Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

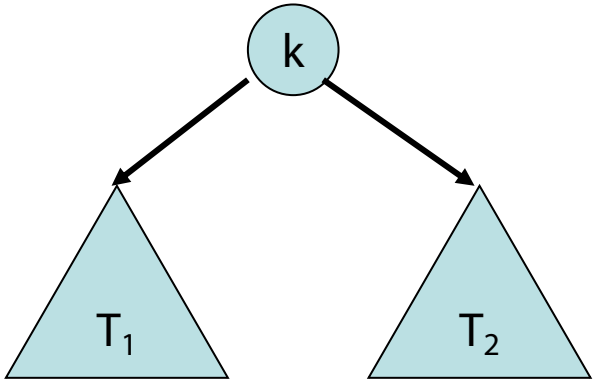
Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Binärer Suchbaum

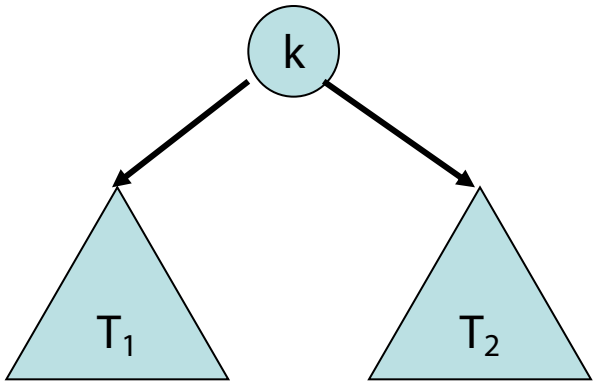
Suchbaum-Regel:



Für alle Schlüssel k' in T_1 und k'' in T_2: $k' \leq k < k''$

- Damit lässt sich die search Operation für Mengen einfach implementieren.

search(k) Operation



Für alle Schlüssel k' in T_1 und k'' in T_2: $k' \leq k < k''$

Suchstrategie:

- Starte in Wurzel des Suchbaums
- Für jeden erreichten Knoten v:
  - Falls key(v) ≤ k, gehe zum linken Kind von v, sonst gehe zum rechten Kind

Binärer Suchbaum als Navigationsstruktur

Für einen Baumknoten v sei

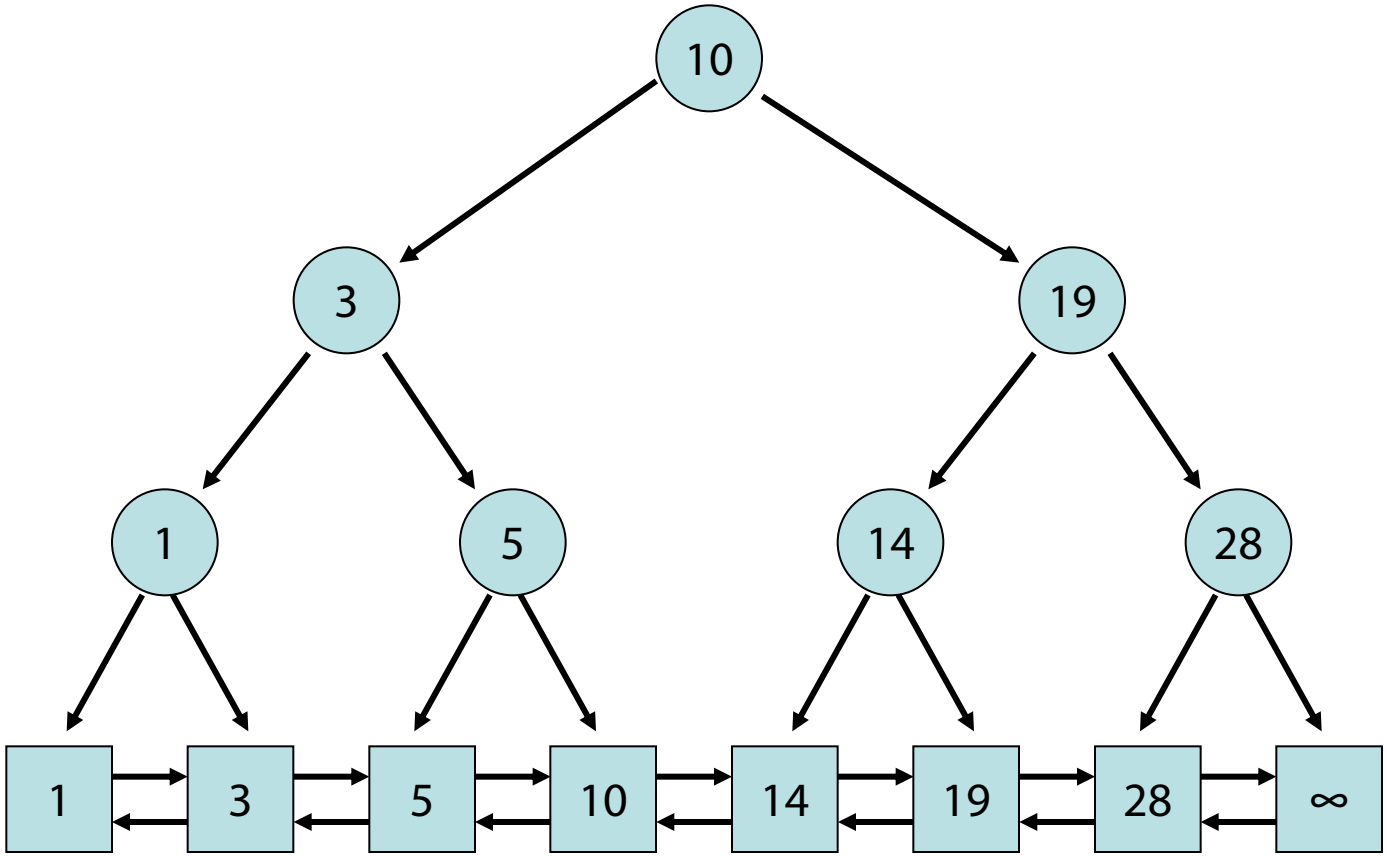
- key(v) der Schlüssel in v
- d(v) die Anzahl Kinder von v

- Suchbaum-Regel: (s.o.)

- Grad-Regel: (Grad = Anzahl der Kinder)
  Alle Baumknoten v haben zwei Kinder: d(v) = 2
  (wenn #Elemente =0 keine Baumknoten vorhanden)
  Unten sind die Knoten der verketteten Liste.

- Schlüssel-Regel:
  Für jedes Element e in der Liste gibt es genau einen Baumknoten v mit key(v)=key(e).

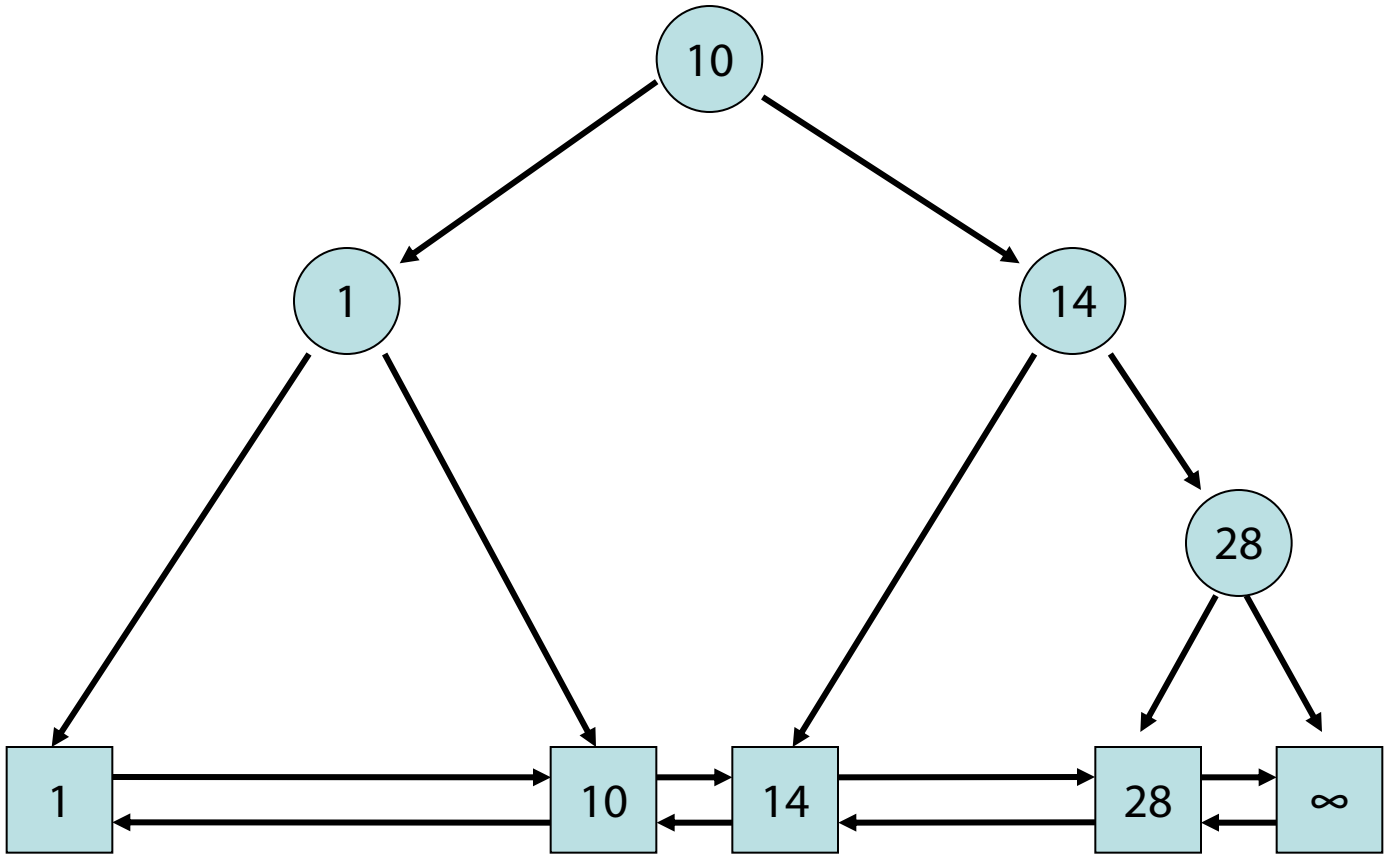
Search(9)



