Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Wiederholung Splay-Baum

- Umstrukturierung bei der Suche
  - Search, insert, delete sind effizient (O(log n))
- Statische Optimalität
  - Für geg. Anfragesequenz ist Splay-Baum-Zugriffszeit aymptotisch gleich zu der Zugriffszeit mit optimalem Baum bzgl. a priori gegebener Gewichte der Knoten
- Dynamische Optimalität?
  - Gilt das auch bzgl. der besten dynamischen Baumstruktur?
  - Offenes Problem

Rot-Schwarz-Baum

Rot-Schwarz-Bäume sind binäre Suchbäume mit roten und schwarzen Knoten, so dass gilt:

- Wurzeleigenschaft: Die Wurzel ist schwarz.
- Externe Eigenschaft: Jeder Listenknoten ist schwarz.
- Interne Eigenschaft: Die Kinder eines roten Knotens sind schwarz.
- Tiefeneigenschaft: Alle Listenknoten haben dieselbe "Schwarztiefe"

"Schwarztiefe" eines Knotens: Anzahl der schwarzen Baumknoten (außer der Wurzel) auf dem Pfad von der Wurzel zu diesem Knoten

Es gibt keine direkten Kanten zu Listenelementen

Leonidas J. Guibas and Robert Sedgewick, A Dichromatic Framework for Balanced Trees, In: Proceedings of the 19th Annual Symposium on Foundations of Computer Science, S. 8–21, **1978**

Danksagung

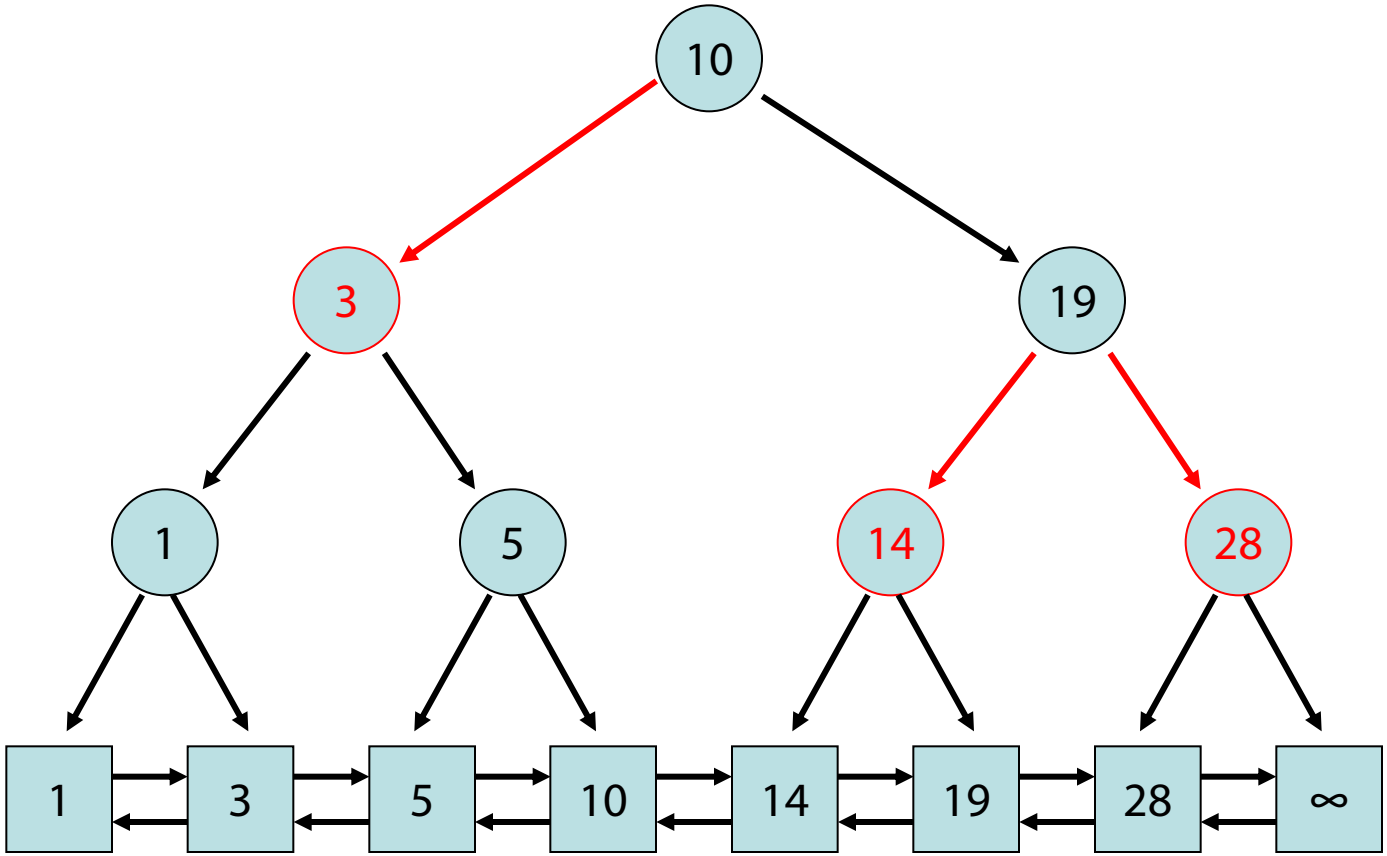
Die nachfolgenden Präsentationen wurden mit einigen Änderungen übernommen aus:

- „Effiziente Algorithmen und Datenstrukturen“ (Kapitel 2: Suchstrukturen) gehalten von Christian Scheideler an der TUM
  http://www14.in.tum.de/lehre/2008WS/ea/index.html.de

- „Algorithmen und Datenstrukturen“
  gehalten von Sven Groppe an der UzL

Rot-Schwarz-Baum

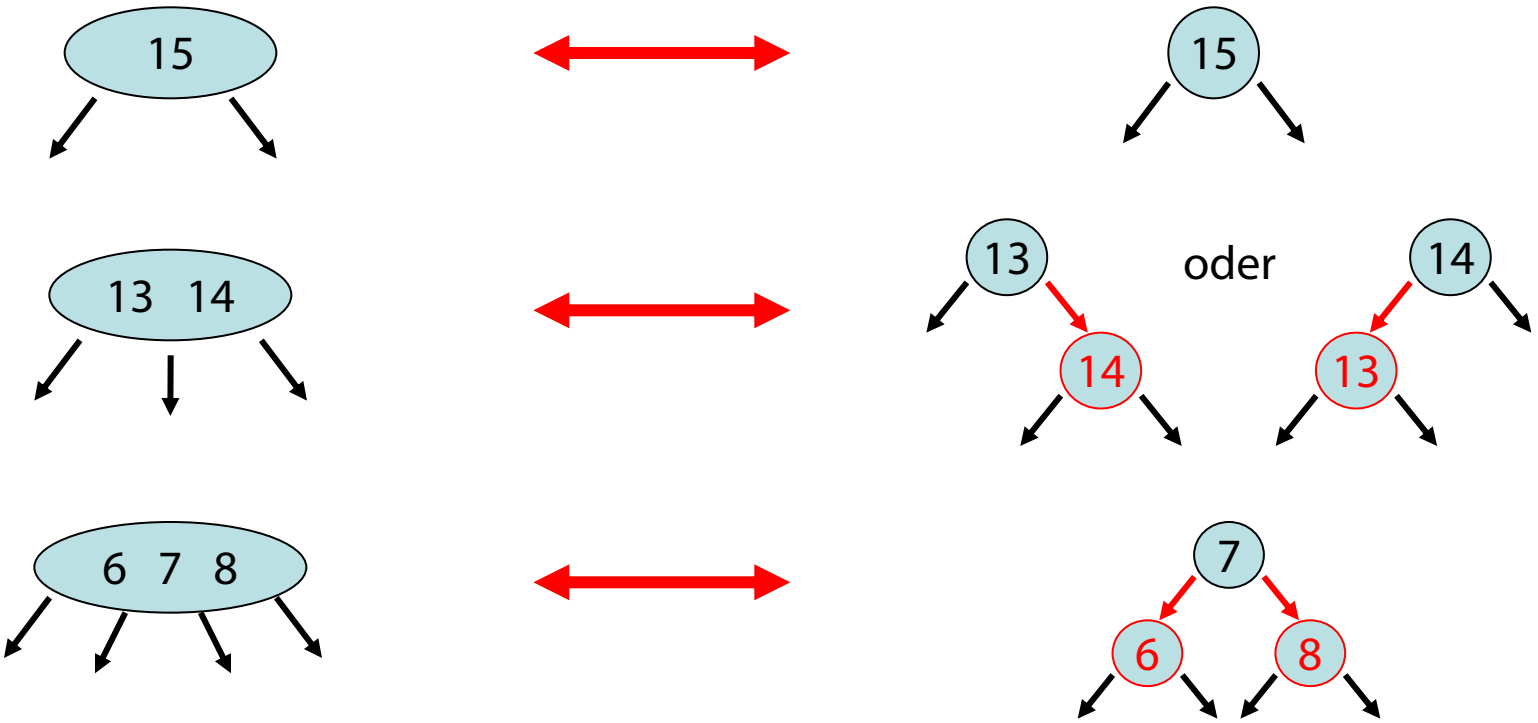
Beispiel:



