Script generated by TTT

Title: Täubig: GAD (26.06.2012)

Date: Tue Jun 26 14:34:47 CEST 2012

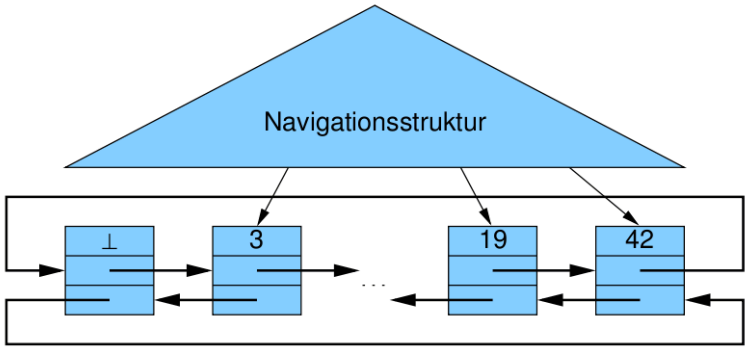
Duration: 85:34 min

Pages: 47

Suchstruktur

Idee:

- füge Navigationsstruktur hinzu, die locate effizient macht



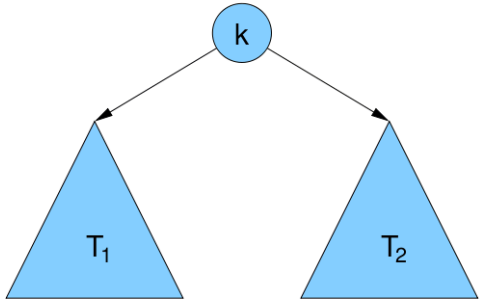
Übersicht

Binärer Suchbaum

Suchbaum-Regel:

Für alle Schlüssel
k_1 in T_1 und k_2 in T_2:

$$k_1 \leq k < k_2$$



locate-Strategie:

- Starte in Wurzel des Suchbaums
- Für jeden erreichten Knoten v:

  Falls key$(v) \geq k_{\text{gesucht}}$, gehe zum linken Kind von v,
  sonst gehe zum rechten Kind

Binärer Suchbaum

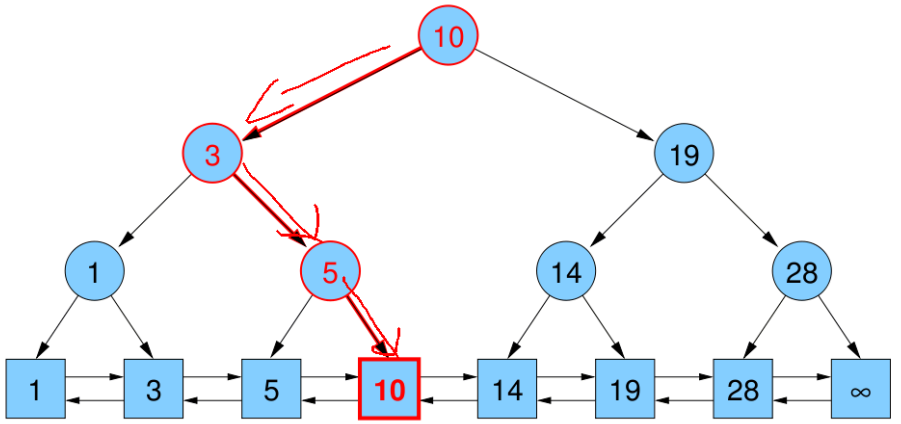
Formal: für einen Baumknoten v sei

- key(v) der Schlüssel von v
- $d(v)$ der Ausgangsgrad (Anzahl Kinder) von v

- Suchbaum-Invariante: $k_1 \leq k < k_2$
  (Sortierung der linken und rechten Nachfahren)
- Grad-Invariante: $d(v) \leq 2$
  (alle Baumknoten haben höchstens 2 Kinder)
- Schlüssel-Invariante:
  (Für jedes Element e in der Liste gibt es *genau einen* Baumknoten v mit key(v) == key(e))

Binärer Suchbaum / locate

locate(9)



Binärer Suchbaum / insert, remove

Strategie:

- insert(e):
  - erst wie locate(key(e)) bis Element e' in Liste erreicht
  - falls key(e') > key(e):
    füge e vor e' ein, sowie ein neues Suchbaumblatt für e und e' mit key(e) als Splitter Key, so dass Suchbaum-Regel erfüllt

- remove(k):
  - erst wie locate(k) bis Element e in Liste erreicht
  - falls key(e) = k, lösche e aus Liste und Vater v von e aus Suchbaum und
  - setze in dem Baumknoten w mit key(w) = k den neuen Wert key(w) = key(v)

