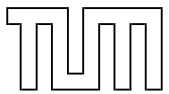
Grundlagen: Algorithmen und Datenstrukturen

Prof. Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2010



Übersicht

Vergleich Wörterbuch / Suchstruktur

- S: Menge von Elementen
- Element e wird identifiziert über eindeutigen Schlüssel key(e)

Operationen:

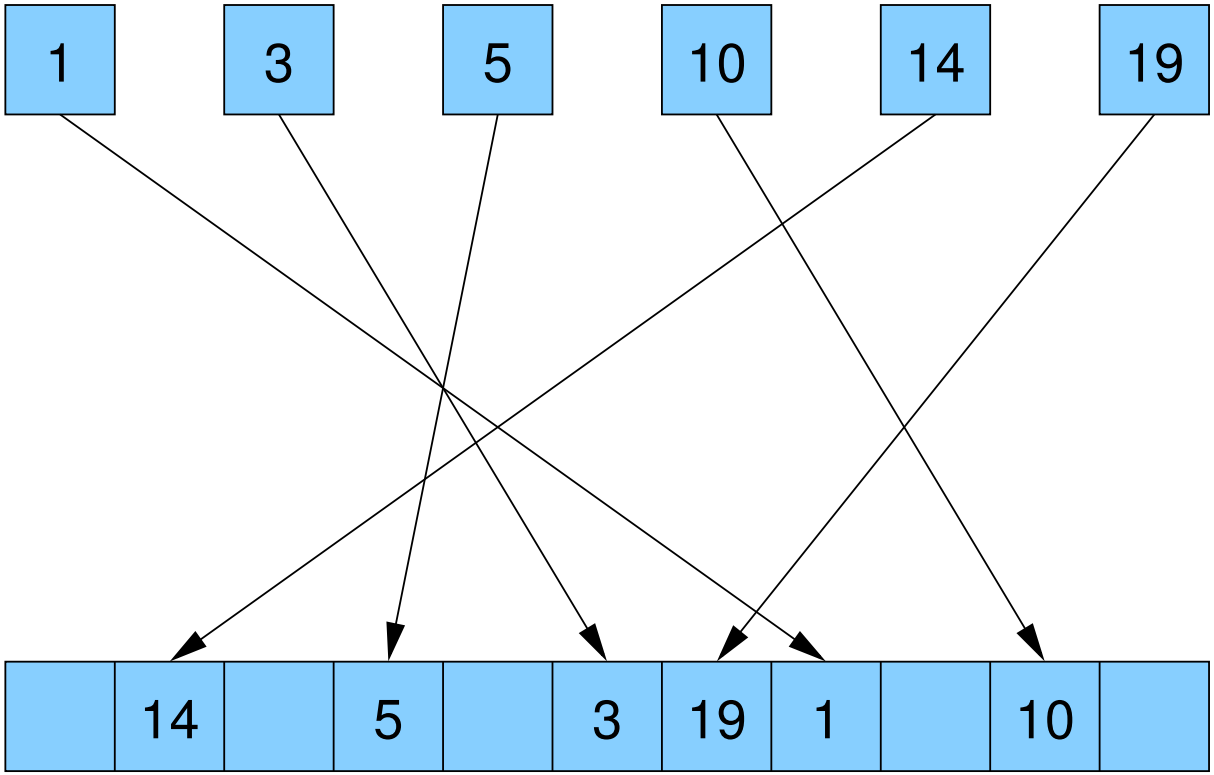
- S.insert(Elem e): $S = S \cup \{e\}$
- S.remove(Key k): $S = S \setminus \{e\}$,
  wobei e das Element mit key(e) $==$ k ist
- S.find(Key k):

  Wörterbuch: gibt das Element $e \in S$ mit key(e) $==$ k zurück, falls es existiert, sonst null

  Suchstruktur: gibt das Element $e \in S$ mit minimalem Schlüssel key(e) zurück, für das key(e) $\geq$ k

Vergleich Wörterbuch / Suchstruktur

- Wörterbuch effizient über Hashing realisierbar



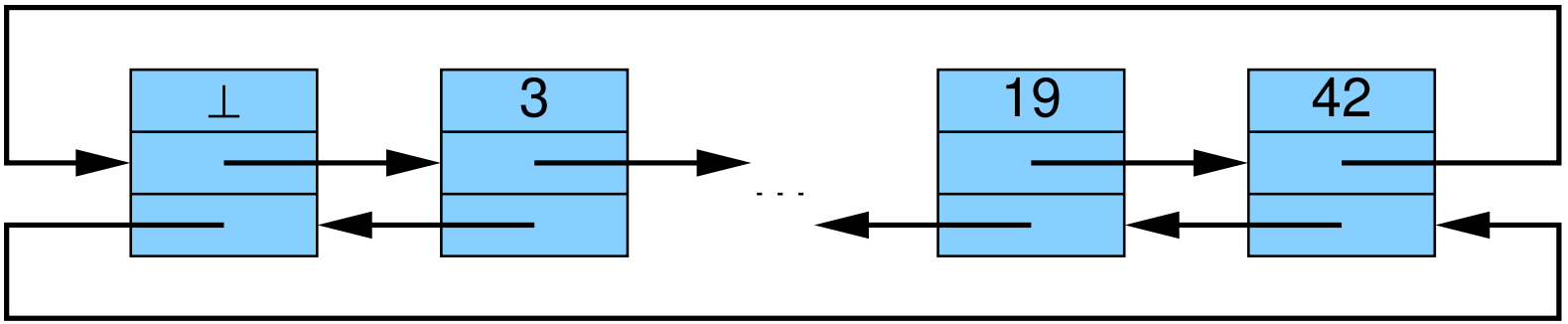
- Hashing zerstört die Ordnung auf den Elementen

$\Rightarrow$ keine effiziente locate-Operation

$\Rightarrow$ keine Intervallanfragen

Suchstruktur

Erster Ansatz: sortierte Liste



Problem:

- insert, remove, locate kosten im worst case $\Theta(n)$ Zeit

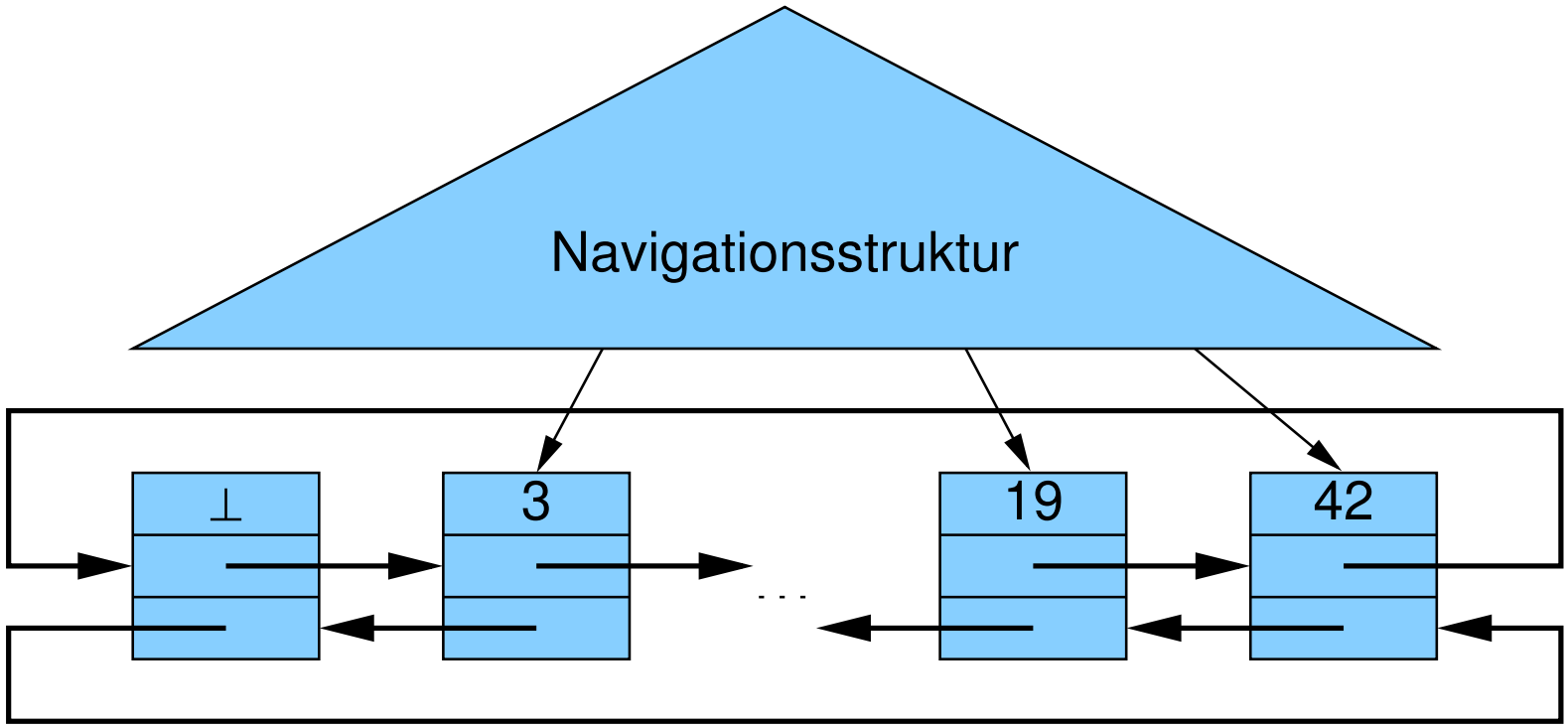
Einsicht:

- wenn locate effizient implementierbar, dann auch die anderen Operationen

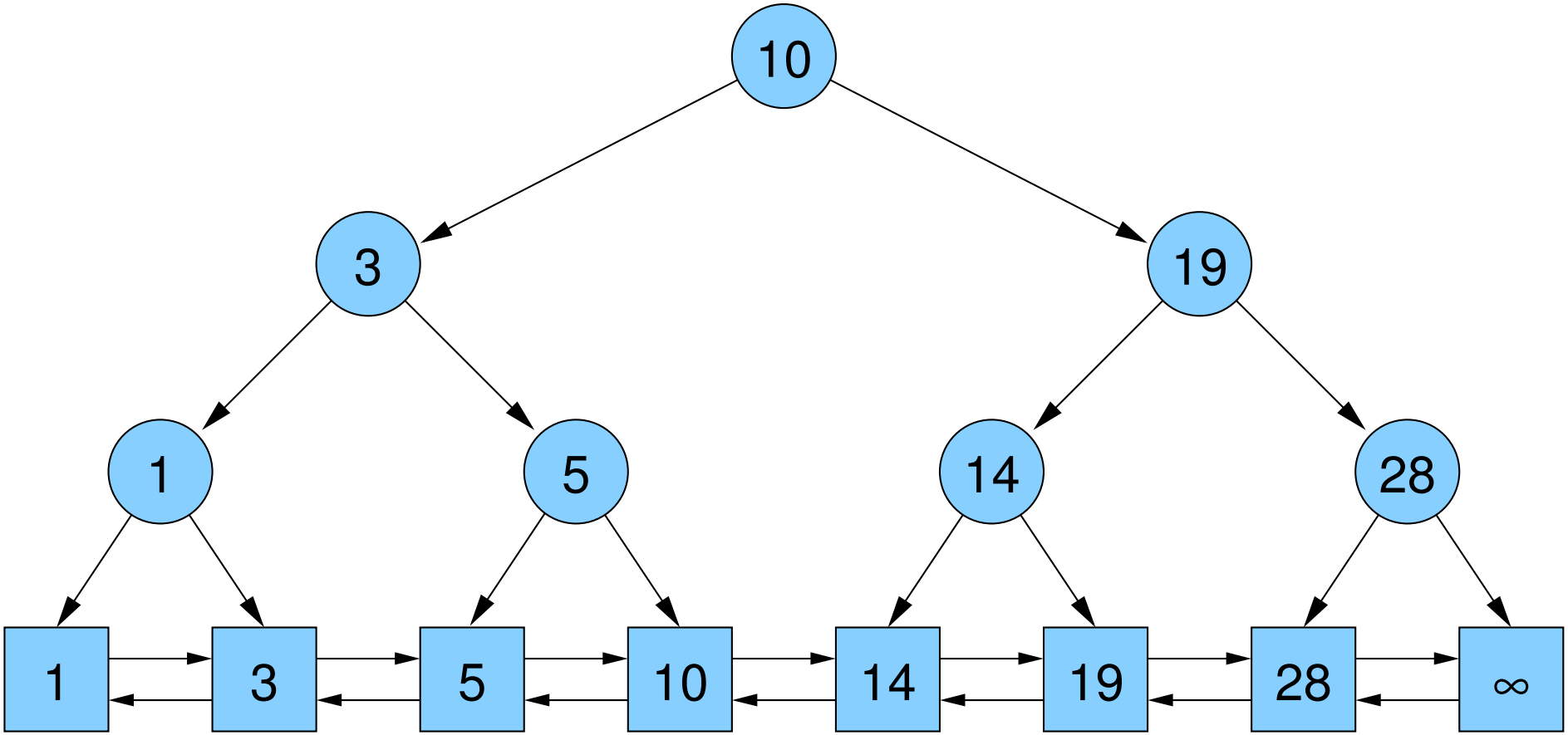
Suchstruktur

Idee:

- füge Navigationsstruktur hinzu, die locate effizient macht



Binärer Suchbaum (ideal)

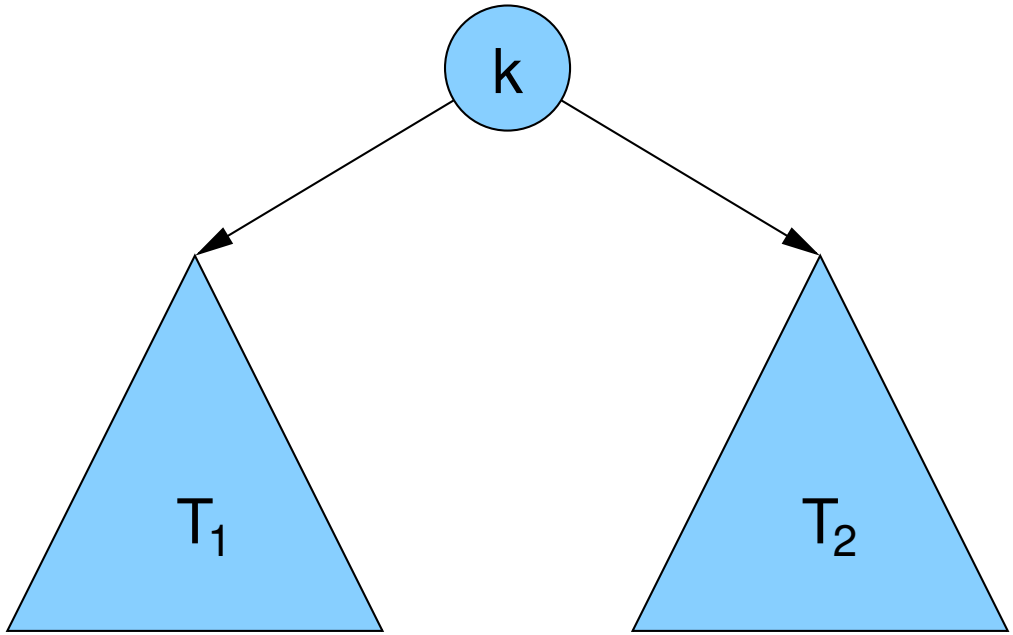


Binärer Suchbaum

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

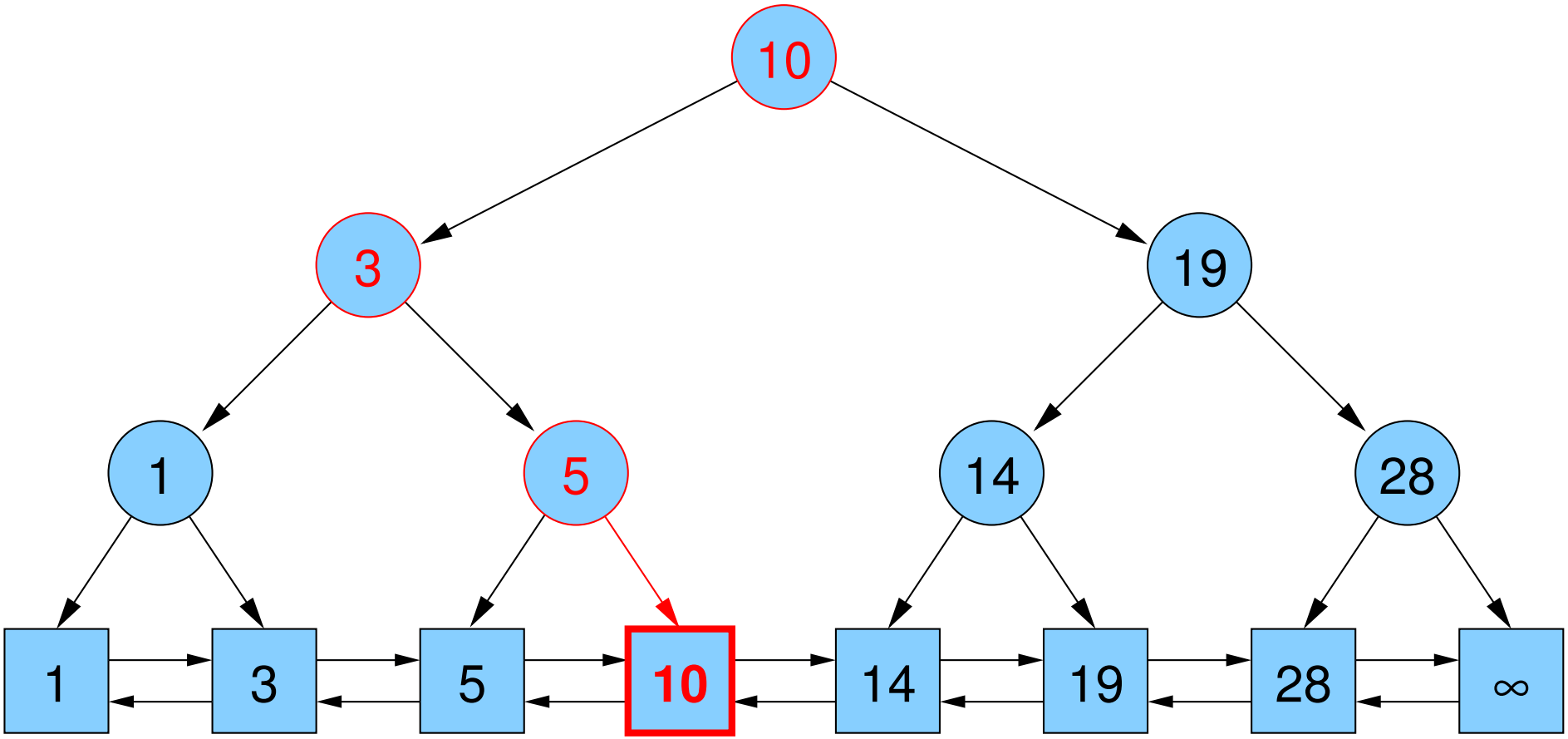
Formal: für einen Baumknoten v sei

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



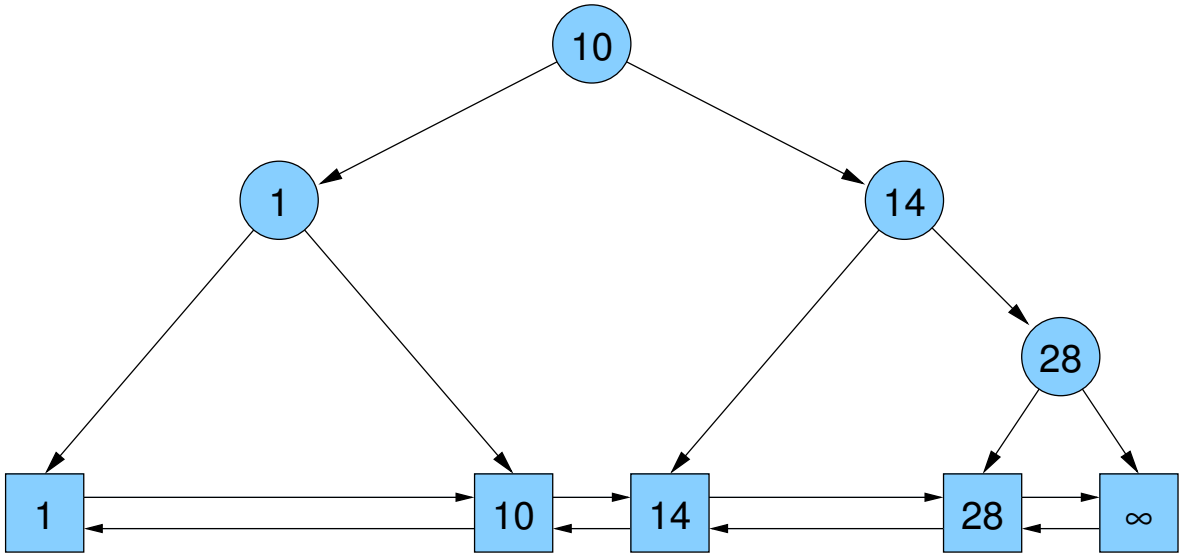
Binärer Suchbaum / insert, remove

Strategie:

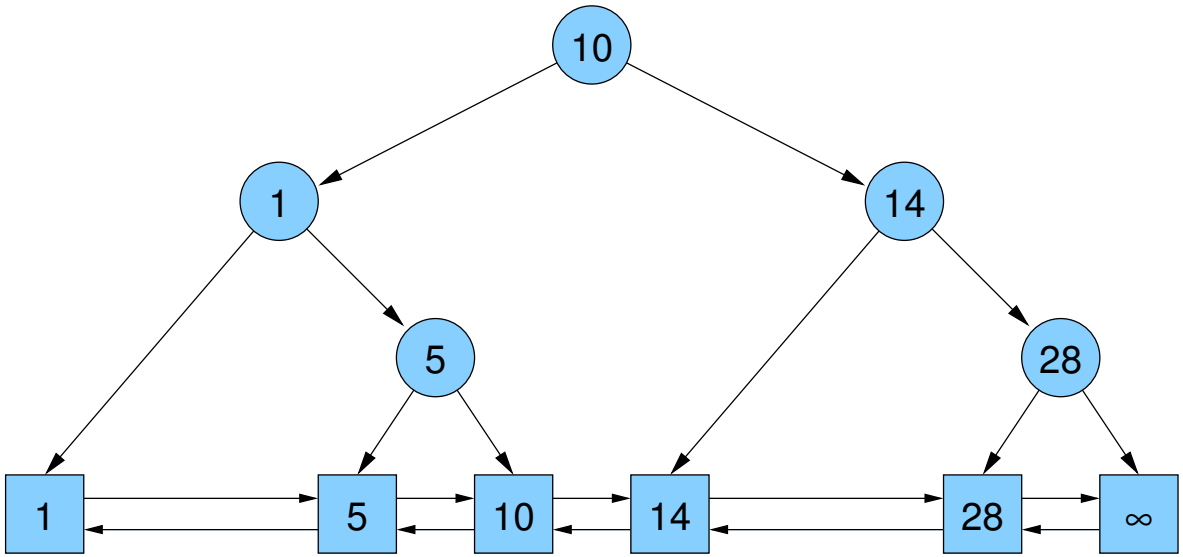
- insert(e):
  - erst locate(key(e)) bis Element e' in Liste erreicht
  - falls key(e') $>$ key(e), füge e vor e' ein und ein neues Suchbaumblatt für e und e' mit key(e), so dass Suchbaum-Regel erfüllt

- remove(k):
  - erst 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)

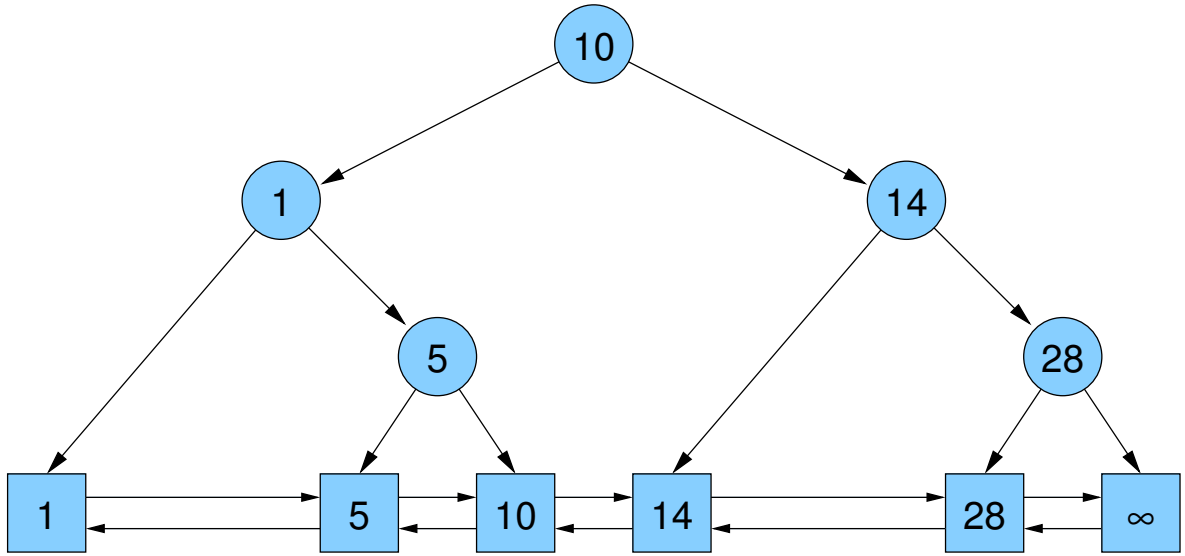
Binärer Suchbaum / insert, remove



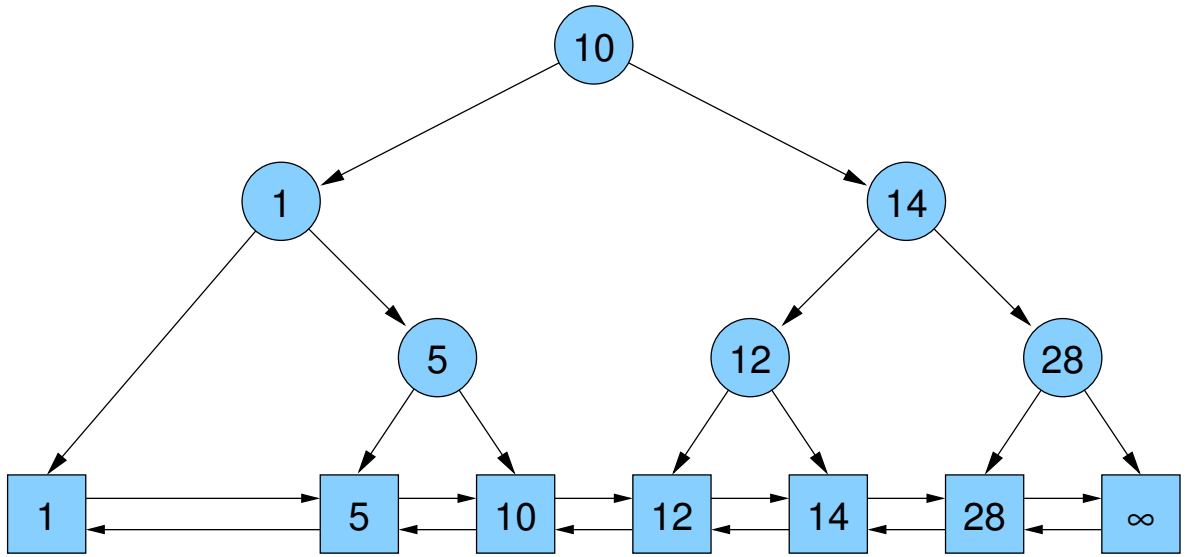
insert(5)



