

Kapitel 4: Dynamische Datenstrukturen

*Algorithmen und Datenstrukturen
WS 2020/21*

Prof. Dr. Sándor Fekete

4.1 Grundoperationen

Aufgabenstellung:

- *Verwalten einer Menge S von Objekten*
- *Ausführen von verschiedenen Operationen (s.u.)*

Im Folgenden:

S	Menge von Objekten
k	Wert eines Elements (“Schlüssel”)
x	Zeiger auf Element
NIL	spezieller, “leerer” Zeiger

4.1 Grundoperationen

SEARCH(S,k): “Suche in S nach k”

Durchsuche die Menge S nach einem Element von Wert k.

**Ausgabe: Zeiger x, falls x existent
NIL, falls kein Element Wert k hat.**

4.1 Grundoperationen

Langsam:

- $O(n)$: *lineare Zeit*

Alle Objekte anschauen

Sehr schnell:

- $O(1)$: *konstante Zeit*

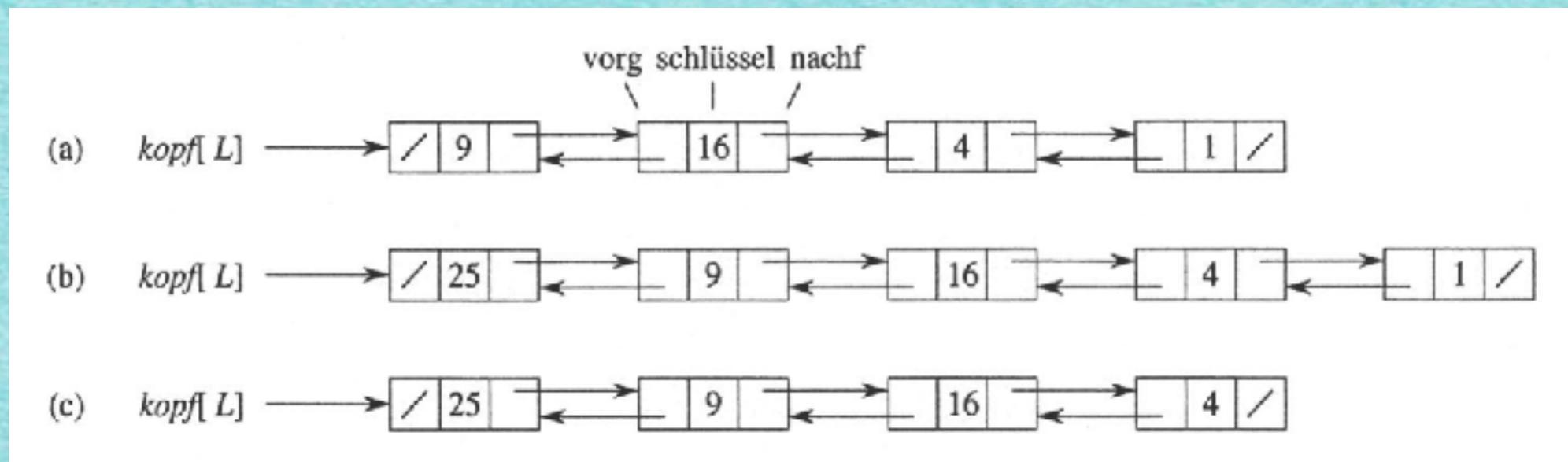
Immer gleich schnell, egal wie groß S ist.

Schnell:

