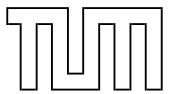
Grundlagen: Algorithmen und Datenstrukturen

Prof. Dr. Hanjo Täubig

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

Sommersemester 2010

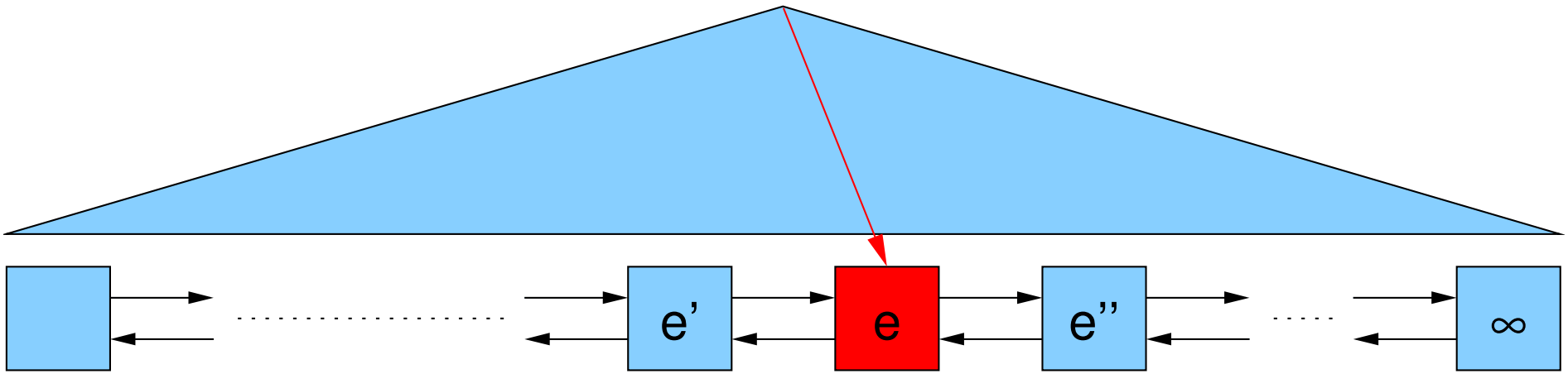


Übersicht

(a,b)-Baum

remove(k)

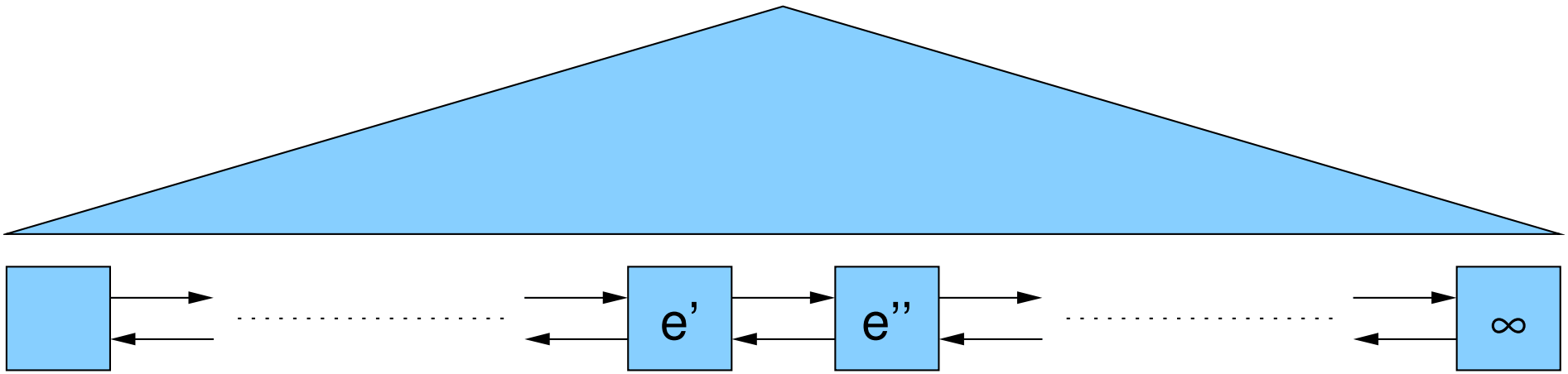
- Abstieg wie bei locate(k) bis Element e in Liste erreicht
- falls key(e) $= k$, entferne e aus Liste (sonst return)



(a,b)-Baum

remove(k)

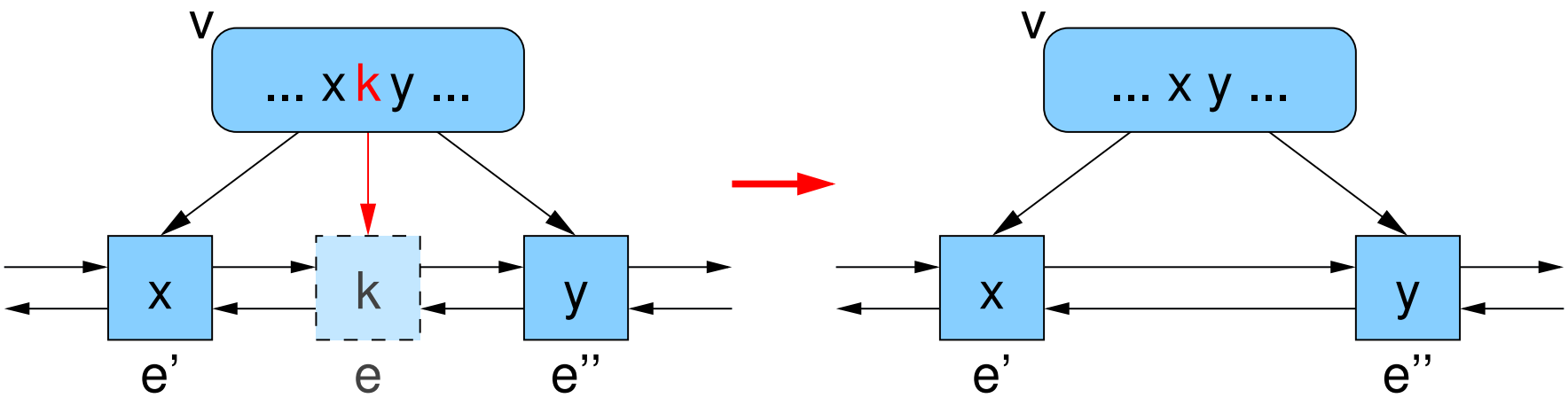
- Abstieg wie bei locate(k) bis Element e in Liste erreicht
- falls key(e) $= k$, entferne e aus Liste (sonst return)