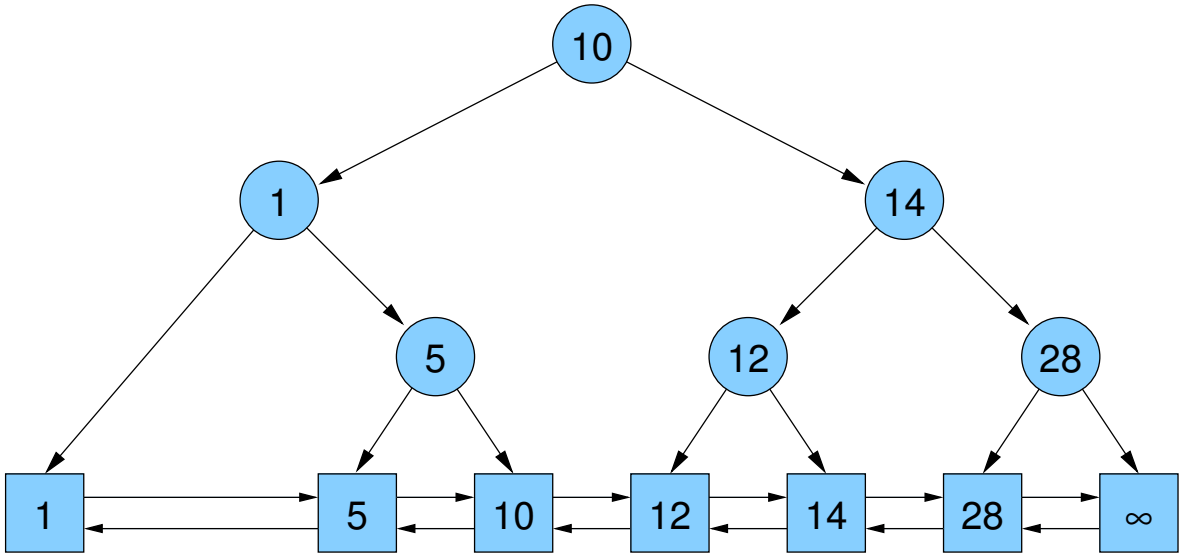
Binärer Suchbaum / insert, remove



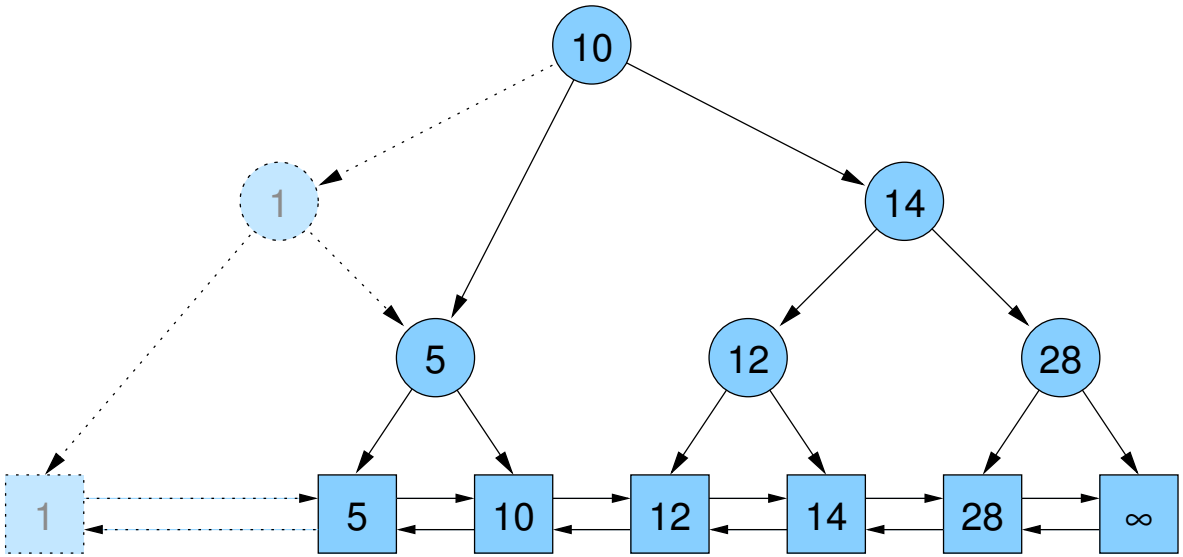
insert(12)



Binärer Suchbaum / insert, remove

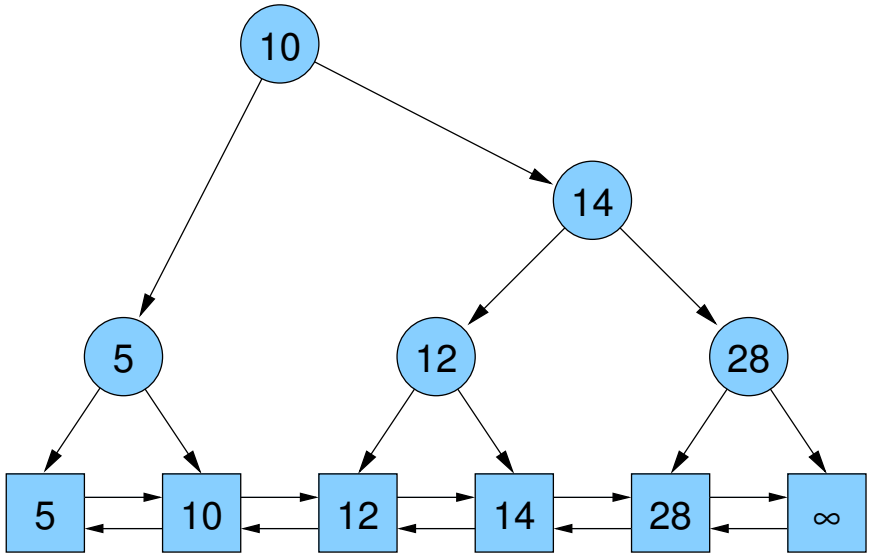


remove(1)

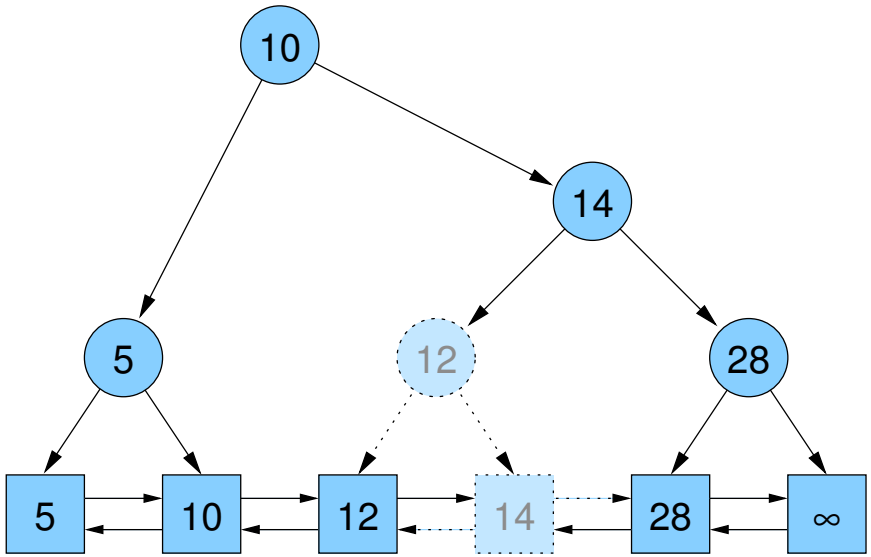


Binärer Suchbaum / insert, remove

1

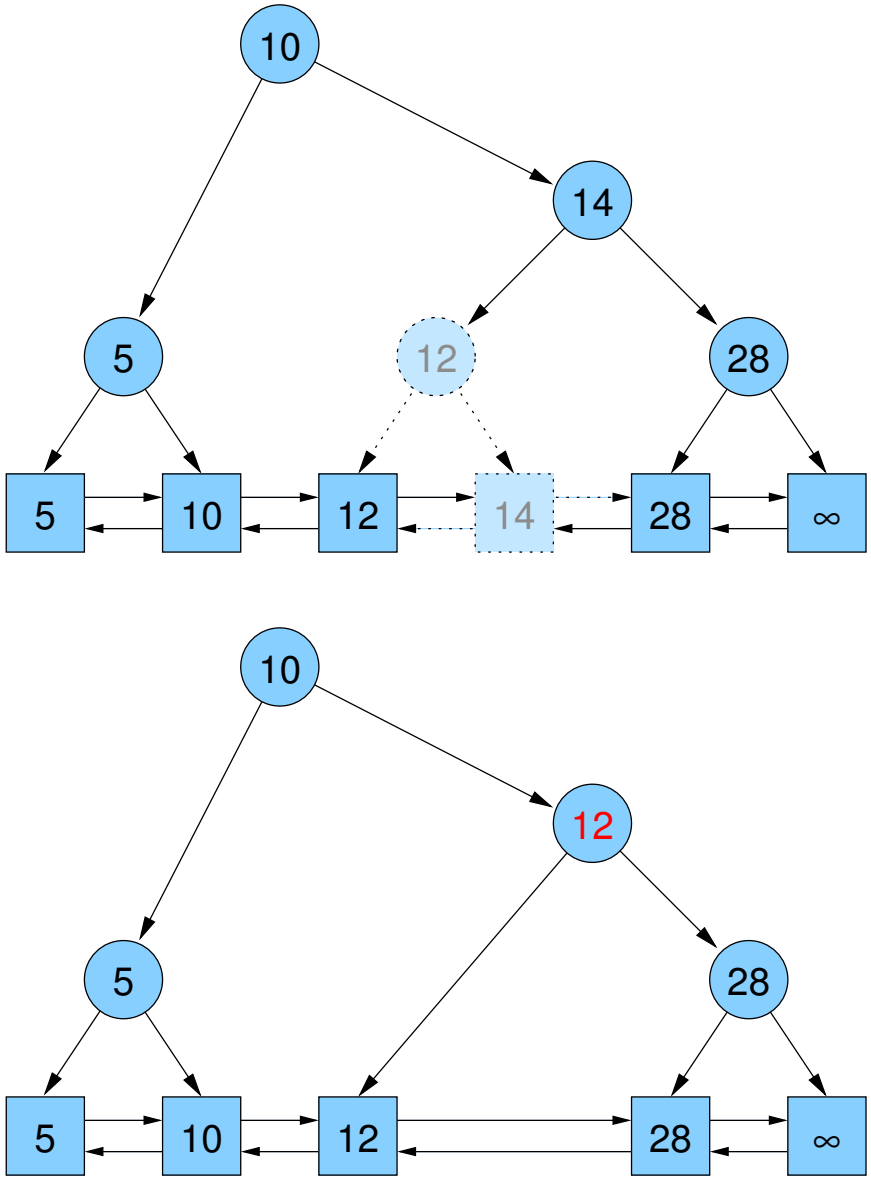


remove(14)



Binärer Suchbaum / insert, remove

remove(14)



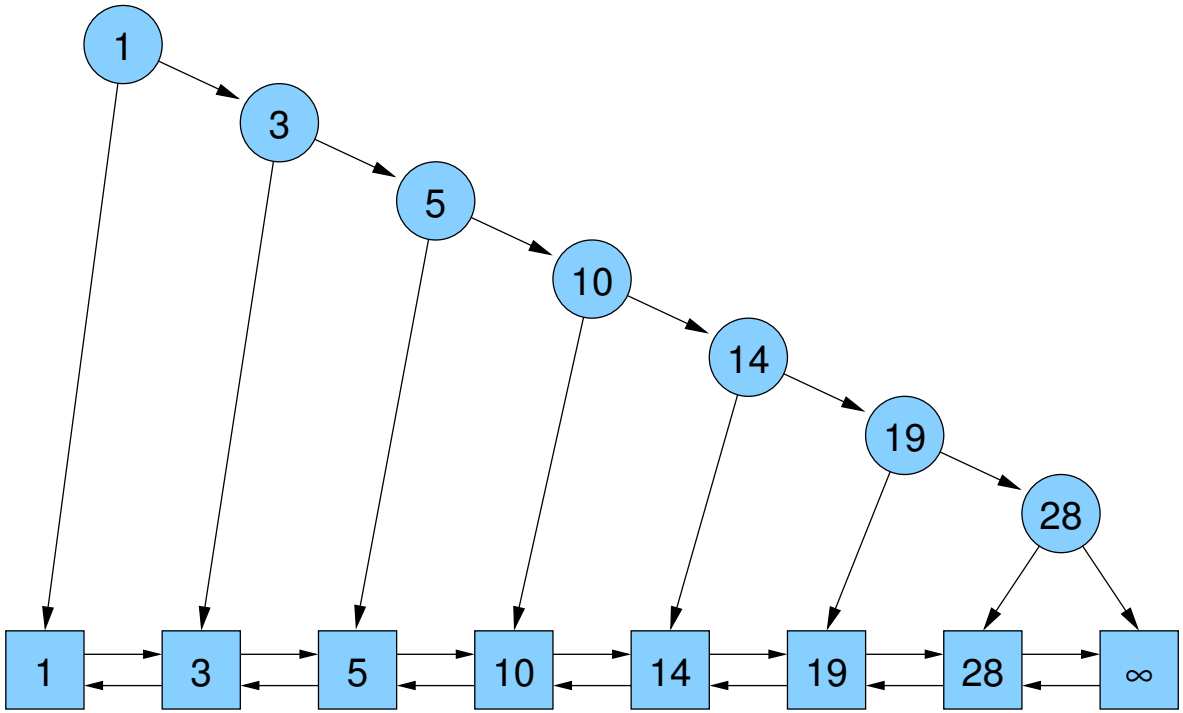
Binärer Suchbaum / entartet

Problem:

- Baumstruktur kann zur Liste entarten

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

Beispiel: Zahlen werden in sortierter Reihenfolge eingefügt



(a,b)-Baum

Problem: Baumstruktur kann zur Liste entarten

Lösung: (a,b)-Baum

Idee:

- $d(v)$: Ausgangsgrad (also die Anzahl der 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: $2 \leq d(r) \leq b$ (sofern #Elemente > 1)

(a,b)-Baum

Lemma

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

Beweis.