- $O(\log n)$: *logarithmische Zeit*

Wiederholtes Halbieren

Struktur einer doppelt verketteten Liste



- **Füge vorne das Element mit Schlüssel 25 ein.**

- **Finde ein Element mit Schlüssel 1 und lösche es.**

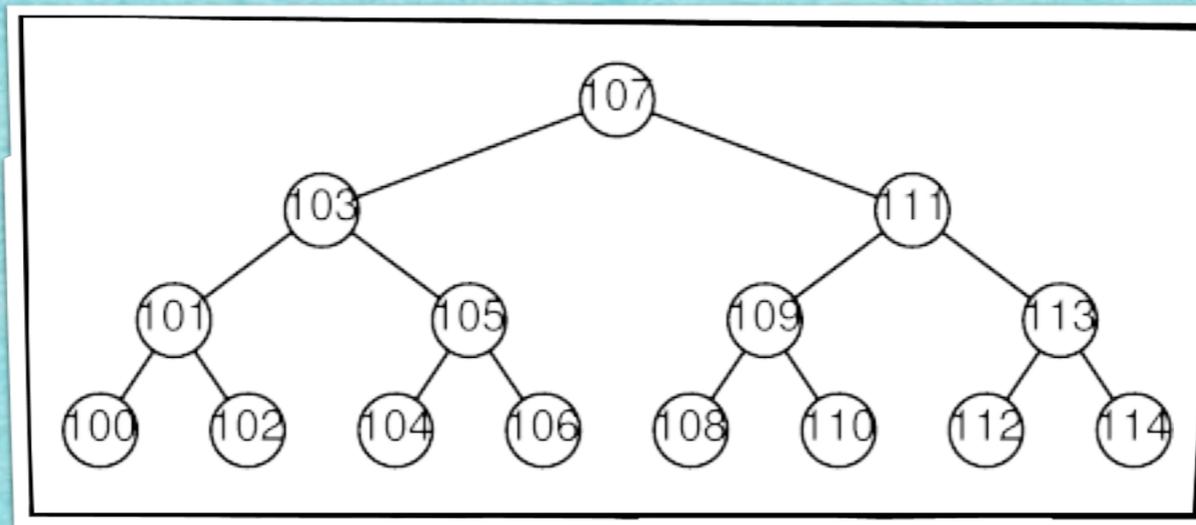
4.4 Binäre Suche

Satz 4.2

Die binäre Suche terminiert in $O(\log(\text{RECHTS-LINKS}))$ Schritten (für $\text{RECHTS} > \text{LINKS}$).

Beweis:

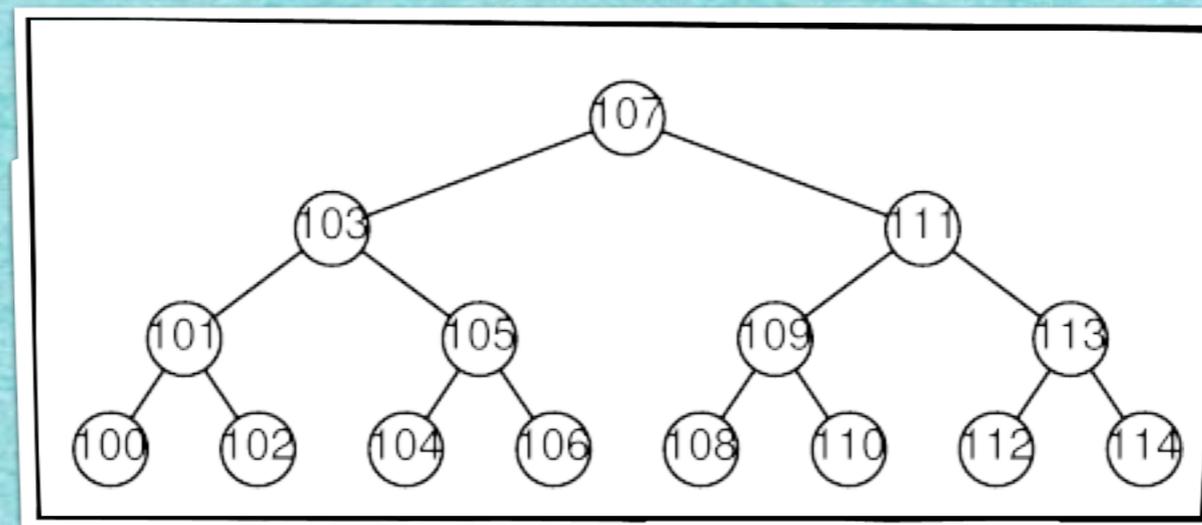
Selbst!