Rot-Schwarz-Baum

Andere Deutung von Rot-schwarz-Mustern:

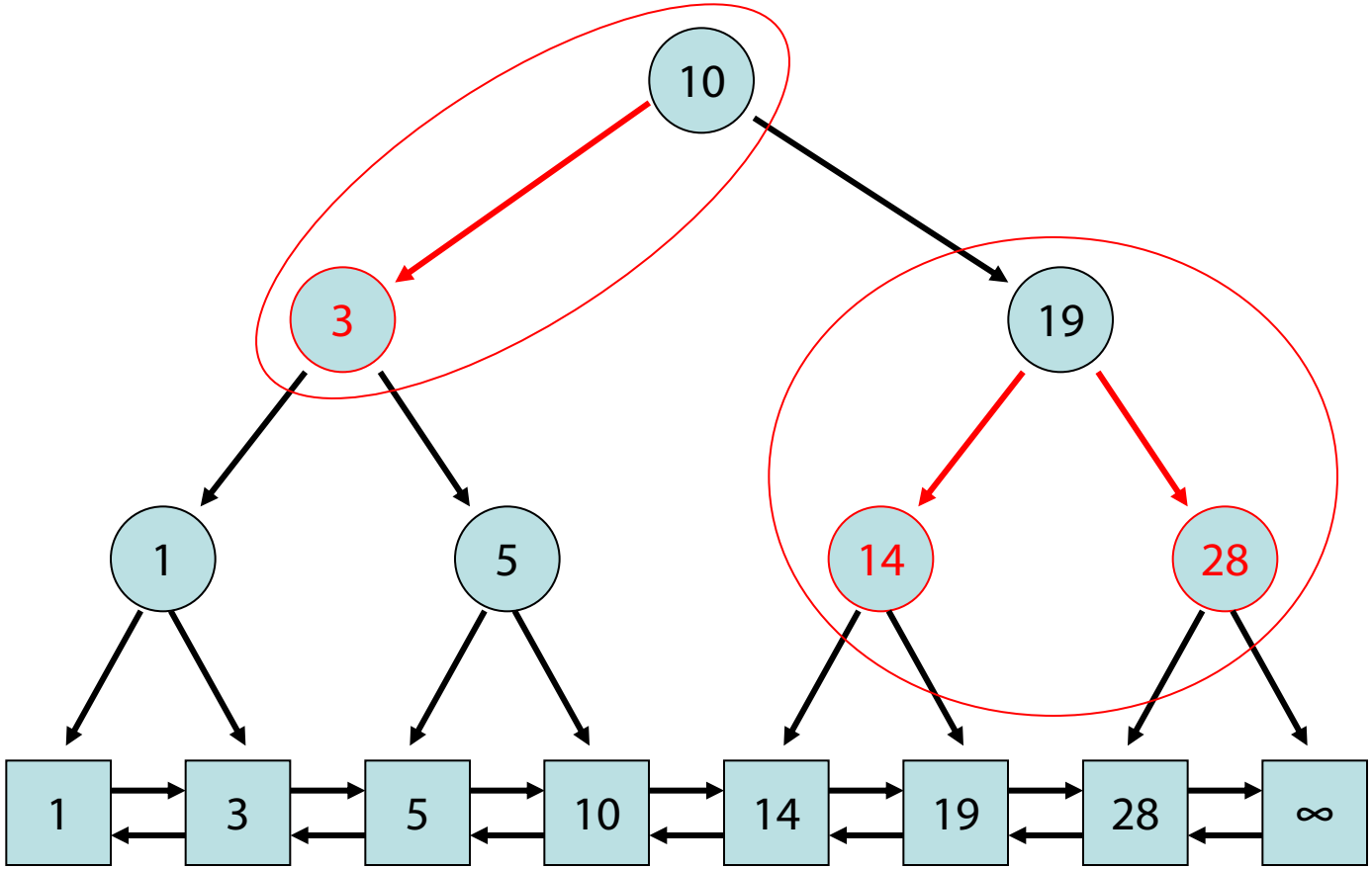
B-Baum (Baum mit “Separatoren” und min 2 max 4 Nachfolger)



Rudolf **B**ayer, Symmetric binary **B**-Trees: Data structure and maintenance algorithms. Acta Informatica 1 (4), S. 290–306, **1972**

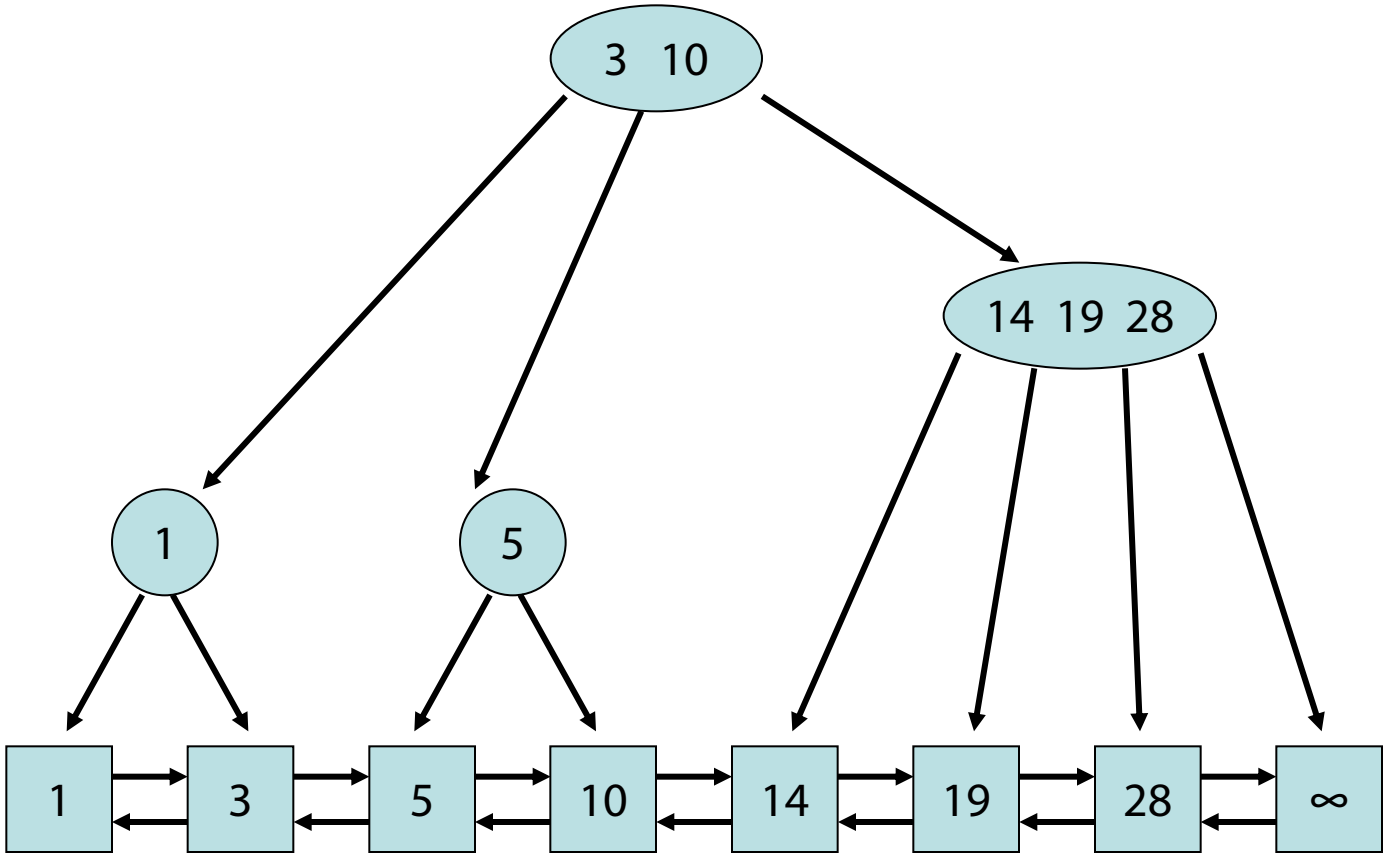
Rot-Schwarz-Baum

R-S-Baum:



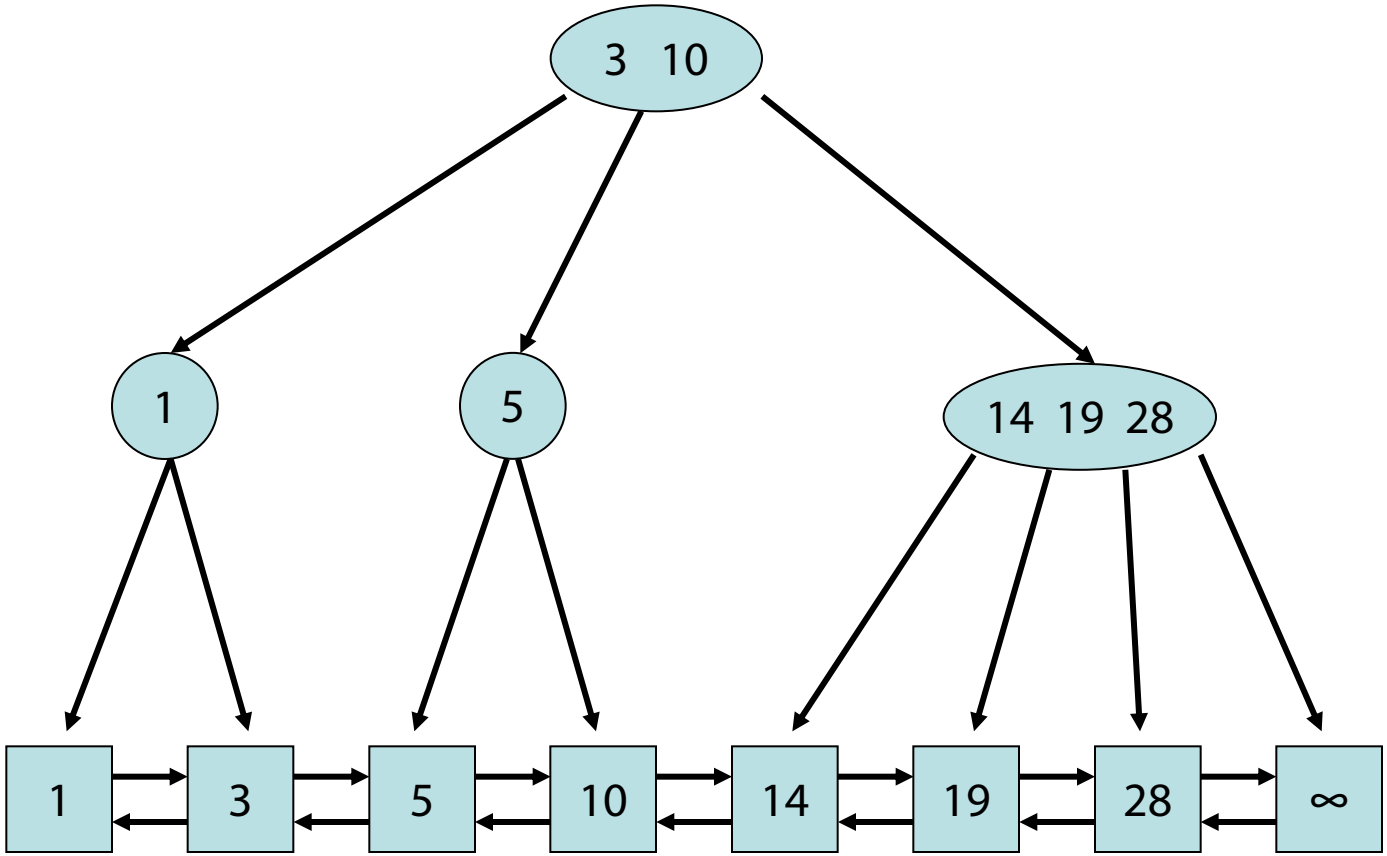
Rot-Schwarz-Baum

B-Baum:



Rot-Schwarz-Baum

B-Baum:



Rot-Schwarz-Baum

Behauptung: Die Tiefe t eines Rot-Schwarz-Baums T mit n Elementen ist in $\Theta(\log n)$. Also: Der Rot-Schwarz-Baum T ist ausgeglichen.

Beweis:

Wir zeigen: $\log(n+1) \leq t \leq 2\log(n+1)$ für die Tiefe t des Rot-Schwarz-Baums mit n Elementen.

- d: Schwarztiefe der Listenknoten
- T': B-Baum zu T
- T' hat Tiefe exakt d überall und $d \leq \log(n+1)$
- Aufgrund der internen Eigenschaft (Kinder eines roten Knotens sind schwarz) gilt: $t \leq 2d$
- Außerdem ist $t \geq \log(n+1)$, da ein Rot-Schwarz-Baum ein Binärbaum ist und rote "dazwischen" sind.

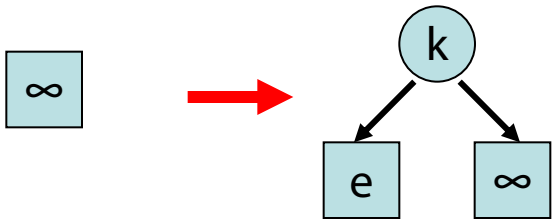
Rot-Schwarz-Baum

search(k): wie im binären Suchbaum

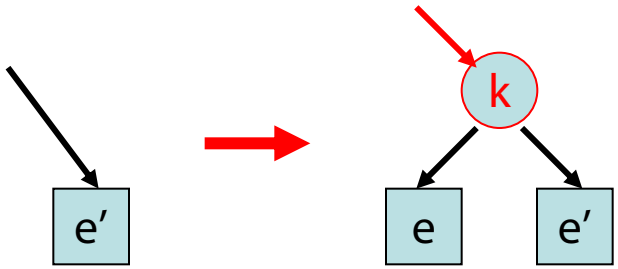
insert(e):

- Führe search(k) mit k=key(e) aus
- Füge e vor Nachfolger e' in Liste ein

Fall 1: Baum leer
(nur Markerelement ∞ enthalten)

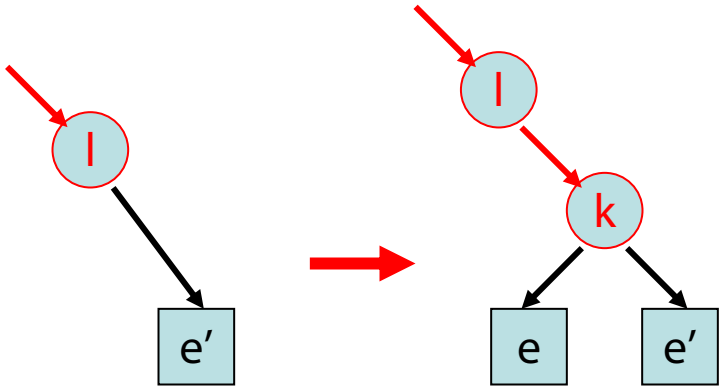


Fall 2: Baum nicht leer



Problem

- Knoten e' kann schon roten Vorgänger haben



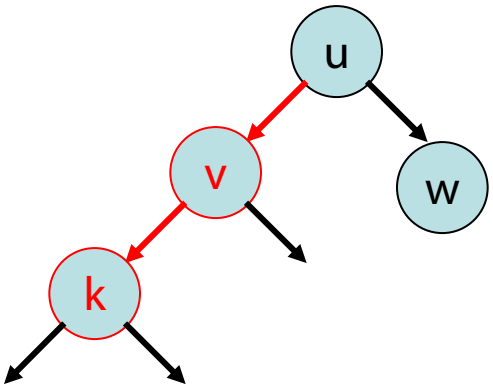
Rot-Schwarz-Baum

insert(e):

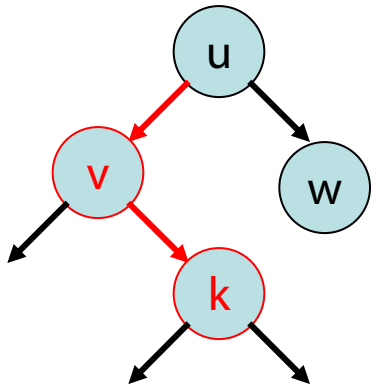
- Führe search(k) mit k=key(e) aus
- Füge e vor Nachfolger e' in Liste ein
  (bewahrt alles bis auf evtl. interne Eigenschaft)
- Interne Eigenschaft verletzt (Fall 2 vorher): 2 Fälle
  - Fall 1: Vater von k in T hat **schwarzen Bruder**
    (Restrukturierung, aber beendet Reparatur)
  - Fall 2: Vater von k in T hat **roten Bruder**
    (setzt Reparatur nach oben fort, aber keine Restrukturierung)

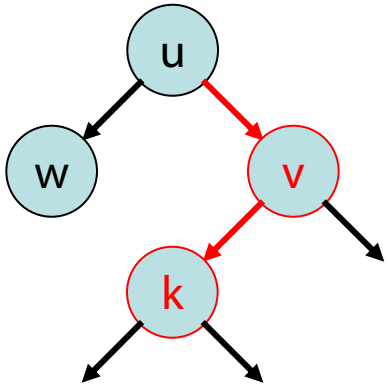
Rot-Schwarz-Baum

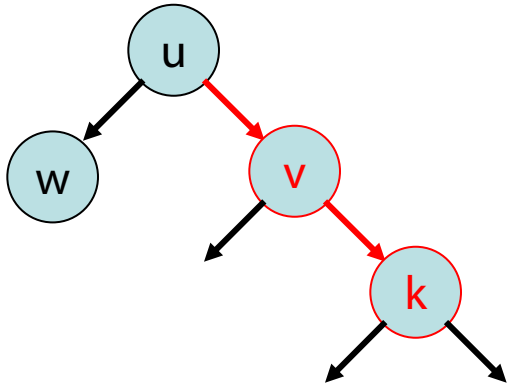
Fall 1: Vater v von k in T hat schwarzen Bruder w