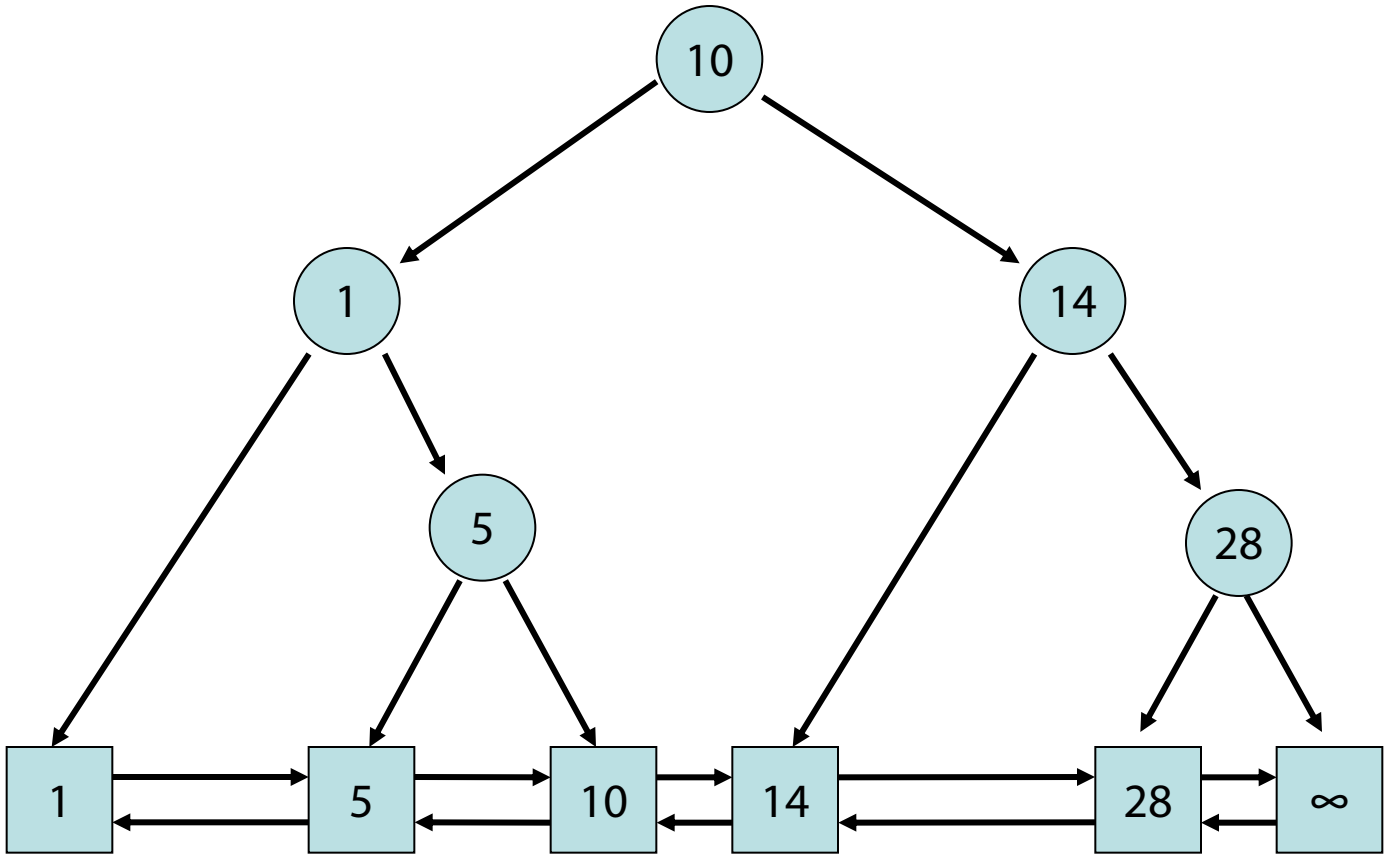
Insert Operation

- insert(e, s:)
  Erst search(key(e), s) bis Element e' in Liste erreicht.
  Falls key (e‘) > key (e) dann:
    Füge e vor e' ein und ein neues Suchbaumblatt
    für e und e' mit key(e), so dass Suchbaum-Regel erfüllt.
  Falls key (e‘) = key (e) dann:
    Ersetze e‘ durch e (keine Duplikate)

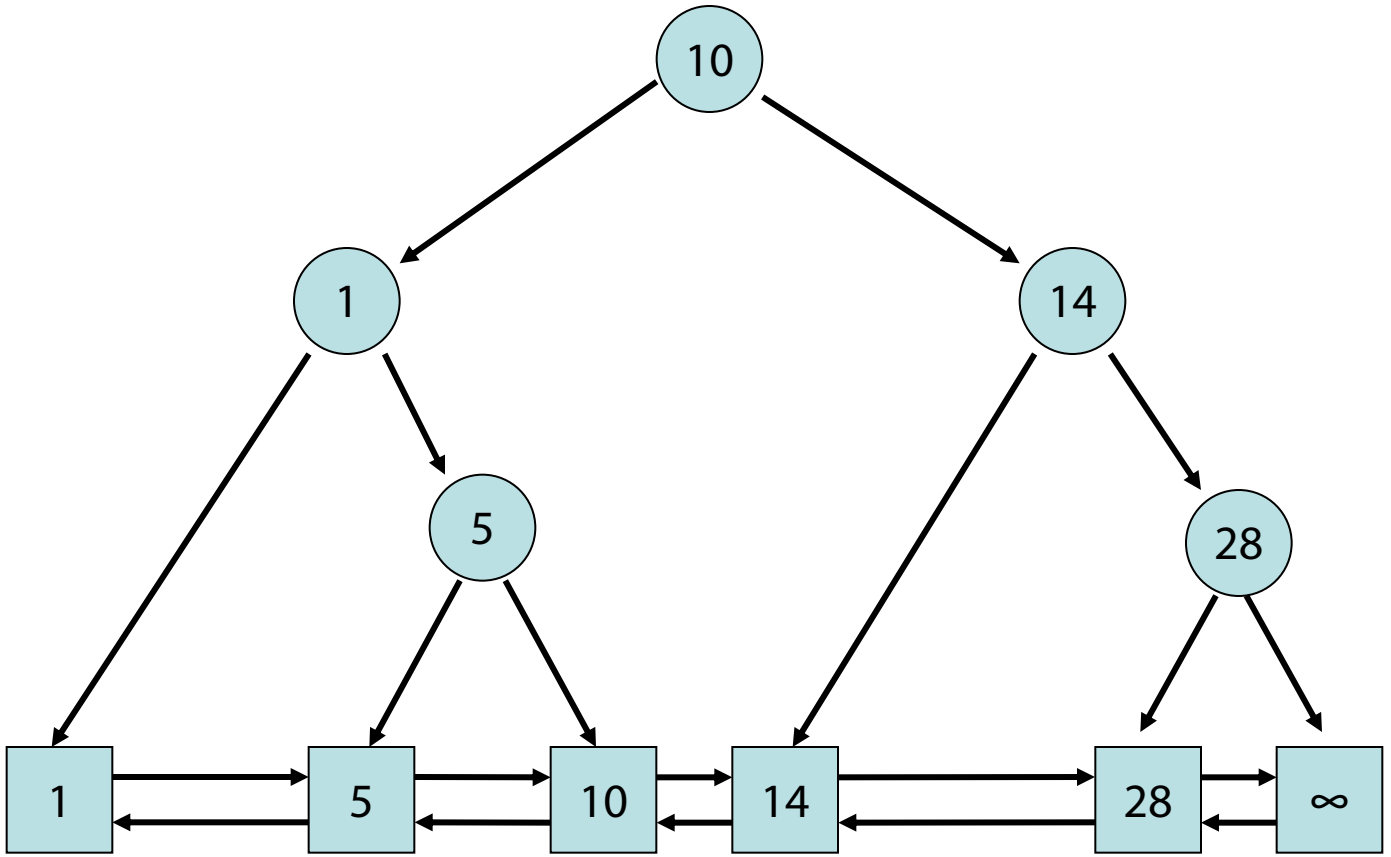
Insert(5)



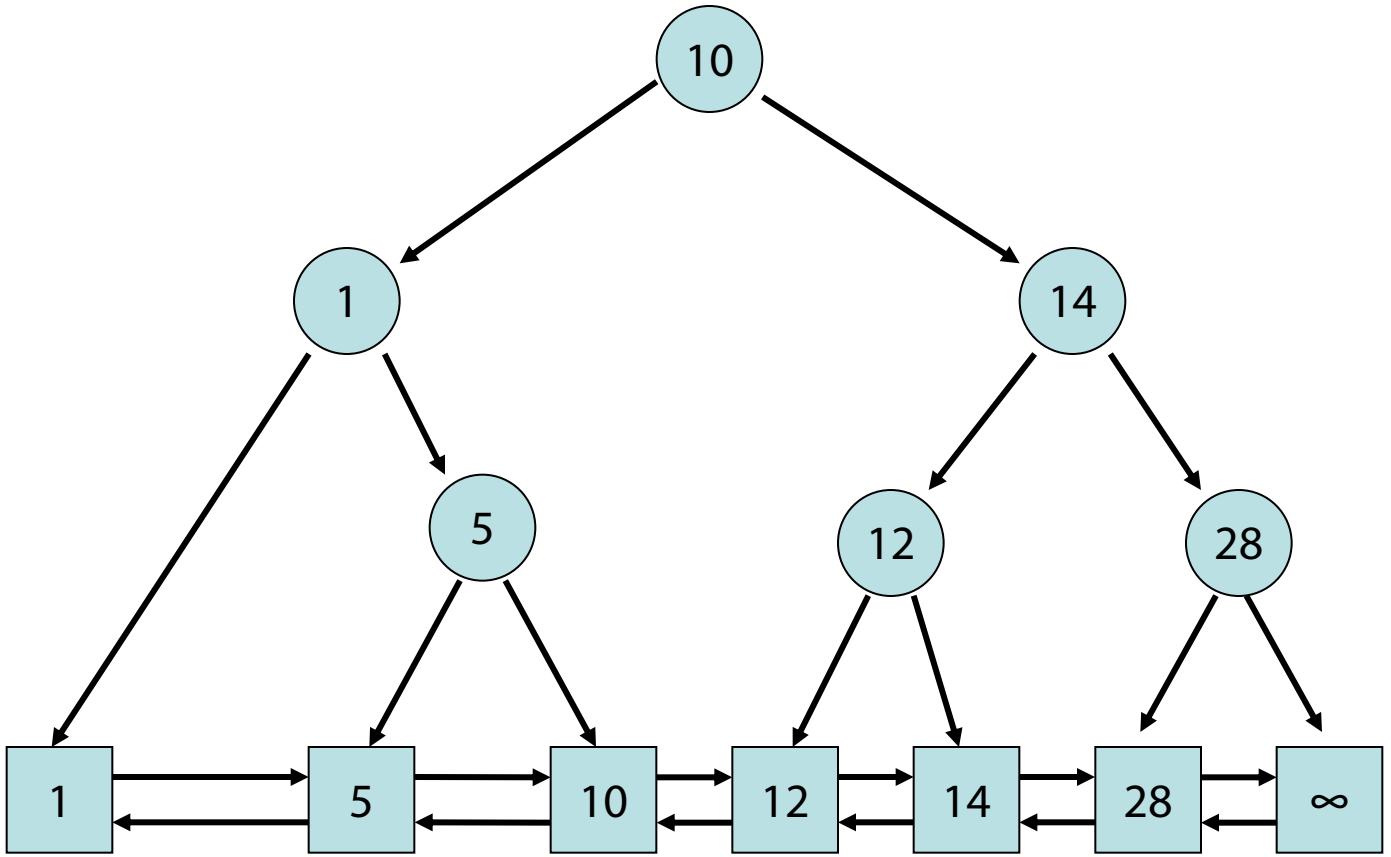
Insert(5)



