6. Binäre Suchbäume

Natürliche binäre Suchbäume

- Begriffe und Definitionen
- Grundoperationen: Einfügen, sequentielle Suche, direkte Suche, Löschen
- Bestimmung der mittleren Zugriffskosten

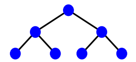
Balancierte Binärbäume

AVL-Baum

- Einfügen mit Rotationstypen
- Löschen mit Rotationstypen
- Höhe von AVL-Bäumen

Gewichtsbalancierte Binärbäume

Positionssuche mit balancierten Bäumen (Lösung des Auswahlproblems)



Binäre Suchbäume

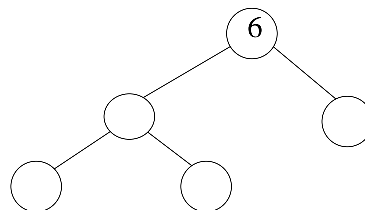
Def.: Ein natürlicher <u>binärer Suchbaum</u> B ist ein Binärbaum;
er ist entweder leer oder jeder Knoten in B enthält einen Schlüssel und:

(1) alle Schlüssel im linken Unterbaum von B sind kleiner als der Schlüssel in der Wurzel von B

(2) alle Schlüssel im rechten Unterbaum von B sind größer als der Schlüssel in der Wurzel von B

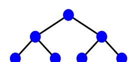
(3) die linken und rechten Unterbäume von B sind auch binäre Suchbäume.

Beispiel:



4 Grundoperationen:

- Einfügen
- direkte Suche
- sequentielle Suche
- Löschen



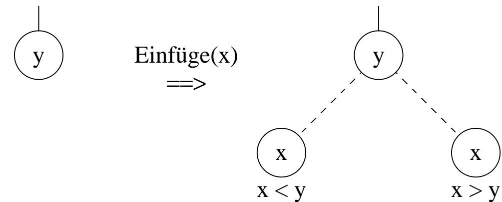
Einfügen in binären Suchbäumen

Neue Knoten werden immer als Blätter eingefügt

Suche der Einfügeposition:

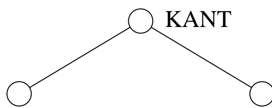
Aussehen des Baumes wird durch die Folge der Einfügungen bestimmt: *reihenfolgeabhängige* Struktur

n Schlüssel erlauben n! verschiedene Schlüsselfolgen



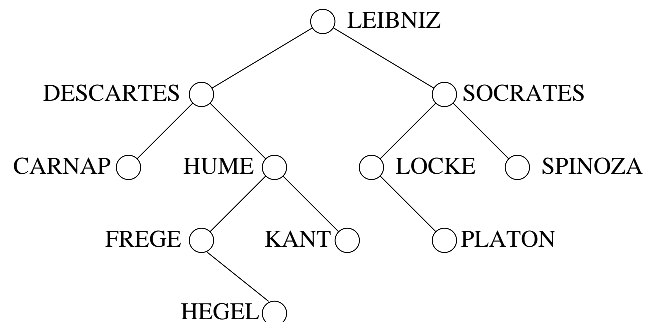
Einfügereihenfolge 1:

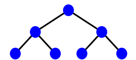
KANT, LEIBNIZ, HEGEL, HUME, LOCKE, SOCRATES, SPINOZA, DESCARTES, CARNAP, FREGE, PLATON



Einfügereihenfolge 2:

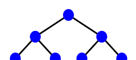
LEIBNIZ, DESCARTES, CARNAP, HUME, SOCRATES, FREGE, LOCKE, KANT, HEGEL, PLATON, SPINOZA





Einfügen in binären Suchbäumen (2)

```
class BinaryNode {
  BinaryNode lChild = null;
  BinaryNode rChild = null;
  Orderable key = null;
   /** Konstruktor */
  BinaryNode(Orderable key) { this.key = key; } }

public class BinarySearchTree {
  private BinaryNode root = null;
  ...
  public void insert(Orderable key) throws TreeException {
     root = insert(root, key); }

  protected BinaryNode insert(BinaryNode node, Orderable key)
  throws TreeException {
     if (node == null) return new BinaryNode(key);
        else
           if (key.less(node.key))
              node.lChild = insert(node.lChild, key);
           else
              if (key.greater(node.key))
                 node.rChild = insert(node.rChild, key);
              else
                 throw new TreeException("Schluessel schon vorhanden!");
     return node; }
  ...
}
```

Suche in binären Suchbäumen

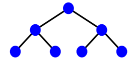
1. Sequentielle Suche

Einsatz eines Durchlauf-Algorithmus (Zwischenordnung)

2. Direkte Suche: **Vorgehensweise wie bei Suche nach Einfügeposition**

Suchen eines Knotens (rekursive Version):

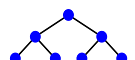
```
/** Rekursive Suche eines Schluessels im Baum */
public boolean searchRec (Orderable key) {
  return searchRec(root, key);
}

/** Rekursive Suche eines Schluessels im Teilbaum */
protected boolean searchRec (BinaryNode node, Orderable key) {
  if (node == null) return false;        // nicht gefunden
  if (key.less(node.key))
    return searchRec(node.lChild, key);  // suche im linken Teilbaum
  if (key.greater(node.key))
    return searchRec(node.rChild, key);  // suche im rechten Teilbaum
  return true;                           // gefunden
}
```

Suchen (2)

Suchen (iterative Version):

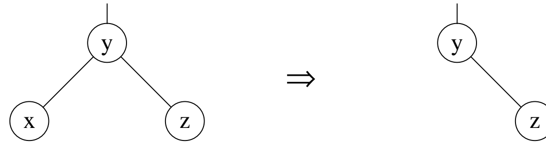
```
public boolean searchIter (Orderable key) {
  BinaryNode node = root;
  do {
    if (node == null) return false;  // nicht gefunden
    if (key.less(node.key))
      node = node.lChild;            // suche im linken Teilbaum
    else if (key.greater(node.key))
      node = node.rChild;            // suche im rechten Teilbaum
    else return true;                // gefunden
  } while (true);
}
```