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

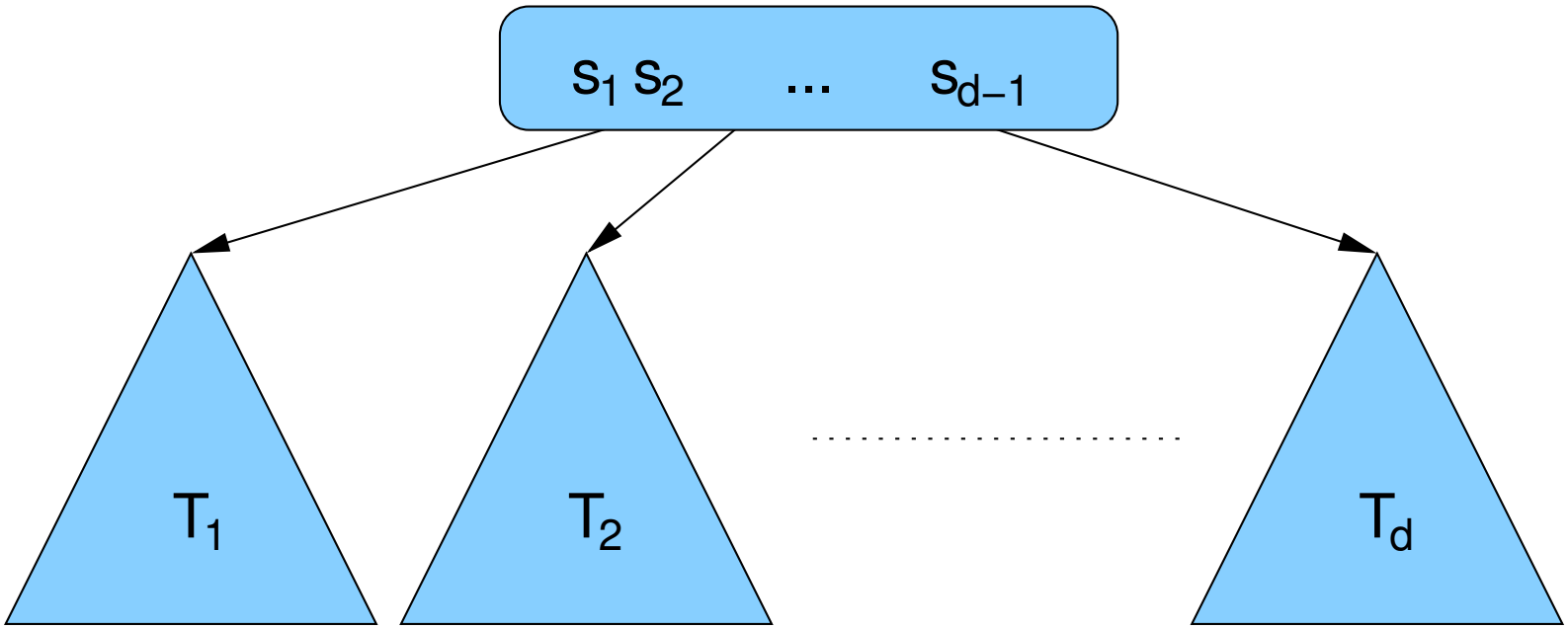
$\Rightarrow$ Bei Tiefe t gibt es $\geq 2a^{t-1}$ Blätter

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

□

(a,b)-Baum: Split-Schlüssel

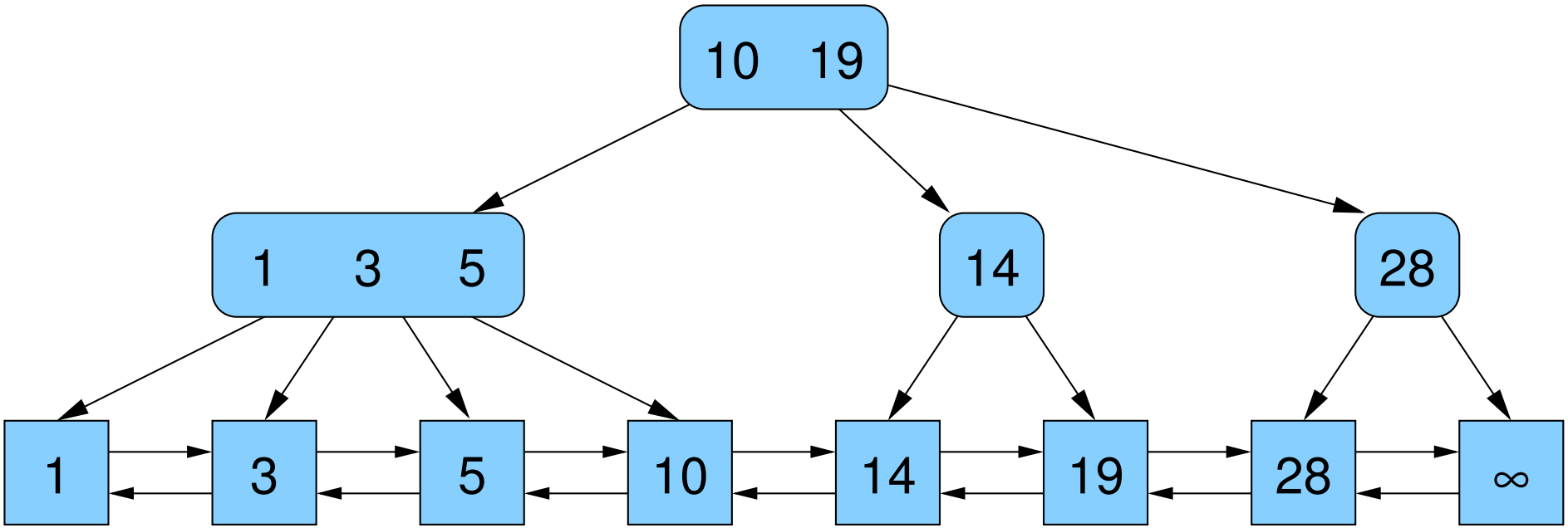
- Jeder Knoten v enthält ein sortiertes Array von $d(v) - 1$ Split-Schlüsseln $s_1, \ldots, s_{d(v)-1}$



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

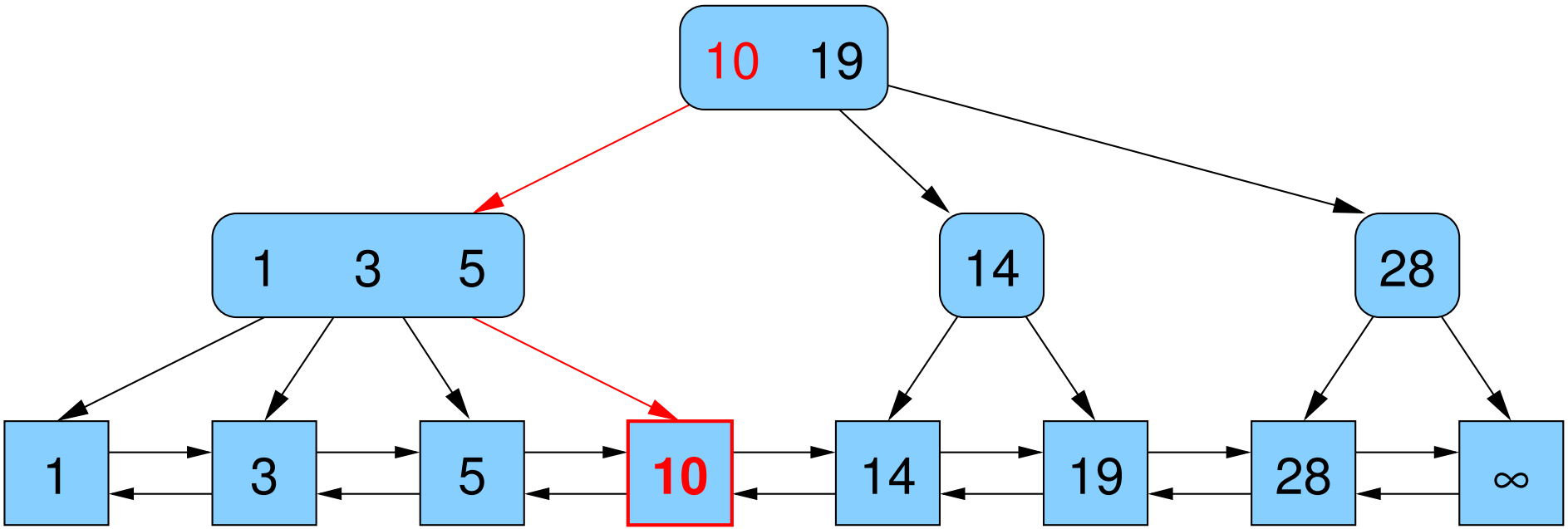
(a,b)-Baum

Beispiel:



(a,b)-Baum

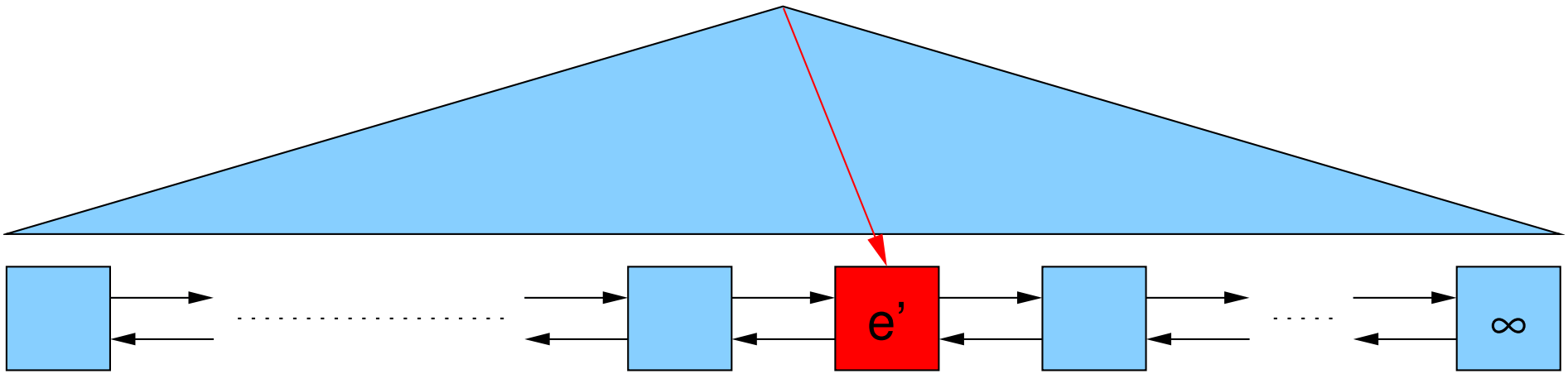
locate(9)



(a,b)-Baum

insert(e)

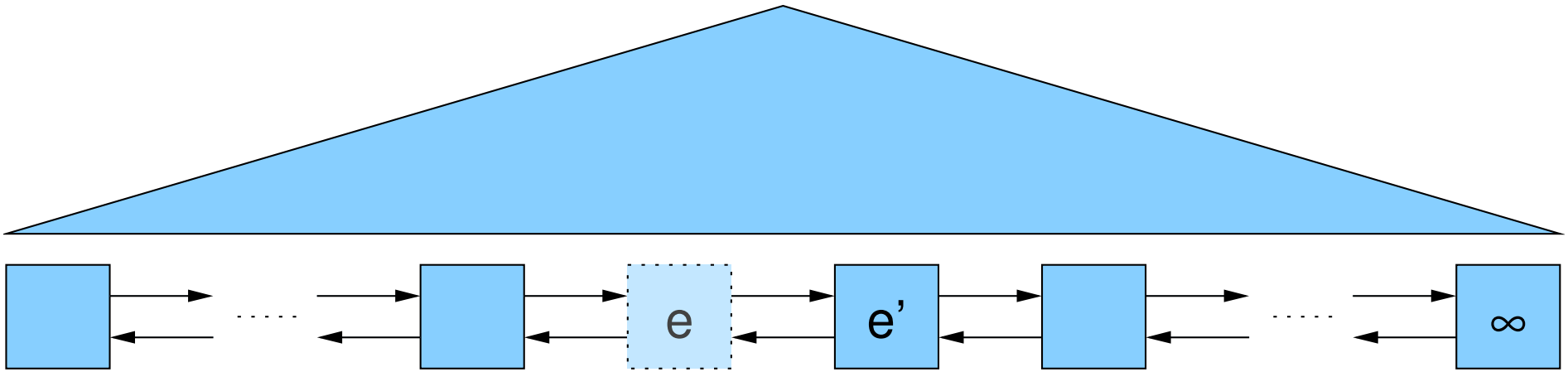
- Abstieg wie bei locate(key(e)) bis Element e' in Liste erreicht
- falls key(e')>key(e), füge e vor e' ein



(a,b)-Baum

insert(e)

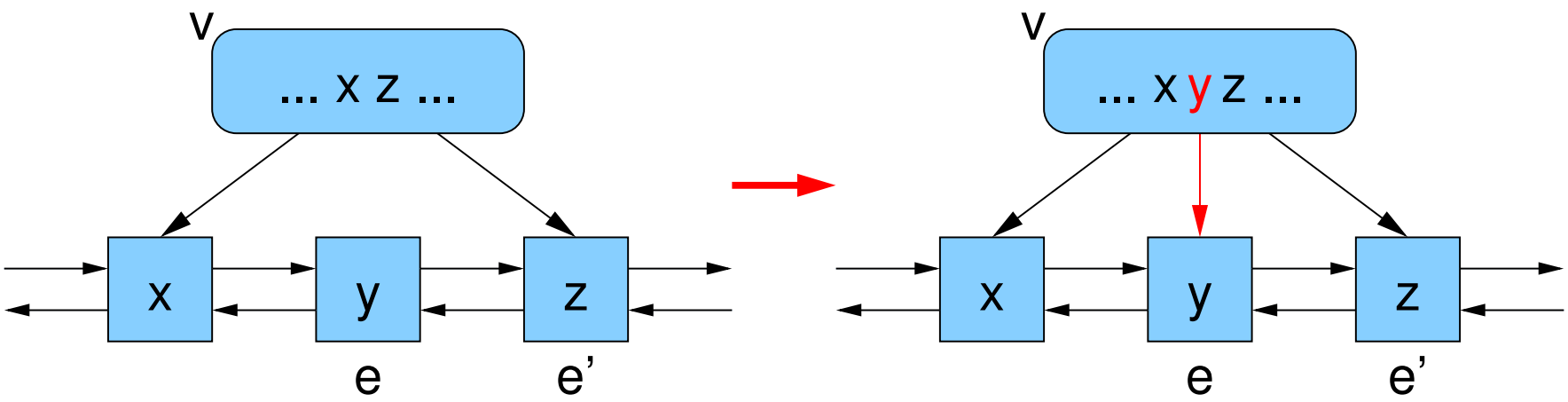
- Abstieg wie bei locate(key(e)) bis Element e' in Liste erreicht
- falls key(e')>key(e), füge e vor e' ein



(a,b)-Baum

insert(e)

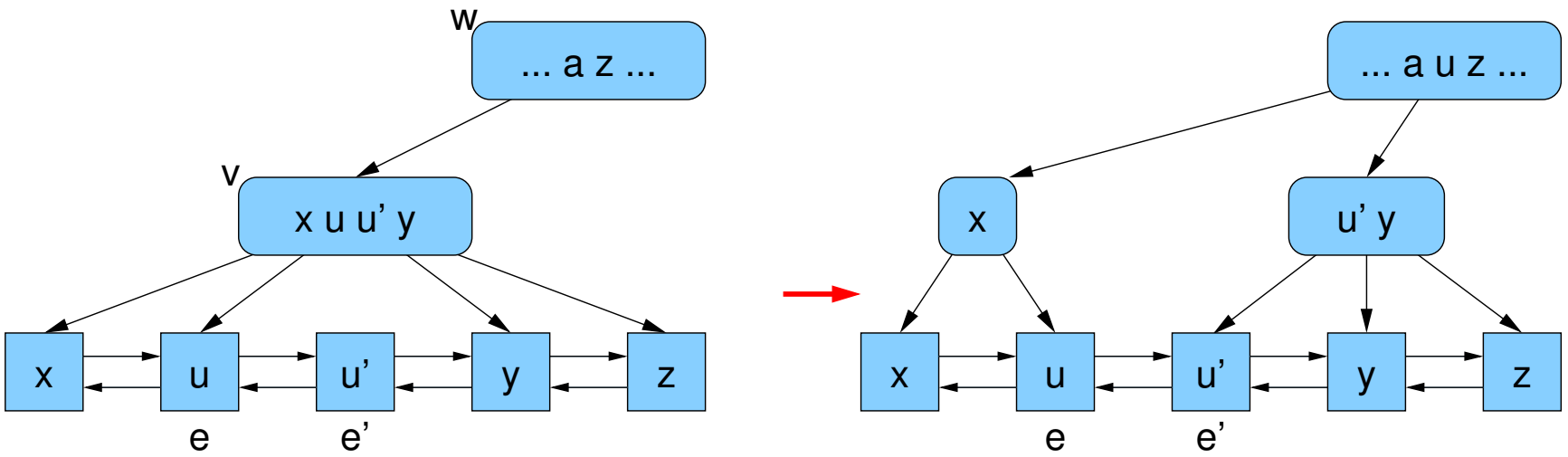
- füge key(e) und Handle auf e in Baumknoten v über e ein
- falls $d(v) \leq b$, dann fertig



(a,b)-Baum

insert(e)

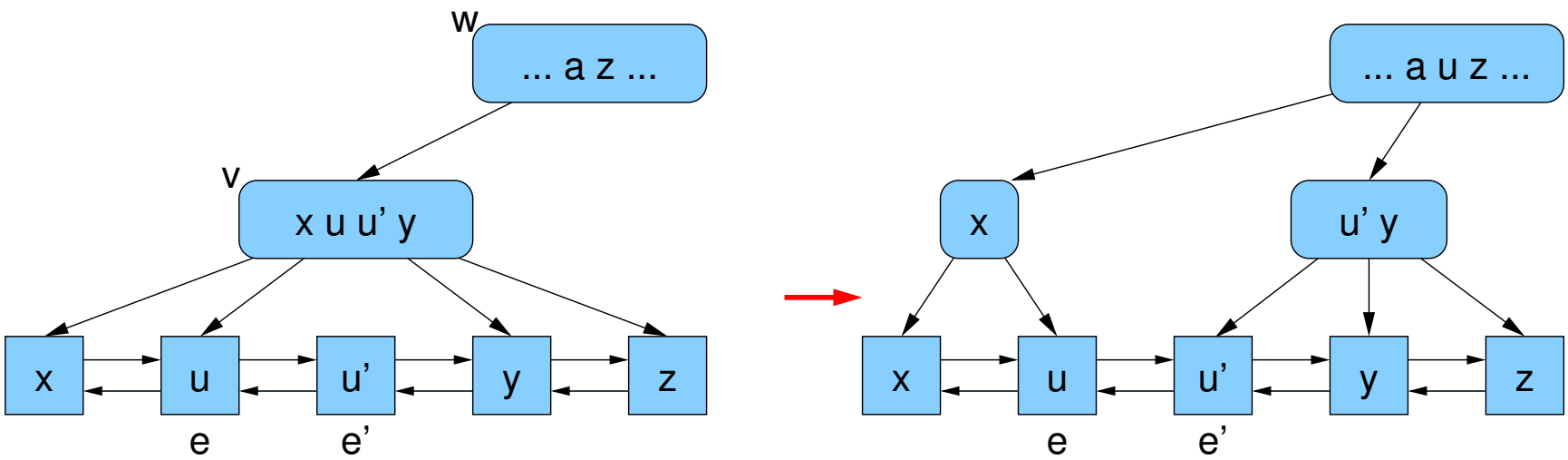
- füge key(e) und Handle auf e in Baumknoten v über e ein
- falls $d(v) > b$, dann teile v in zwei Knoten auf (Bsp.: $a = 2$, $b = 4$)