Insert(12)



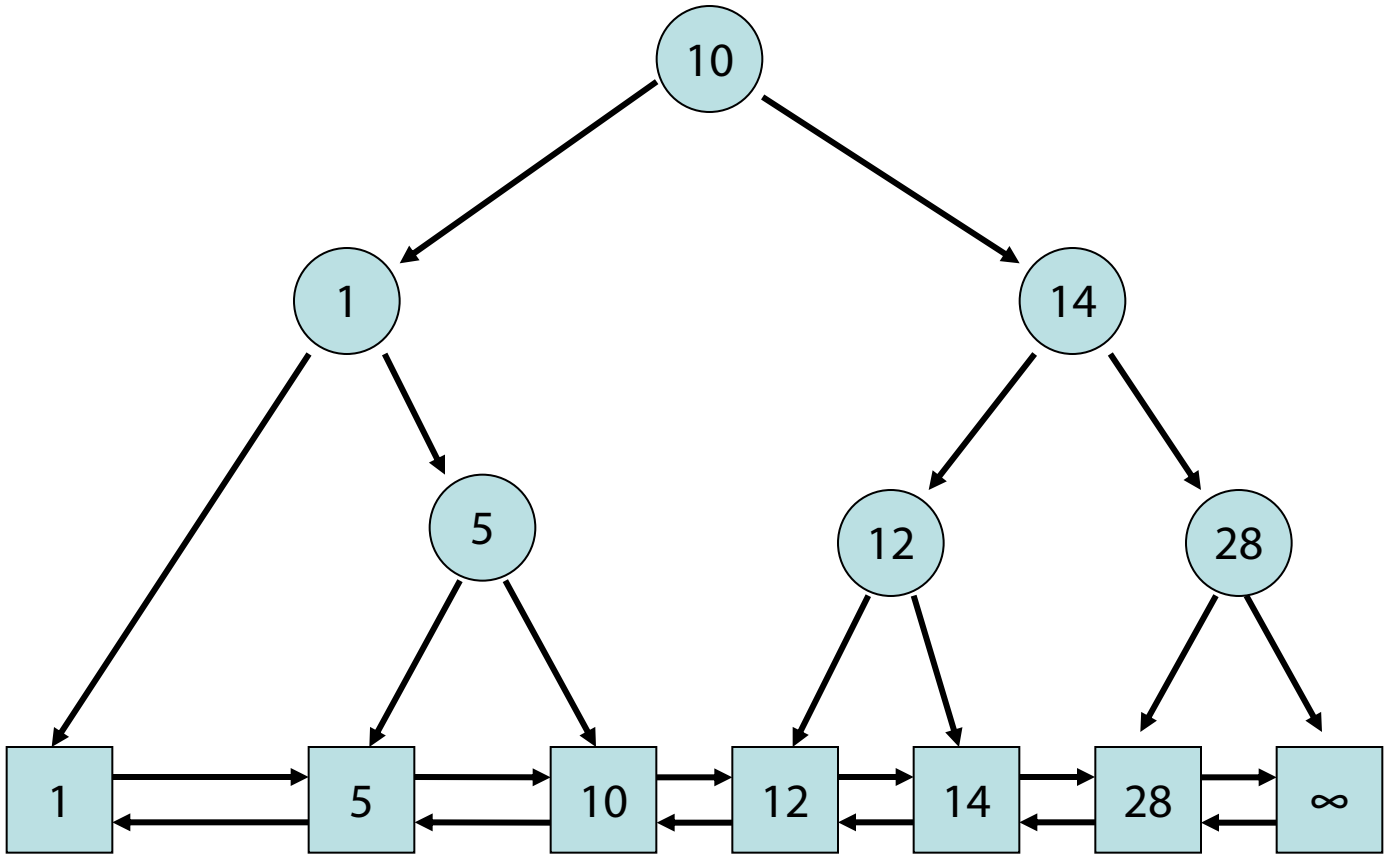
Insert(12)



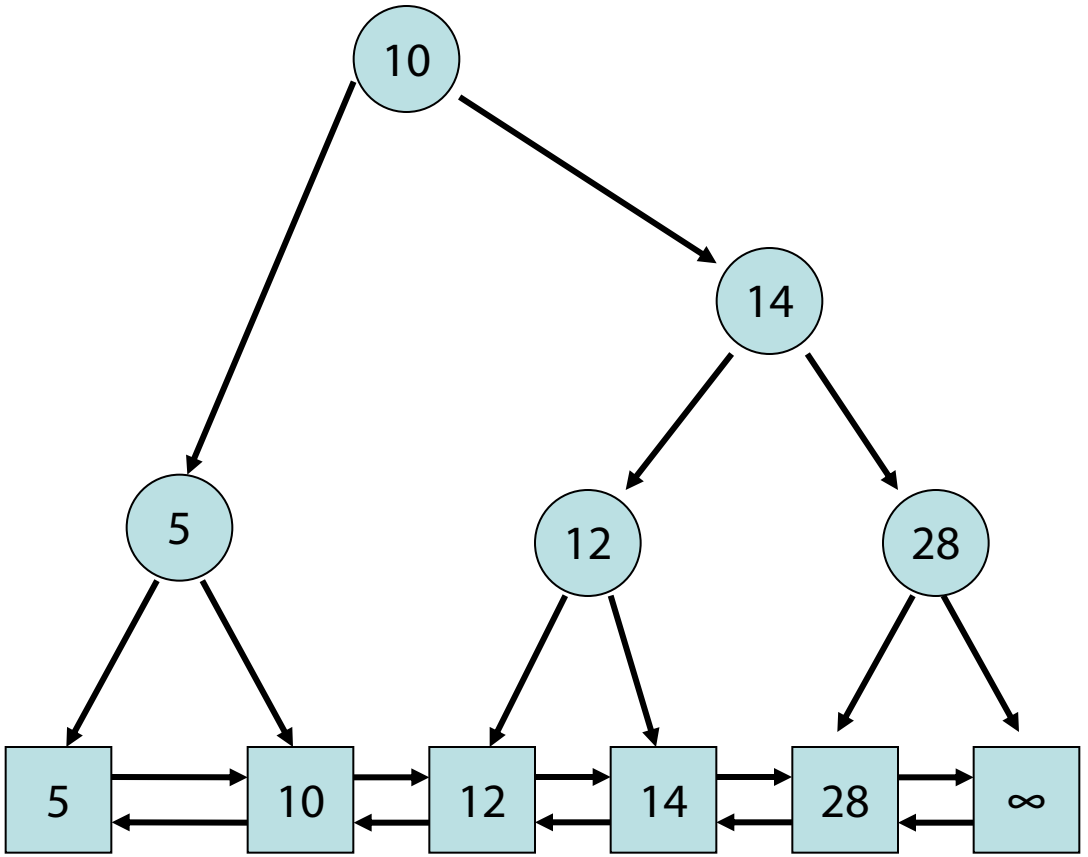
Delete Operation

- delete(k, s:)
  Erst search(k, s) bis ein Element e in Liste erreicht.
  Falls key (e) = k, lösche e aus Liste und Vater v von e aus Suchbaum, und setze den Baumknoten w mit key (w) = k: key(w) := key(v)

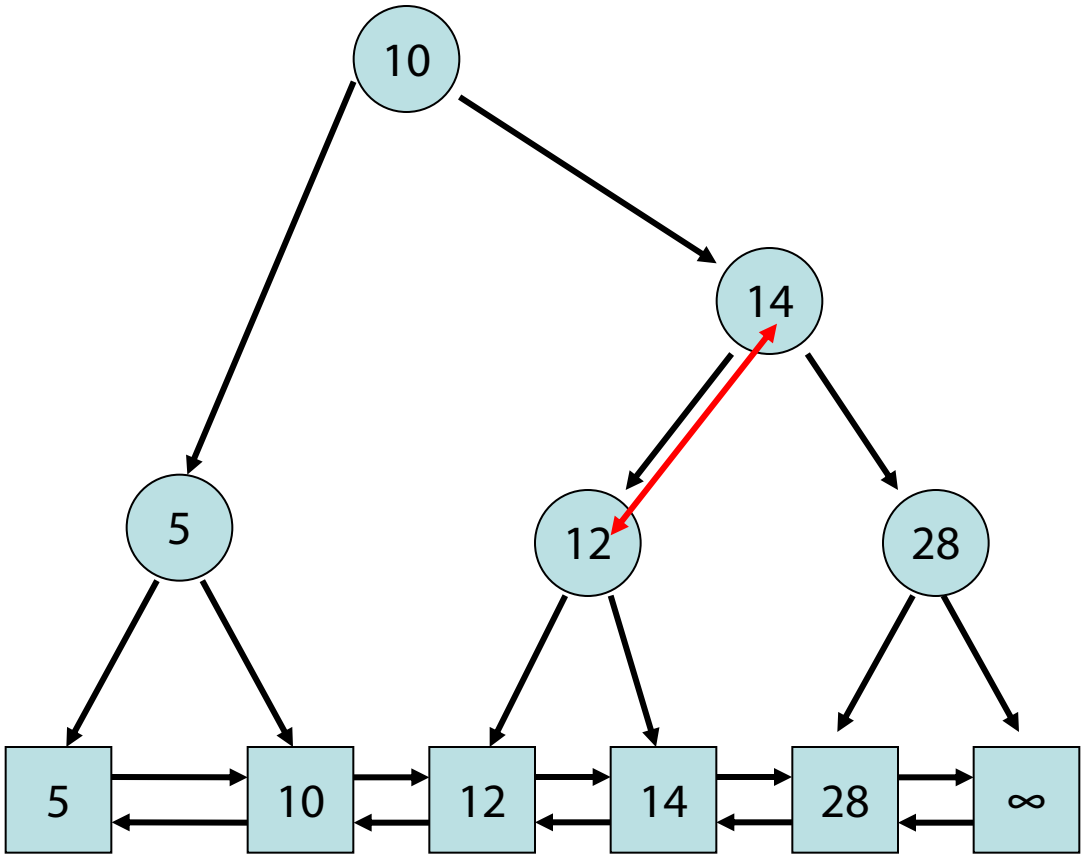
Delete(1)