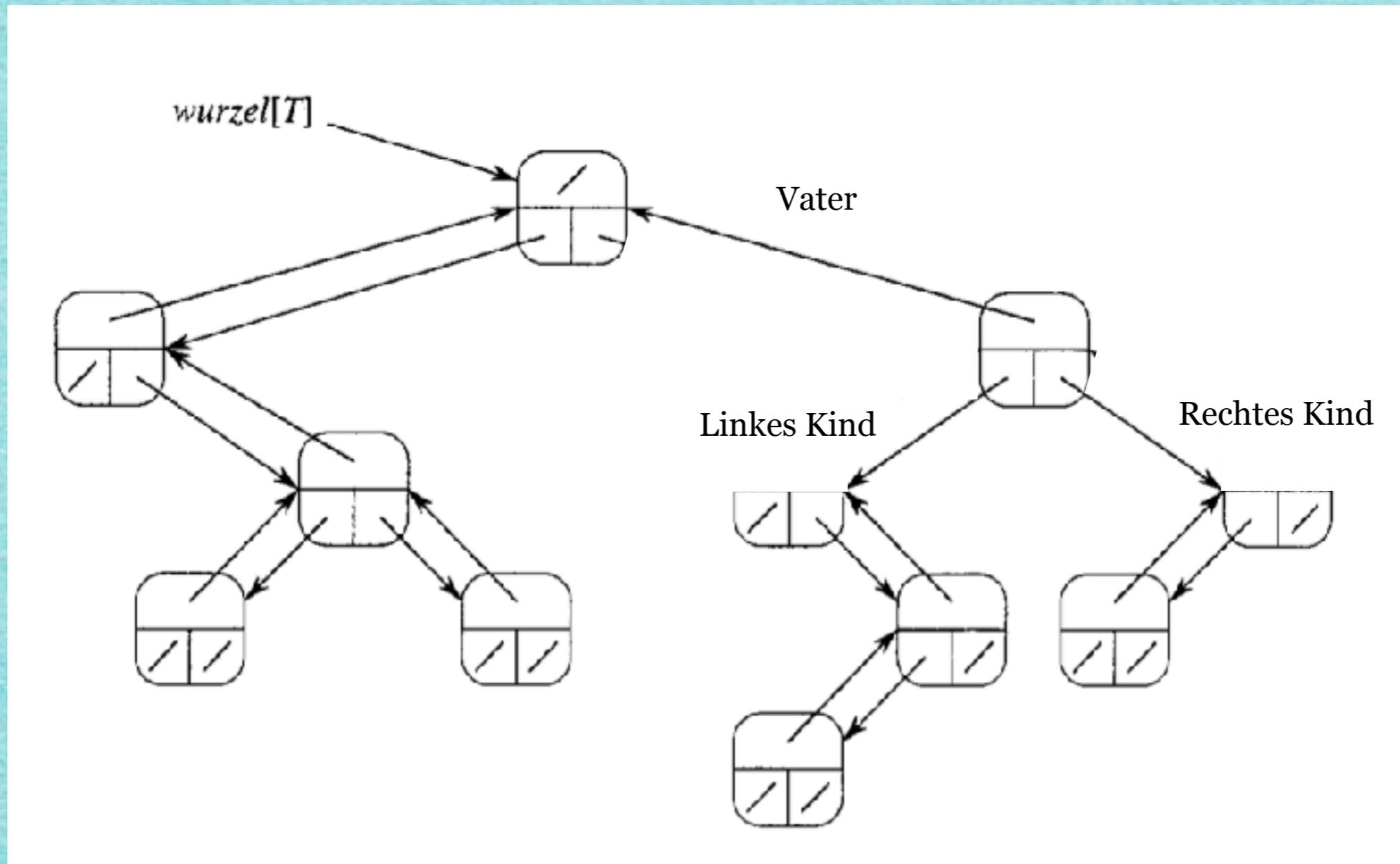
4.5 Binäre Suchbäume

Ideen:

- **Strukturiere Daten wie im möglichen Ablauf einer binären Suche!**
- **Erziele logarithmische Zeiten!**



Binärer Suchbaum

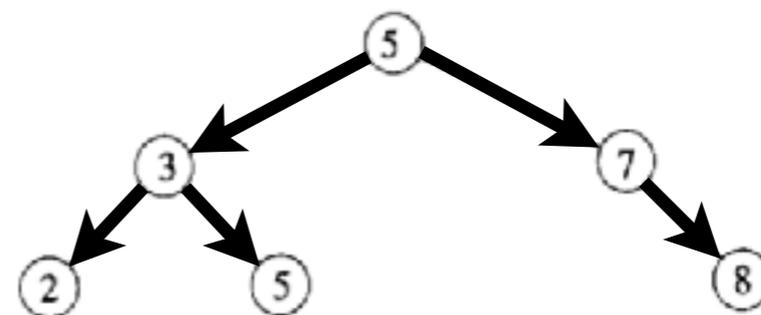


Außerdem wichtig: Struktur der Schlüsselwerte!

4.5 Binäre Suchbäume

Definition 4.3

- (1)** Ein gerichteter Graph $D=(V,A)$ besteht aus einer endlichen Menge V von Knoten v und einer endlichen Menge von gerichteten Kanten, $a=(v,w)$. (v ist Vorgänger von w .)
- (2)** Ein gerichteter Baum $B=(V,T)$ hat folgende Eigenschaften:
 - (i) Es gibt einen eindeutigen Knoten w ohne Vorgänger.
 - (ii) Jeder Knoten außer w ist auf einem eindeutigen Weg von w aus erreichbar; das heißt insbesondere, dass v einen eindeutigen Vorgänger (Vaterknoten) hat.
- (3)** Die Höhe eines gerichteten Baumes ist die maximale Länge eines gerichteten Weges von der Wurzel.
- (4)** Ein binärer Baum ist ein gerichteter Baum, in dem jeder Knoten höchstens zwei Nachfolger (“Kindknoten”) hat. Einer ist der “linke” $l[v]$, der andere der “rechte”, $r[v]$.