Alternativen für Fall 1



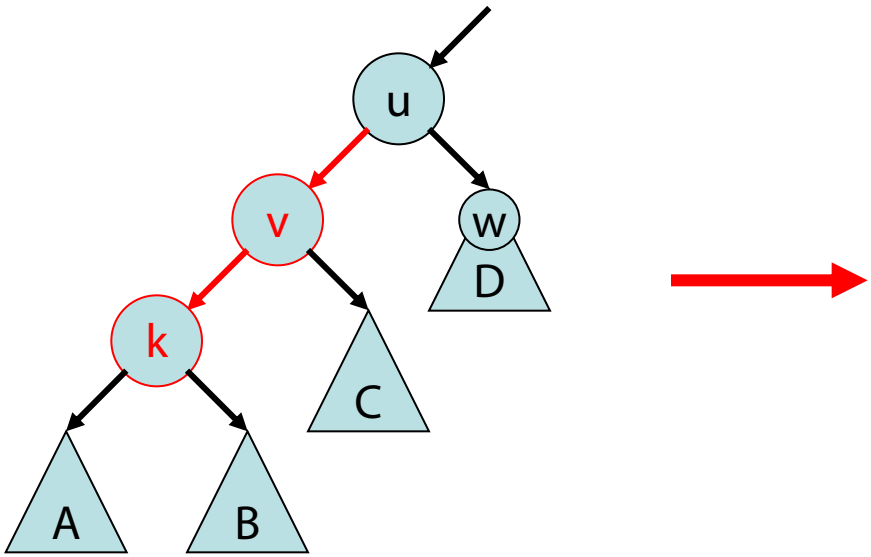




Rot-Schwarz-Baum

Fall 1: Vater v von k in T hat schwarzen Bruder w

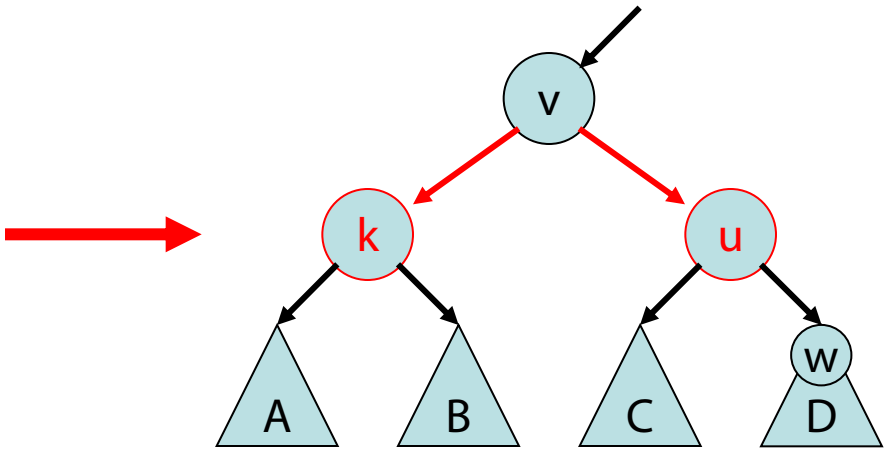
Lösung:



Rot-Schwarz-Baum

Fall 1: Vater v von k in T hat schwarzen Bruder w

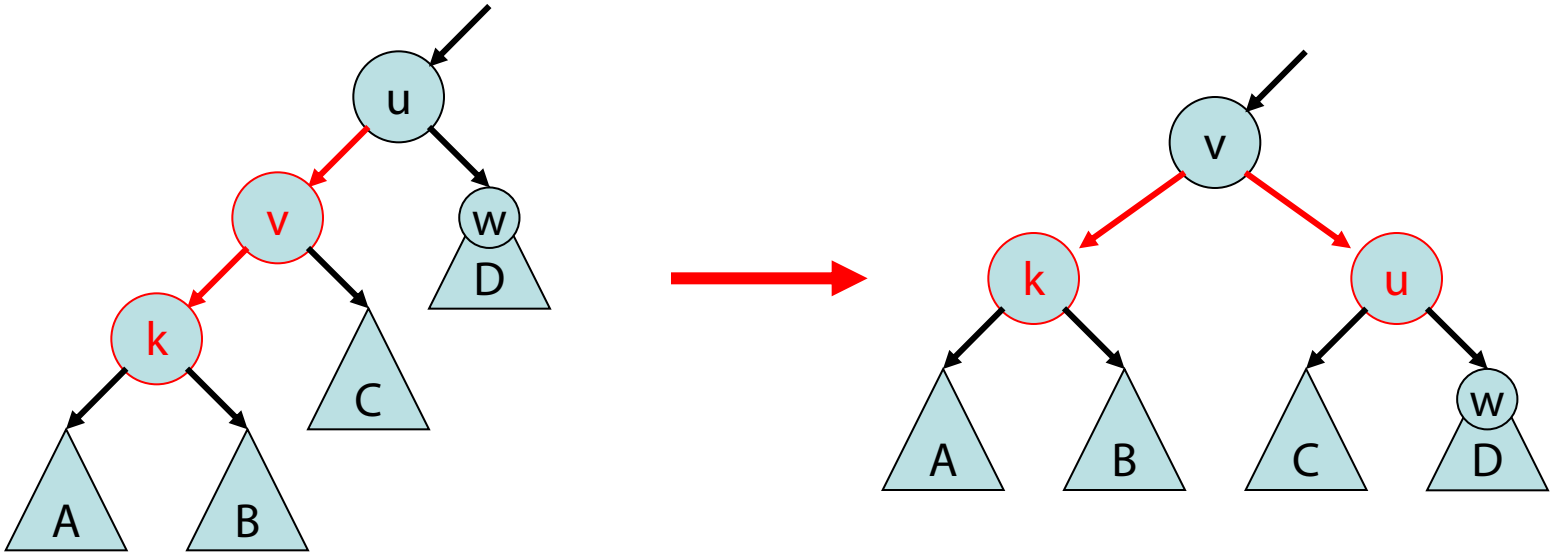
Lösung:



Rot-Schwarz-Baum

Fall 1: Vater v von k in T hat schwarzen Bruder w

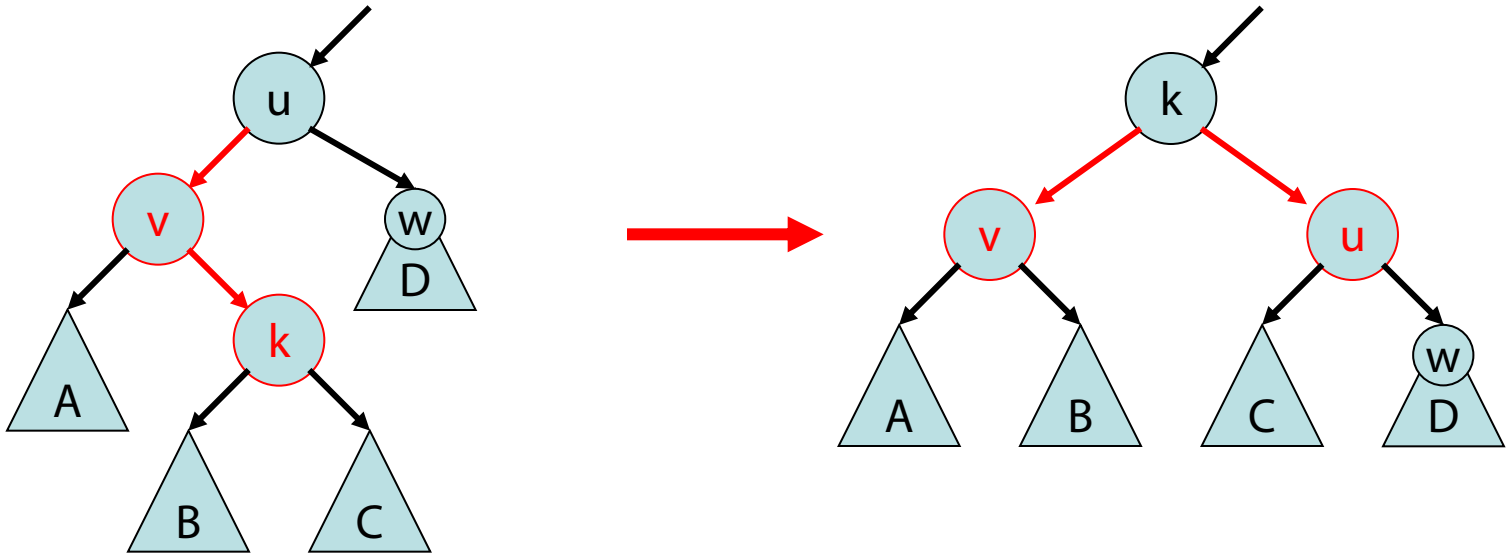
Lösung:



Rot-Schwarz-Baum

Fall 1: Vater v von k in T hat schwarzen Bruder w

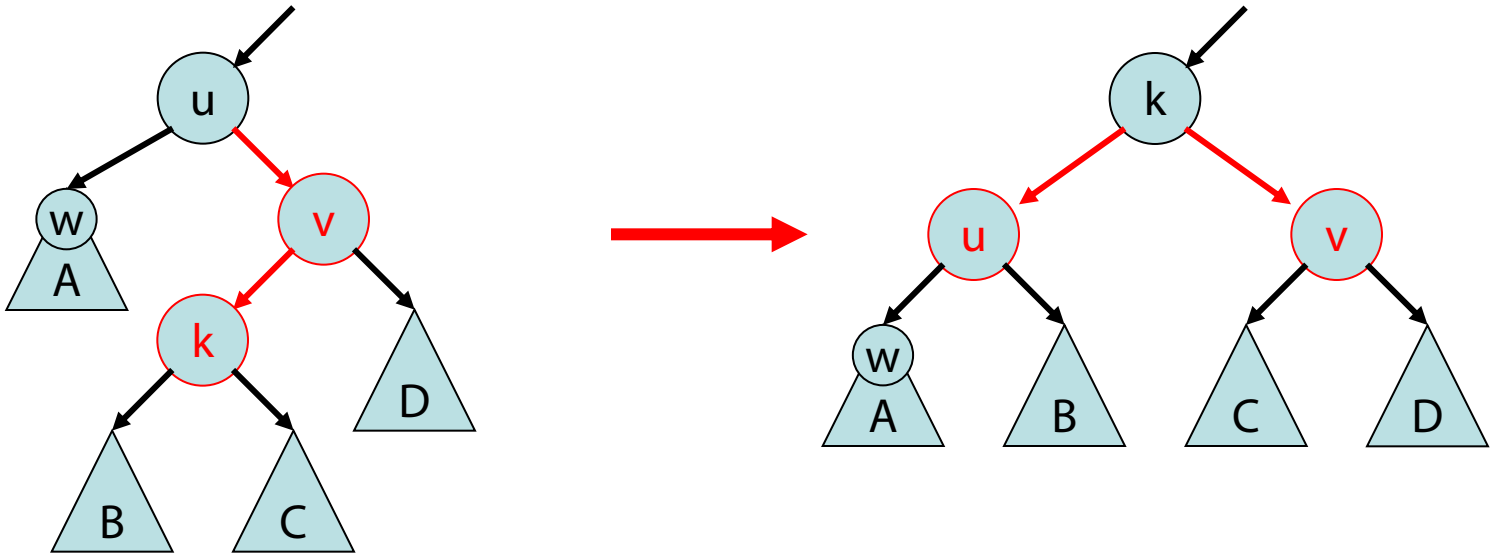
Lösung:



Rot-Schwarz-Baum

Fall 1: Vater v von k in T hat schwarzen Bruder w

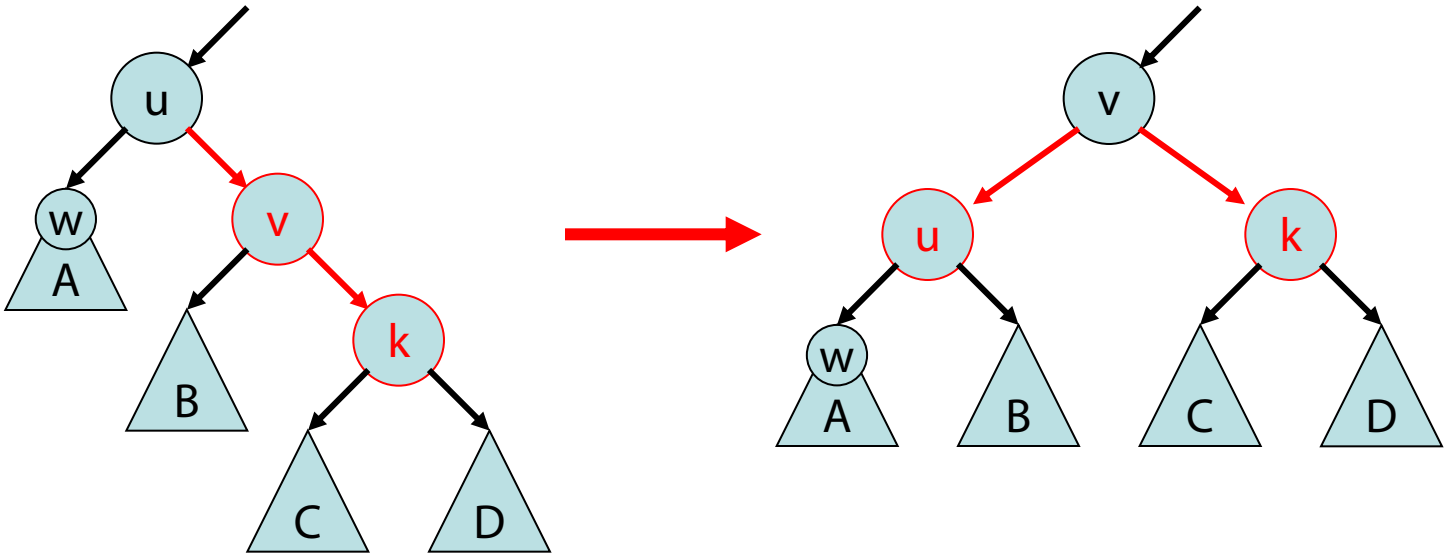
Lösung:



Rot-Schwarz-Baum

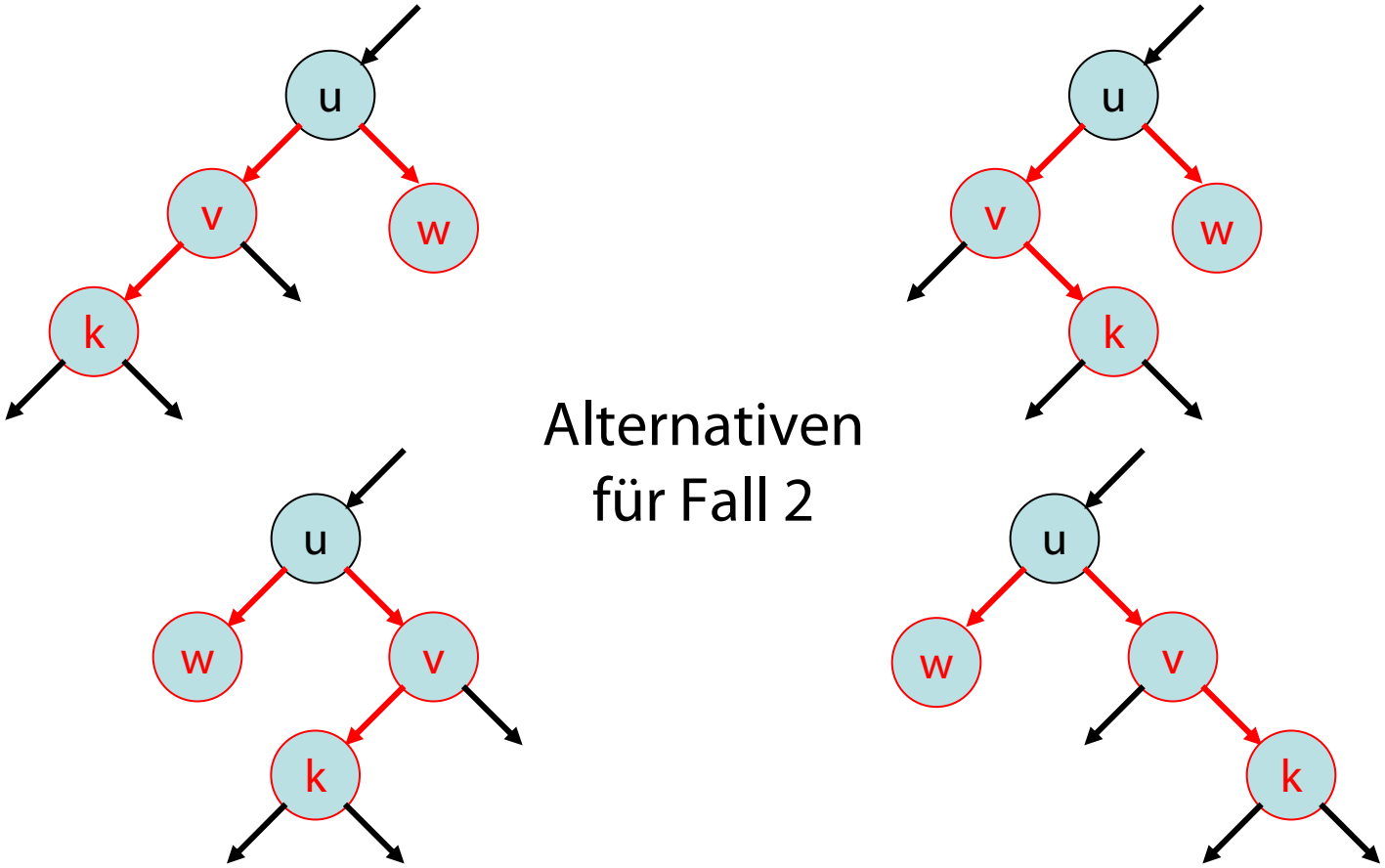
Fall 1: Vater v von k in T hat schwarzen Bruder w

Lösung:



Rot-Schwarz-Baum

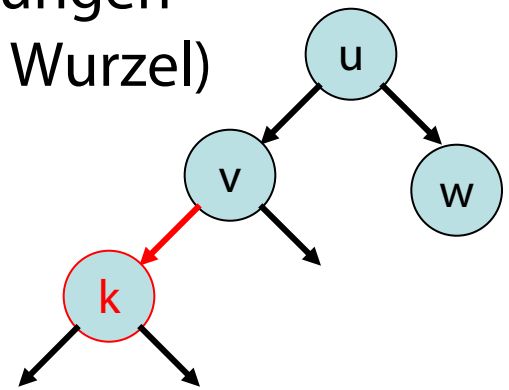
Fall 2: Vater v von k in T hat roten Bruder w