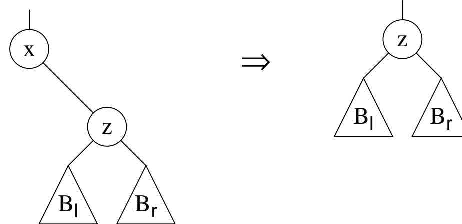
Löschen in binären Suchbäumen

Löschen ist am kompliziertesten

Fall 1: x ist Blatt

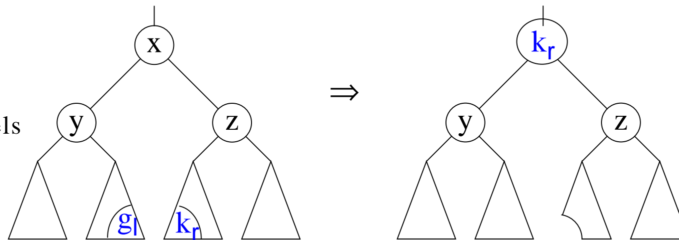


Fall 2/3: x hat leeren linken/rechten Unterbaum

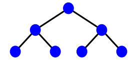


Fall 4: x hat zwei nicht-leere Unterbäume

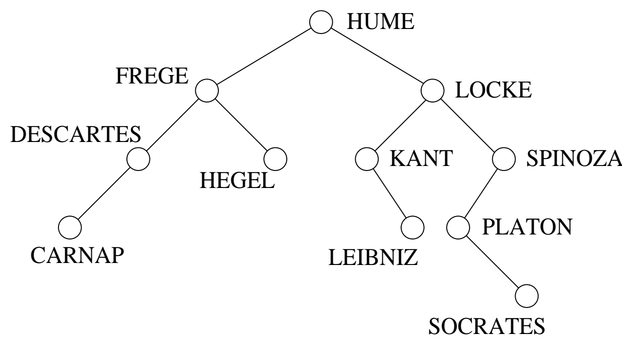
Heranziehen des größten Schlüssels im linken Unterbaum (g_l) oder des kleinsten Schlüssels im rechten Unterbaum (k_r)



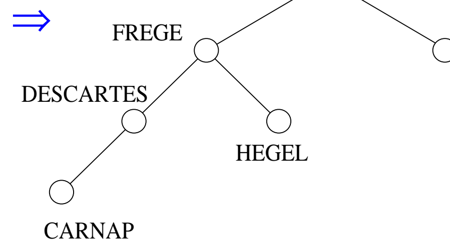
- Alternative: Jeder zu löschende Knoten wird speziell markiert; bei Such- und Einfügevorgängen wird er gesondert behandelt



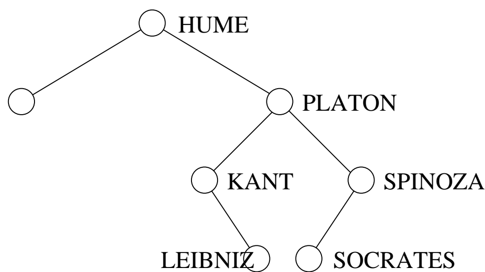
Löschen in binären Suchbäumen (2)



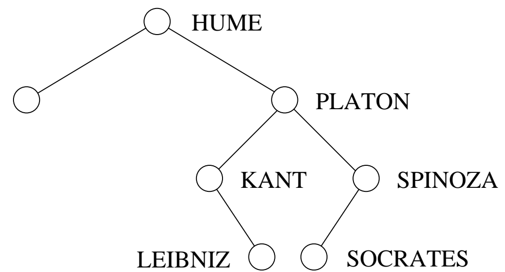
Lösche (LOCKE)

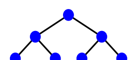


Lösche (DESCARTES)



Lösche (FREGE)





Binäre Suchbäume: Zugriffskosten

Kostenmaß: Anzahl der aufgesuchten Knoten bzw. Anzahl der benötigten Suchschritte oder Schlüsselvergleiche.

Kosten der Grundoperationen

- sequentielle Suche:
- Einfügen, Löschen, direkte Suche

Bestimmung der mittleren Zugriffskosten (direkte Suchkosten)

- Mittlere Zugriffskosten $\bar{z}$ eines Baumes B erhält man durch Berechnung seiner gesamten Pfadlänge PL als Summe der Längen der Pfade von der Wurzel bis zu jedem Knoten K_i.

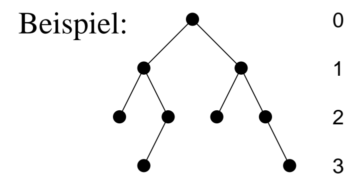
$$PL(B) = \sum_{i=1}^{n} Stufe(K_i)$$

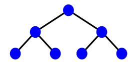
- mit n_i = Zahl der Knoten auf Stufe i gilt

$$PL(B) = \sum_{i=0}^{h-1} i \cdot n_i \quad \text{und} \quad \sum_{i=0}^{h-1} n_i = n \quad \text{= gesamte Knotenzahl}$$

- Die mittlere Pfadlänge ergibt sich zu $p = PL / n$
- Da bei jedem Zugriff noch auf die Wurzel zugegriffen werden muß, erhält man

$$\bar{z} = p + 1 = \frac{1}{n} \cdot \sum_{i=0}^{h-1} (i+1) \cdot n_i$$

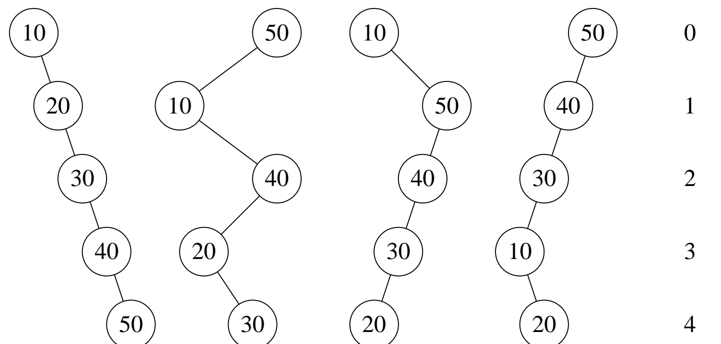




Binäre Suchbäume: Zugriffskosten (2)

Maximale Zugriffskosten

- Die längsten Suchpfade und damit die maximalen Zugriffskosten ergeben sich, wenn der binäre Suchbaum zu einer linearen Liste entartet

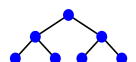


- Höhe: $h = l_{max} + 1 = n$
- Maximale mittlere Zugriffskosten:

$$\bar{z}_{max} = \frac{1}{n} \cdot \sum_{i=0}^{n-1} (i+1) \cdot 1 = n \cdot \frac{(n+1)}{2n} = \frac{(n+1)}{2} = O(n)$$

Minimale (mittlere) Zugriffskosten: können in einer fast vollständigen oder ausgeglichenen Baumstruktur erwartet werden

- Gesamtzahl der Knoten: $2^{h-1} - 1 < n \leq 2^h - 1$
- Höhe $h = \lfloor \log_2 n \rfloor + 1$
- Minimale mittlere Zugriffskosten: $\bar{z}_{min} \approx \log_2 n - 1$

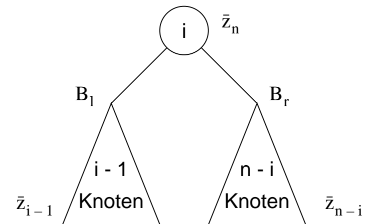


Binäre Suchbäume: Zugriffskosten (3)

Durchschnittliche Zugriffskosten

- Extremfälle der mittleren Zugriffskosten sind wenig aussagekräftig:
- Wie groß sind $\bar{z}_{min}$ und $\bar{z}_{max}$ bei n= 10, 10^3, 10^6, . . . ?
- Differenz der mittleren zu den minimalen Zugriffskosten ist ein Maß für Dringlichkeit von Balancierungstechniken

Bestimmung der mittleren Zugriffskosten



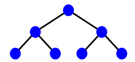
- n verschiedene Schlüssel mit den Werten 1, 2, ..., n seien in zufälliger Reihenfolge gegeben. Die Wahrscheinlichkeit, daß der erste Schlüssel den Wert i besitzt, ist 1/n (Annahme: gleiche Zugriffswahrscheinlichkeit auf alle Knoten)
- Für den Baum mit i als Wurzel erhalten wir

$$\bar{z}_n(i) = \frac{1}{n} \cdot ((\bar{z}_{i-1} + 1) \cdot (i-1) + 1 + (\bar{z}_{n-i} + 1) \cdot (n-i))$$

- Die Rekursionsgleichung läßt sich in nicht-rekursiver, geschlossener Form mit Hilfe der harmonischen Funktion $H_n = \sum_{i=1}^{n} \frac{1}{i}$ darstellen.