(a,b)-Baum

remove(k)

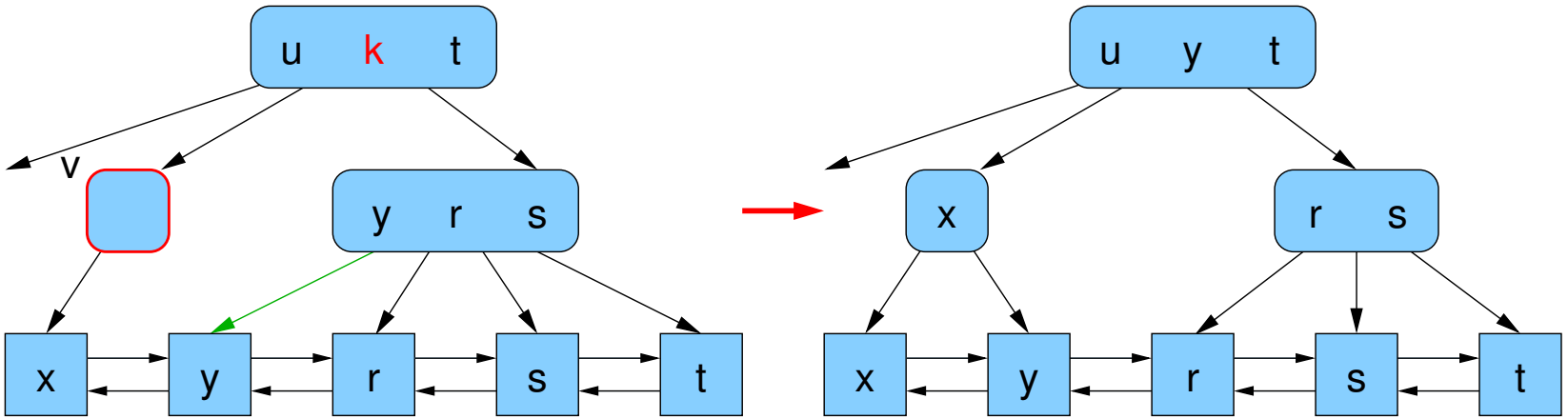
- entferne Handle auf e und Schlüssel k vom Baumknoten v über e (wenn e rechtestes Kind: Schlüsselvertauschung wie bei binärem Suchbaum)
- falls $d(v) \geq a$, dann fertig



(a, b)-Baum

remove(k)

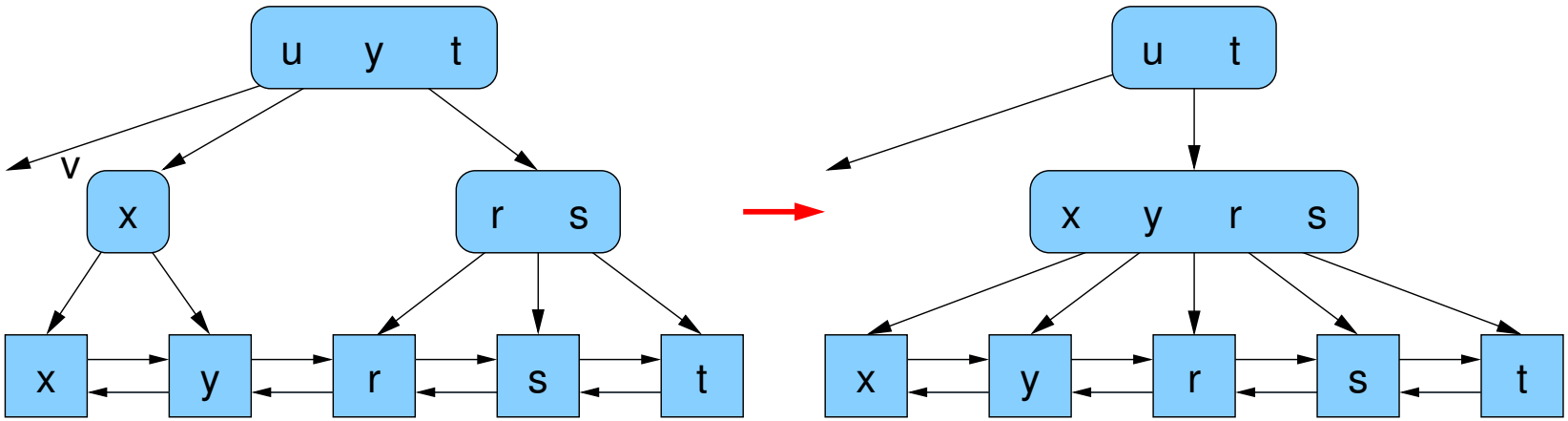
- falls $d(v) < a$ und ein direkter Nachbar v' von v hat Grad $> a$, nimm Kante von v' (Bsp.: $a = 2$, $b = 4$)



(a, b)-Baum

remove(k)

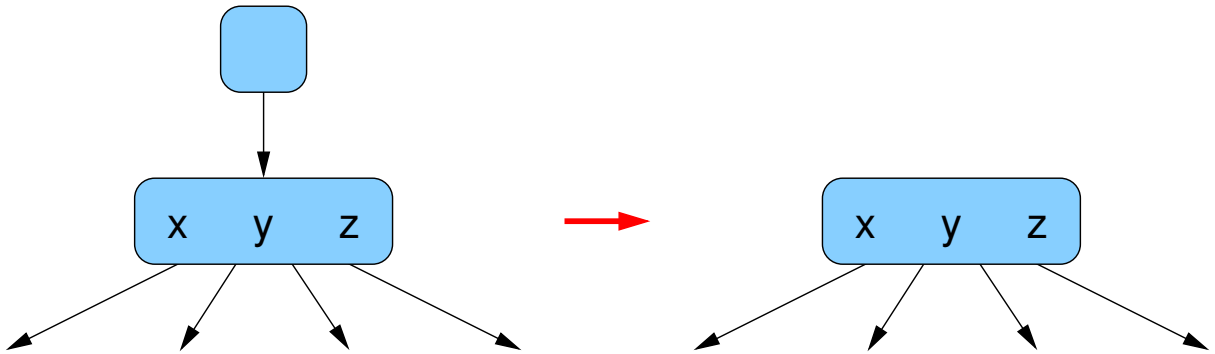
- falls $d(v) < a$ und kein direkter Nachbar von v hat Grad $> a$, merge v mit Nachbarn (Bsp.: $a = 3$, $b = 5$)