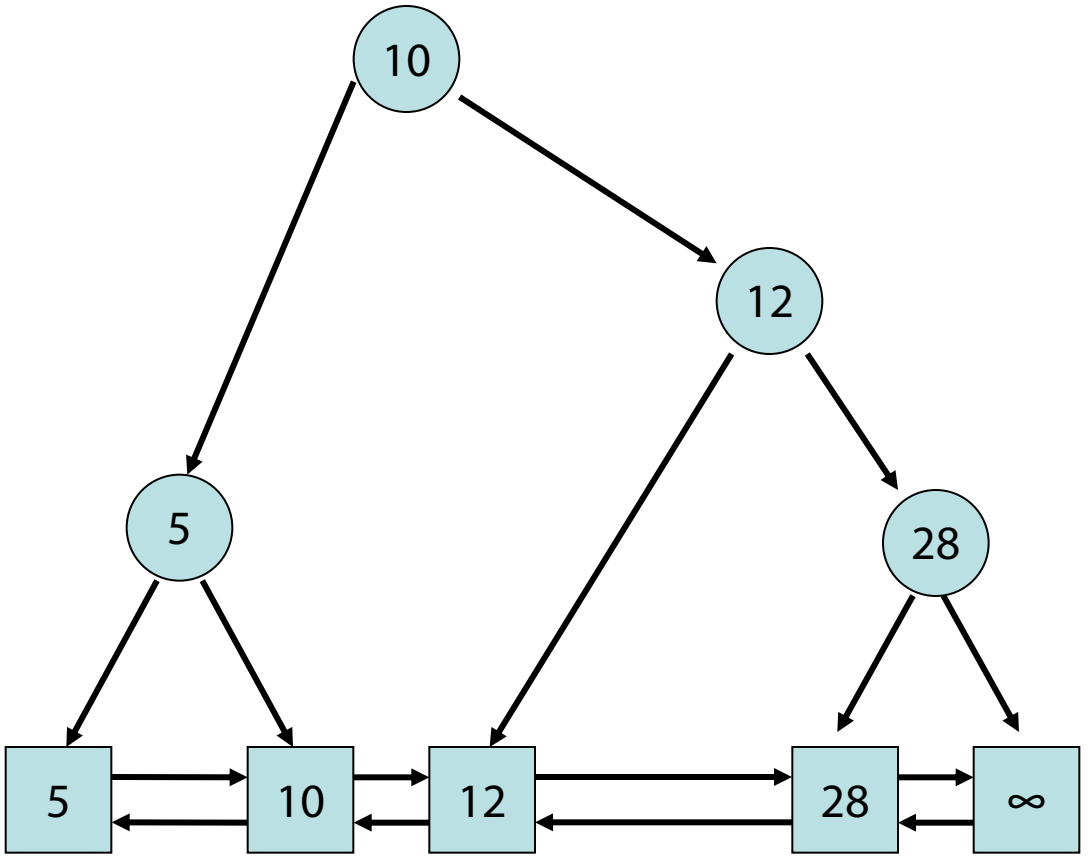
Delete(1)



Delete(14)



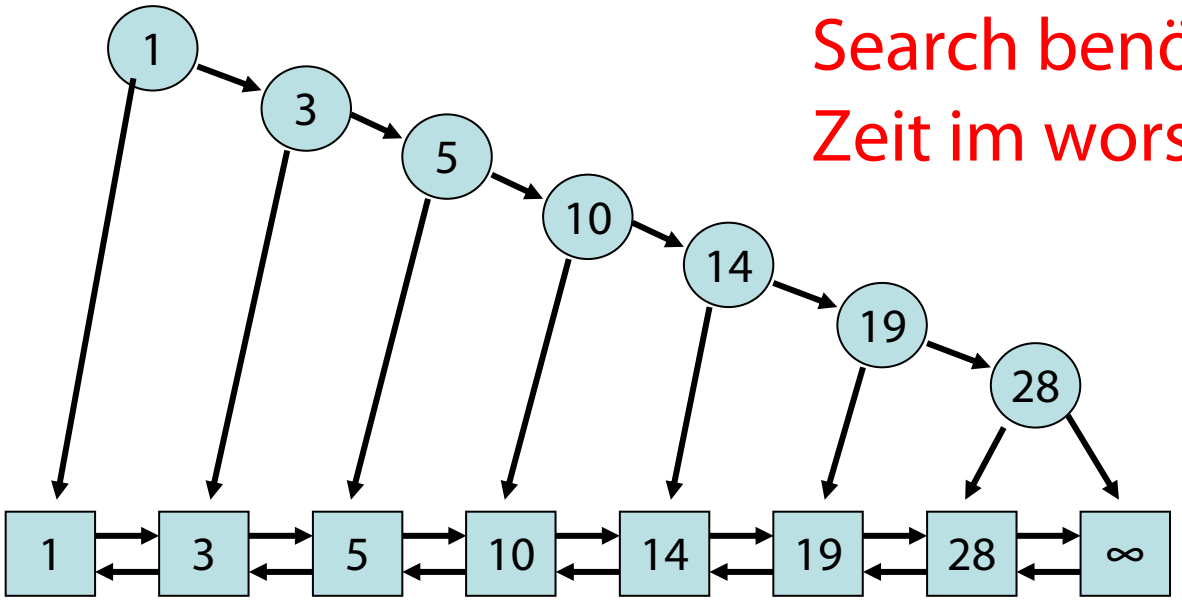
Delete(14)



Entarteter Binärbaum

Problem: Binärbaum kann bei bestimmter Einfügereihenfolge in entarteter Form aufgebaut werden!

Beispiel: Zahlen werden in sortierter Folge eingefügt



Search benötigt $\theta(n)$ Zeit im worst case

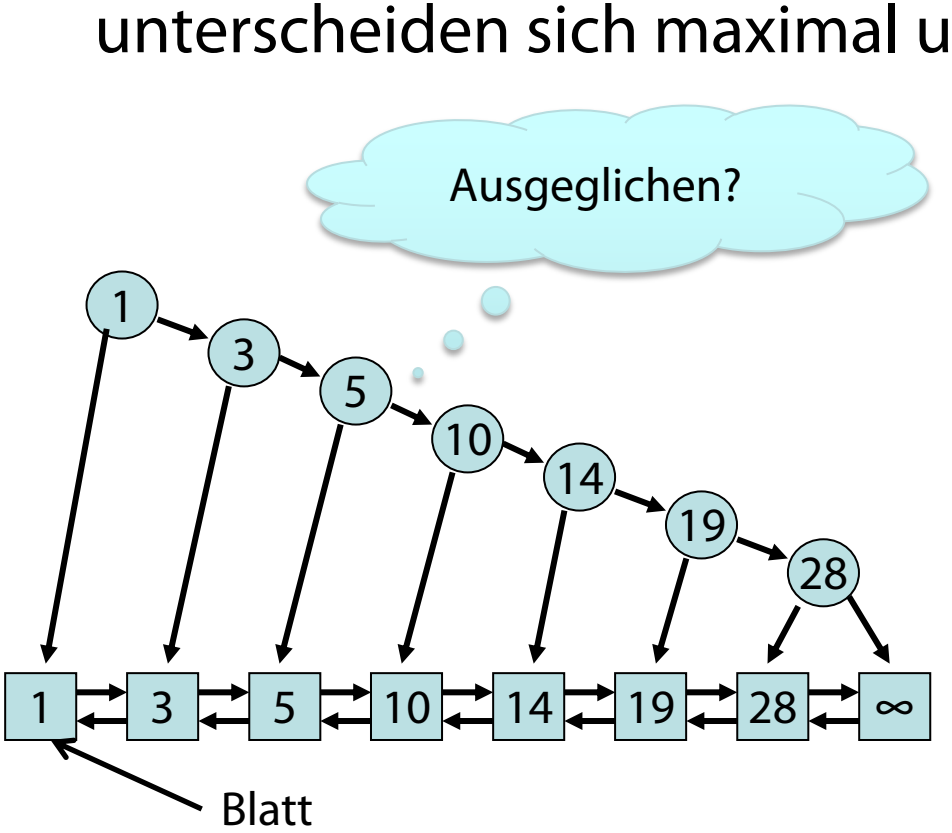
Automatische Umstrukturierung

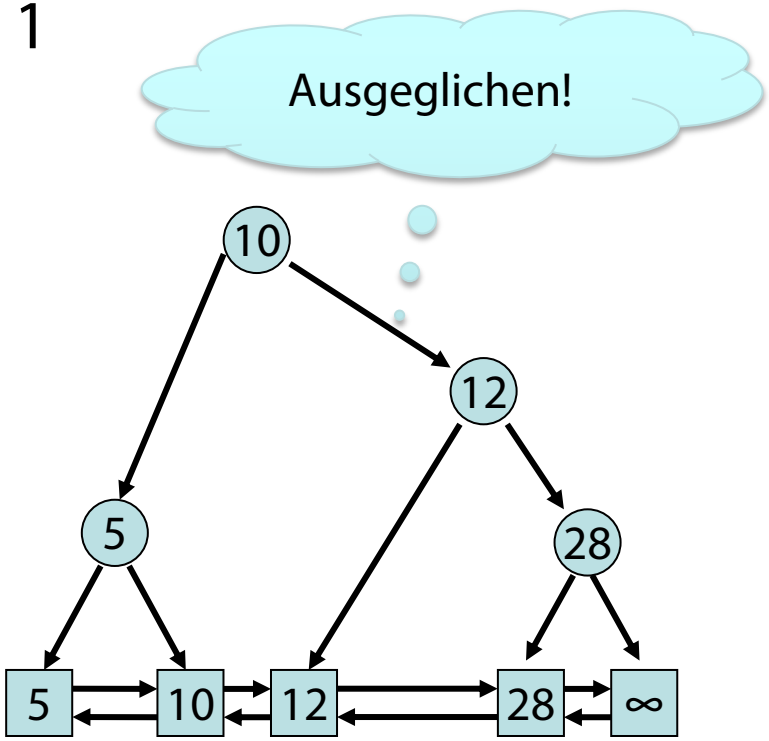
Optionen

- Anpassung bei Anfragen (Operationen search oder test)
- Anpassung beim Einfügen (insert) oder Löschen (delete) neuer Elemente