- Es ergibt sich $\bar{z}_n = 2 \cdot \frac{(n+1)}{n} \cdot H_n - 3$ $= 2 \ln(n) - c$.

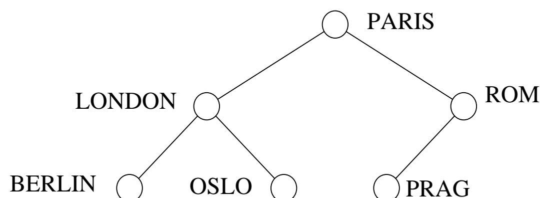
- Relative Mehrkosten: $\frac{\bar{z}_n}{\bar{z}_{min}} = \frac{2\ln(n) - c}{\log_2(n) - 1} \sim \frac{2\ln(n) - c}{\log_2(n)} = 2\ln(2) = 1{,}386\ldots$



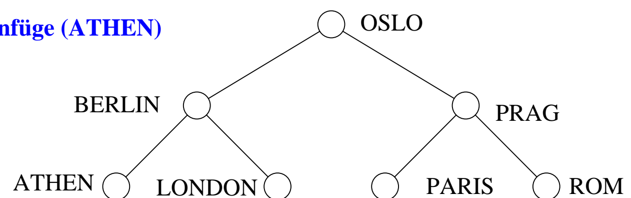
Balancierte Binärbäume

Der ausgeglichene binäre Suchbaum verursacht für alle Grundoperationen die geringsten Kosten

Perfekte Balancierung zu jeder Zeit kommt jedoch sehr teuer.



Einfüge (ATHEN)



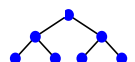
- In welchem Maße sollen Strukturabweichungen bei Einfügungen und Löschungen toleriert werden ?

Balancierte Bäume

- Ziel: schneller direkten Zugriff mit $\bar{z}_{max} \sim O(\log_2 n)$ sowie Einfüge- und Löschoperationen mit logarithmischen Aufwand
- Heuristik: für jeden Knoten im Baum soll die Anzahl der Knoten in jedem seiner beiden Unterbäume möglichst gleich gehalten werden
- Zwei unterschiedliche Vorgehensweisen:

  (1) die zulässige Höhendifferenz der beiden Unterbäume ist beschränkt (=> höhenbalancierte Bäume)

  (2) das Verhältnis der Knotengewichte der beiden Unterbäume erfüllt gewisse Bedingungen (=> gewichtsbalancierte Bäume)

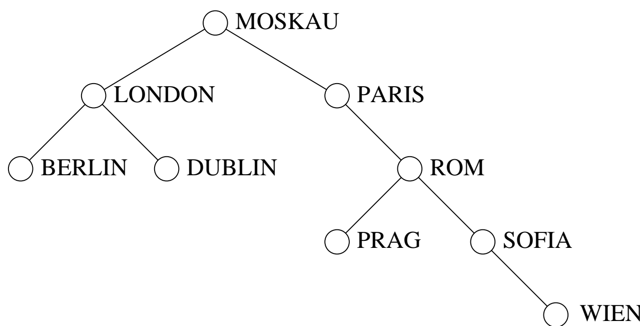


k-balancierter Binärbaum

Def.: Seien $B_l(x)$ und $B_r(x)$ die linken und rechten Unterbäume eines Knotens x. Weiterhin sei h (B) die Höhe eines Baumes B. Ein k-balancierter Binärbaum ist entweder leer oder es ist ein Baum, bei dem für jeden Knoten x gilt: $|h(B_l(x)) - h(B_r(x))| \leq \mathbf{k}$

k läßt sich als Maß für die zulässige Entartung im Vergleich zur ausgeglichenen Baumstruktur auffassen

Prinzip

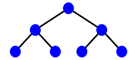


$|h(B_l(SOFIA)) - h(B_r(SOFIA))|$ =

$|h(B_l(ROM)) - h(B_r(ROM))|$ =

$|h(B_l(PARIS)) - h(B_r(PARIS))|$ =

$|h(B_l(MOSKAU)) - h(B_r(MOSKAU))|$ =



AVL-Baum

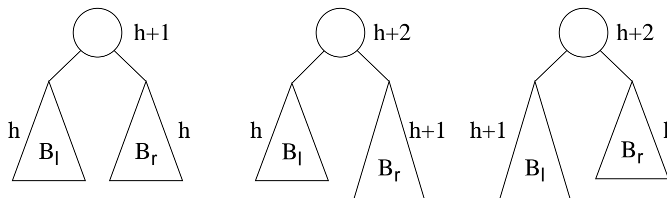
benannt nach russischen Mathematikern: Adelson-Velski und Landis

Def.: Ein 1-balancierter Binärbaum heißt AVL-Baum

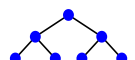
-> Balancierungskriterium: $|h(B_l(x)) - h(B_r(x))| \leq 1$

Konstruktionsprinzip:

- B_l und B_r seien AVL-Bäume der Höhe h und h+1. Dann sind die nachfolgend dargestellten Bäume auch AVL-Bäume:



Suchoperationen wie für allgemeine binäre Suchbäume



AVL-Baum: Wartungsalgorithmen

Wann und wo ist das AVL-Kriterium beim Einfügen verletzt ?

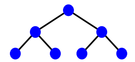
- Es kann sich nur die Höhe von solchen Unterbäumen verändert haben, deren Wurzeln auf dem Suchpfad von der Wurzel des Baumes zum neu eingefügten Blatt liegen
- Reorganisationsoperationen lassen sich lokal begrenzen; es sind höchstens h Knoten betroffen

Def.: Der Balancierungsfaktor BF(x) eines Knotens x ergibt sich zu

$$BF(x) = h(B_l(x)) - h(B_r(x)).$$

Knotendefinition

```
class AVLNode {
  int BF = 0;
  AVLNode lChild = null;
  AVLNode rChild = null;
  Orderable key = null;
  /** Konstruktor */
  AVLNode(Orderable key) { this.key = key; }
}
```

Einfügen in AVL-Bäumen

Sobald ein BF(x) durch eine Einfügung verletzt wird, muß eine Rebalancierung des Baumes durch sog. Rotationen durchgeführt werden.