4.5 Binäre Suchbäume

Definition 4.3 (Forts.)

(5) Ein binärer Baum heißt voll, wenn jeder Knoten zwei oder keinen Kindknoten hat.

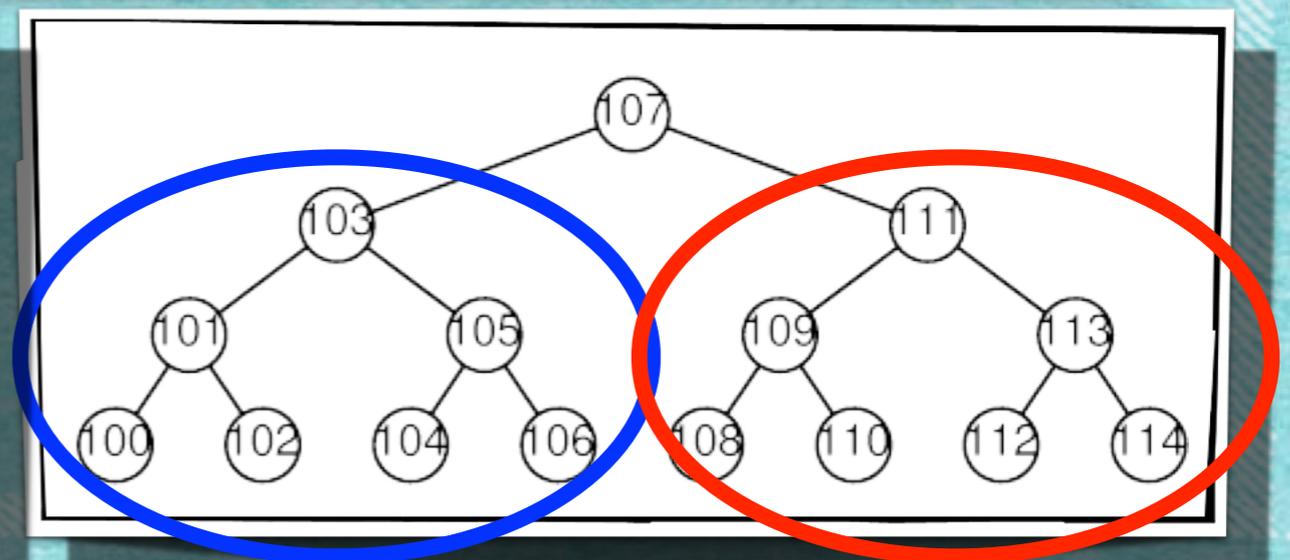
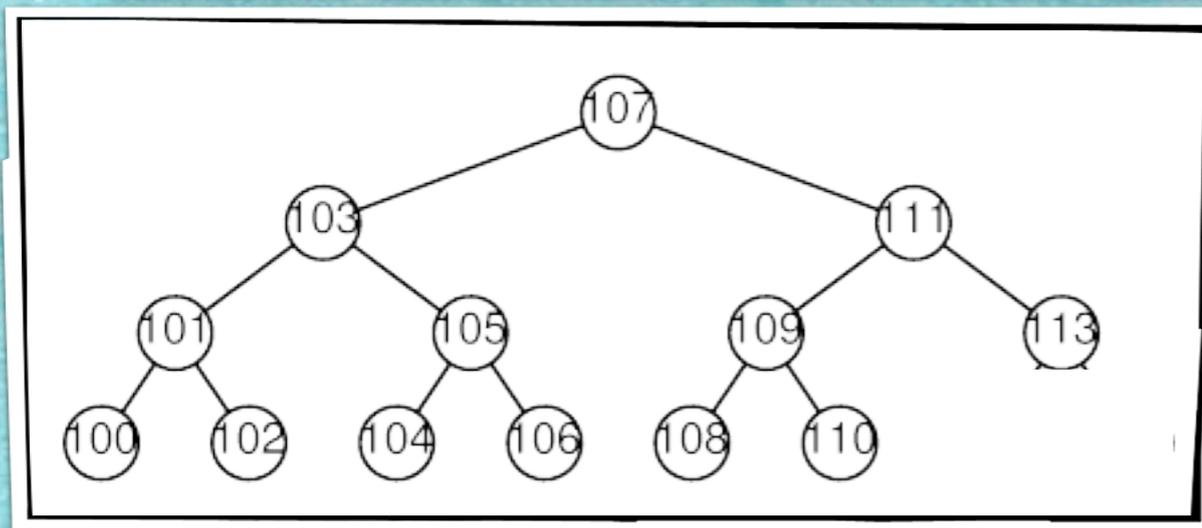
(6) Ein binärer Baum heißt vollständig, wenn zusätzlich alle Blätter gleichen Abstand zur Wurzel haben.