(a,b)-Baum

remove(k)

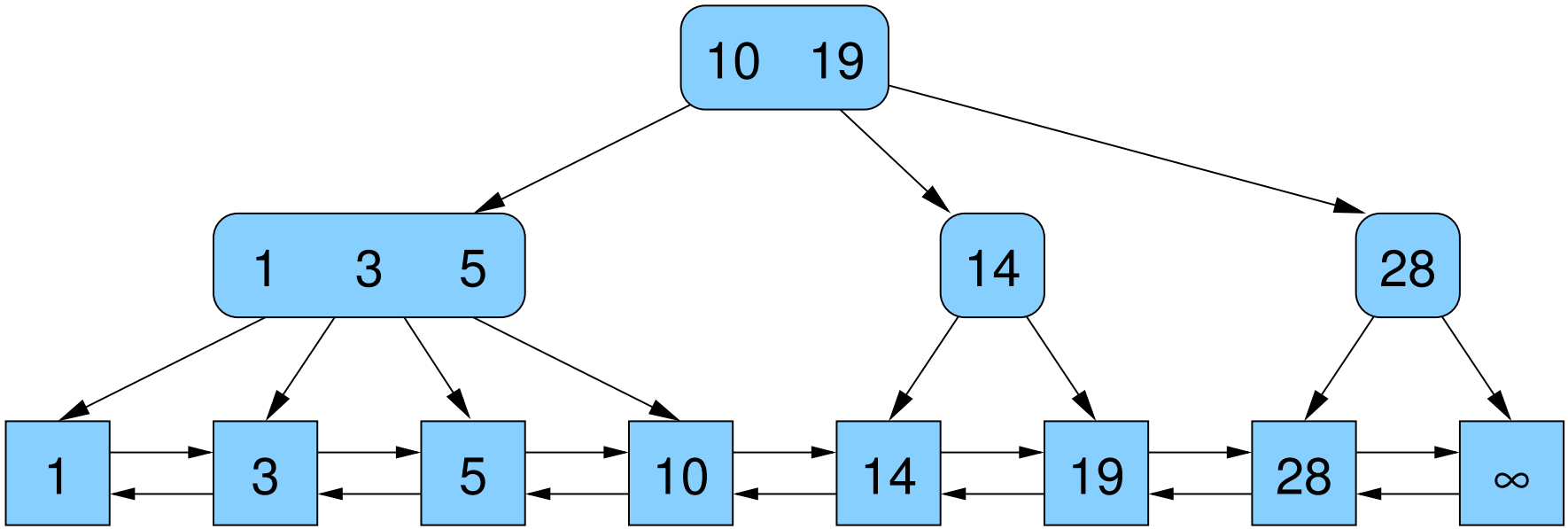
- Veränderungen hoch bis Wurzel
- falls Grad der Wurzel < 2: entferne Wurzel
  neue Wurzel wird das einzige Kind der alten Wurzel



(a,b)-Baum / remove

$a = 2, b = 4$

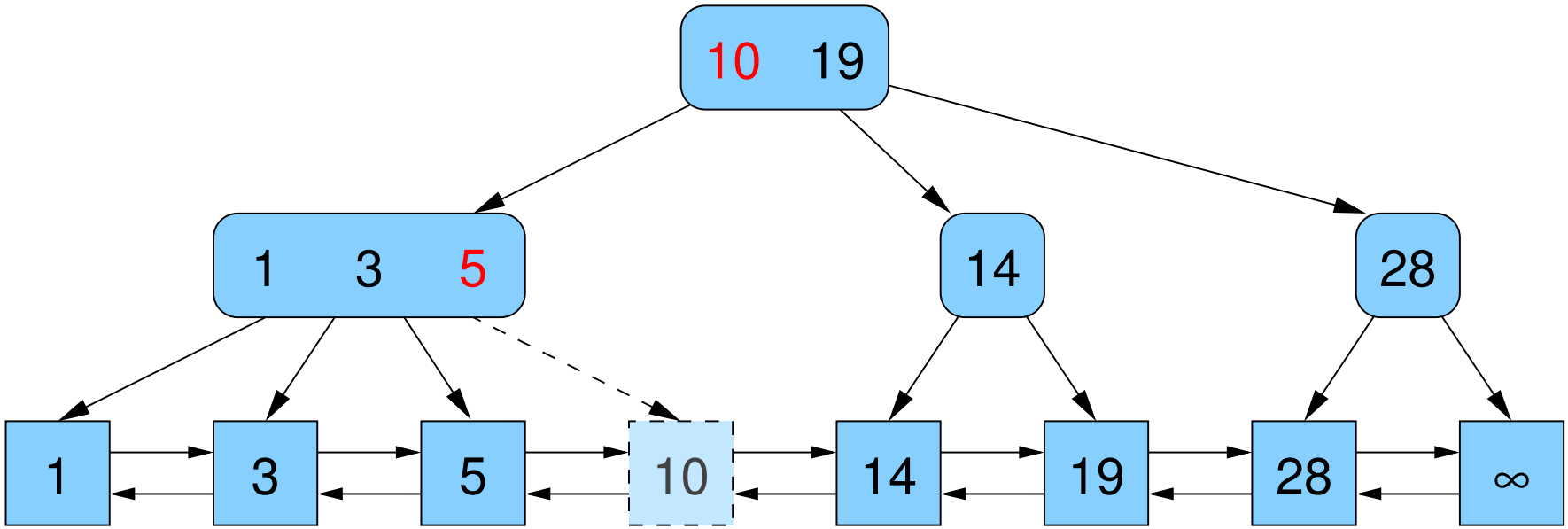
remove(10)