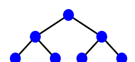
- Ausgangspunkt der Rotation ist der näheste Vater des neu eingefügten Knotens mit $BF = \mp 2$.
- Dieser Knoten dient zur Bestimmung des Rotationstyps. Er wird durch die von diesem Knoten ausgehende Kantenfolge auf dem Pfad zum neu eingefügten Knoten festgelegt.

Rotationstypen

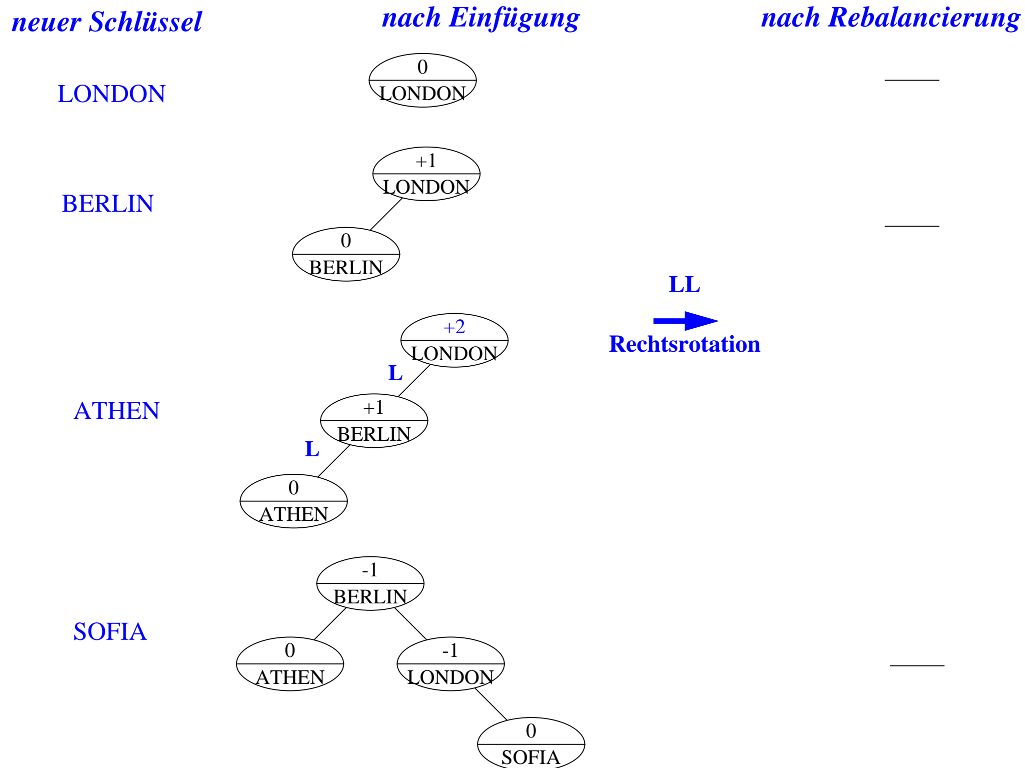
Es treten vier verschiedene Rotationstypen auf. Der neu einzufügende Knoten sei X. Y sei der bezüglich der Rotation kritische Knoten - der näheste Vater von X mit $BF = \mp 2$. Dann bedeutet:

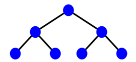
- RR: X wird im rechten Unterbaum des rechten Unterbaums von Y eingefügt (Linksrotation)
- LL: X wird im linken Unterbaum des linken Unterbaums von Y eingefügt (Rechtsrotation)
- RL: X wird im linken Unterbaum des rechten Unterbaums von Y eingefügt (Doppelrotation)
- LR: X wird im rechten Unterbaum des linken Unterbaums von Y eingefügt (Doppelrotation)

Die Typen LL und RR sowie LR und RL sind symmetrisch zueinander.

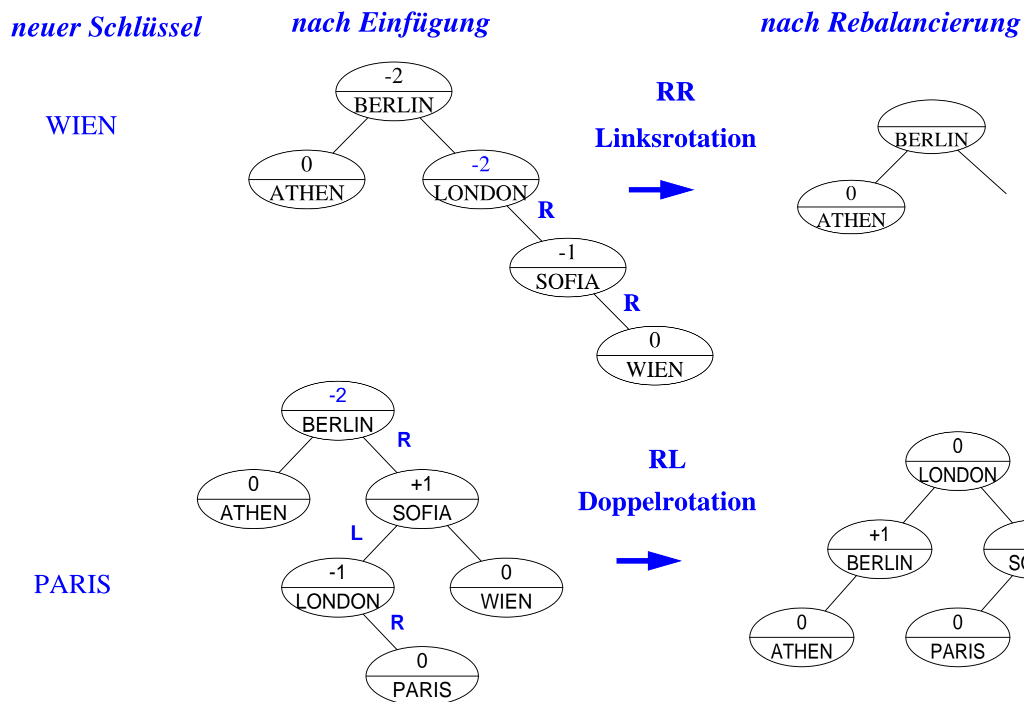


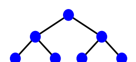
Einfügen in AVL-Bäumen (2)





Einfügen in AVL-Bäumen (3)