Definition: Ausgeglichener Suchbaum

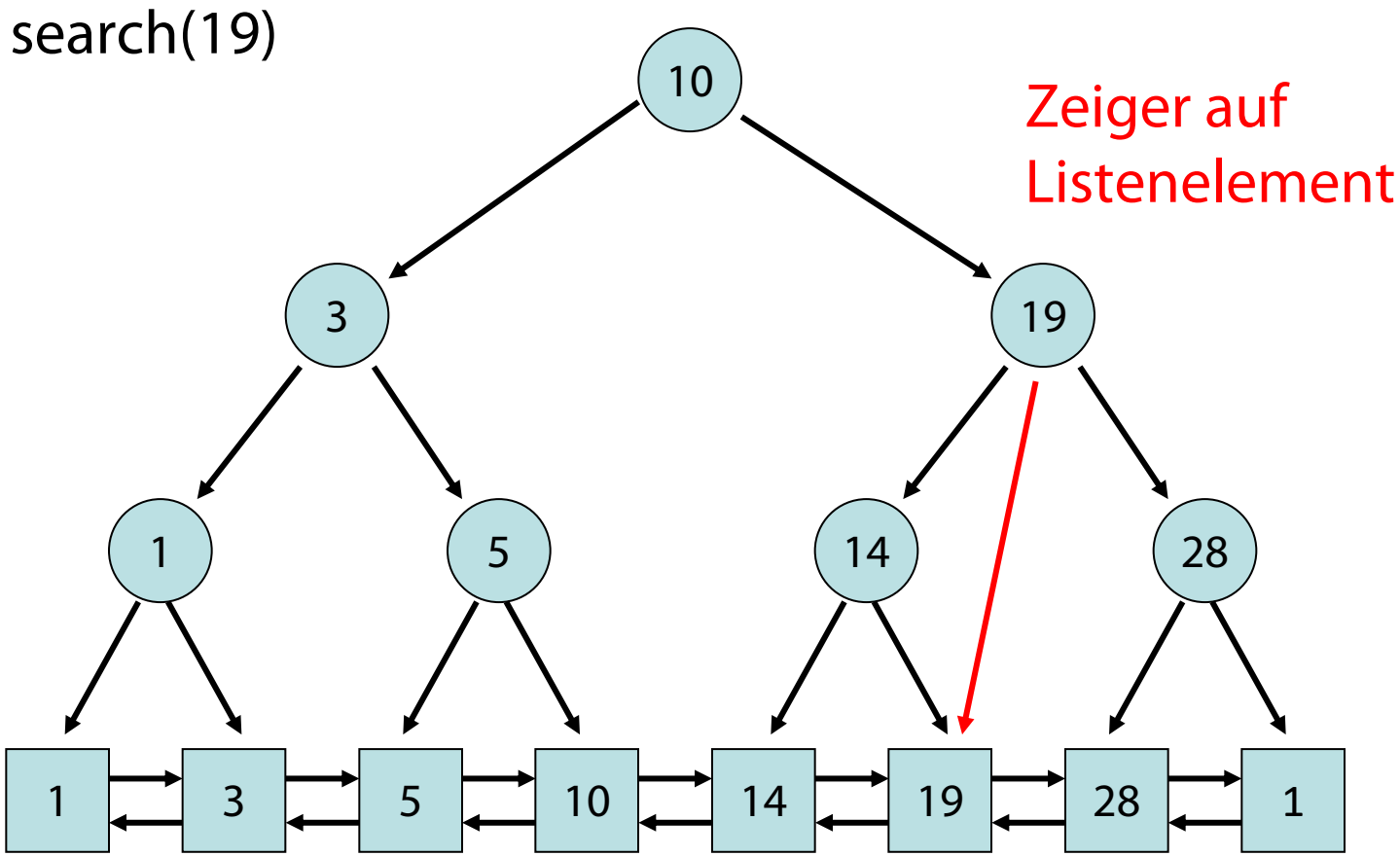
- Die Längen der Pfade von den Blättern zur Wurzel unterscheiden sich maximal um 1





- „Alle Ebenen bis auf Blattebene voll gefüllt“

Ein weiterer Zeiger pro Knoten…



Navigationsbäume mit Zeigern auf Listenelemente

- Ausgeglichenheit nur optimal, wenn relative Häufigkeit des Zugriffs bei allen Schlüsseln gleich
- Ist dies nicht der Fall, sollte relative Zugriffshäufigkeit bei der Baumkonstruktion berücksichtigt werden
- Idee: Ordne den Schlüsseln Gewichte zu
  - Häufiger zugegriffene Schlüssel: hohes Gewicht
  - Weniger oft zugegriffene Schlüssel: kleines Gewicht
- Knoten mit Schlüsseln, denen ein höheres Gewicht gegeben wird, sollen weiter oben stehen

Selbstorganisierende Bäume

- Man beachte: Suchaufwand $\Theta(\log n)$
  - Elementtests mehrfach mit dem gleichen Element:
    → dann $O(\log n)$ „zu teuer“
- Weiterhin: Mit bisheriger Technik des Einfügens kann Ausgeglichenheit nicht garantiert werden: Zugriff für bestimmte Elemente $> \log n$
  - Elementtest für diese Elemente häufig:
    → Performanz sinkt
- Idee: Häufig zugegriffene Elemente sollten trotz Unausgeglichenheit schneller gefunden werden (amortisiert betrachtet dann insgesamt $O(\log n)$)
- Umsetzung: Splay-Baum (selbstorganisierend)

Daniel D. Sleator, Robert Tarjan: Self-Adjusting Binary Search Trees, In: Journal of the ACM (Association for Computing Machinery). 32, Nr. 3, S. 652–686, **1985**

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

Splay-Baum

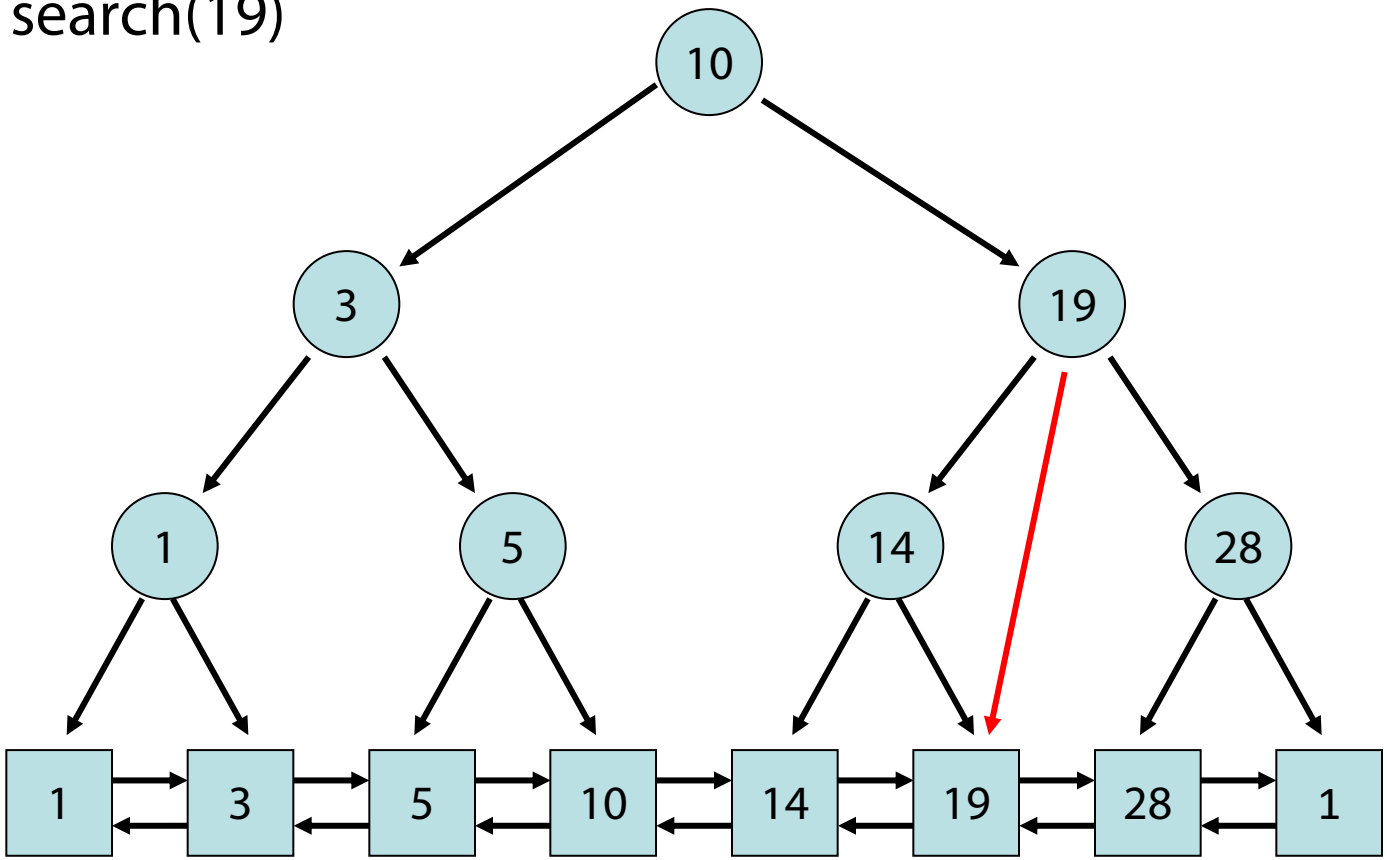
Üblicherweise: Implementierung als interner Suchbaum (d.h. Elemente direkt integriert in Baum und nicht in extra Liste “unten”)