(7) Ein Knoten ohne Kindknoten heißt Blatt.

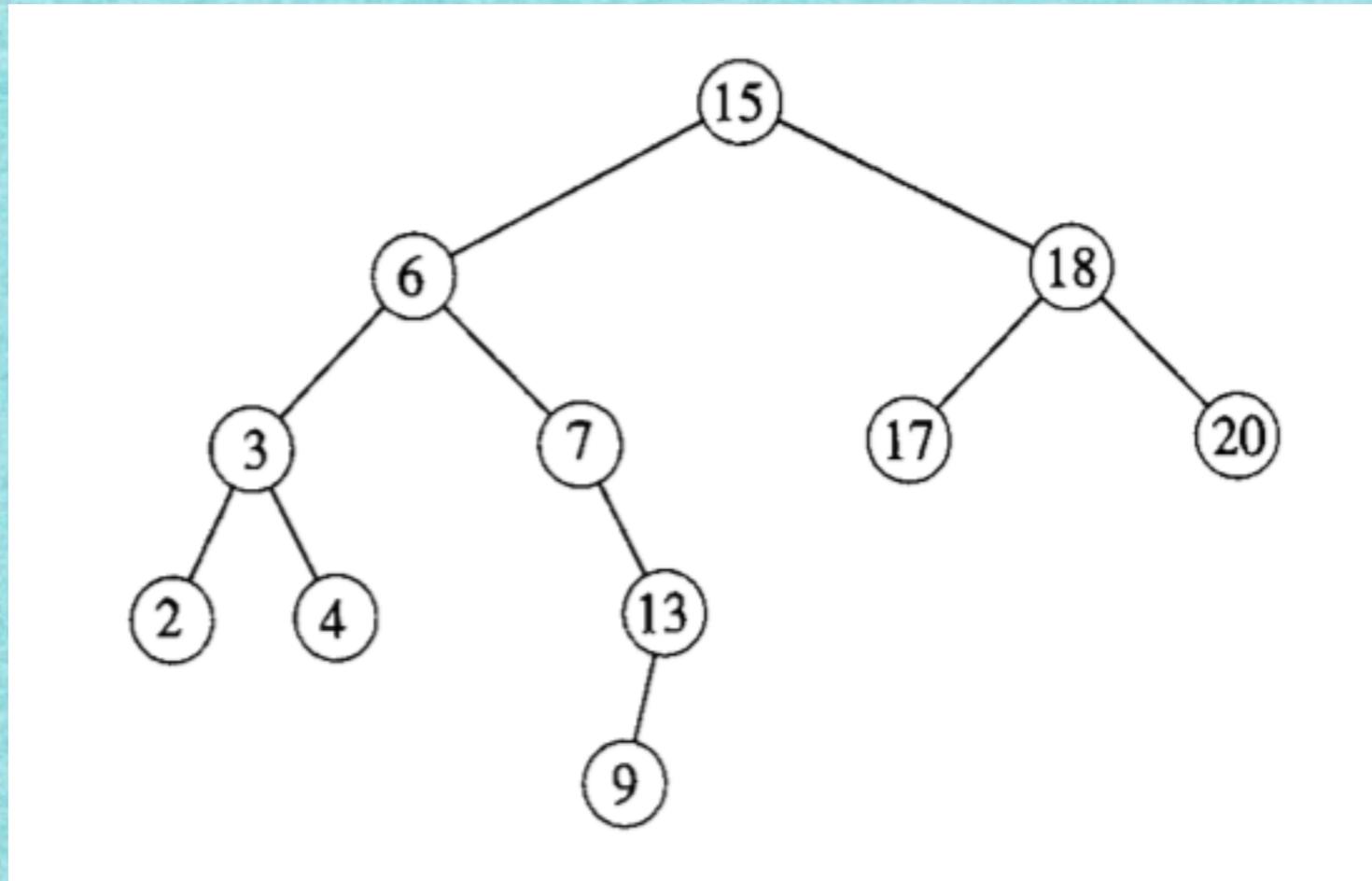
(8) Der Teilbaum eines Knotens v ist durch die Menge der von v erreichbaren Knoten und der dabei verwendeten Kanten definiert; der linke Teilbaum ist der Teilbaum von $l[v]$.