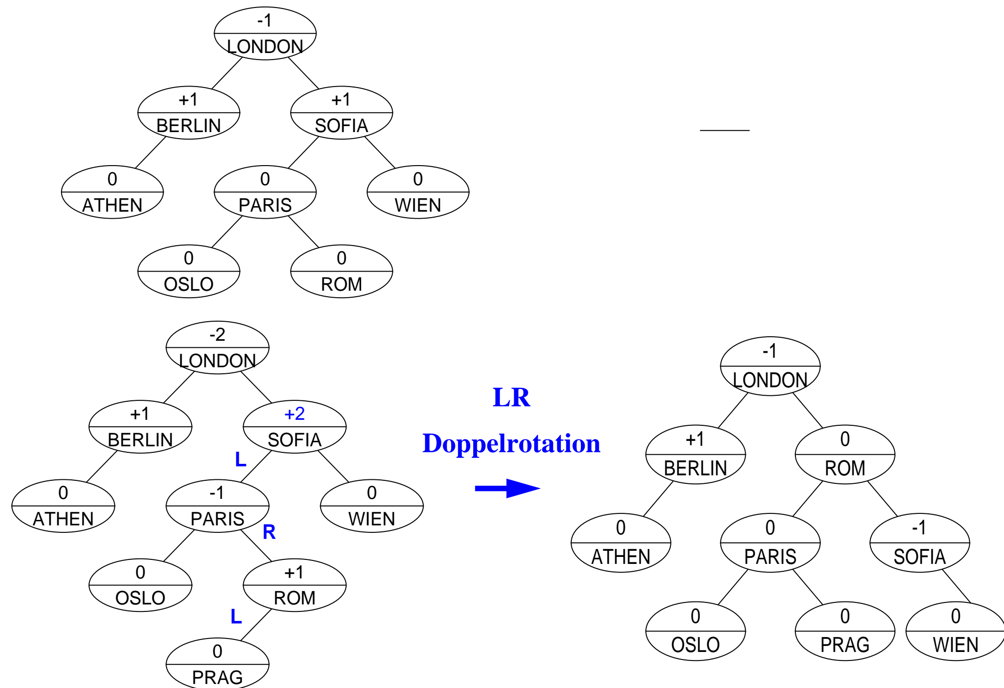
Einfügen in AVL-Bäumen (3)

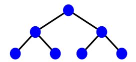
neuer Schlüssel | *nach Einfügung* | *nach Rebalancierung*

OSLO

ROM

PRAG

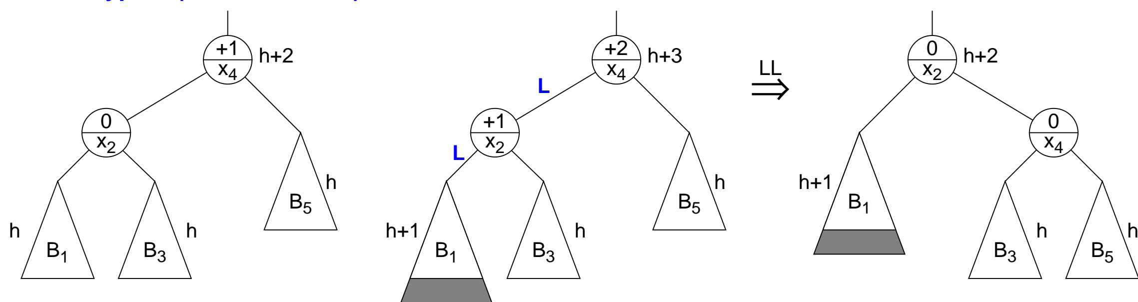




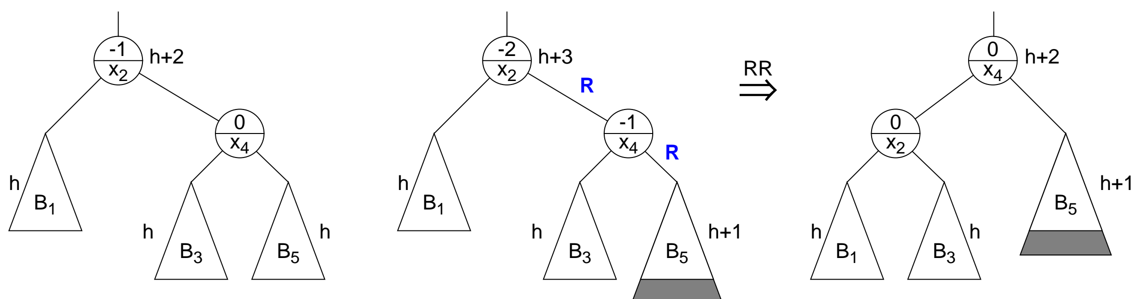
Einfügen in AVL-Bäumen (4)

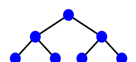
Balancierter Unterbaum | nach Einfügung | Rebalancierter Unterbaum

Rotationstyp LL (Rechtsrotation)

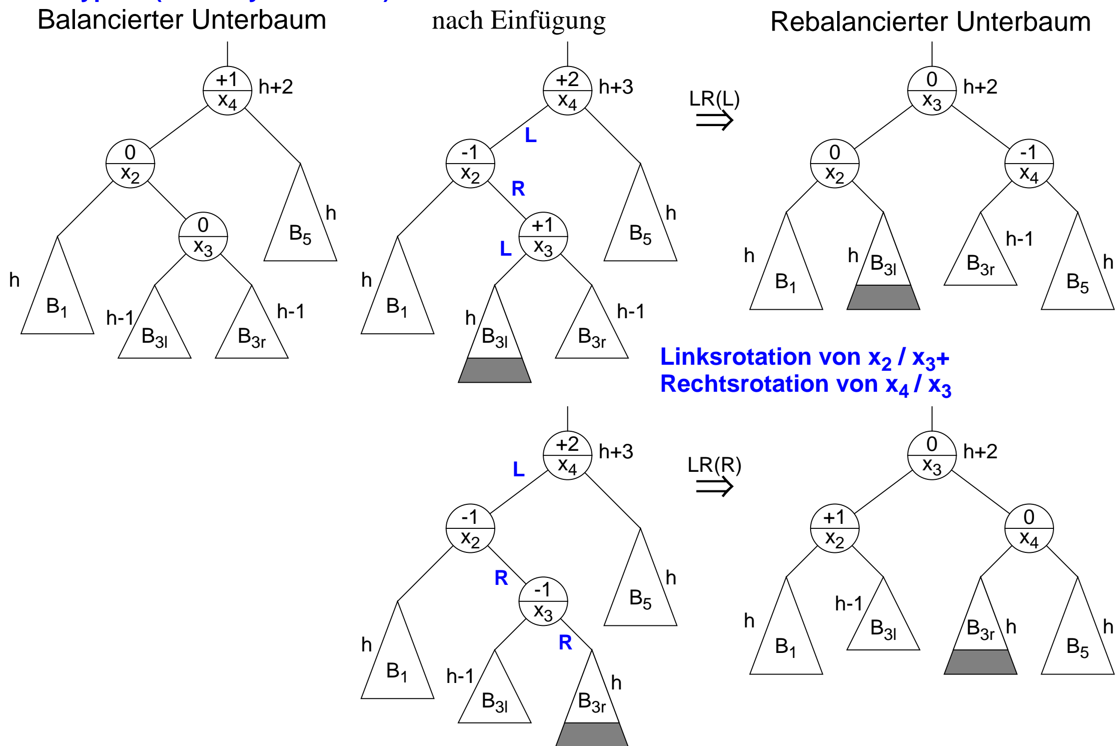


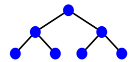
Rotationstyp RR (Linksrotation)





Einfügen in AVL-Bäumen (5)