(a,b)-Baum / remove

$a = 2, b = 4$

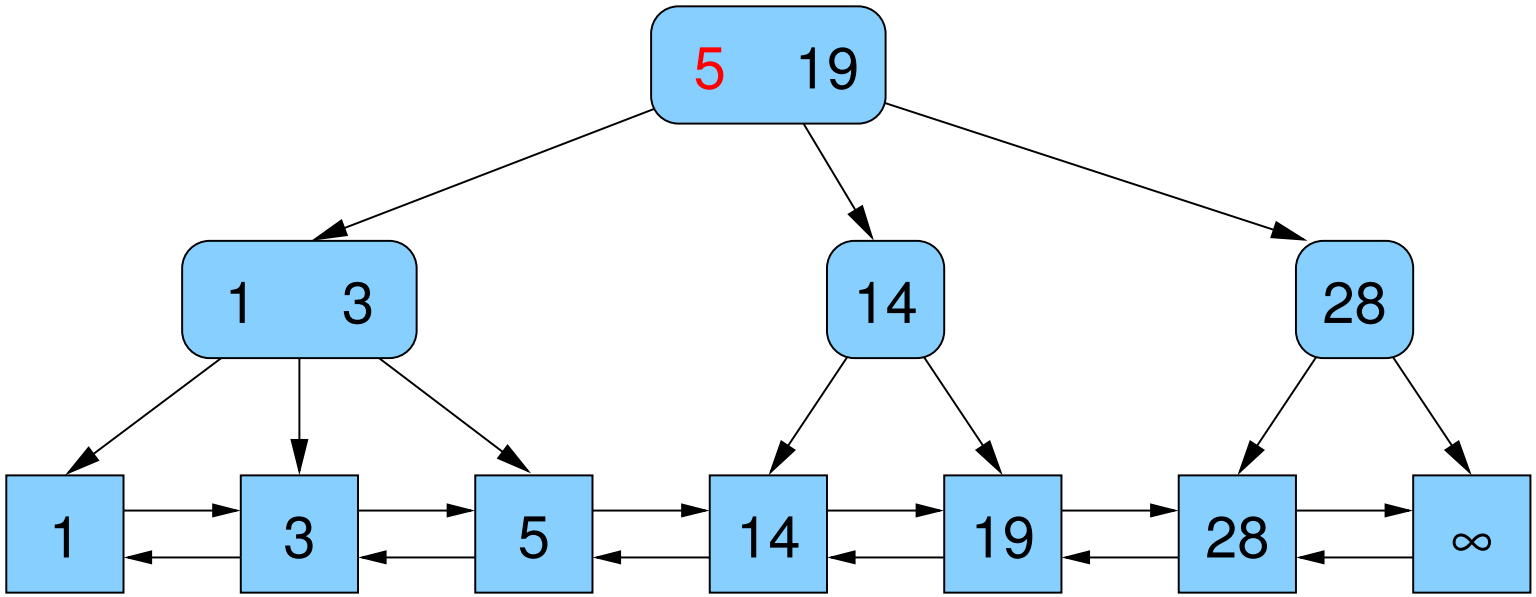
remove(10)



(a,b)-Baum / remove

$a = 2, b = 4$

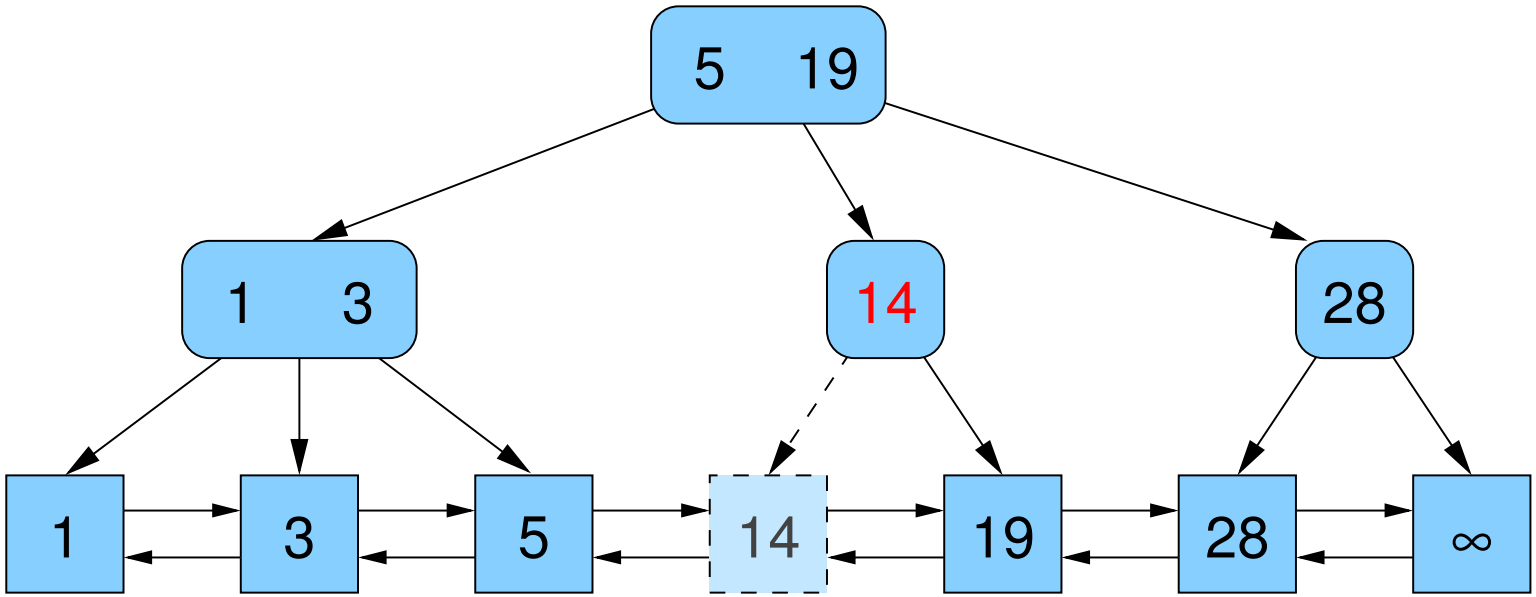
remove(10)