(9) In einem binären Suchbaum hat jeder Knoten v einen Schlüsselwert $S[v]$, und es gilt:

- $S[u] \leq S[v]$ für Knoten u im linken Teilbaum von v
- $S[u] > S[v]$ für Knoten u im rechten Teilbaum von v



Minimum und Maximum



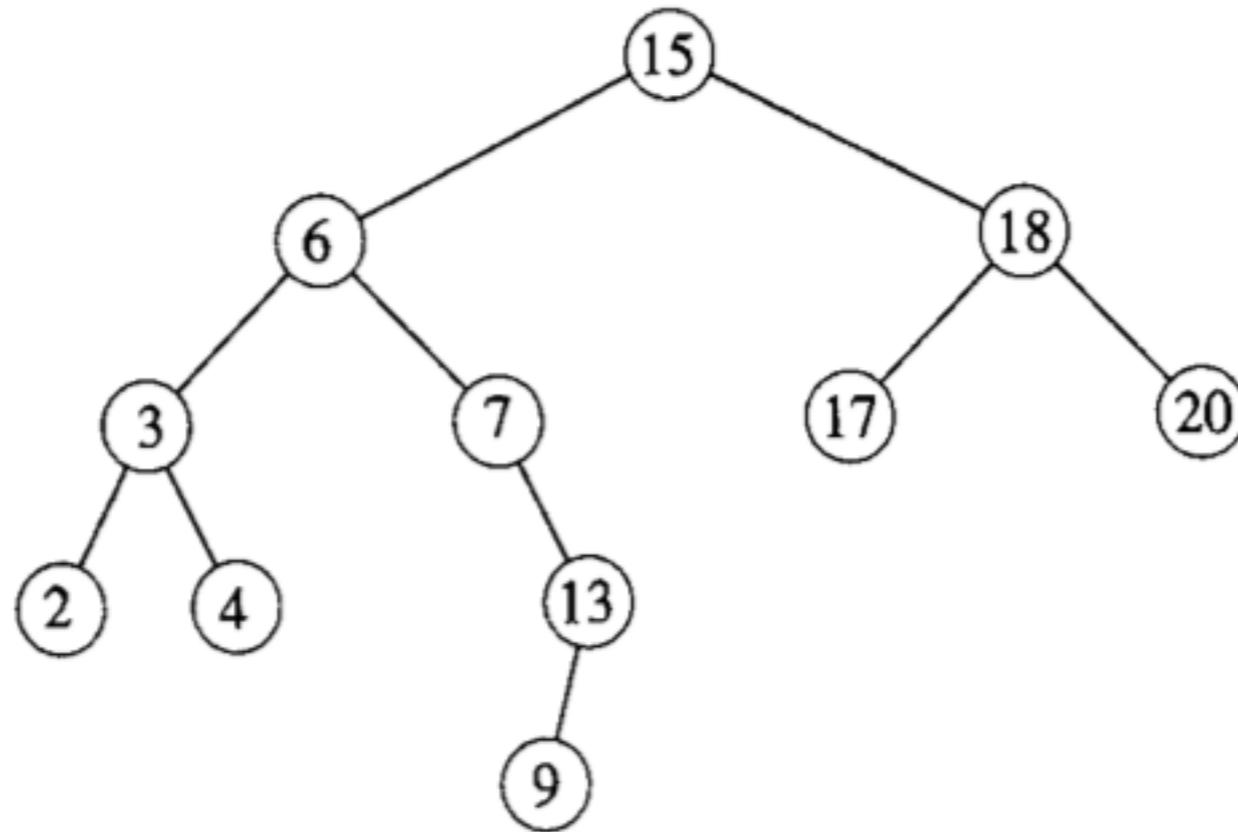
TREE-MINIMUM(x)

```
1 while  $links[x] \neq NIL$ 
2   do  $x \leftarrow links[x]$ 
3 return  $x$ 
```

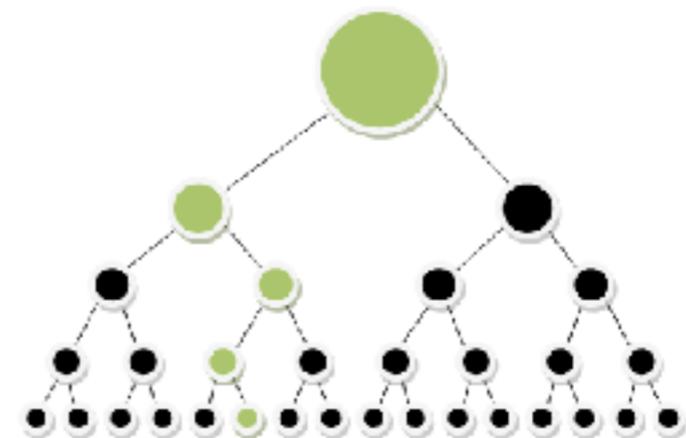
TREE-MAXIMUM(x)

```
1 while  $rechts[x] \neq NIL$ 
2   do  $x \leftarrow rechts[x]$ 
3 return  $x$ 
```