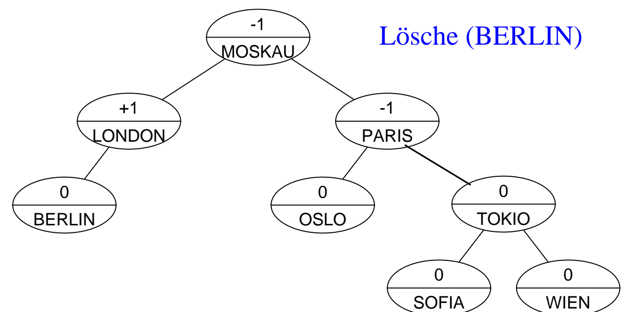




Löschen in AVL-Bäumen

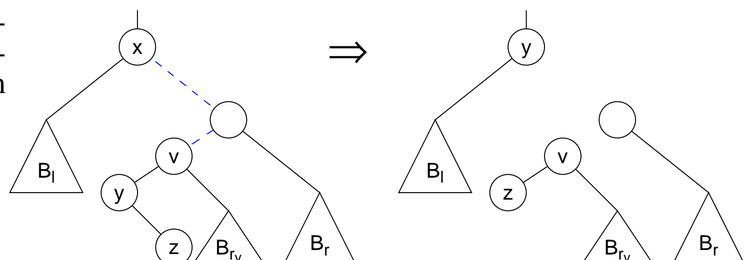
Löschen eines Blattes bzw. eines Randknotens (max. 1 Sohn)

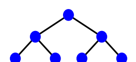
- Höhenreduzierung ändert Balancierungsfaktoren der Vaterknoten
- Rebalancierung für UB der Vorgängerknoten mit BF = +/- 2
- ggf. fortgesetzte Rebalancierung (nur möglich für Knoten mit BF = +/- 2 auf dem Weg vom zu löschenden Element zur Wurzel)



Löschen eines Knotens (Schlüssel x) mit 2 Söhnen kann auf Löschen für Blatt/Randknoten zurückgeführt werden

- x wird ersetzt durch kleinsten Schlüssel y im rechten Unterbaumbaum von x (oder größten Schlüssel im linken Unterbaum)
- führt zur Änderung des Balancierungsfaktors für v (Höhe des linken Unterbaums von v hat sich um 1 reduziert)

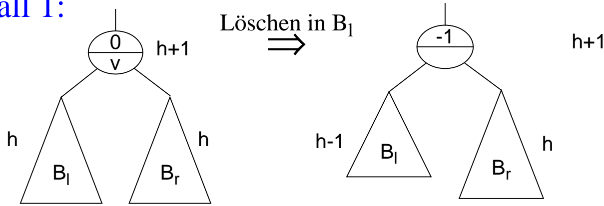




Löschen in AVL-Bäumen (2)

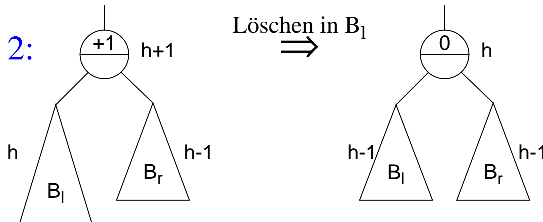
Bis auf Symmetrie treten nur 3 Fälle auf:

Fall 1:



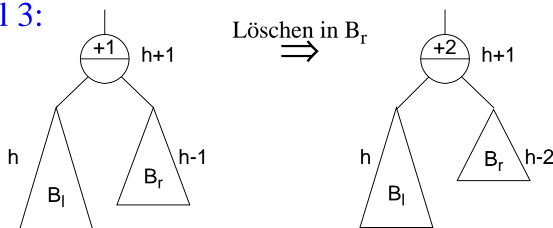
In diesem Fall pflanzt sich Höhenerniedrigung nicht fort, da in der Wurzel das AVL-Kriterium erfüllt bleibt.
-> kein Rebalancieren erforderlich.

Fall 2:

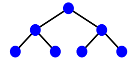


Die Höhenerniedrigung von B_l pflanzt sich hier zur Wurzel hin fort. Sie kann auf diesem Pfad eine Rebalancierung auslösen.

Fall 3:

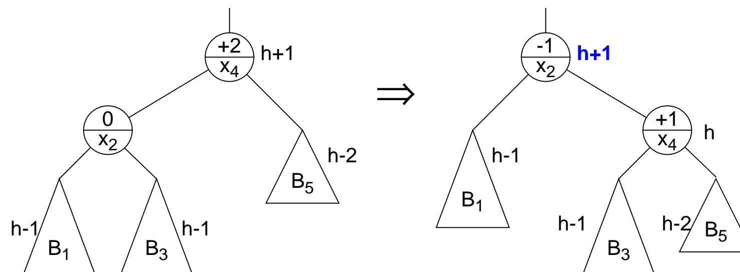


Für die Behandlung dieser Situation ist der linke Unterbaum B_l in größerem Detail zu betrachten. Dabei ergeben sich die 3 Unterfälle:



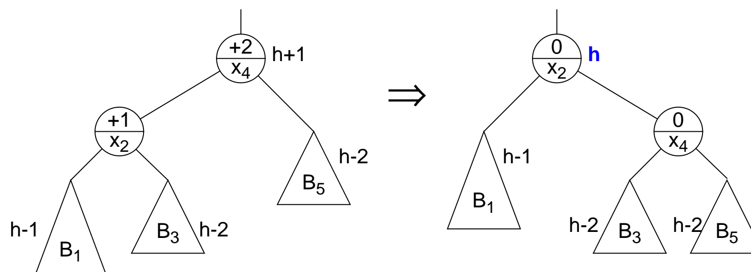
Löschen in AVL-Bäumen (3)

Fall 3a:

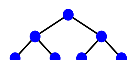


- Rechtsrotation führt zur Erfüllung des AVL-Kriteriums
- Unterbaum behält ursprüngliche Höhe
- keine weiteren Rebalancierungen erforderlich

Fall 3b:

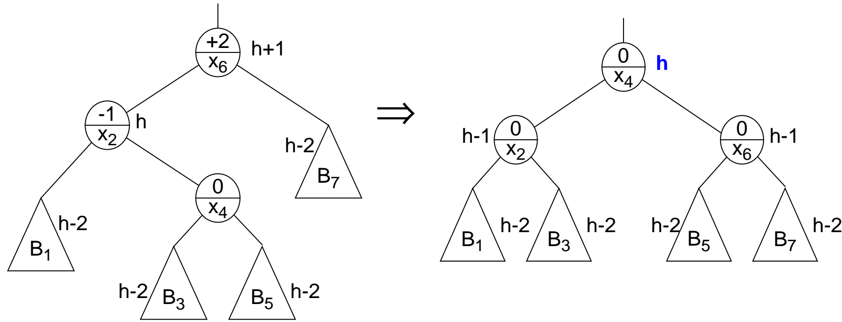


- Rechtsrotation reduziert Höhe des gesamten UB von h+1 nach h
- Höhenreduzierung pflanzt sich auf dem Pfad zur Wurzel hin fort und kann zu weiteren Rebalancierungen führen.