Hier: Implementierung als externer Suchbaum (wie beim binären Navigationsbaum oben)

Modifikation bei Anfragen

Splay-Baum

search(19)



Splay-Baum

Idee:

Bewege Schlüssel vom zugegriffenem Element zur Wurzel

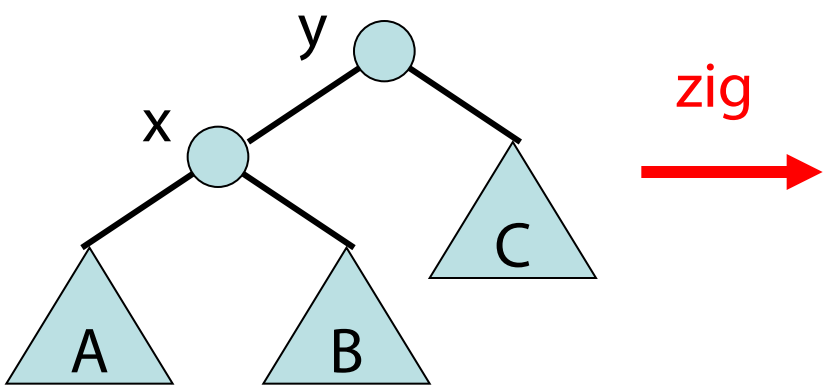
- Können wir einfach nach oben sieben wie beim Heap?
  - Nein
- Wie dann Bewegung zur Wurzel realisieren?
  - Idee: über sog. Splay-Operation

Splay-Operation

Bewegung von Schlüssel x nach oben:

Wir unterscheiden zwischen 3 Fällen.

1a. x ist Kind der Wurzel:

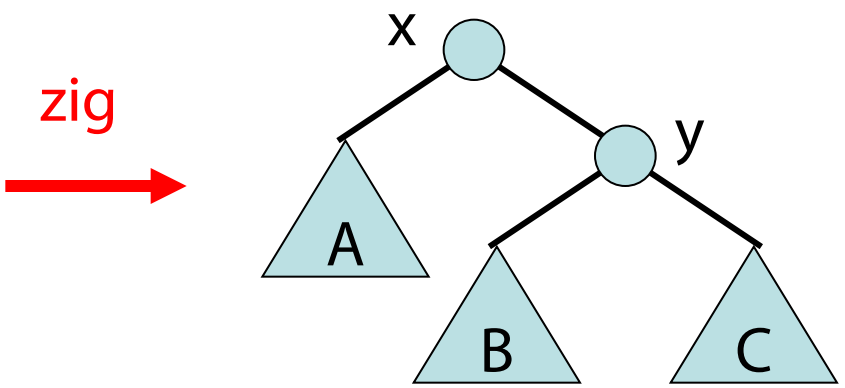


Splay-Operation

Bewegung von Schlüssel x nach oben:

Wir unterscheiden zwischen 3 Fällen.

1a. x ist Kind der Wurzel:

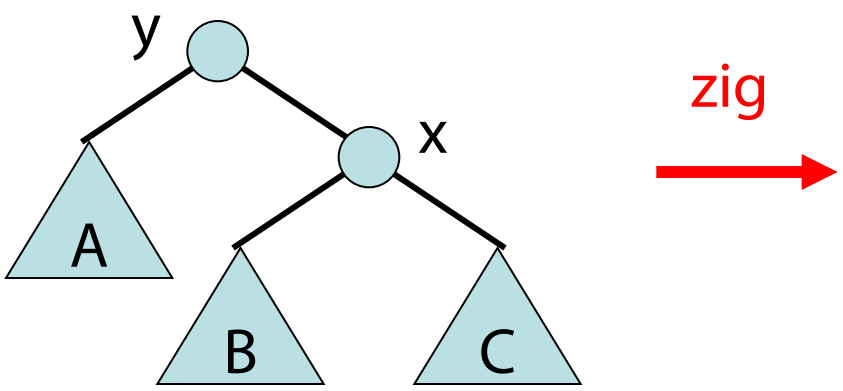


Splay-Operation

Bewegung von Schlüssel x nach oben:

Wir unterscheiden zwischen 3 Fällen.

1b. x ist Kind der Wurzel:

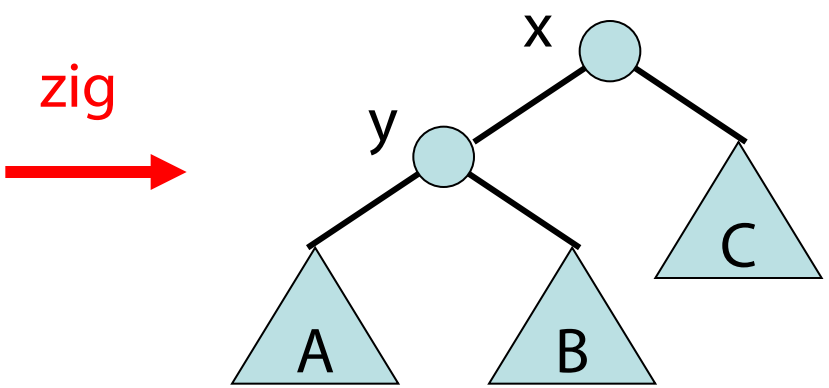


Splay-Operation

Bewegung von Schlüssel x nach oben:

Wir unterscheiden zwischen 3 Fällen.

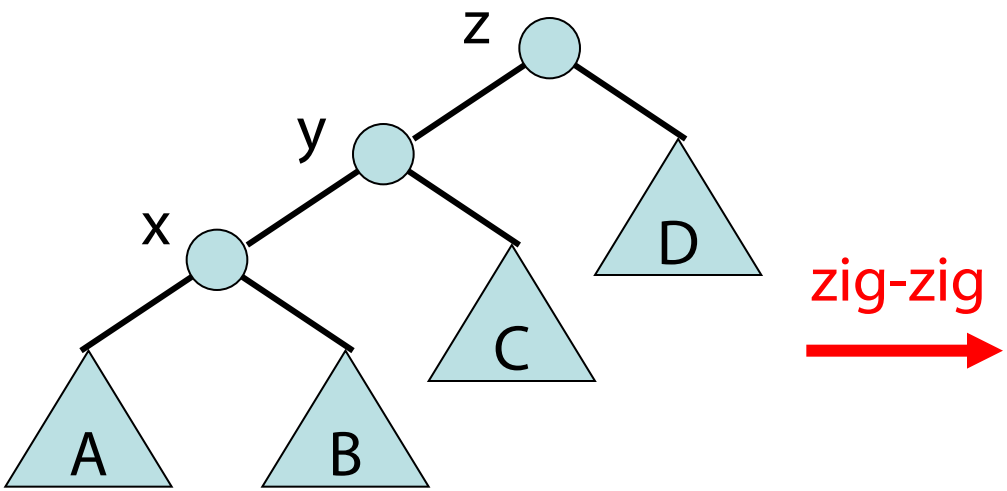
1b. x ist Kind der Wurzel:



Splay-Operation

Wir unterscheiden zwischen 3 Fällen.

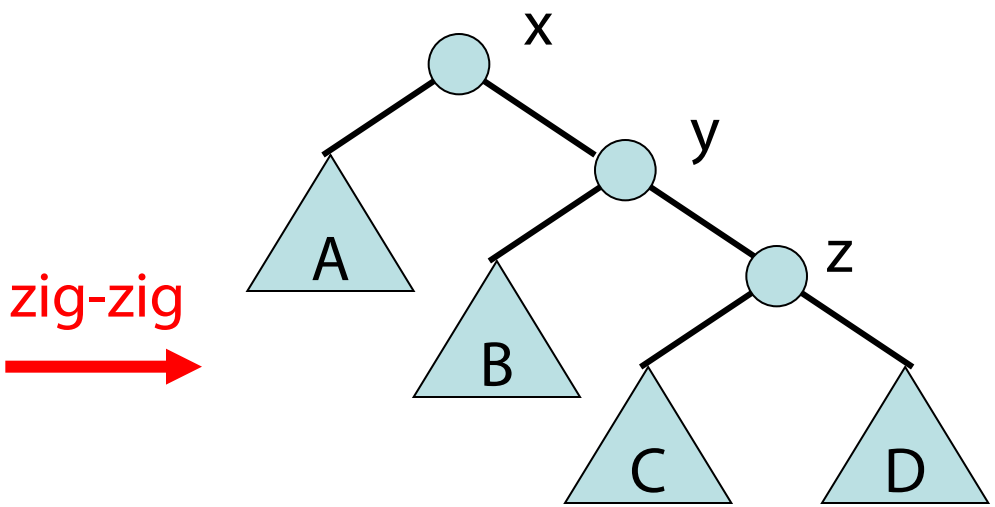
2a. x hat Vater und Großvater rechts:



Splay-Operation

Wir unterscheiden zwischen 3 Fällen.

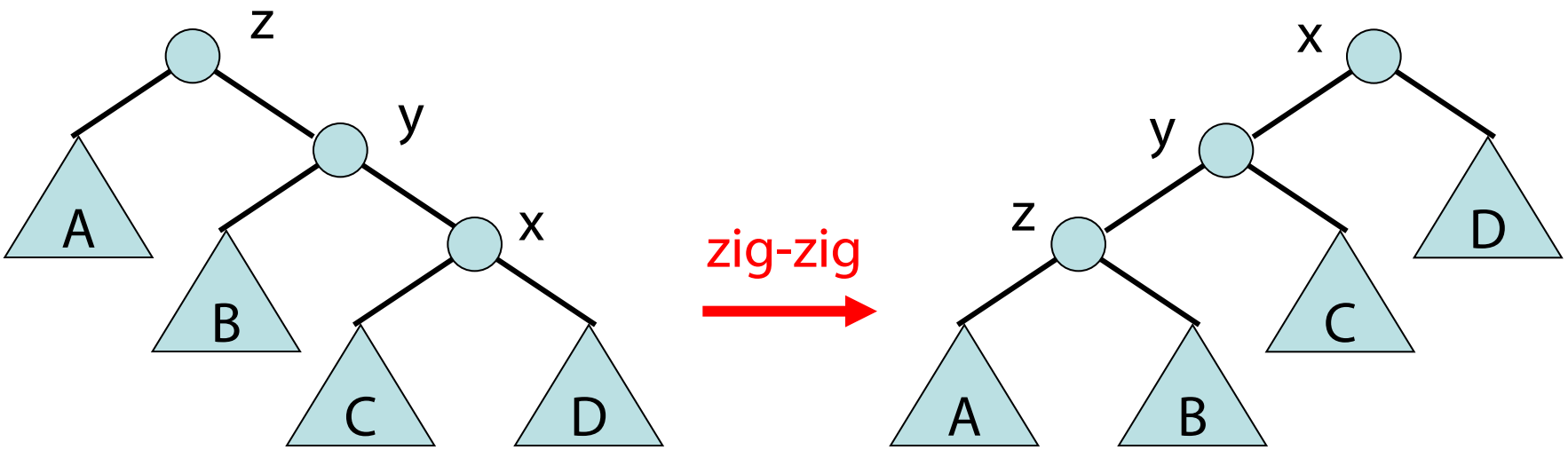
2a. x hat Vater und Großvater rechts:



Splay-Operation

Wir unterscheiden zwischen 3 Fällen.