Binärer Suchbaum / insert, remove

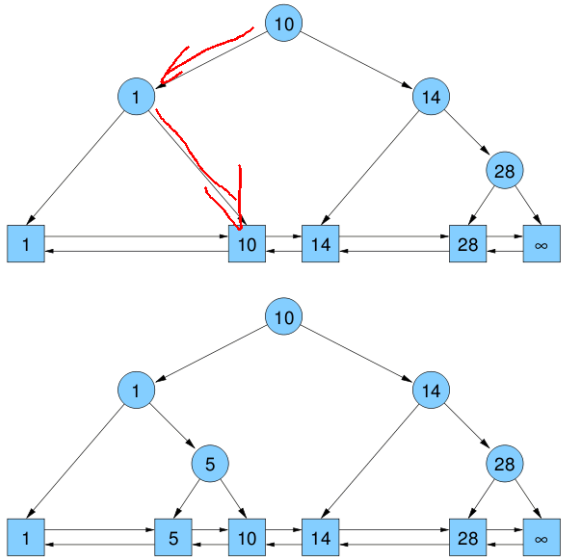
Strategie:

- insert(e):
  - erst wie locate(key(e)) bis Element e' in Liste erreicht
  - falls key(e') > key(e):
    füge e vor e' ein, sowie ein neues Suchbaumblatt für e und e' mit key(e) als Splitter Key, so dass Suchbaum-Regel erfüllt

- remove(k):
  - erst wie locate(k) bis Element e in Liste erreicht
  - falls key(e) = k, lösche e aus Liste und Vater v von e aus Suchbaum und
  - setze in dem Baumknoten w mit key(w) = k den neuen Wert key(w) = key(v)

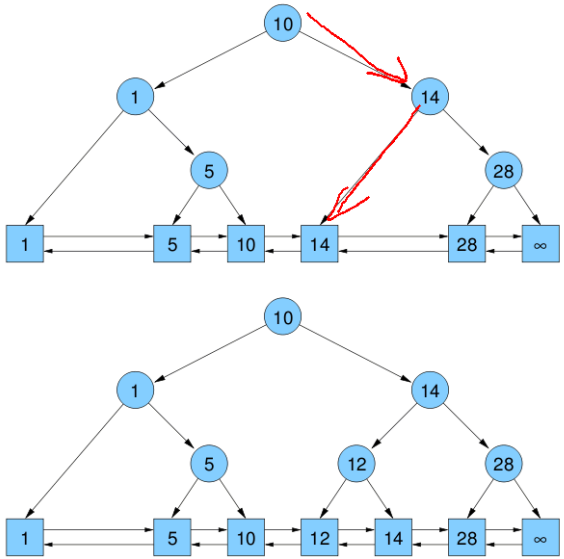
Binärer Suchbaum / insert, remove

insert(5)



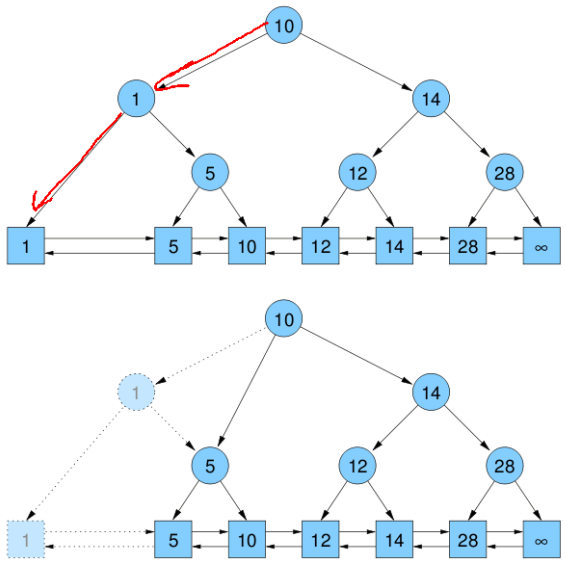
Binärer Suchbaum / insert, remove

insert(12)



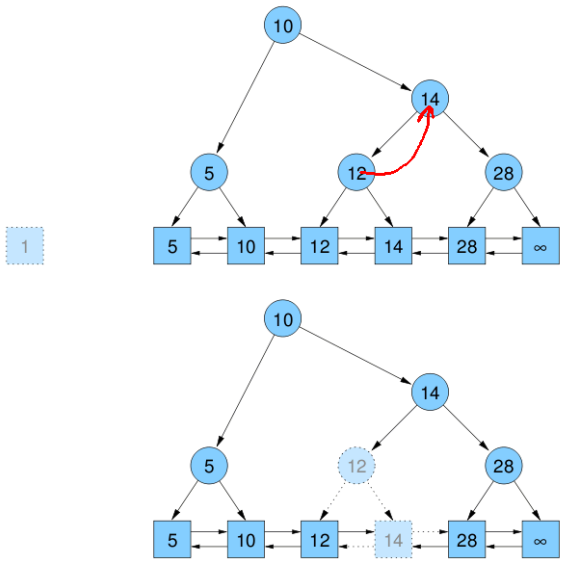
Binärer Suchbaum / insert, remove

remove(1)