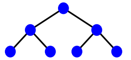


Löschen in AVL-Bäumen (5)

Fall 3c:



- Doppelrotation (-> Höhenerniedrigung)
- ggf. fortgesetzte Rebalancierungen

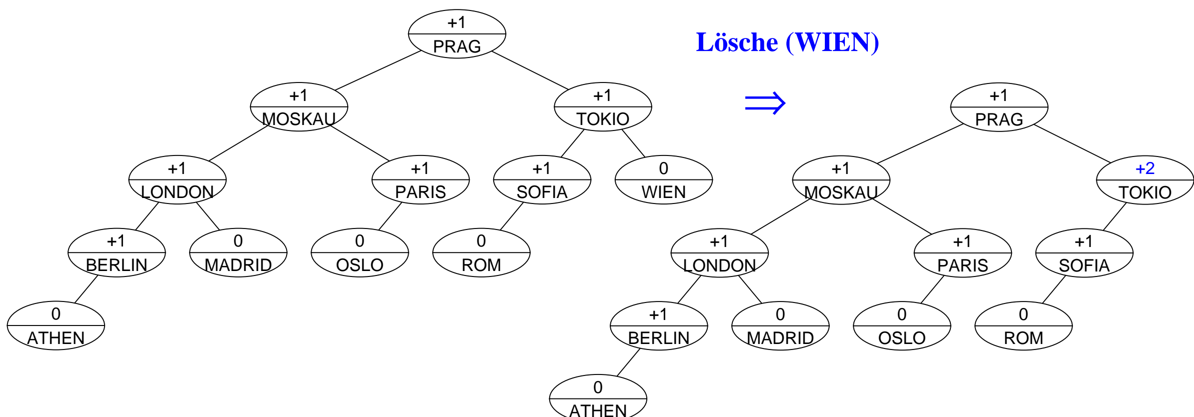


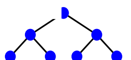
Löschen in AVL-Bäumen (6)

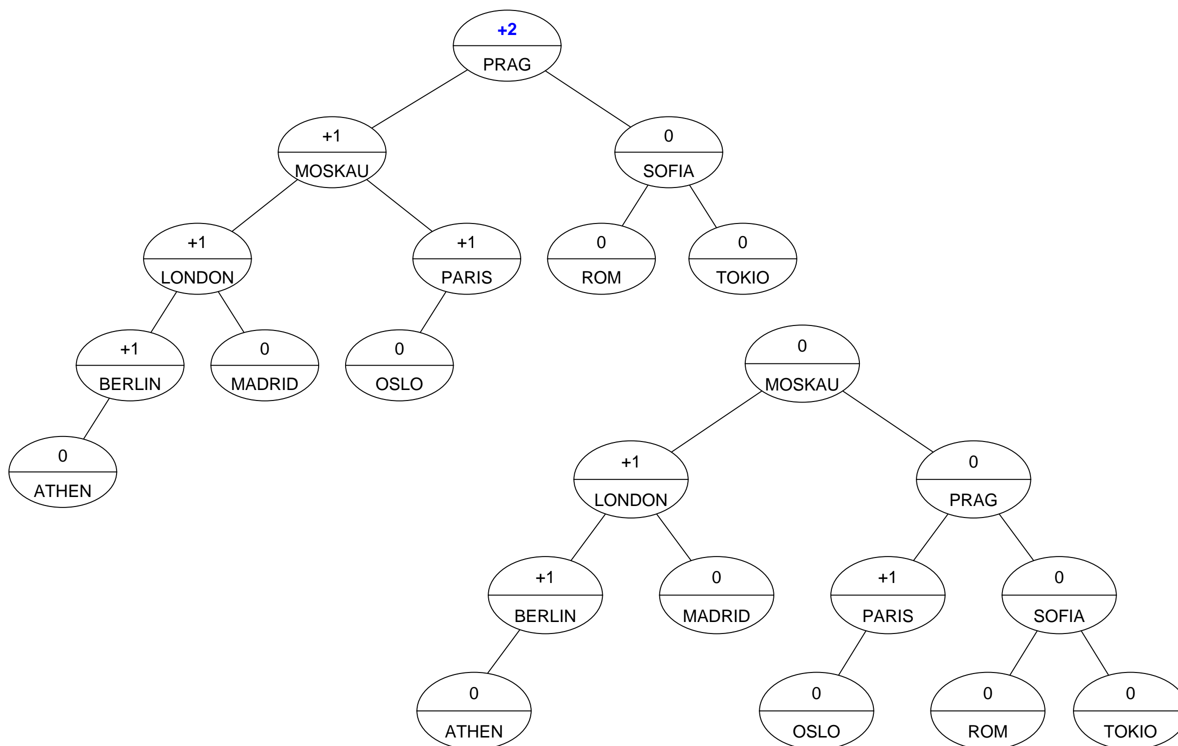
Rebalancierungsschritte beim Löschen:

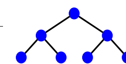
1. Suche im Löschpfad nähesten Vater mit BF = ∓2 .
2. Führe Rotation im gegenüberliegenden Unterbaum dieses Vaters aus.
Im Gegensatz zum Einfügevorgang kann hier eine Rotation wiederum eine Rebalancierung auf dem Pfad zur Wurzel auslösen, da sie in gewissen Fällen auf eine Höhenerniedrigung des transformierten Unterbaums führt. Die Anzahl der Rebalancierungsschritte ist jedoch durch die Höhe h des Baums begrenzt

Beispiel-Löschvorgang:









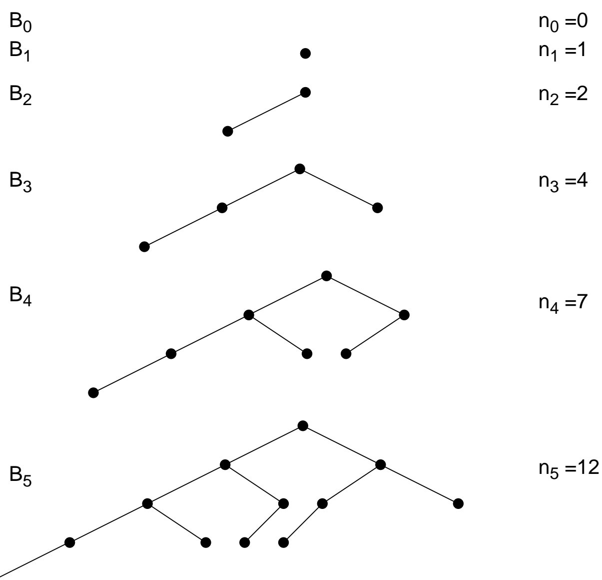
Höhe von AVL-Bäumen

Balancierte Bäume wurden als Kompromiß zwischen ausgeglichenen und natürlichen Suchbäumen eingeführt, wobei logarithmischer Suchaufwand im schlechtesten Fall gefordert wurde

Für die Höhe h_b eines AVL-Baumes mit n Knoten gilt:

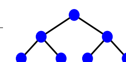
$$\lfloor \log_2(n) \rfloor + 1 \le h_b \le 1{,}44 \cdot \log_2(n+1)$$

- Die obere Schranke läßt sich durch sog. Fibonacci-Bäume, eine Unterklasse der AVL-Bäume, herleiten.