Suche im Suchbaum



```
ITERATIVE-TREE-SEARCH( $x, k$ )  
1  while  $x \neq \text{NIL}$  und  $k \neq \text{schlüssel}[x]$   
2      do if  $k < \text{schlüssel}[x]$   
3          then  $x \leftarrow \text{links}[x]$   
4          else  $x \leftarrow \text{rechts}[x]$   
5  return  $x$ 
```



4.1 Grundoperationen

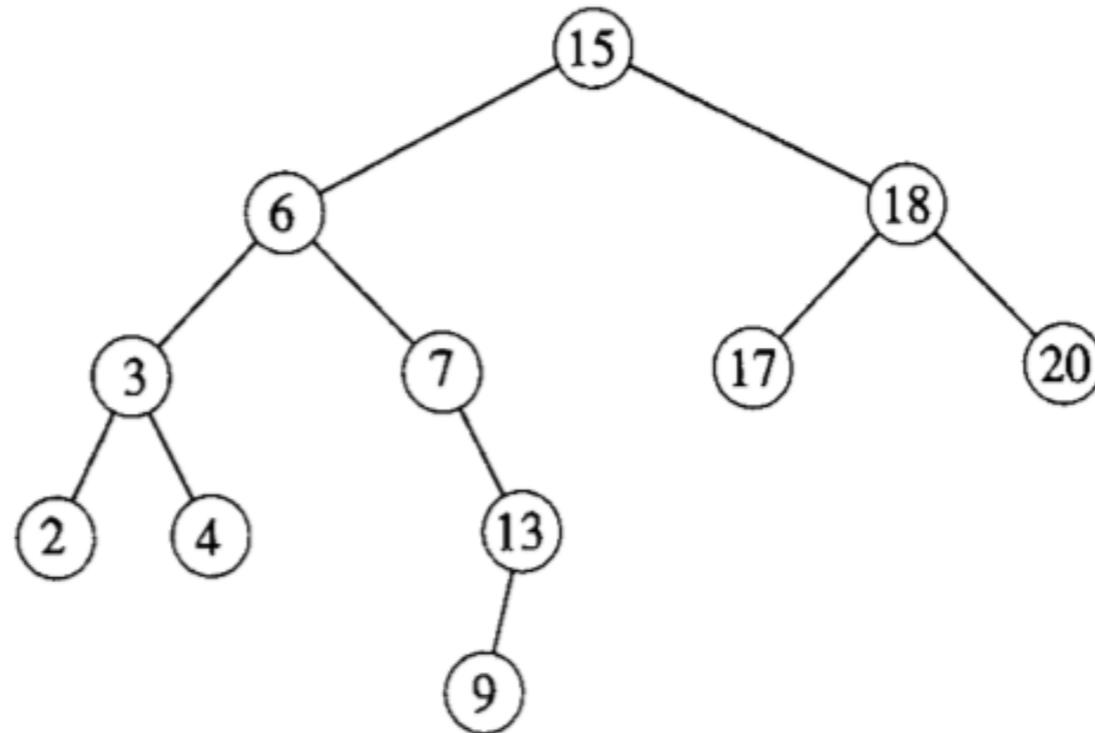
SUCCESSOR(S,x):

“Finde das nächstgrößere Element”

**Für ein in x stehendes Element in S ,
bestimme ein Element von nächstgrößerem
Wert in S .**

**Ausgabe: Zeiger y auf Element
NIL, falls x Maximum von S angibt**

Nachfolger im Suchbaum



TREE-SUCCESSOR(x)

```
1  if  $rechts[x] \neq \text{NIL}$ 
2    then return TREE-MINIMUM( $rechts[x]$ )
3   $y \leftarrow p[x]$ 
4  while  $y \neq \text{NIL}$  und  $x = rechts[y]$ 
5    do  $x \leftarrow y$ 
6     $y \leftarrow p[y]$ 
7  return  $y$ 
```