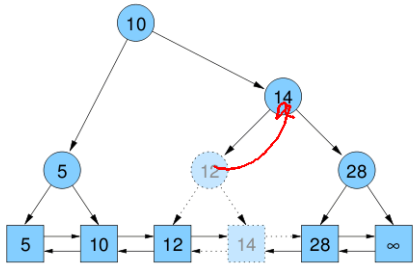
Binärer Suchbaum / insert, remove

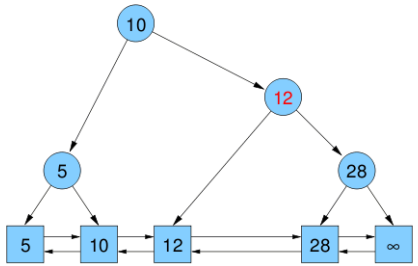
remove(14)



Binärer Suchbaum / insert, remove

remove(14)





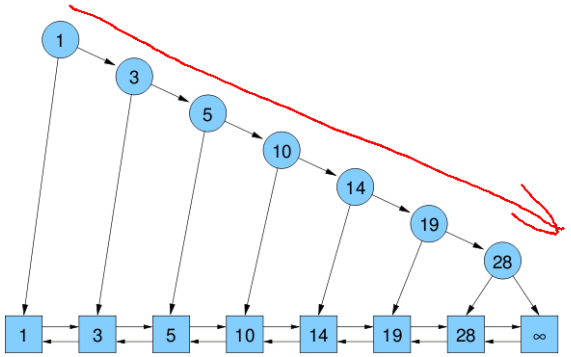
Binärer Suchbaum / worst case

Problem:

- Baumstruktur kann zur Liste entarten
- Höhe des Baums kann linear in der Anzahl der Elemente werden

$\Rightarrow$ locate kann im worst case Zeitaufwand $\Theta(n)$ verursachen

Beispiel: Zahlen werden in sortierter Reihenfolge eingefügt



Übersicht

AVL-Bäume

Balancierte binäre Suchbäume

Strategie zur Lösung des Problems:

- Balancierung des Baums

G. M. **A**delson-**V**elsky & E. M. **L**andis (1962):

- Beschränkung der Höhenunterschiede für Teilbäume auf $[-1, 0, +1]$

$\Rightarrow$ führt nicht unbedingt zu einem idealen unvollständigen Binärbaum (wie wir ihn von array-basierten Heaps kennen), aber zu einem hinreichenden Gleichgewicht

AVL-Bäume

Balancierte binäre Suchbäume

Worst Case: Fibonacci-Baum

- Laufzeit der Operation hängt von der Baumhöhe ab
- Was ist die schlimmste Höhe bei gegebener Anzahl von Elementen?
- bzw: Wieviel Elemente hat ein Baum bei gegebener Höhe h mindestens?

$\Rightarrow$ Für mindestens ein Kind hat der Unterbaum Höhe $h-1$.
Im worst case hat das andere Kind Höhe $h-2$ (kleiner geht nicht wegen Höhendifferenzbeschränkung)

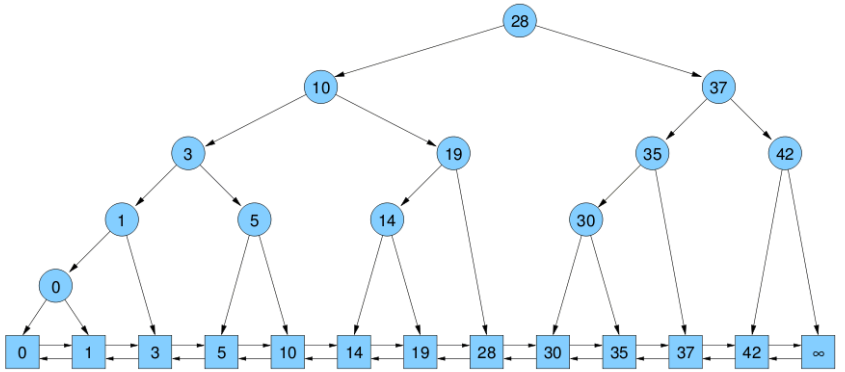
$\Rightarrow$ Anzahl der (inneren) Knoten entspricht den Fibonacci-Zahlen:

$$F_n = F_{n-1} + F_{n-2}$$

AVL-Bäume

Balancierte binäre Suchbäume

Worst Case: Fibonacci-Baum



AVL-Bäume

Balancierte binäre Suchbäume

Worst Case: Fibonacci-Baum

- Fibonacci-Baum der Stufe 0 ist der leere Baum
- Fibonacci-Baum der Stufe 1 ist ein einzelner Knoten
- Fibonacci-Baum der Stufe $h+1$ besteht aus einer Wurzel, deren Kinder Fibonacci-Bäume der Stufen h und $h-1$ sind