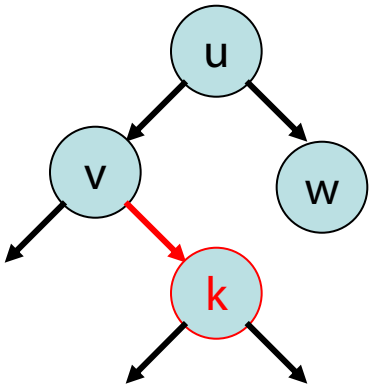
Rot-Schwarz-Baum

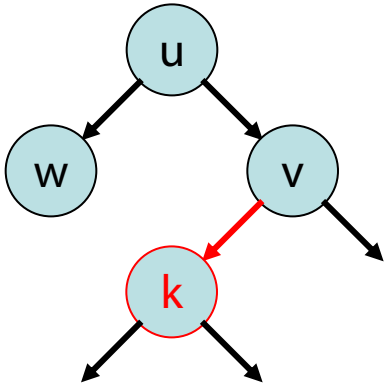
Fall 2: Vater v von k in T hat roten Bruder w

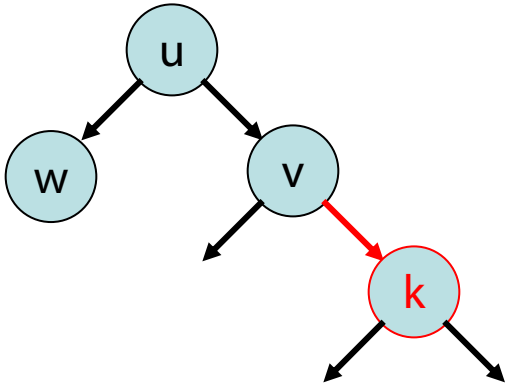
Lösungen
(u ist Wurzel)



Schwarztiefe+1



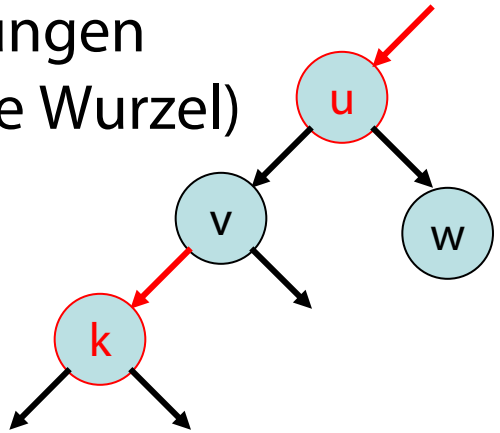




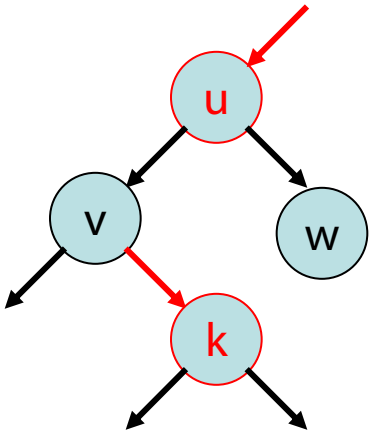
Rot-Schwarz-Baum

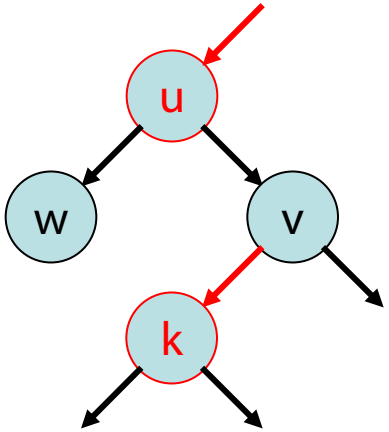
Fall 2: Vater v von k in T hat roten Bruder w

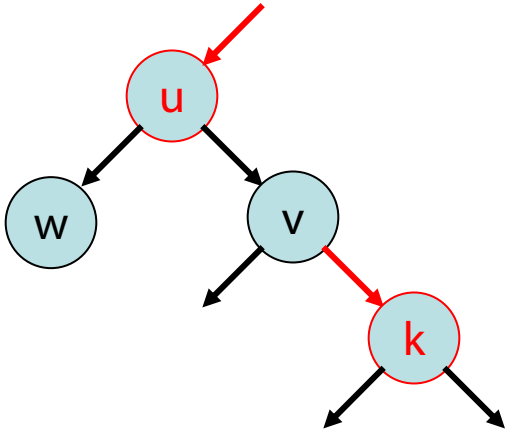
Lösungen
(u keine Wurzel)



bewahrt
Schwarztiefe!

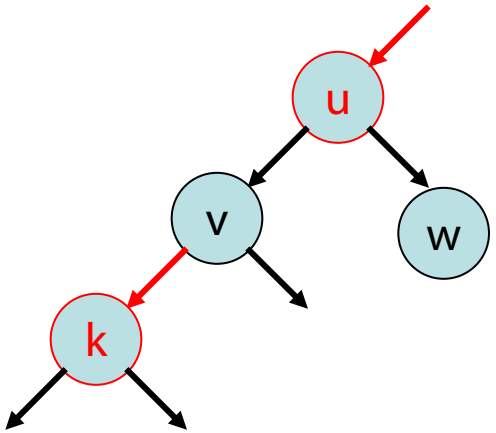




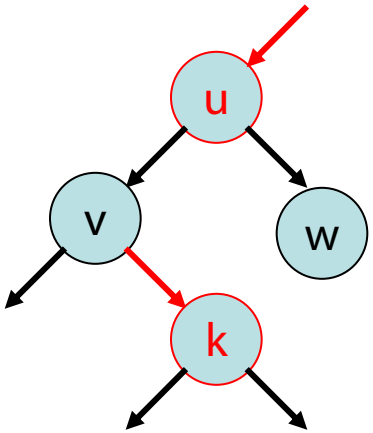


Rot-Schwarz-Baum

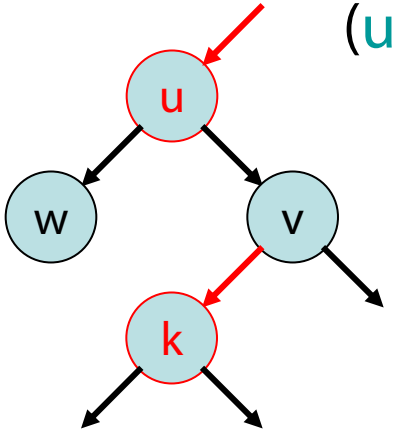
Fall 2: Vater v von k in T hat roten Bruder w

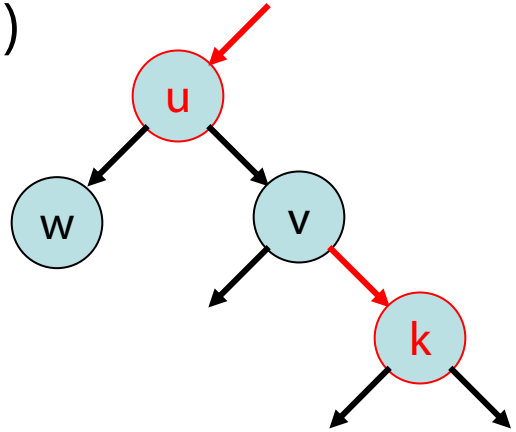


weiter mit u
wie mit k



Lösungen
(u keine Wurzel)

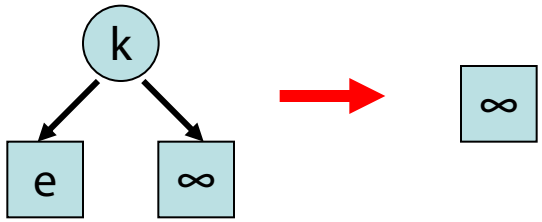




Rot-Schwarz-Baum

delete(k):

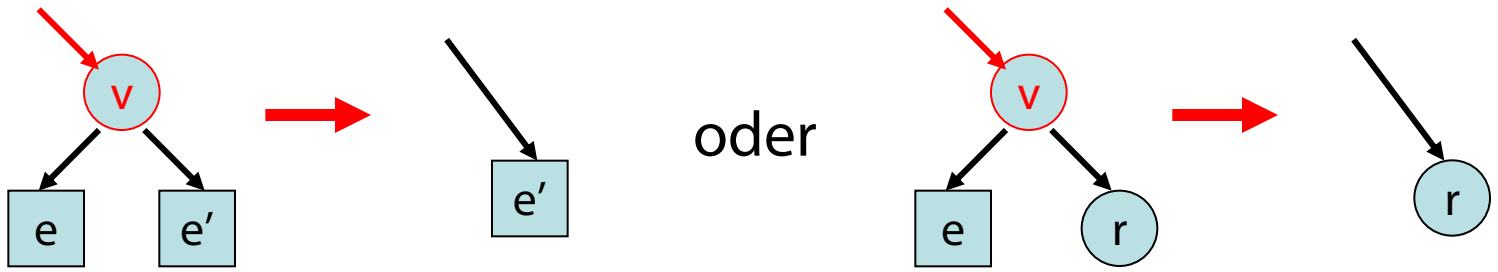
- Führe search(k) auf Baum aus
- Lösche Element e mit key(e)=k wie im binären Suchbaum
- Fall 1: Baum ist dann leer



Rot-Schwarz-Baum

delete(k):