(a,b)-Baum / remove

$a = 2, b = 4$

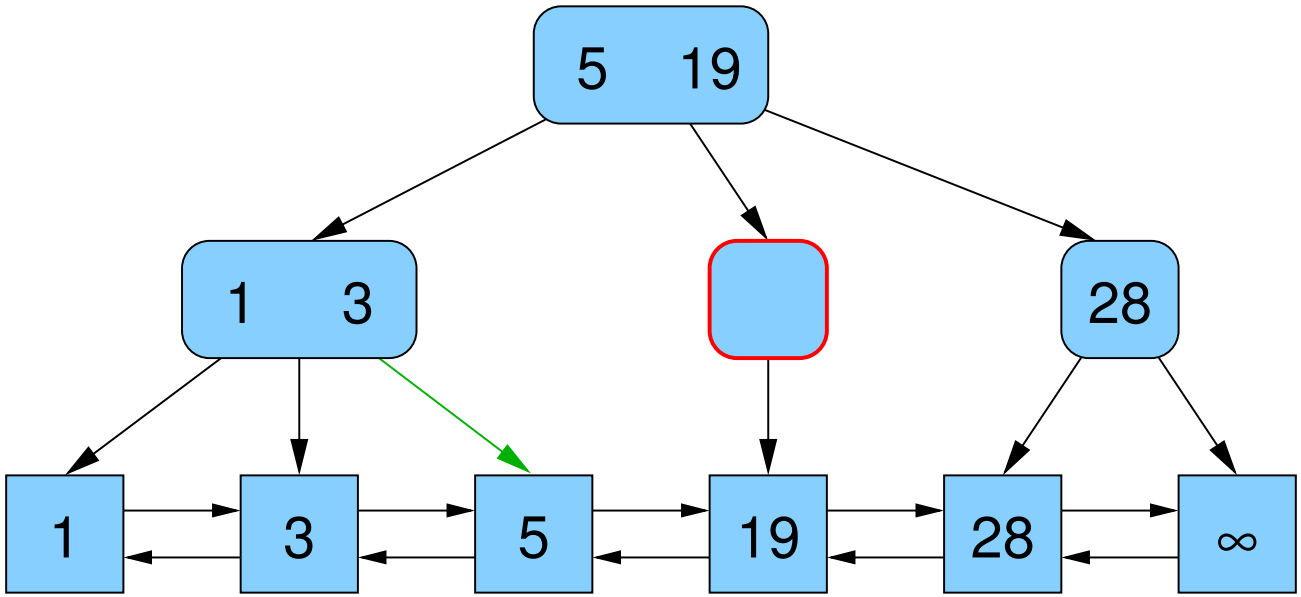
remove(14)



(a,b)-Baum / remove

$a = 2, b = 4$

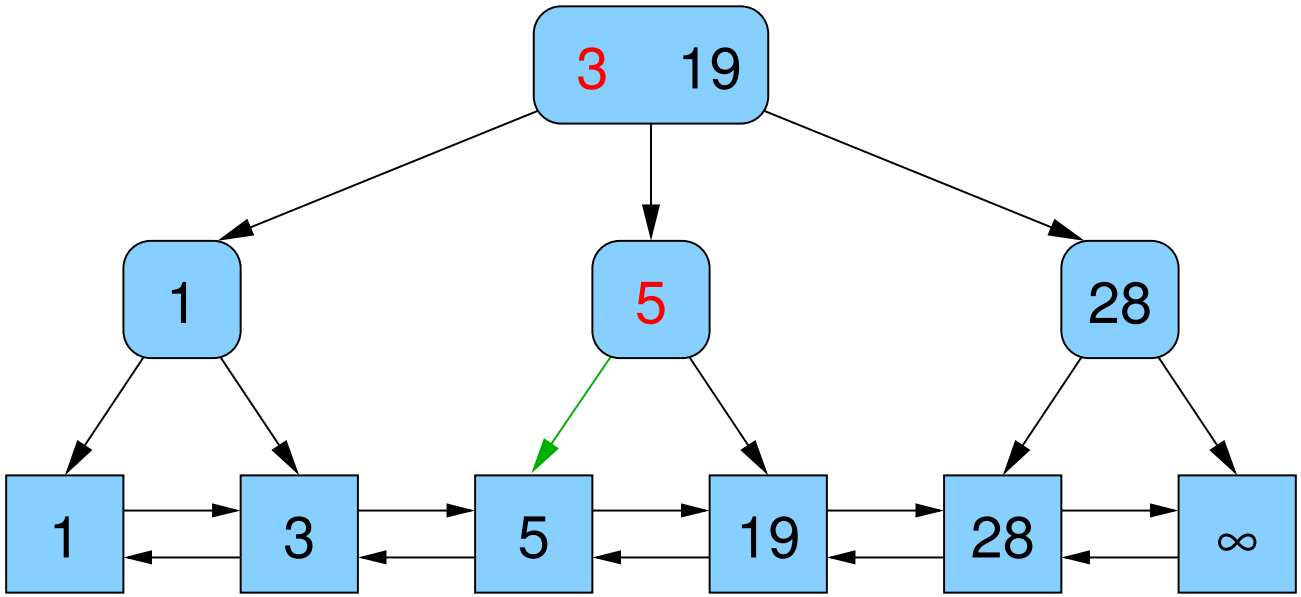
remove(14)



(a,b)-Baum / remove

$a = 2, b = 4$

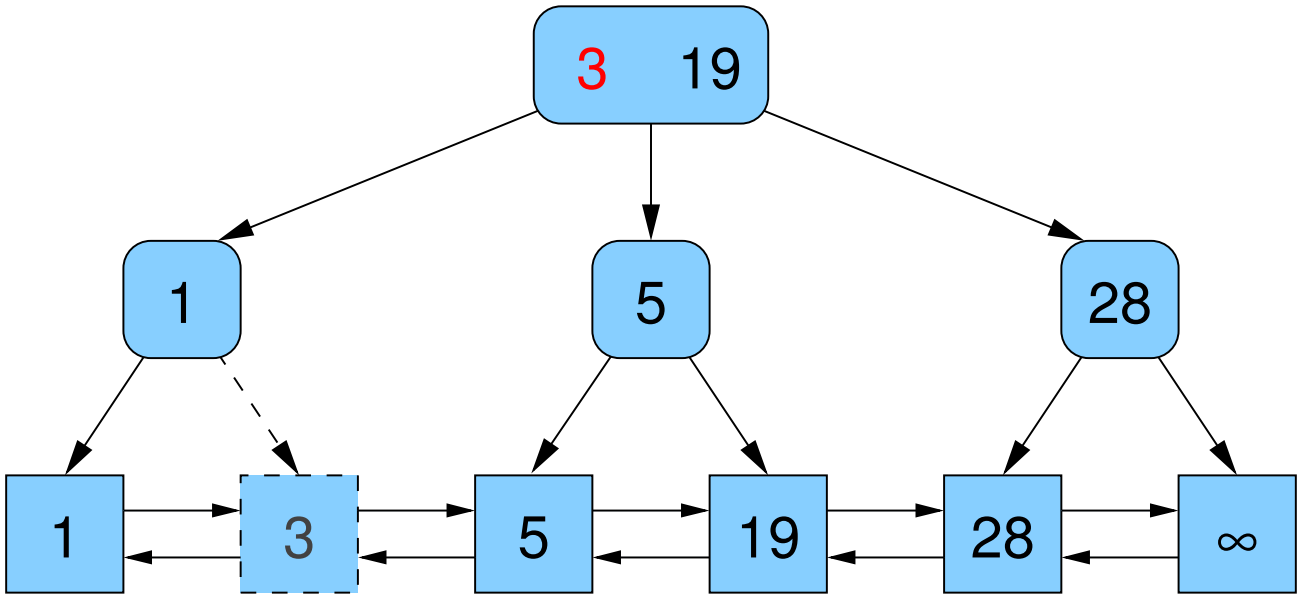
remove(14)