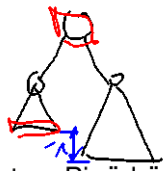
Explizite Formel:

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

$\Rightarrow$ Die Anzahl der Elemente ist exponentiell in der Höhe bzw. die Höhe ist logarithmisch in der Anzahl der Elemente.

AVL-Bäume

Balancierte binäre Suchbäume



Operationen auf einem AVL-Baum:

- insert und remove können zunächst zu Binärbäumen führen, die die Balance-Bedingung für die Höhendifferenz der Teilbäume verletzen

$\Rightarrow$ Teilbäume müssen umgeordnet werden, um das Kriterium für AVL-Bäume wieder zu erfüllen (Rebalancierung / Rotation)

- Dazu wird an jedem Knoten die Höhendifferenz der beiden Unterbäume vermerkt ($-1, 0, +1$, mit 2 Bit / Knoten)

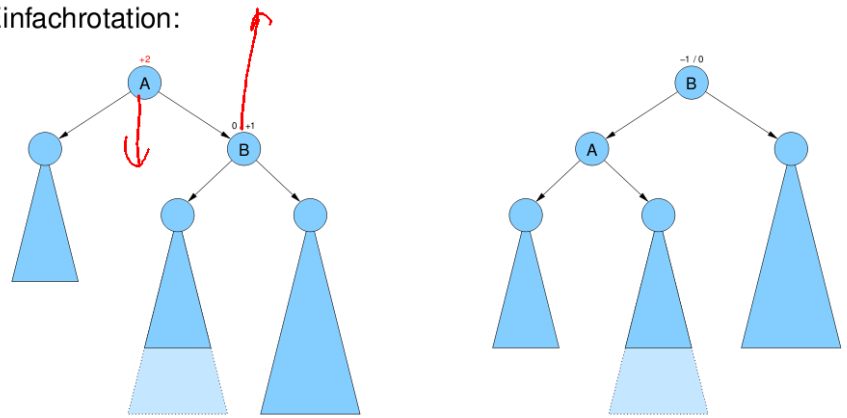
(Hausaufgabe . . .)

- Operationen locate, insert und remove haben Laufzeit $O(\log n)$

AVL-Bäume

Balancierte binäre Suchbäume

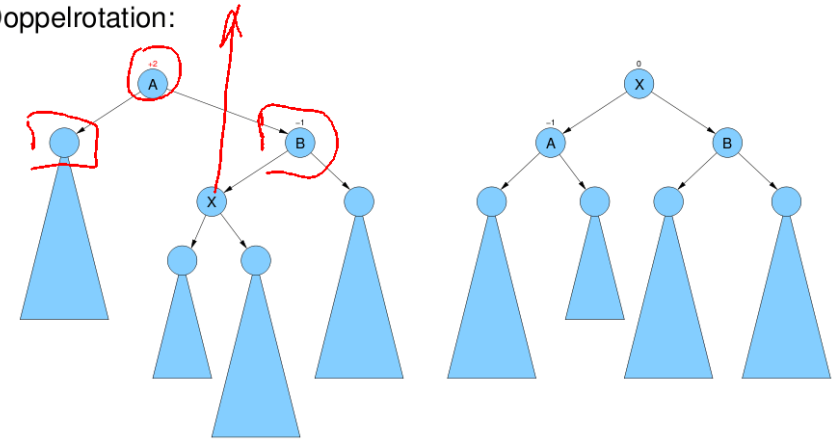
Einfachrotation:



AVL-Bäume

Balancierte binäre Suchbäume

Doppelrotation:



Übersicht

(a,b)-Baum

Problem: Baumstruktur kann zur Liste entarten

Lösung: (a,b)-Baum

Idee:

- $d(v)$: Ausgangsgrad (Anzahl Kinder) von Knoten v
- $t(v)$: Tiefe (in Kanten) von Knoten v
- Form-Invariante:
  alle Blätter in derselben Tiefe: $t(v) = t(w)$ für Blätter v, w
- Grad-Invariante:
  Für alle internen Knoten v (außer Wurzel) gilt:

$$a \leq d(v) \leq b \qquad \text{(wobei } a \geq 2 \text{ und } b \geq 2a-1\text{)}$$

Für Wurzel r: $2 \leq d(r) \leq b$ (außer wenn nur 1 Blatt im Baum)

(a,b)-Baum

Lemma

Ein (a,b)-Baum für n Elemente hat Tiefe $\leq 1 + \left\lfloor \log_a \frac{n+1}{2} \right\rfloor$.