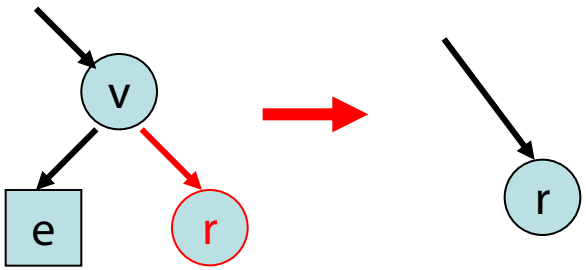
- Führe search(k) auf Baum aus
- Lösche Element e mit key(e)=k wie im binären Suchbaum
- Fall 2: Vater v von e ist rot (d.h. Bruder schwarz)



Rot-Schwarz-Baum

delete(k):

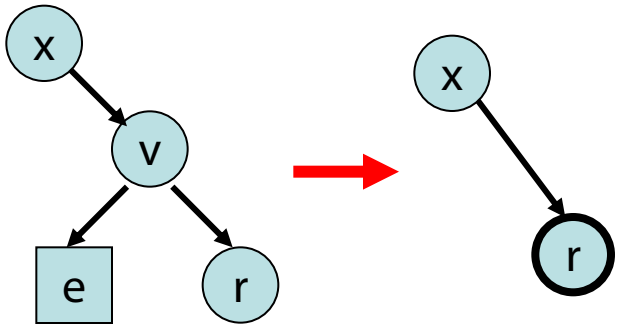
- Führe search(k) auf Baum aus
- Lösche Element e mit key(e)=k wie im binären Suchbaum
- Fall 3: Vater v von e ist schwarz und Bruder rot



Rot-Schwarz-Baum

delete(k):

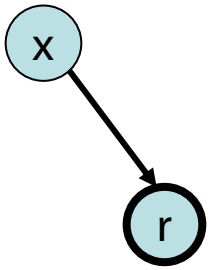
- Führe search(k) auf Baum aus
- Lösche Element e mit key(e)=k wie im binären Suchbaum
- Fall 4: Vater v von e und Bruder r sind schwarz



Tiefenregel verletzt!
r heißt dann doppelt schwarz

Rot-Schwarz-Baum

delete(k):



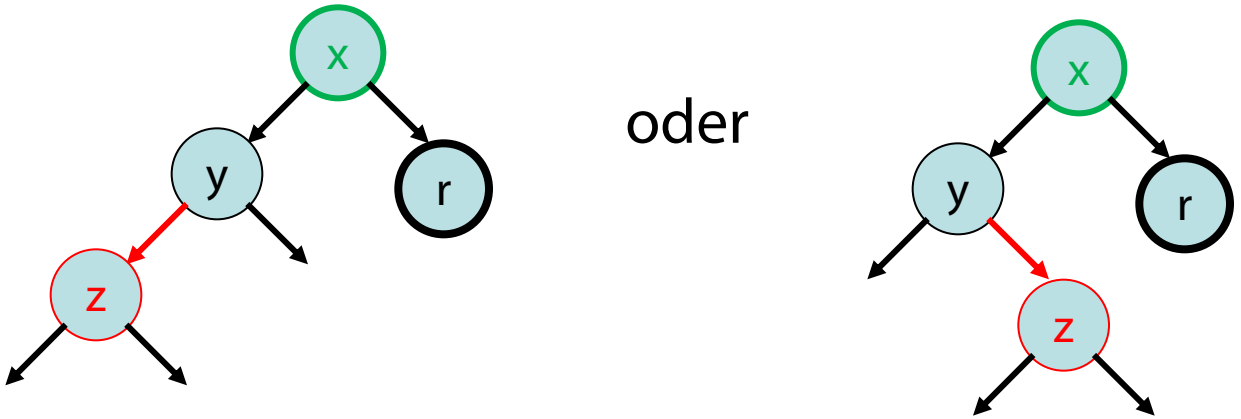
- Führe search(k) auf Baum aus
- Lösche Element e mit key(e)=k wie im binären Suchbaum
- Falls Vater v von e und Bruder r schwarz (s.o.), dann 3 weitere Fälle:
  - Fall 1: Bruder y von r ist schwarz und hat rotes Kind
  - Fall 2: Bruder y von r ist schwarz und beide Kinder von y sind schwarz (evtl. weiter, aber keine Restrukt.)
  - Fall 3: Bruder y von r ist rot

Rot-Schwarz-Baum

Fall 1: Bruder y von r ist schwarz, hat rotes Kind z

O.B.d.A. sei r rechtes Kind von x (links: analog)

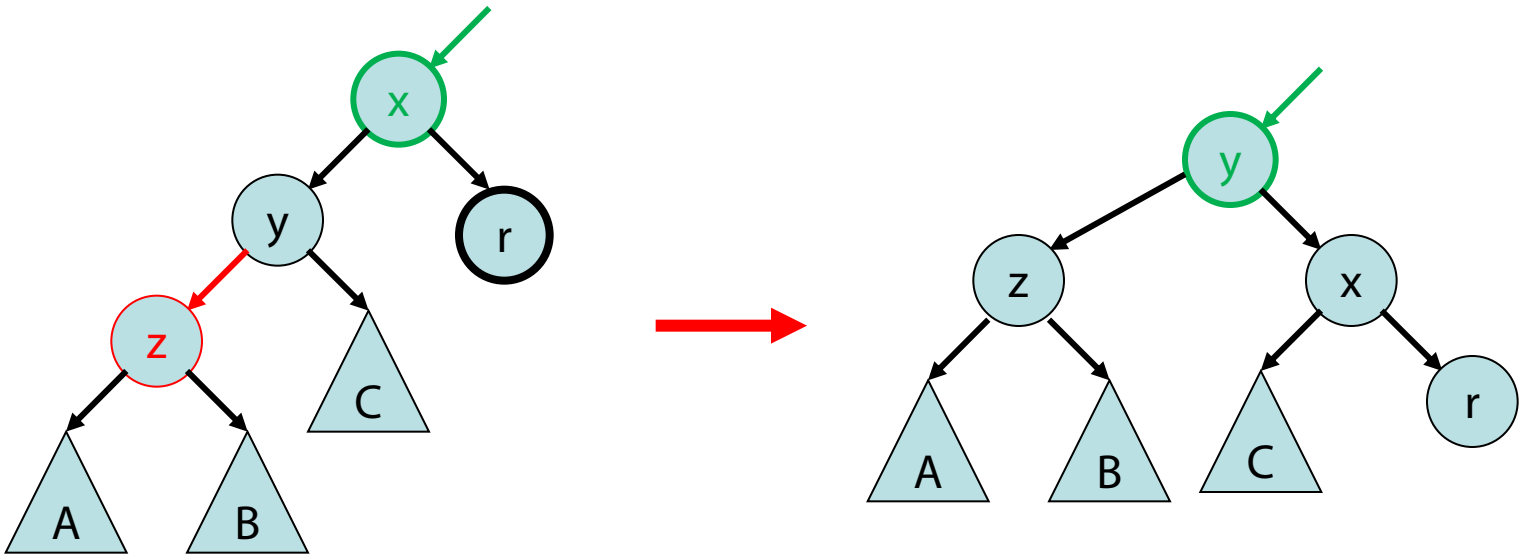
Alternativen für Fall 1: (x: beliebig gefärbt)



Rot-Schwarz-Baum

Fall 1: Bruder y von r ist schwarz, hat rotes Kind z

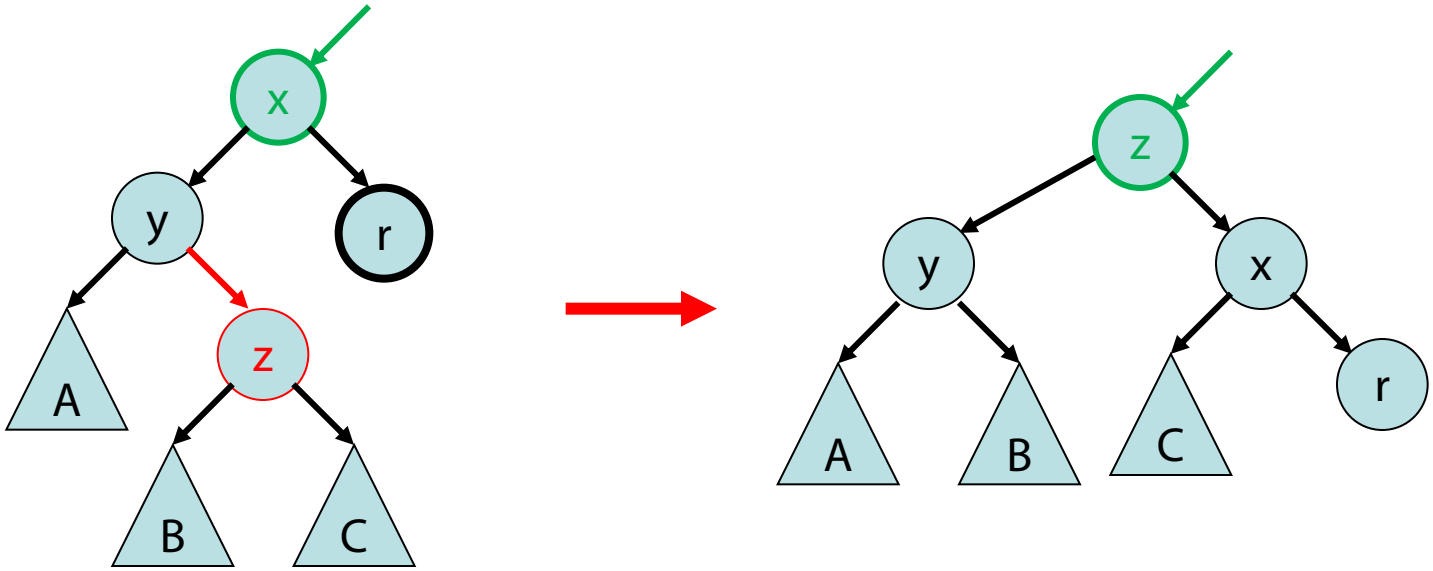
O.B.d.A. sei r rechtes Kind von x (links: analog)