(a,b)-Baum

insert(e)

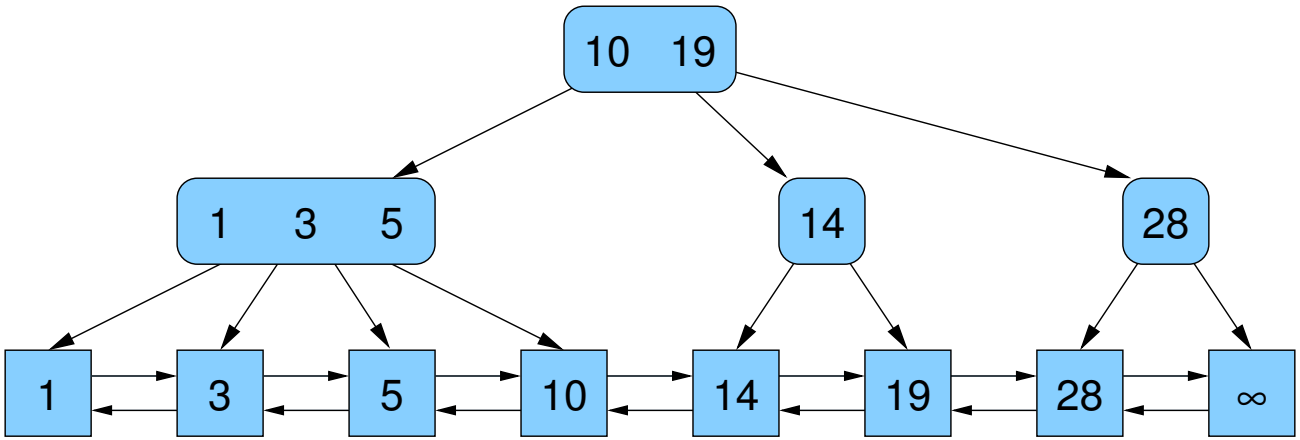
- falls $d(w) > b$, dann teile w in zwei Knoten auf usw.
  bis Grad $\leq b$
  oder Wurzel aufgeteilt wurde



(a,b)-Baum / insert

$a = 2, b = 4$

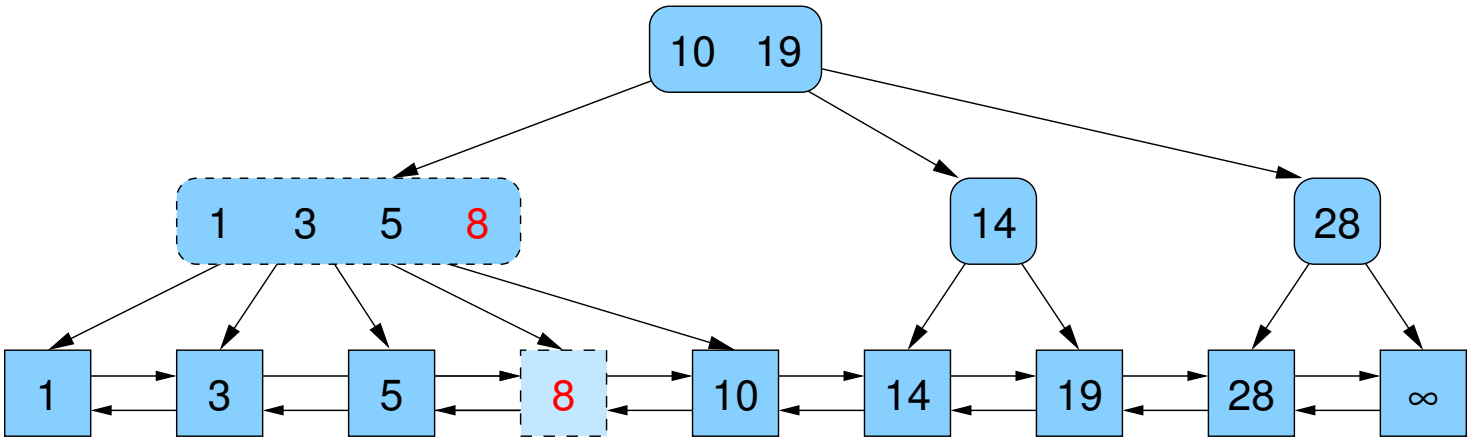
insert(8)



(a,b)-Baum / insert

$a = 2, b = 4$

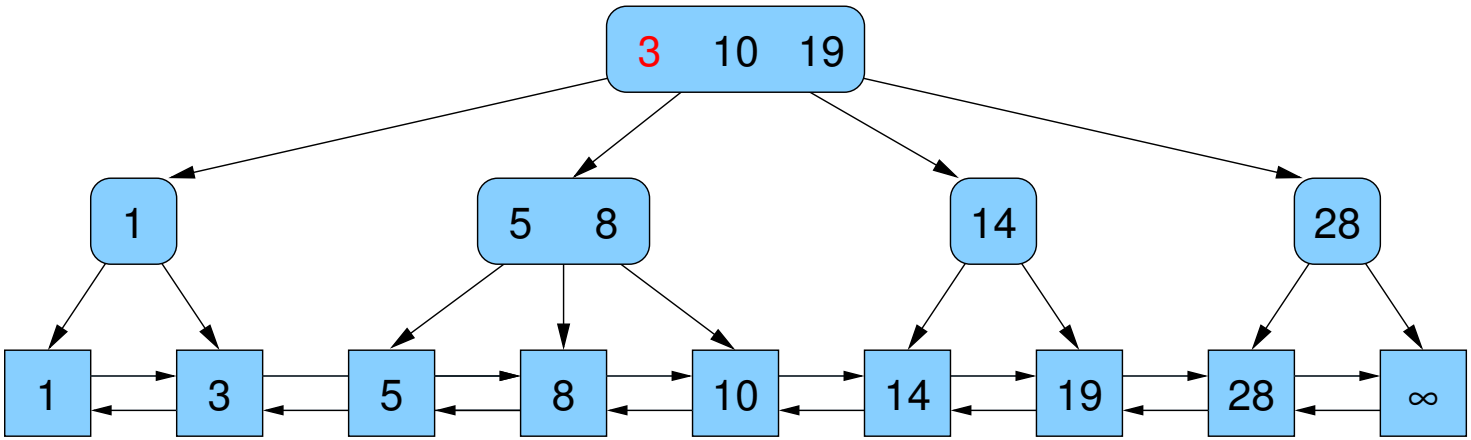
insert(8)



(a,b)-Baum / insert

$a = 2, b = 4$

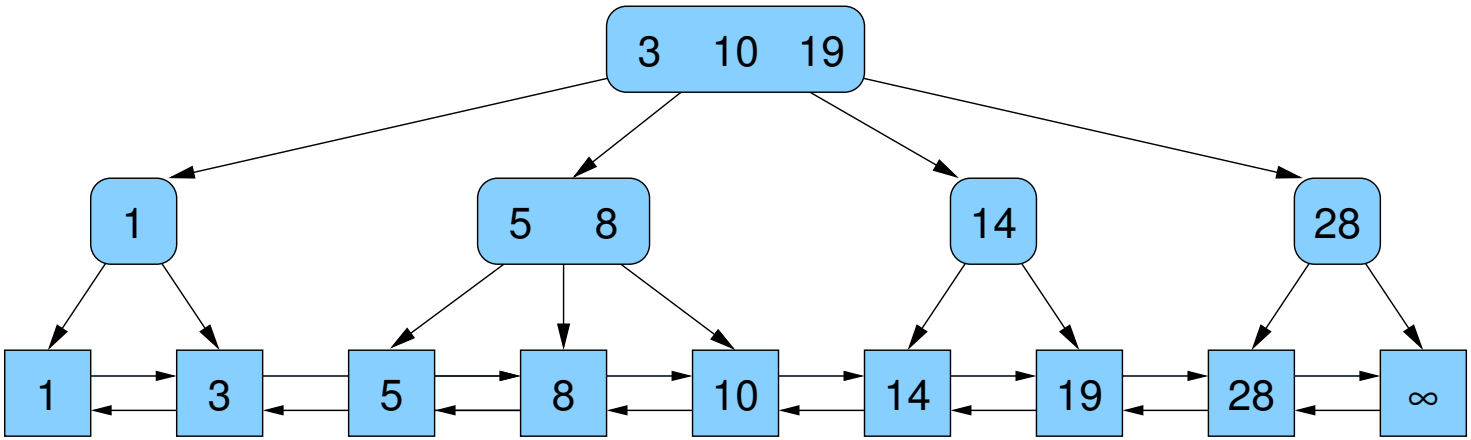
insert(8)