Beweis.

- Baum hat $n+1$ Blätter ($+1$ wegen ∞-Dummy)
- Für $n = 0$ gilt die Ungleichung (1 Dummy-Blatt, Höhe 1)
- sonst hat Wurzel Grad ≥ 2,
  die anderen inneren Knoten haben Grad $\geq a$

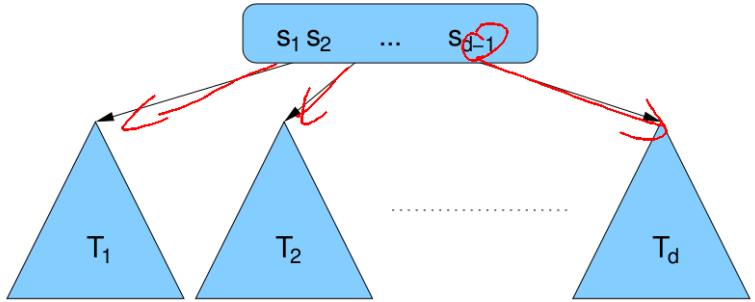
$\Rightarrow$ Bei Tiefe t gibt es $\geq 2a^{t-1}$ Blätter

- $n+1 \geq 2a^{t-1} \quad \Leftrightarrow \quad t \leq 1 + \left\lfloor \log_a \frac{n+1}{2} \right\rfloor$

□

(a,b)-Baum: Split-Schlüssel

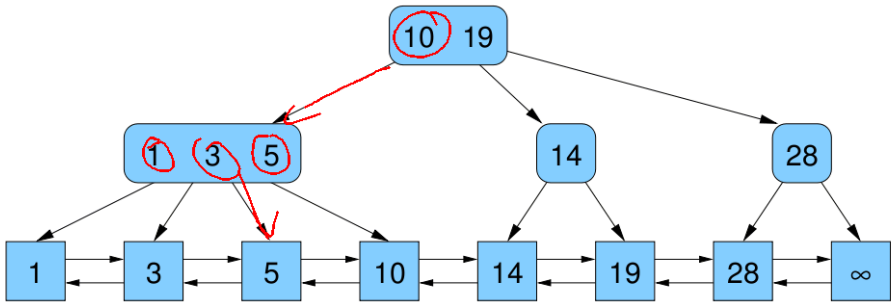
- Jeder Knoten v enthält ein sortiertes Array von $d(v) - 1$ Split-Schlüsseln $s_1, \ldots, s_{d(v)-1}$



- (a,b)-Suchbaum-Regel:
  Für alle Schlüssel k in T_i und k' in T_{i+1} gilt:
  $k \leq s_i < k'$ bzw. $s_{i-1} < k \leq s_i$ $\quad (s_0 = -\infty, s_d = \infty)$

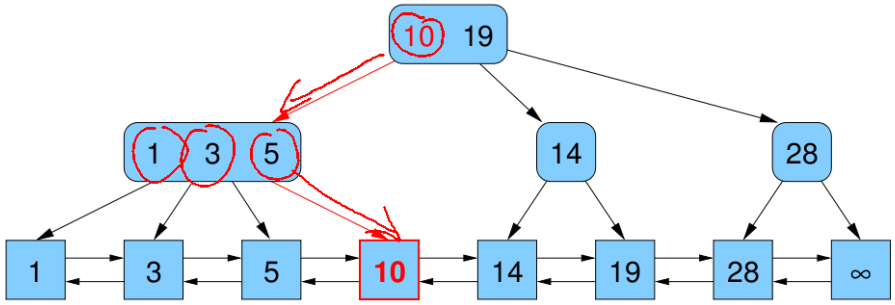
(a,b)-Baum

Beispiel:



(a,b)-Baum

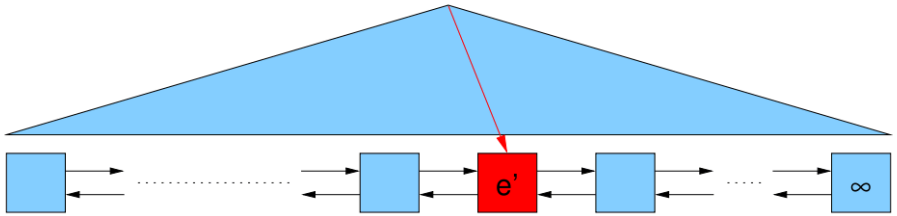
locate(9)



(a,b)-Baum

insert(e)