Definition für **Fibonacci-Bäume** (Konstruktionsvorschrift)

- Der leere Baum ist ein Fibonacci-Baum der Höhe 0.
- Ein einzelner Knoten ist ein Fibonacci-Baum der Höhe 1.
- Sind B_{h-1} und B_{h-2} Fibonacci-Bäume der Höhe h-1 und h-2, so ist $B_h = \langle B_{h-1}, x, B_{h-2} \rangle$ ein Fibonacci-Baum der Höhe h
- Keine anderen Bäume sind Fibonacci-Bäume



Gewichtsbalancierte Suchbäume

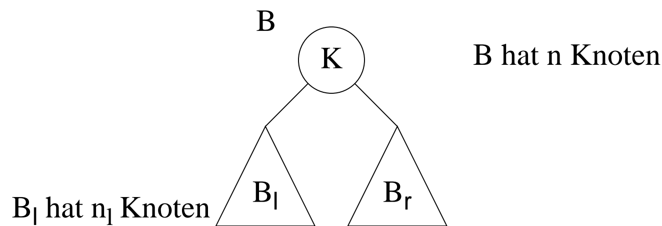
Gewichtsbalancierte oder BB-Bäume (bounded balance)

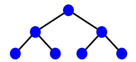
Zulässige Abweichung der Struktur vom ausgeglichenen Binärbaum wird als Differenz zwischen der Anzahl der Knoten im rechten und linken Unterbaum festgelegt

Def.: Sei B ein binärer Suchbaum mit linkem Unterbaum B_l und sei n (n_l) die Anzahl der Knoten in B (B_l).

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

$$\alpha <= \rho\,(B') <= 1 - \alpha$$





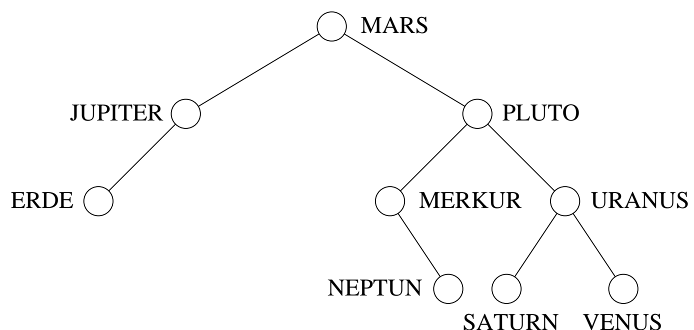
Gewichtsbalancierte Suchbäume (2)

Parameter α als Freiheitsgrad im Baum

- α = 1/2: Balancierungskriterium akzeptiert nur vollständige Binärbäume
- α < 1/2: Strukturbeschränkung wird zunehmend gelockert

Welche Auswirkungen hat die Lockerung des Balancierungskriteriums auf die Kosten ?

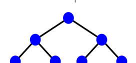
Beispiel: Gewichtsbalancierter Baum in BB (α) für α = 3 / 10



Rebalancierung

- ist gewährleistet durch eine Wahl von $\alpha <= 1 - \sqrt{2}\ /2$
- Einsatz derselben Rotationstypen wie beim AVL-Baum

Kosten für Suche und Aktualisierung: $O\,(\log_2 n)$



Positionssuche mit balancierten Bäumen

Balancierte Suchbäume

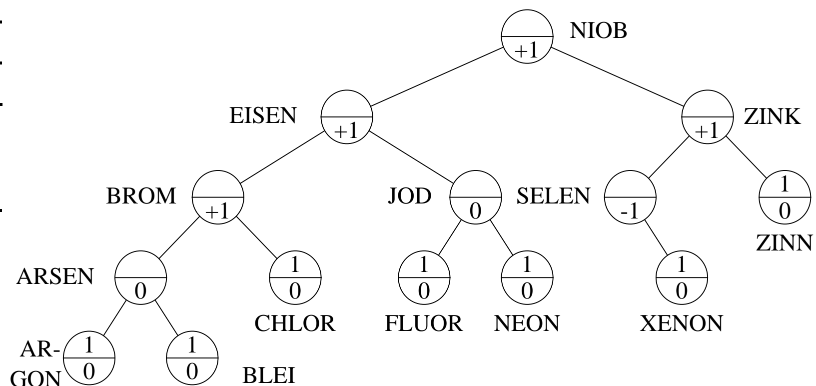
- sind linearen Listen in fast allen Grundoperationen überlegen
- Lösung des Auswahlproblems bzw. Positionssuche (Suche nach k-tem Element der Sortierreihenfolge) kann jedoch noch verbessert werden

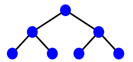
Def.: Der Rang eines Knotens ist die um 1 erhöhte Anzahl der Knoten seines linken Unterbaums

- Blattknoten haben Rang 1

Verbesserung bei Positionssuche durch Aufnahme des Rangs in jedem Knoten

Beispiel für AVL-Baum:





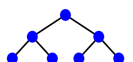
Positionssuche (2)

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 UB des Knotens weiter
- r < p => p := p - r und Fortsetzung der Suche im rechten UB

Wartungsoperationen etwas komplexer

Änderung im linken Unterbaum erfordert Ranganpassung aller betroffenen Väter bis zur Wurzel



Zusammenfassung

Binäre Suchbäume

- Einfügen / direkte Suche durch Baumdurchgang auf einem Pfad
- Reihenfolgeabhängigkeit bezüglich Einfügeoperationen
- sequentielle Suche / sortierte Ausgabe aller Elemente: Inorder-Baumtraversierung (O (n))
- minimale Zugriffskosten der direkten Suche O (log n); mittlere Kosten Faktor 1,39 schlechter

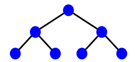
Balancierte Suchbäume zur Sicherstelleung günstiger Zugriffskosten

- höhenbalancierte Bäume (z.B. k-balancierte Binärbäume)
- gewichtsbalancierte Bäume (BB-Bäume)

AVL-Baum: 1-balancierter Binärbaum

- Mitführen eines Balancierungsfaktors B in Baumknoten (zulässige Werte: -1, 0, oder 1)
- dynamische Rebalancierung bei Einfüge- und Löschoperationen (Fallunterscheidungen mit unterschiedlichen Rotationen)
- Anzahl der Rebalancierungsschritte durch Höhe des Baumes begrenzt
- maximale Höhe für Fibonacci-Bäume: 1,44 log (n)

schnelle Positionssuche über Mitführen des Rangs von Knoten



Zusammenfassung: Listenoperationen auf verschiedenen Datenstrukturen

Operation	sequent. Liste	gekettete Liste	balancierter Baum mit Rang
Suche von K_i			
Suche nach k-tem Element			
Einfügen von K_i			
Löschen von K_i			
Löschen von k-tem Element			
sequentielle Suche