(a,b)-Baum / remove

$a = 2, b = 4$

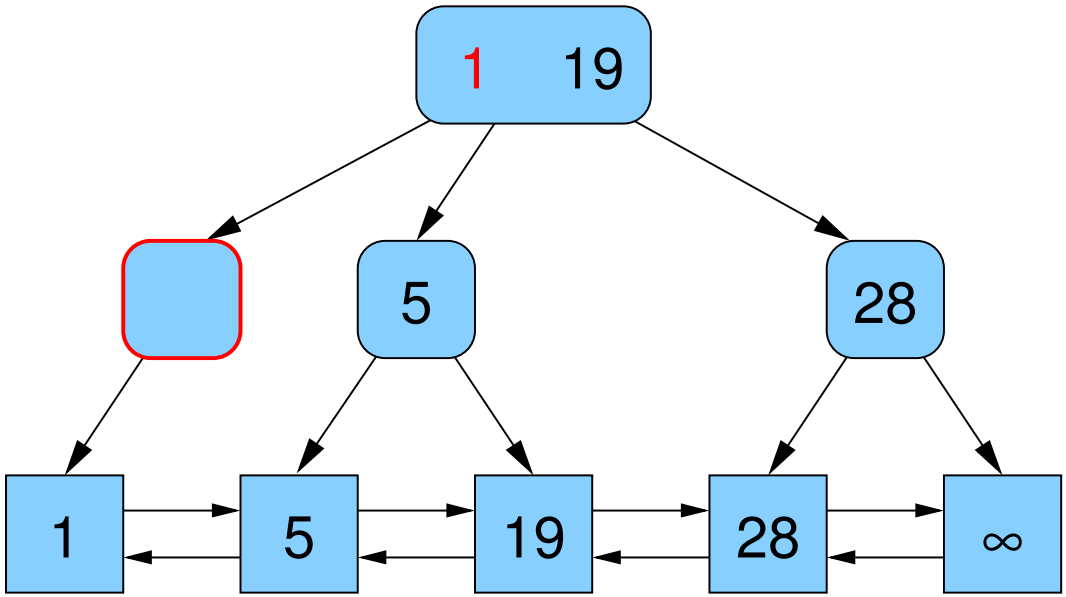
remove(3)



(a,b)-Baum / remove

$a = 2, b = 4$

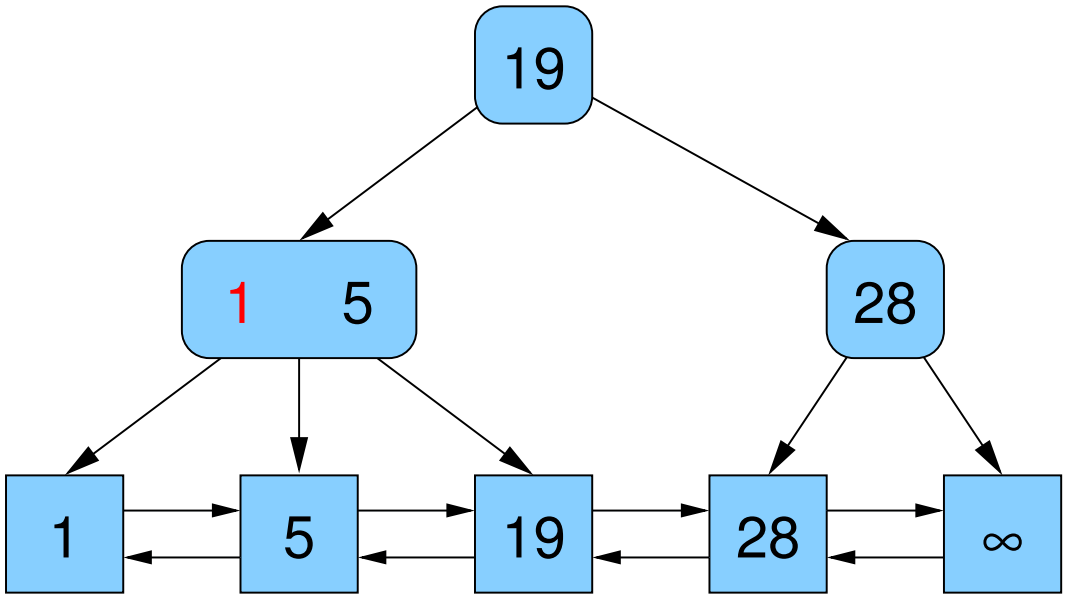
remove(3)



(a,b)-Baum / remove

$a = 2, b = 4$

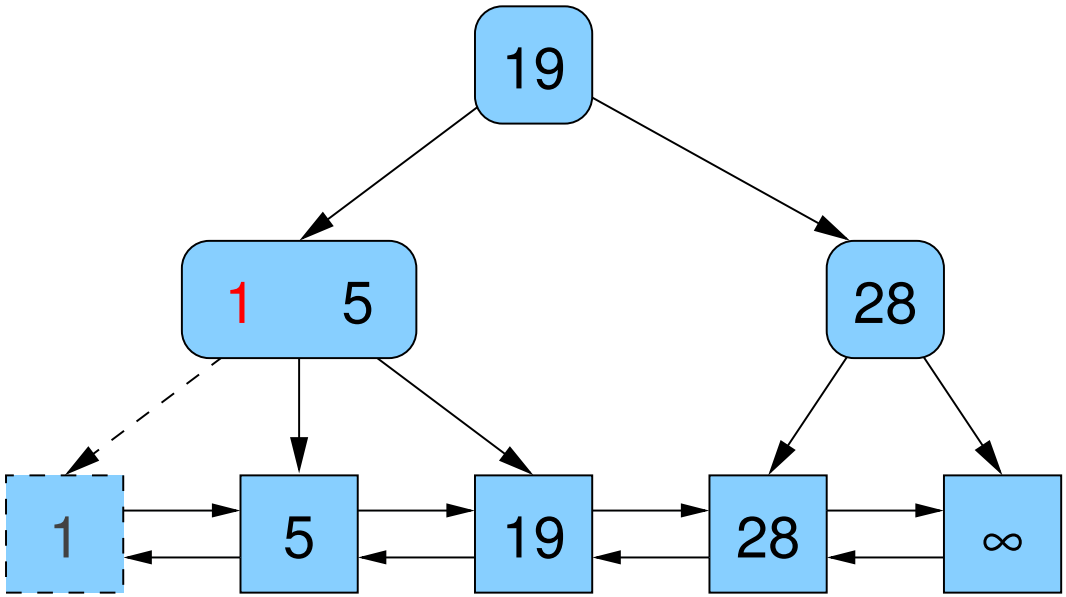
remove(3)