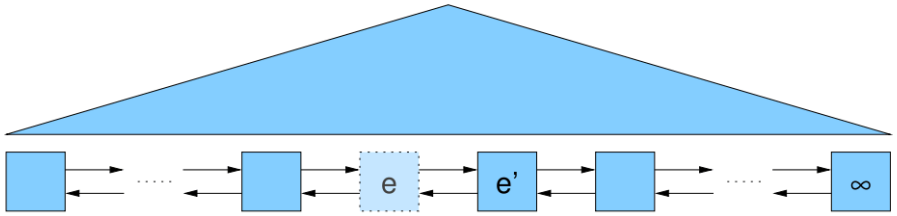
- Abstieg wie bei locate(key(e)) bis Element e' in Liste erreicht
- falls key(e')>key(e), füge e vor e' ein



(a,b)-Baum

insert(e)

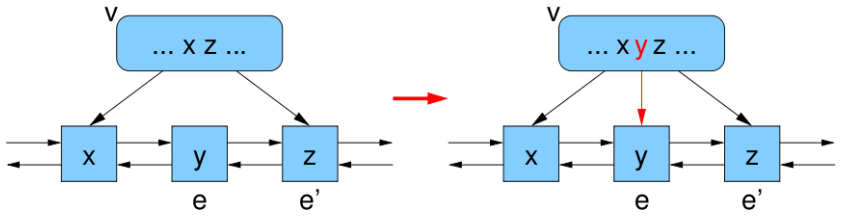
- Abstieg wie bei locate(key(e)) bis Element e' in Liste erreicht
- falls key(e')>key(e), füge e vor e' ein



(a,b)-Baum

insert(e)

- füge key(e) und Handle auf e in Baumknoten v über e ein
- falls $d(v) \leq b$, dann fertig

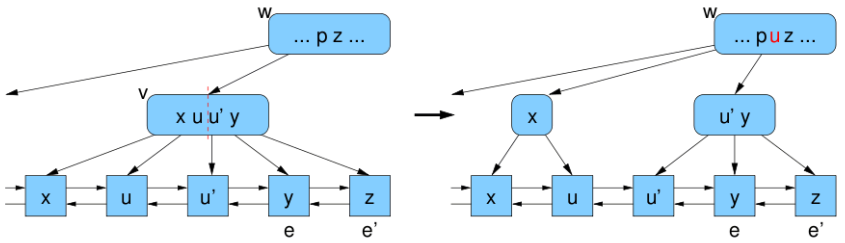


(a,b)-Baum

insert(e)

- füge key(e) und Handle auf e in Baumknoten v über e ein
- falls $d(v) > b$, dann teile v in zwei Knoten auf und
- verschiebe den Splitter (größter Key im linken Teil) in den Vaterknoten

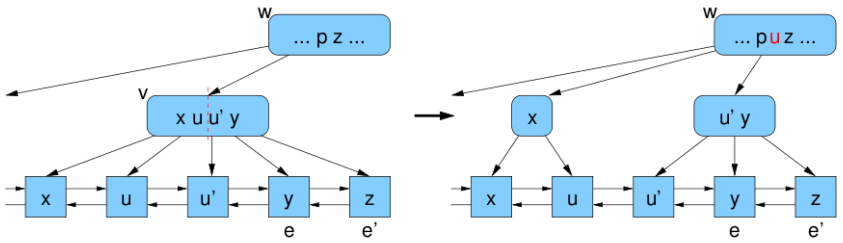
Beispiel: (2,4)-Baum



(a, b)-Baum

insert(e)

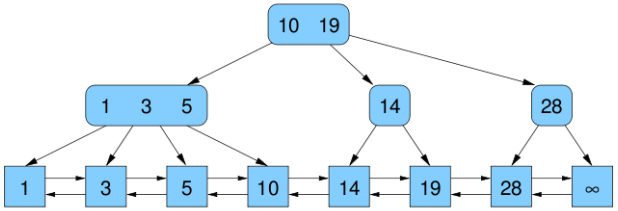
- falls $d(w) > b$, dann teile w in zwei Knoten auf usw.
  bis Grad $\leq b$
  oder Wurzel aufgeteilt wurde



(a, b)-Baum / insert

$a = 2$, $b = 4$

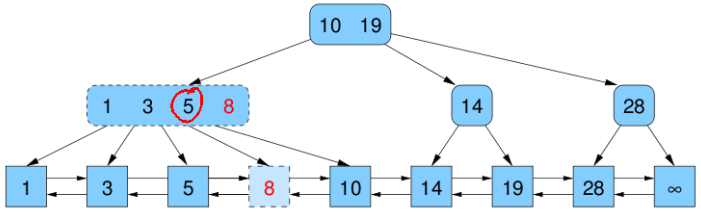
insert(8)