Rot-Schwarz-Baum

Fall 1: Bruder y von r ist schwarz, hat rotes Kind z

O.B.d.A. sei r rechtes Kind von x (links: analog)



Rot-Schwarz-Baum

Fall 2: Bruder y von r ist schwarz und beide Kinder von y sind schwarz

O.B.d.A. sei r rechtes Kind von x (links: analog)

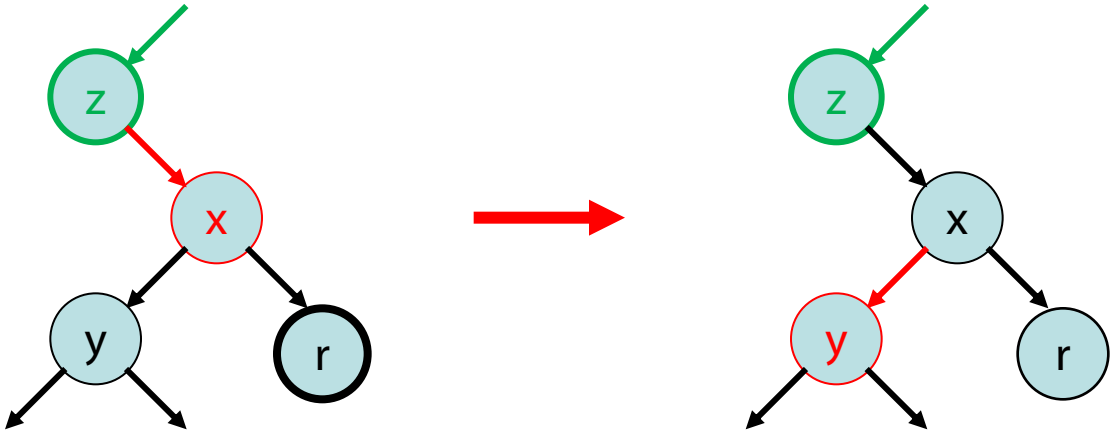
Alternativen für Fall 2: (z beliebig gefärbt)

Rot-Schwarz-Baum

Fall 2: Bruder y von r ist schwarz und beide Kinder von y sind schwarz

O.B.d.A. sei r rechtes Kind von x (links: analog)

2a)

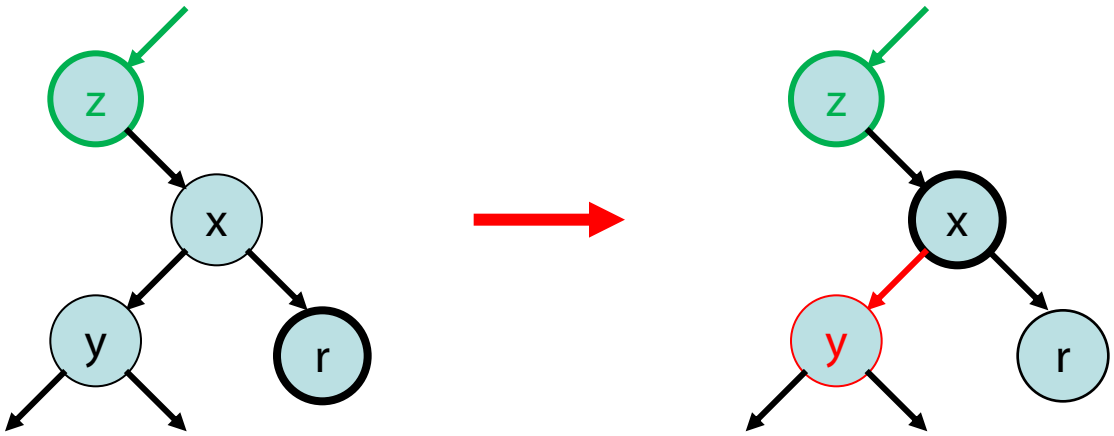


Rot-Schwarz-Baum

Fall 2: Bruder y von r ist schwarz und beide Kinder von y sind schwarz

O.B.d.A. sei r rechtes Kind von x (links: analog)

2b)



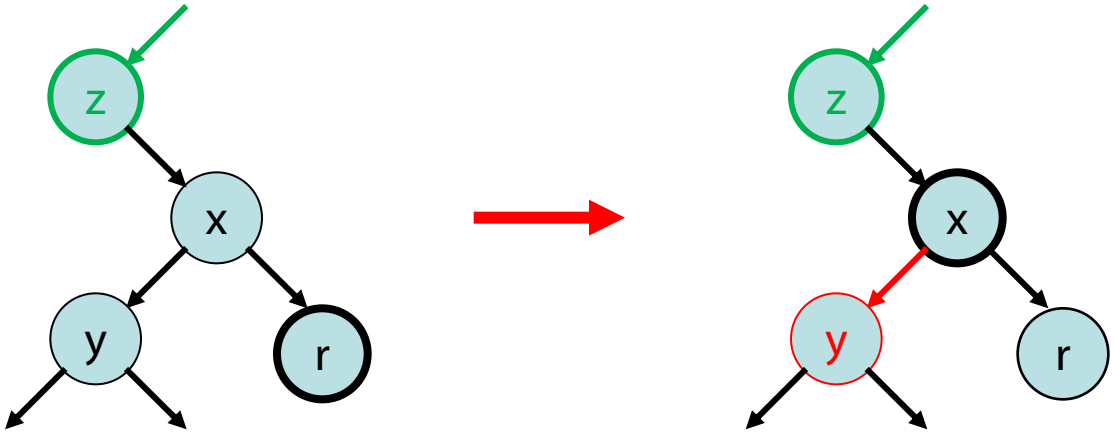
x ist Wurzel: fertig (Schwarztiefe-1)

Rot-Schwarz-Baum

Fall 2: Bruder y von r ist schwarz und beide Kinder von y sind schwarz

O.B.d.A. sei r rechtes Kind von x (links: analog)

2b)

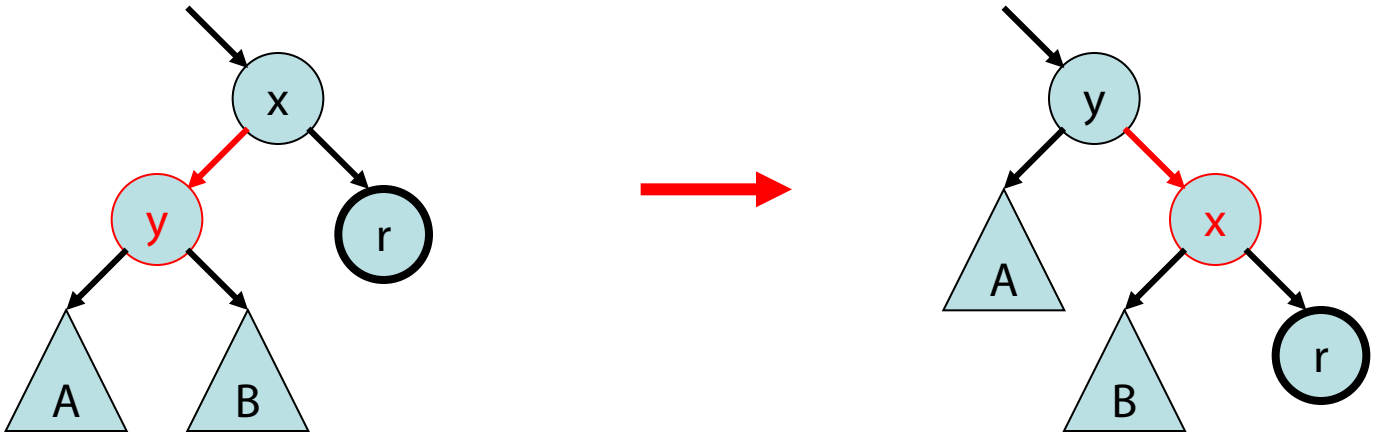


x keine Wurzel: weiter wie mit r

Rot-Schwarz-Baum

Fall 3: Bruder y von r ist rot

O.B.d.A. sei r rechtes Kind von x (links: analog)



Fall 1 oder 2a
→ terminiert dann

Rot-Schwarz-Baum

Laufzeiten der Operationen:

- search(k): O(log n)
- insert(e): O(log n)
- delete(k): O(log n)

Zum Vergleich:

Splay-Bäume

- search: O(log n) amort.
- insert: O(log n)
- delete: O(log n)

Restrukturierungen (Drehoperationen)

- insert(e): max. 1
- delete(k): max. 2