(a,b)-Baum / remove

$a = 2, b = 4$

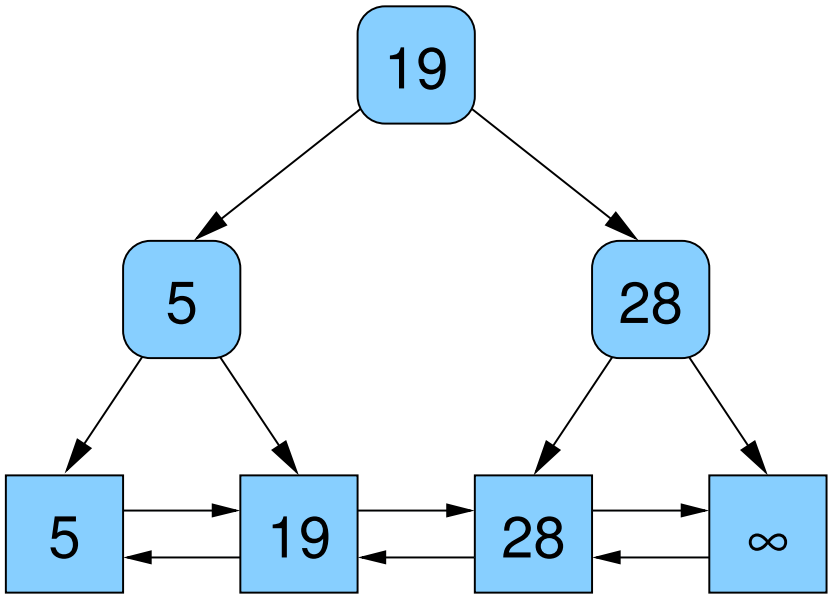
remove(1)



(a,b)-Baum / remove

$a = 2, b = 4$

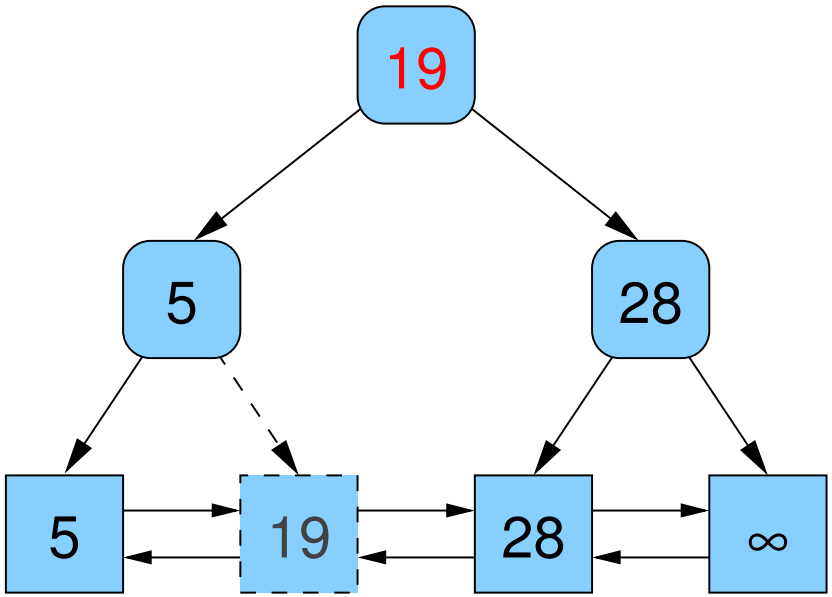
remove(1)



(a,b)-Baum / remove

$a = 2, b = 4$

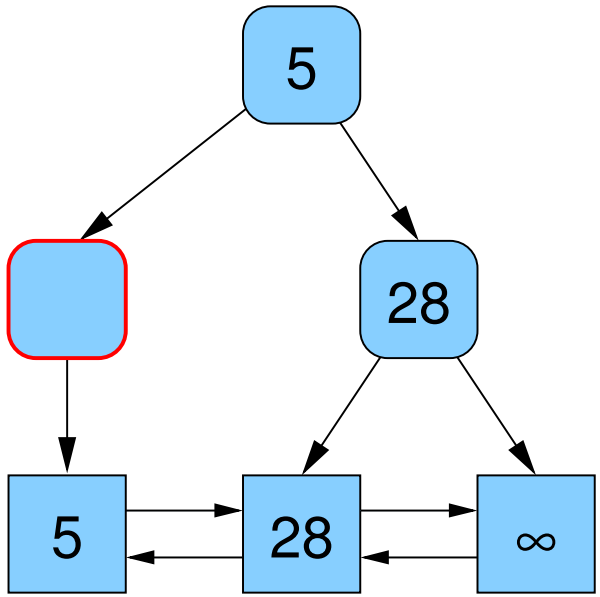
remove(19)