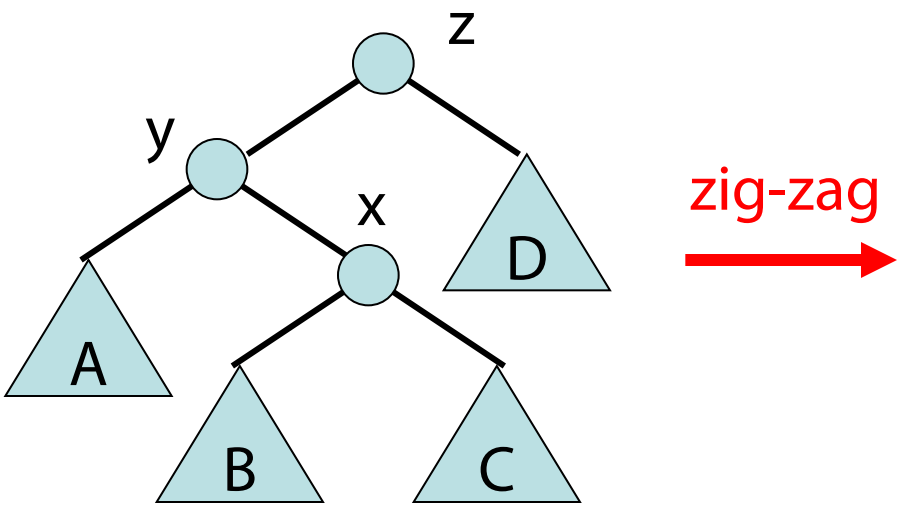
2b. x hat Vater und Großvater links:



Splay-Operation

Wir unterscheiden zwischen 3 Fällen.

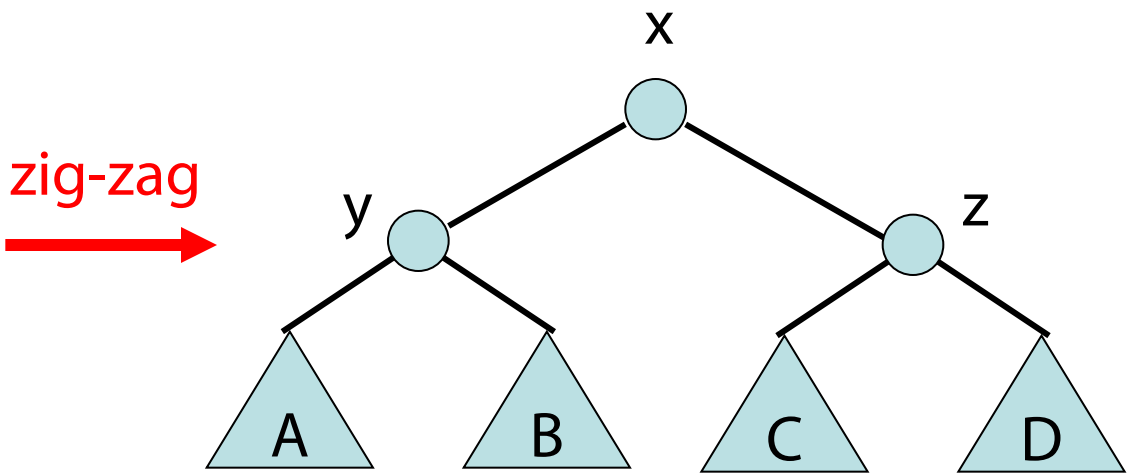
3a. x hat Vater links, Großvater rechts:



Splay-Operation

Wir unterscheiden zwischen 3 Fällen.

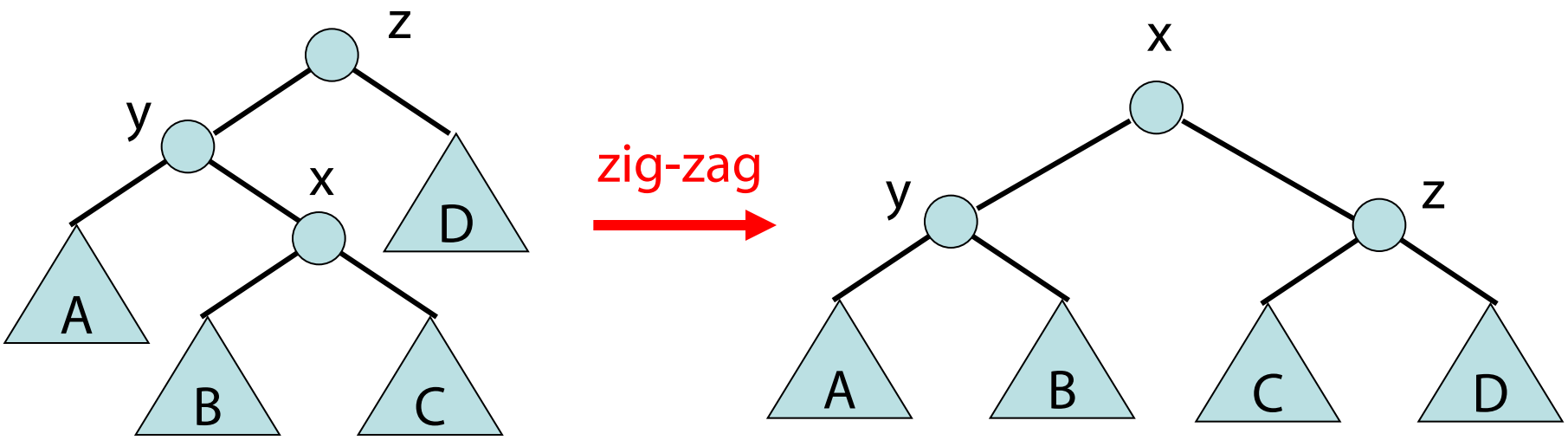
3a. x hat Vater links, Großvater rechts:



Splay-Operation

Wir unterscheiden zwischen 3 Fällen.

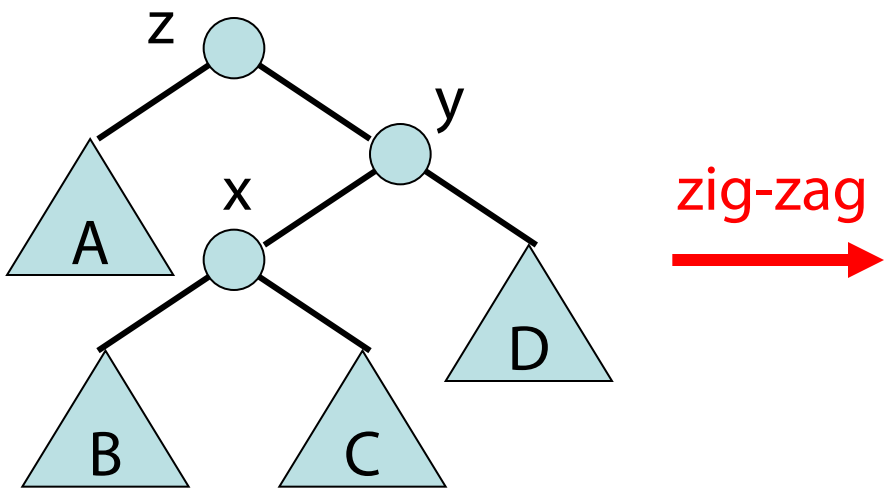
3a. x hat Vater links, Großvater rechts:



Splay-Operation

Wir unterscheiden zwischen 3 Fällen.

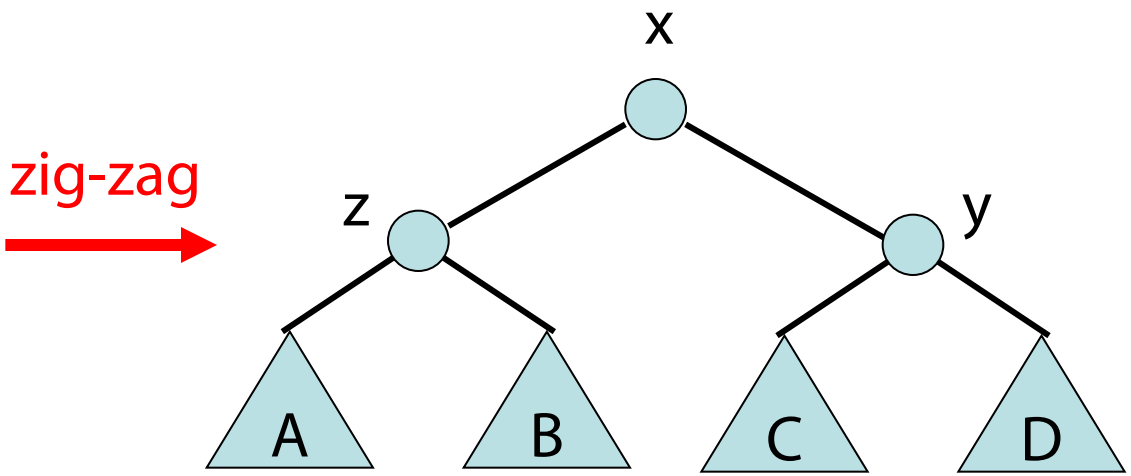
3b. x hat Vater rechts, Großvater links:



Splay-Operation

Wir unterscheiden zwischen 3 Fällen.

