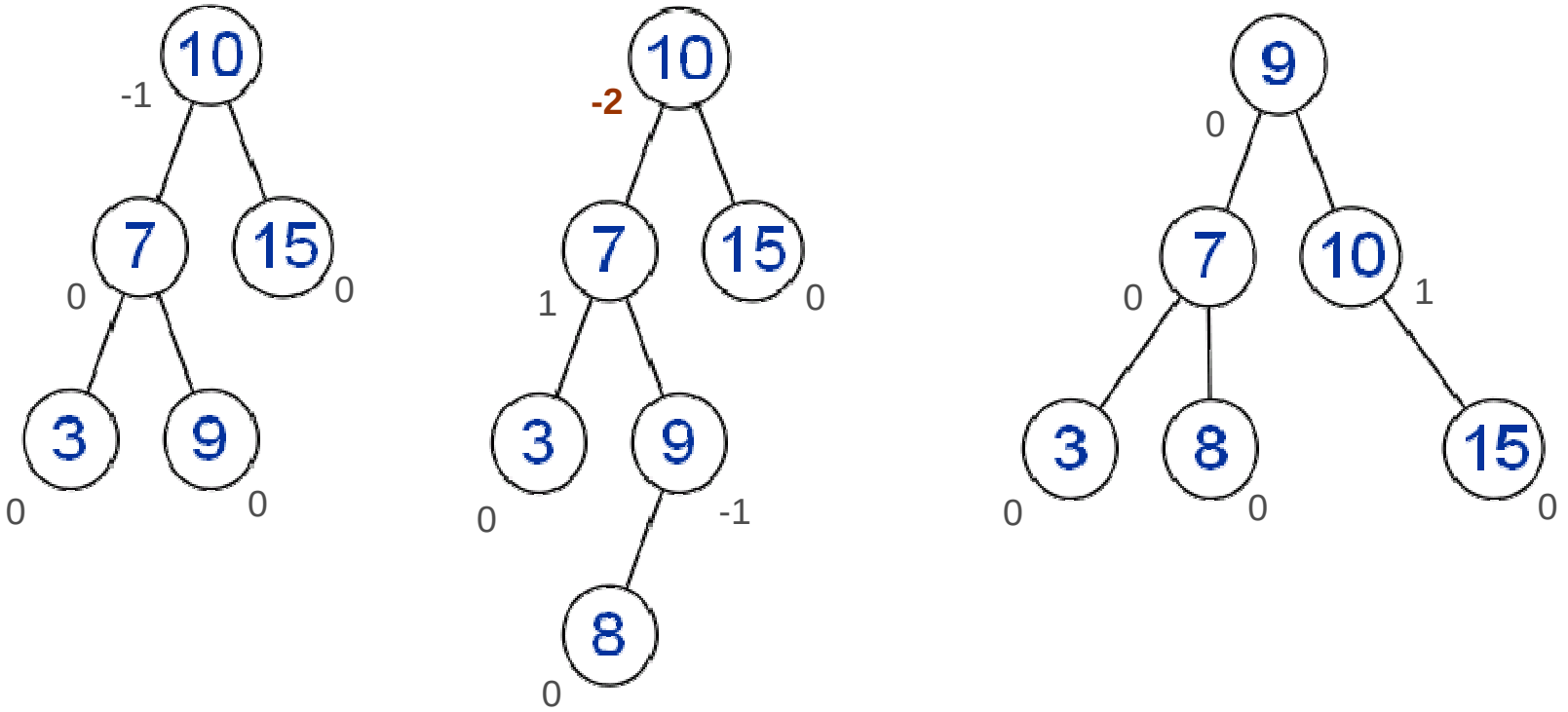






Beispiel

- Links-Rotation für (7, 9), Rechts-Rotation für (9, 10)