(a, b)-Baum / insert

$a = 2$, $b = 4$

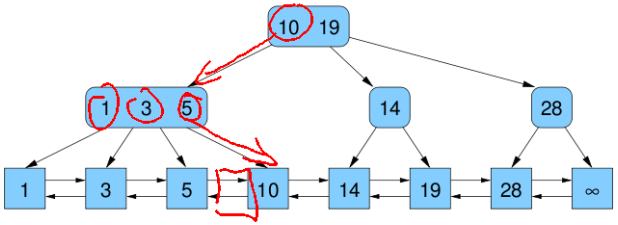
insert(8)



(a, b)-Baum / insert

$a = 2$, $b = 4$

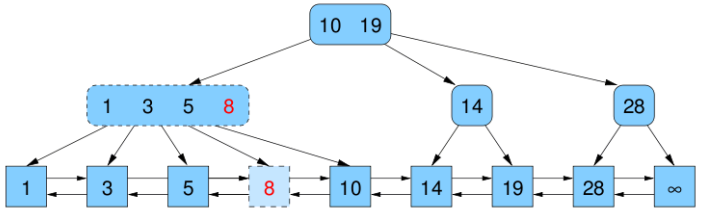
insert(8)



(a, b)-Baum / insert

$a = 2$, $b = 4$

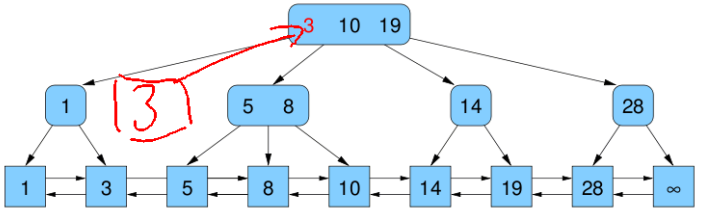
insert(8)



(a, b)-Baum / insert

$a = 2$, $b = 4$

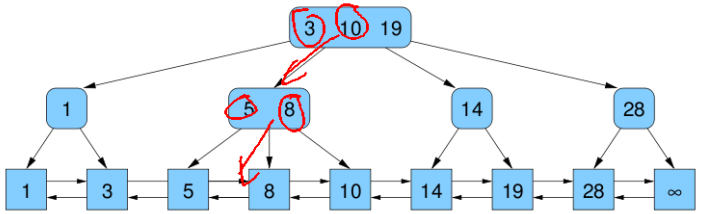
insert(8)



(a, b)-Baum / insert

$a = 2$, $b = 4$

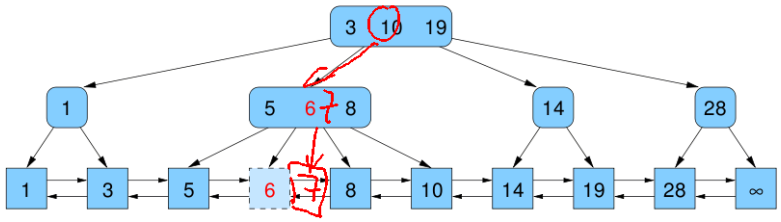
insert(6)



(a, b)-Baum / insert

$a = 2$, $b = 4$

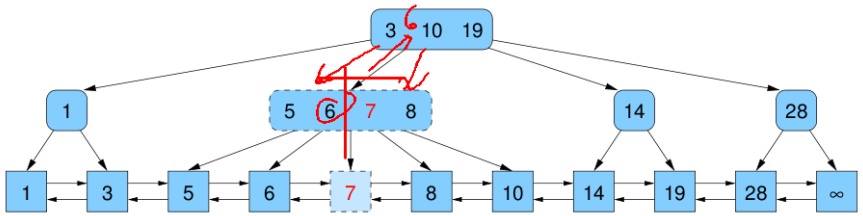
insert(6)



(a, b)-Baum / insert

$a = 2, b = 4$

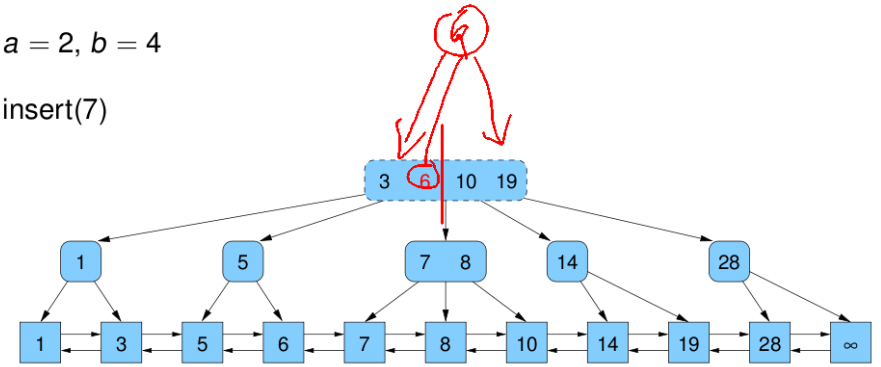
insert(7)