(a,b)-Baum / remove

$a = 2, b = 4$

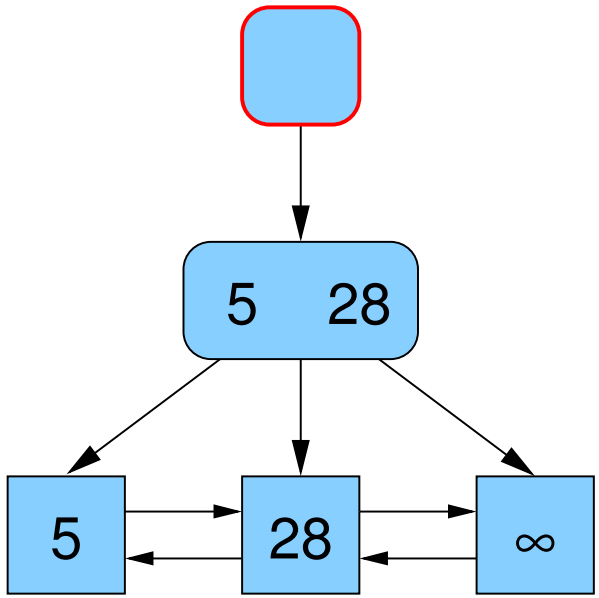
remove(19)



(a,b)-Baum / remove

$a = 2, b = 4$

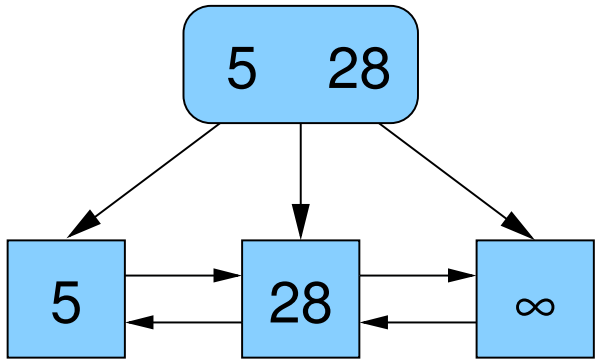
remove(19)



(a,b)-Baum / remove

$a = 2, b = 4$

remove(19)



(a,b)-Baum / remove

Form-Invariante

- durch remove erfüllt: alle Blätter haben dieselbe Tiefe

Grad-Invariante

- remove verschmilzt Knoten, die Grad $a-1$ und a haben
- wenn $b \geq 2a-1$, dann ist der resultierende Grad $\leq b$
- remove verschiebt eine Kante von Knoten mit Grad $> a$ zu Knoten mit Grad $a-1$, danach sind beide Grade in $[a,b]$
- wenn Wurzel gelöscht, wurden vorher die Kinder verschmolzen, Grad vom letzten Kind ist also $\geq a$ (und $\leq b$)

(a,b)-Baum

Weitere Operationen:

- min / max-Operation

  verwende first / last-Methode der Liste, um das kleinste bzw. größte Element auszugeben

  Zeit: $\mathcal{O}(1)$

- Bereichsanfragen

  suche alle Elemente im Bereich $[x,y]$:

  - führe locate(x) aus und
  - durchlaufe die Liste, bis Element $> y$ gefunden wird

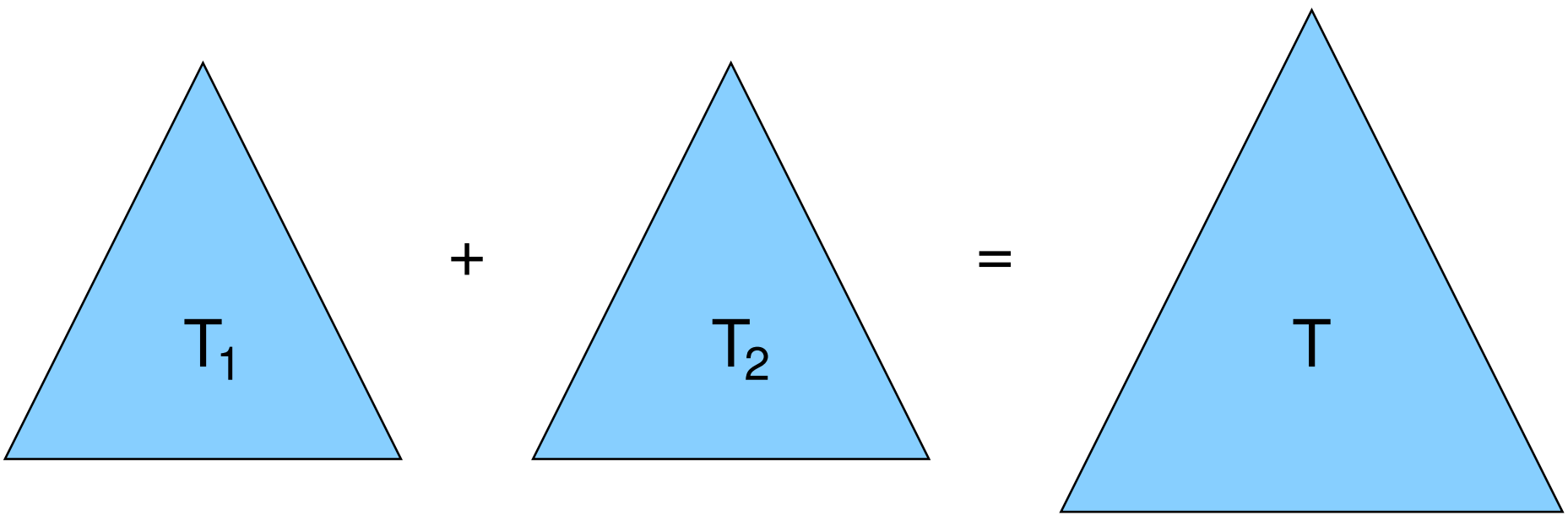
  Zeit: $\mathcal{O}(\log n + \text{Ausgabegröße})$

(a,b)-Baum

Weitere Operationen:

- Konkatenation

  verknüpfe zwei (a,b)-Bäume T_1 und T_2 mit s_1 und s_2 Elementen zu (a,b)-Baum T
  (Schlüssel in $T_1 \leq$ Schlüssel in T_2)



(a,b)-Baum

Weitere Operationen:

Konkatenation

- lösche in T_1 das ∞-Dummy-Element
- wenn danach dessen Vater-Knoten $< a$ Kinder hat, behandle dies wie bei remove
- verschmelze die Wurzel des niedrigeren Baums mit einem entsprechenden äußersten Knoten des anderen Baums, der sich auf dem gleichen Level befindet
- wenn dieser Knoten danach $> b$ Kinder hat, behandle dies wie bei insert
- Zeit: $\mathcal{O}(\log \max\{s_1, s_2\}) \in \mathcal{O}(s_1 + s_2)$
- wenn man die Höhe der Bäume explizit speichert, kann man concatenate sogar mit Zeit $\mathcal{O}(1 + |h_1 - h_2|)$ implementieren