Überblick

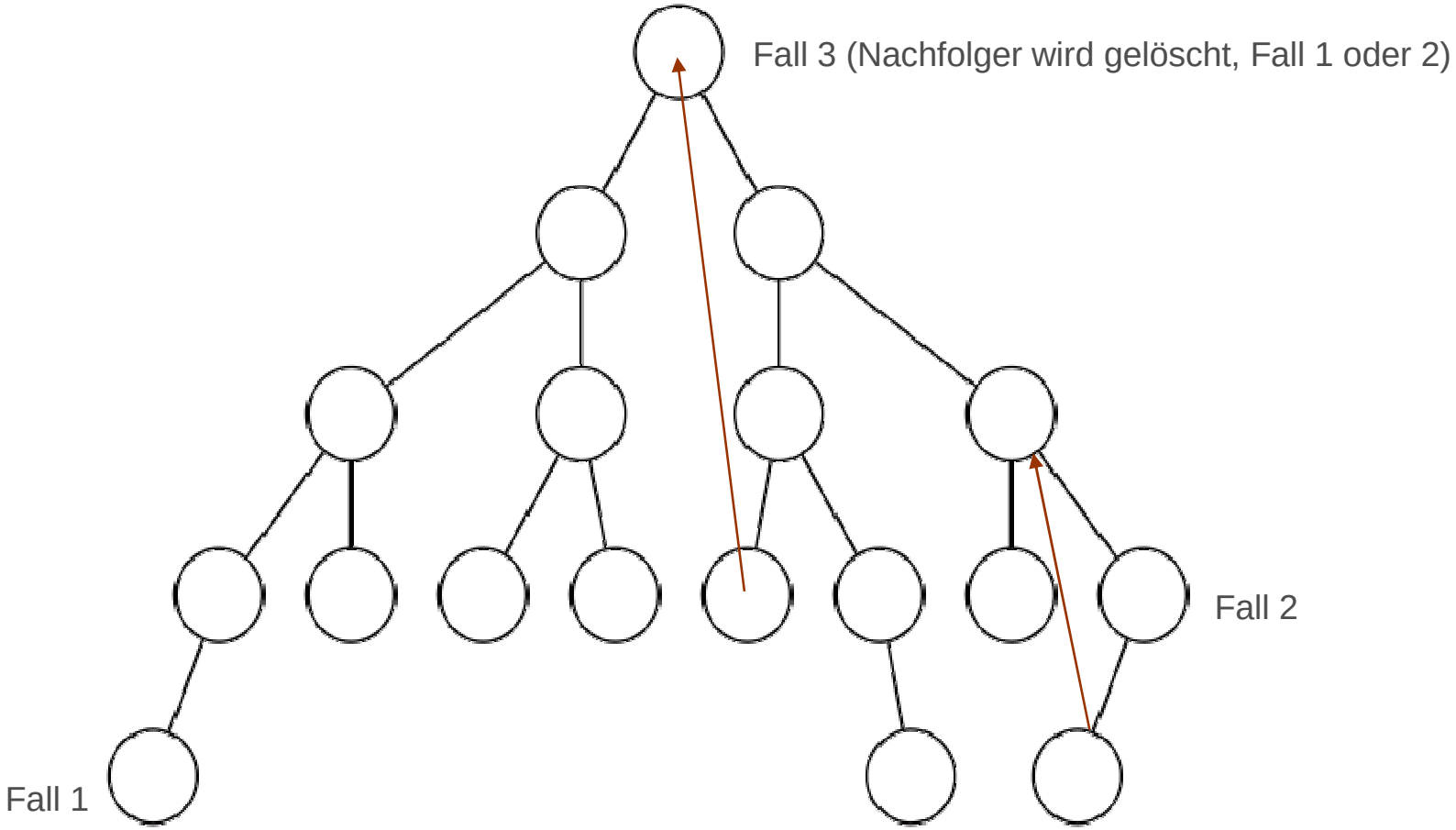
- Einführung
- Einfügen und Löschen
- Einfügen
- Löschen

Löschen im Suchbaum

- 3 Fälle
- Fall 1. Knoten hat keine Kinder
  - lösche Knoten
- Fall 2. Knoten hat ein Kind
  - lösche Knoten p
  - Vater von p wird mit Kind von p verbunden
- Fall 3. Knoten hat zwei Kinder
  - Knoten p wird durch Nachfolger q von p ersetzt
  - Knoten q wird gelöscht (Fall 1 oder 2)

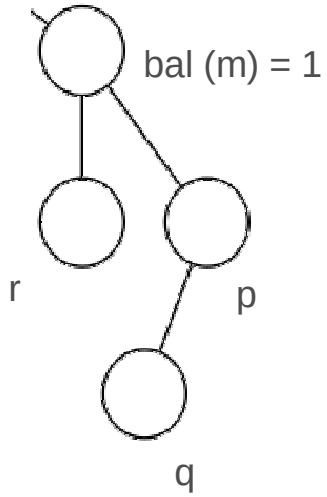
- Löschen im AVL-Baum muss nur Fälle 1 und 2 behandeln.

Löschen im Suchbaum



Löschen im AVL-Baum

- Balance-Grade müssen für die Fälle 1 und 2 aktualisiert werden.
- Wenn AVL-Bedingung verletzt ist, wird der Baum umstrukturiert (Rotation).



Beispiel:

Löschen von p oder q: bal (m) = 0
Löschen von r: Links-Rotation mit (m, p)

Rekursive Aktualisierung
aller Vorgänger bis zur Wurzel.

Zusammenfassung

- AVL-Baum
  - Balancierter Suchbaum
  - `insert, delete` unterliegen Bedingungen an die Höhendifferenz von linken und rechten Teilbäumen eines Knotens
  - für jeden Knoten soll die Höhendifferenz von linkem und rechtem Teilbaum maximal eins sein
  - verbessert schlechtesten Fall aller Grundoperationen zu O (log n)
    - `insert, delete, search, maximum, minimum, …`
  - `insert, delete` führen unter Umständen Strukturveränderungen am Baum durch (Rotationen)
  - Strukturveränderung in O (log n) · O(1)

Nächstes Thema

- Algorithmen / Datenstrukturen
  - Graphen