(a, b)-Baum / insert

$a = 2, b = 4$

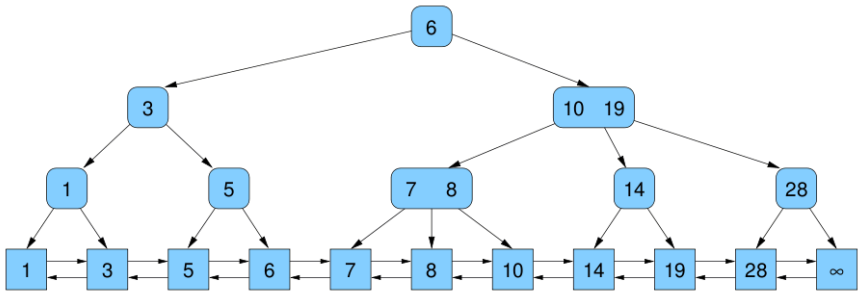
insert(7)



(a, b)-Baum / insert

$a = 2, b = 4$

insert(7)



(a, b)-Baum / insert

Form-Invariante

- alle Blätter haben dieselbe Tiefe, denn
  neues Blatt wird auf der Ebene der anderen eingefügt und
  im Fall einer neuen Wurzel erhöht sich die Tiefe aller Blätter um 1

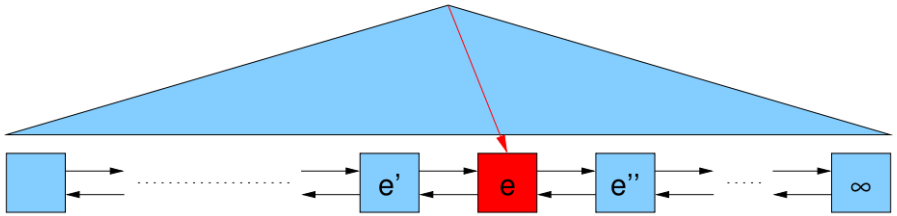
Grad-Invariante

- insert splittet Knoten mit Grad $b + 1$ in zwei Knoten
  mit Grad $\lfloor (b + 1)/2 \rfloor$ und $\lceil (b + 1)/2 \rceil$ $b+1 \geq 2a$
- wenn $b \geq 2a - 1$, dann sind beide Werte $\geq a$
- wenn Wurzel Grad $b + 1$ erreicht und gespalten wird,
  wird neue Wurzel mit Grad 2 erzeugt

(a,b)-Baum

remove(k)

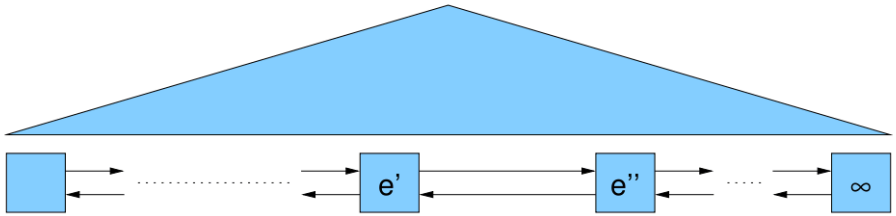
- Abstieg wie bei locate(k) bis Element e in Liste erreicht
- falls key(e) $= k$, entferne e aus Liste (sonst return)



(a,b)-Baum

remove(k)

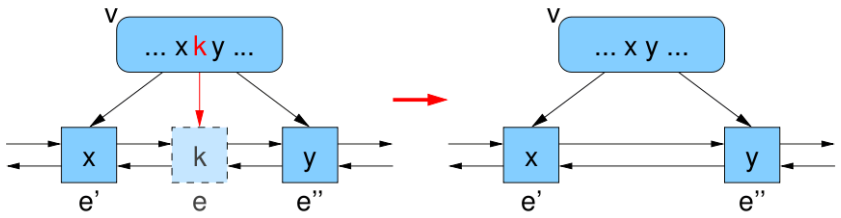
- Abstieg wie bei locate(k) bis Element e in Liste erreicht
- falls key(e) $= k$, entferne e aus Liste (sonst return)



(a,b)-Baum

remove(k)

- entferne Handle auf e und Schlüssel k vom Baumknoten v über e (wenn e rechtestes Kind: Schlüsselvertauschung wie bei binärem Suchbaum)
- falls $d(v) \geq a$, dann fertig



(a,b)-Baum

remove(k)

- falls $d(v) < a$ und ein direkter Nachbar v' von v hat Grad $> a$, nimm Kante von v'

Beispiel: (2,4)-Baum