(a,b)-Baum / insert

$a = 2, b = 4$

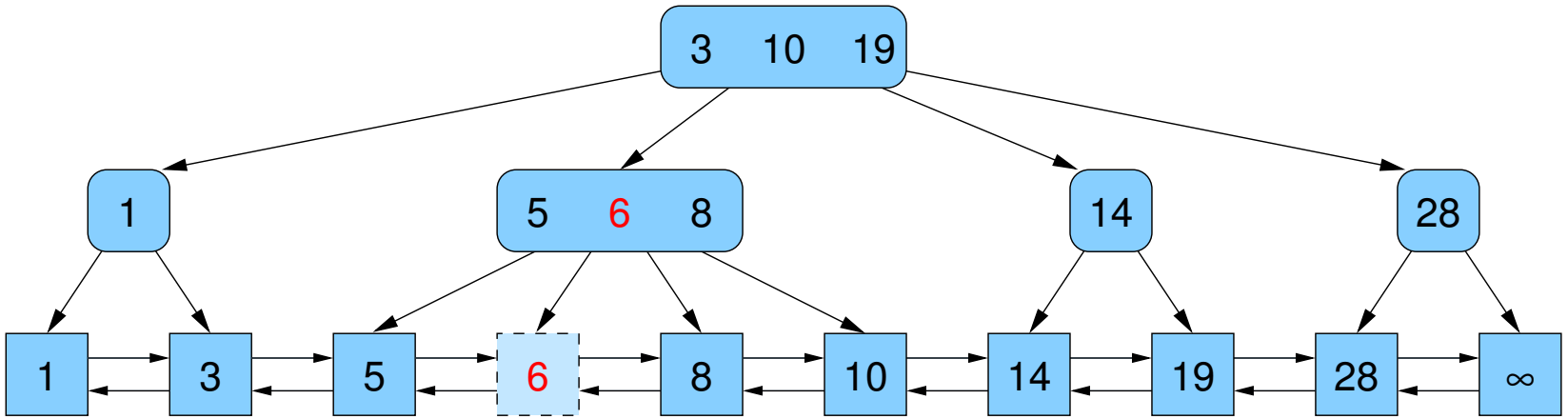
insert(6)



(a,b)-Baum / insert

$a = 2, b = 4$

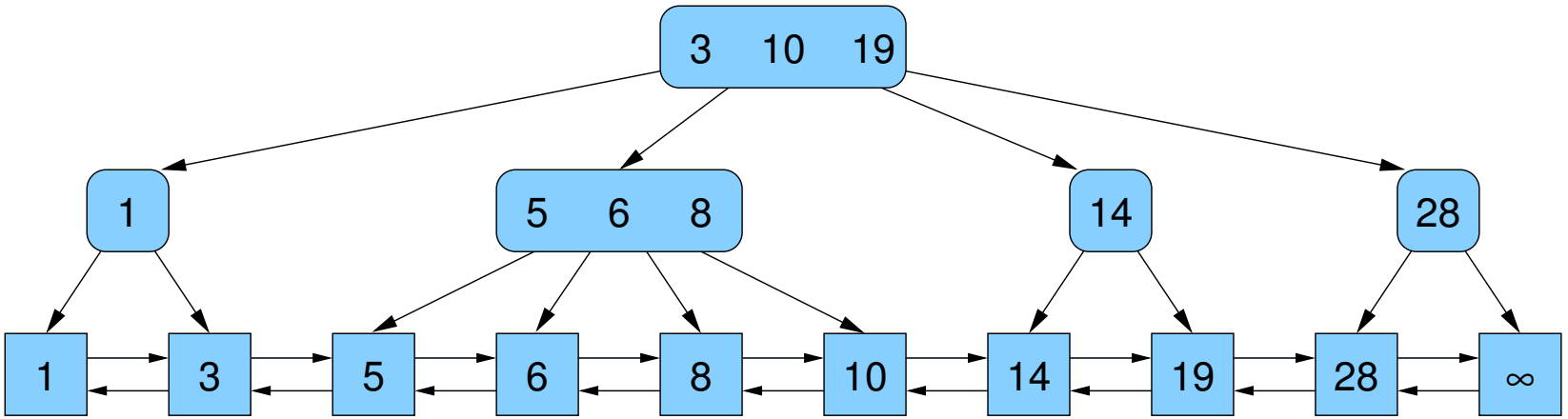
insert(6)



(a, b)-Baum / insert

$a = 2, b = 4$

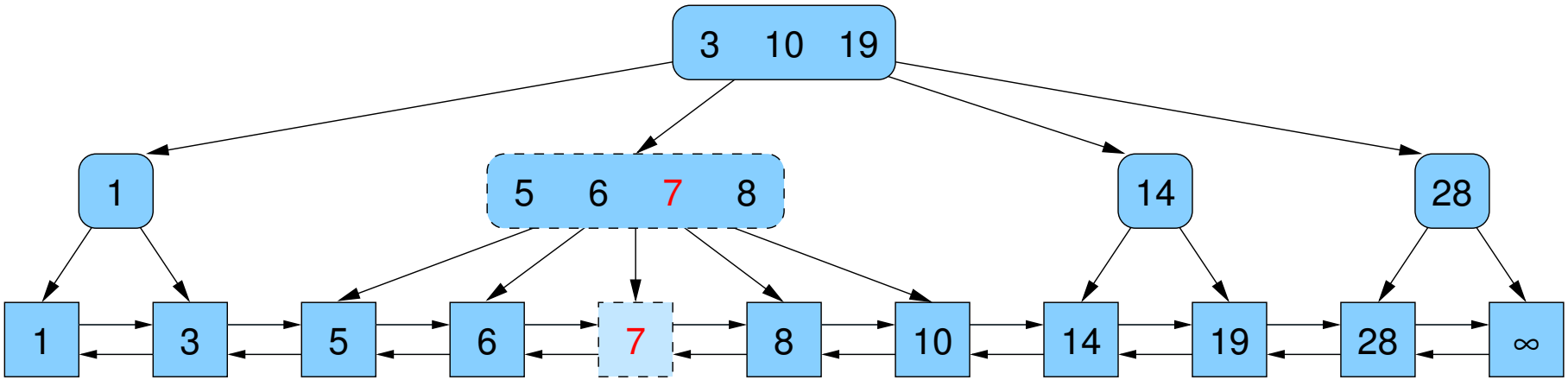
insert(7)



(a, b)-Baum / insert

$a = 2, b = 4$

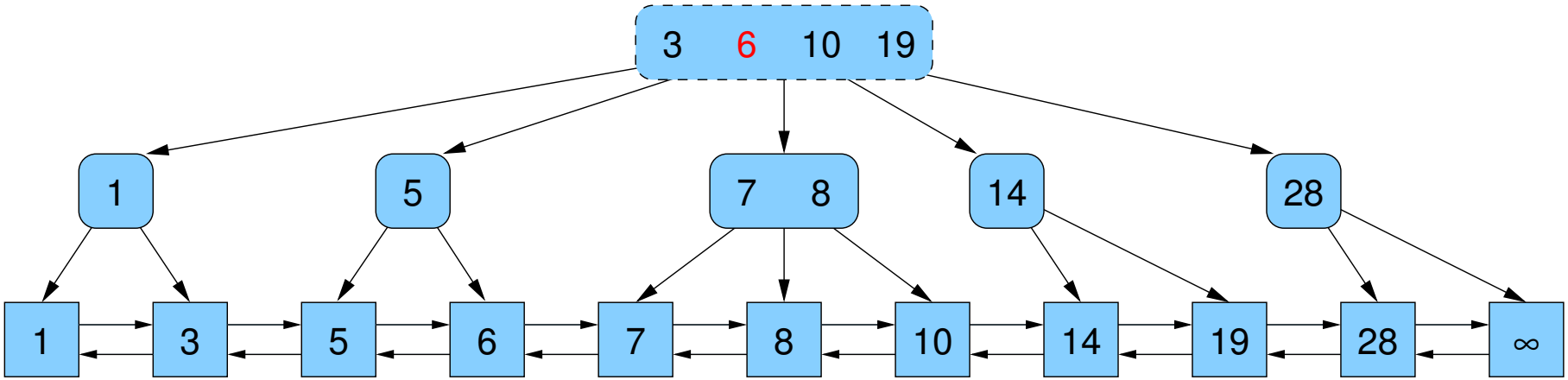
insert(7)



(a,b)-Baum / insert

$a = 2, b = 4$

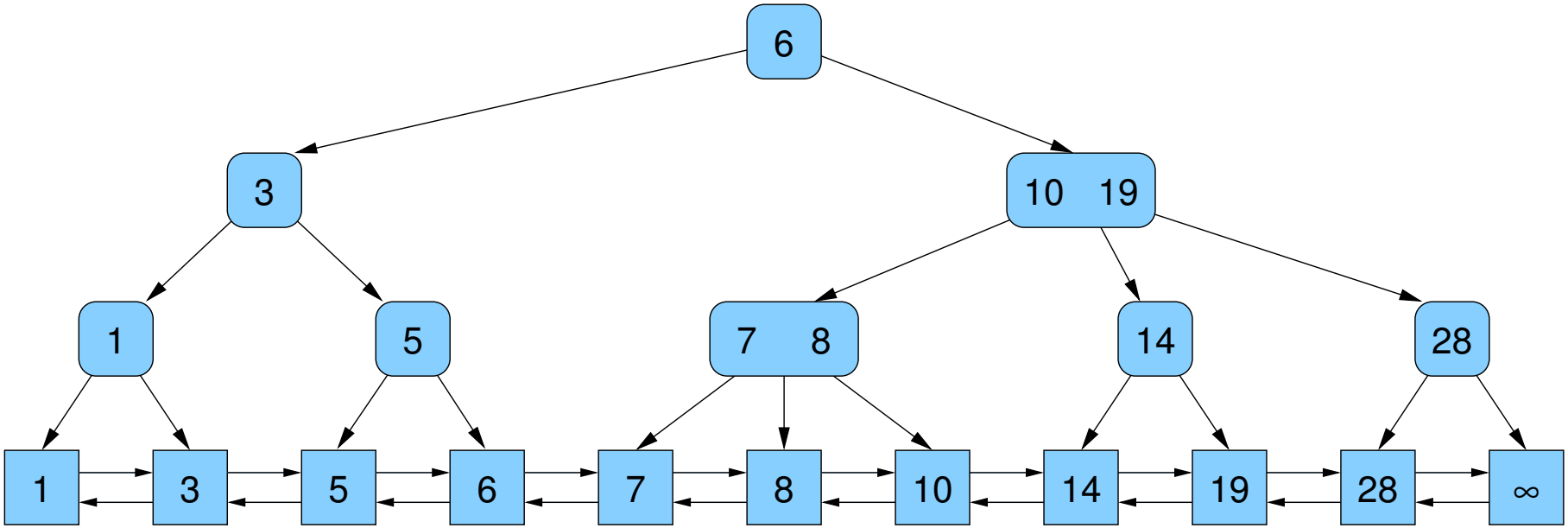
insert(7)



(a,b)-Baum / insert

$a = 2, b = 4$

insert(7)



(a,b)-Baum / insert

Form-Invariante

- durch insert erfüllt: alle Blätter haben dieselbe Tiefe

Grad-Invariante

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