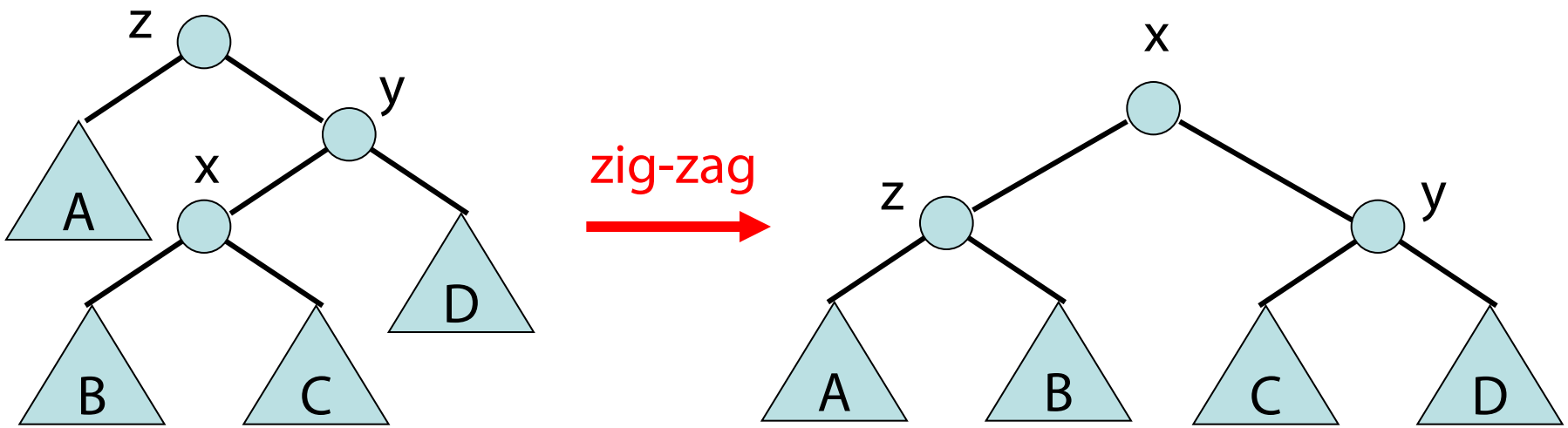
3b. x hat Vater rechts, Großvater links:



Splay-Operation

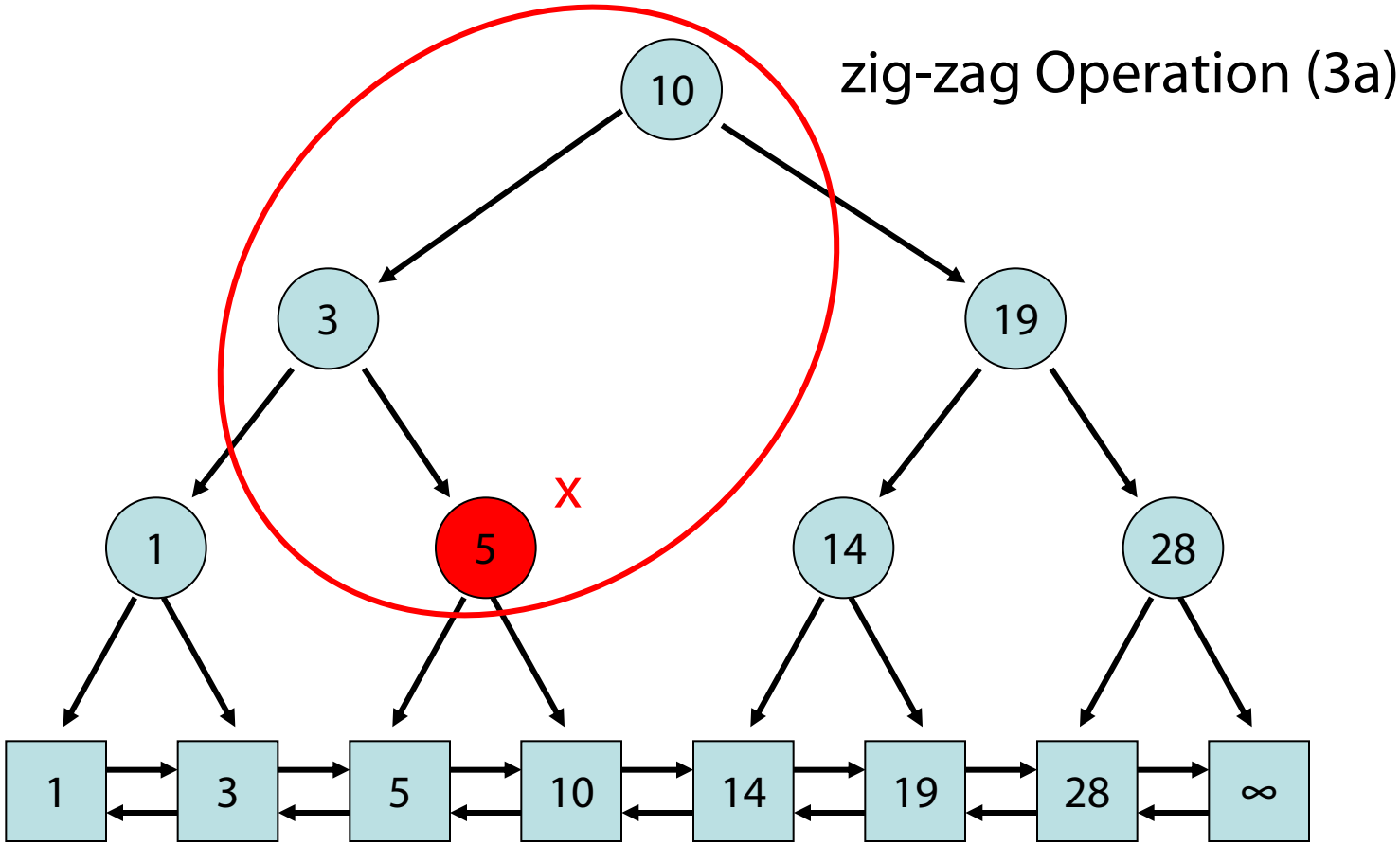
Wir unterscheiden zwischen 3 Fällen.

3b. x hat Vater rechts, Großvater links:

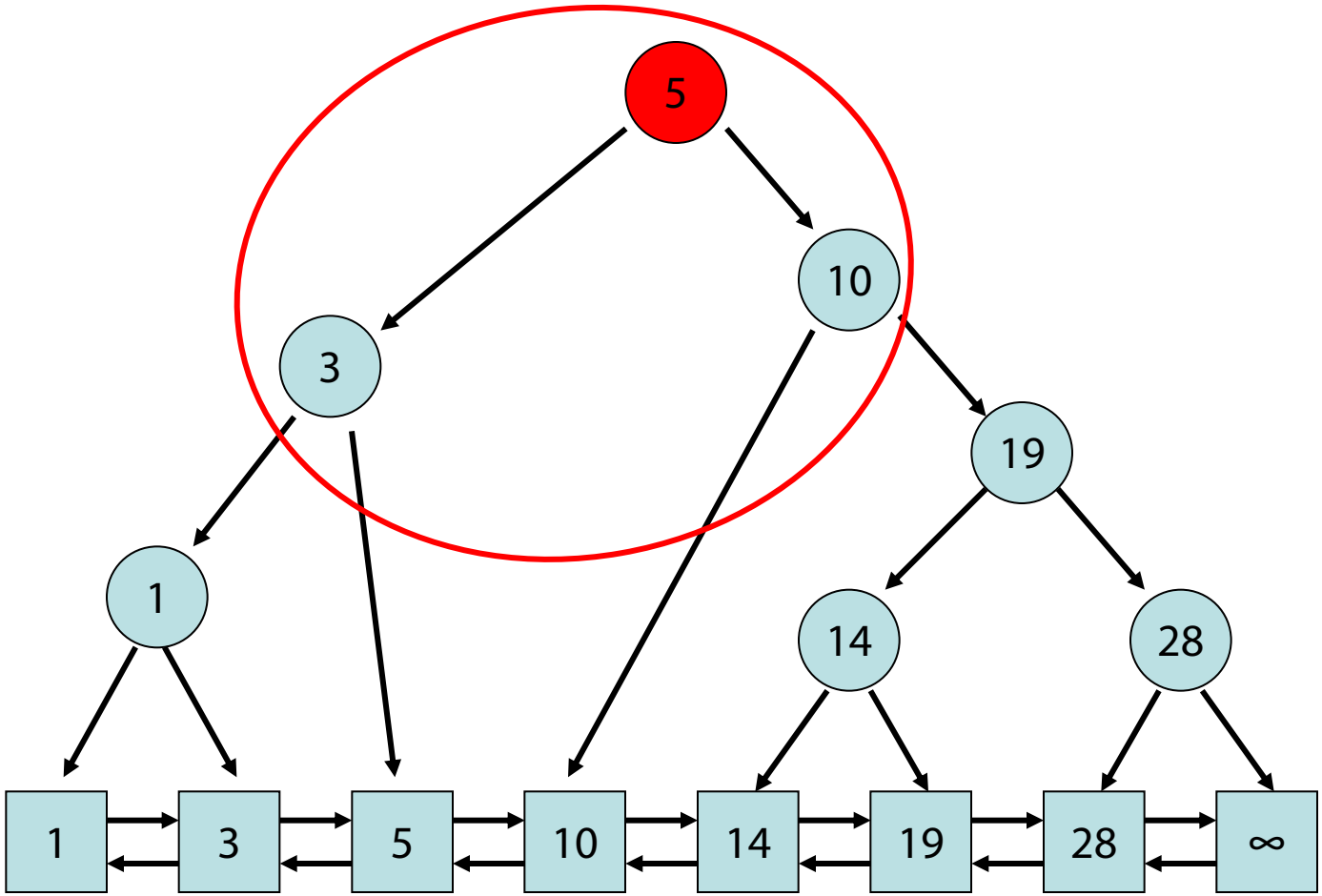


Splay-Operation

Beispiel:



Splay-Operation



Zusammenfassung

- Ausgeglichenheit für Navigationsbäume
- Splay-Bäume
  - Umorganisation beim Zugriff mit search
- Unser Eindruck:
  - Ja, Knoten wandern nach oben
- Noch zu zeigen:
  - Zeitfunktion von search amortisiert in O(log n)