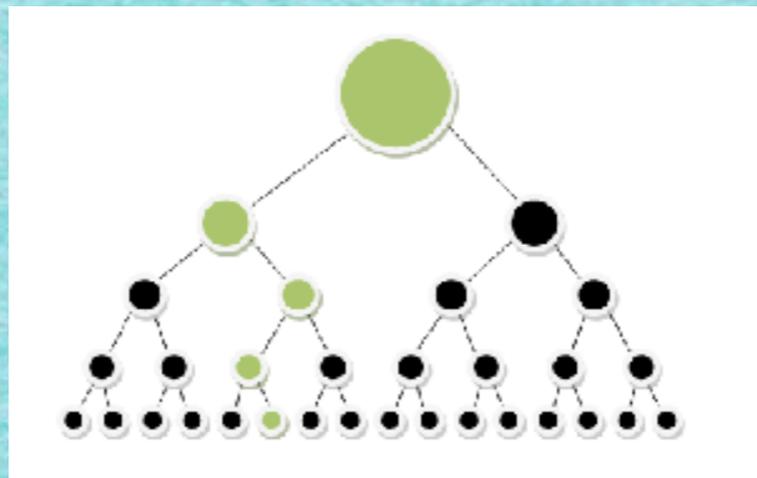
4.5 Binäre Suchbäume

Satz 4.4

Suchen, Minimum, Maximum, Nachfolger, Vorgänger können in einem binären Suchbaum der Höhe h in Zeit $O(h)$ durchlaufen werden.

Beweis:

Klar, der Baum wird nur einmal vertikal durchlaufen!

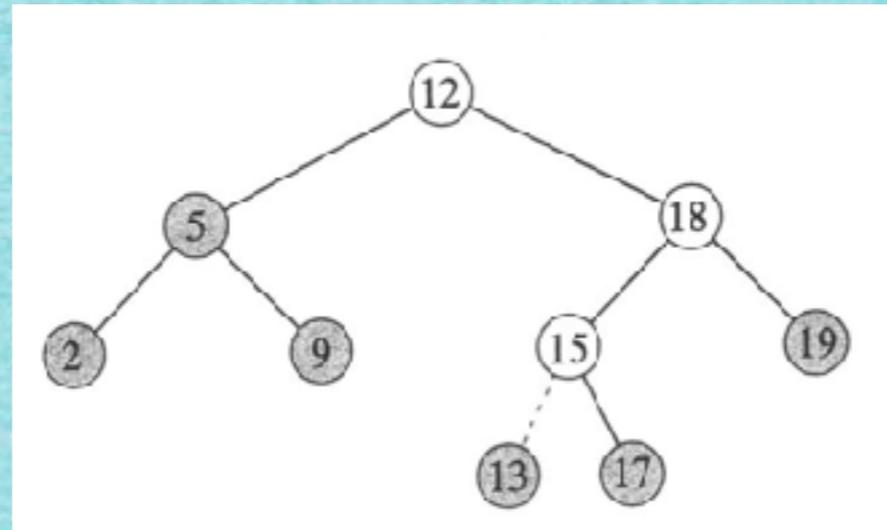


4.1 Grundoperationen

INSERT(S,x): “Füge x in S ein”

Erweitere S um das Element, das unter der Adresse x steht.

Einfügen im Suchbaum



TREE-INSERT(T, z)

```
2  $x \leftarrow \text{wurzel}[T]$ 
3 while  $x \neq \text{NIL}$ 
4   do  $y \leftarrow x$ 
5     if  $\text{schlüssel}[z] < \text{schlüssel}[x]$ 
6       then  $x \leftarrow \text{links}[x]$ 
7     else  $x \leftarrow \text{rechts}[x]$ 
8  $p[z] \leftarrow y$ 
9 if  $y = \text{NIL}$ 
10 then  $\text{wurzel}[T] \leftarrow z$ 
11 else if  $\text{schlüssel}[z] < \text{schlüssel}[y]$ 
12   then  $\text{links}[y] \leftarrow z$ 
13   else  $\text{rechts}[y] \leftarrow z$ 
```

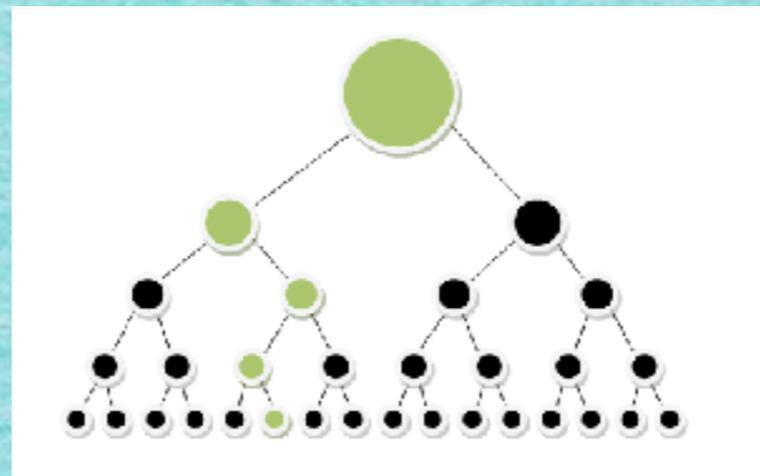
4.5 Binäre Suchbäume

Satz 4.5

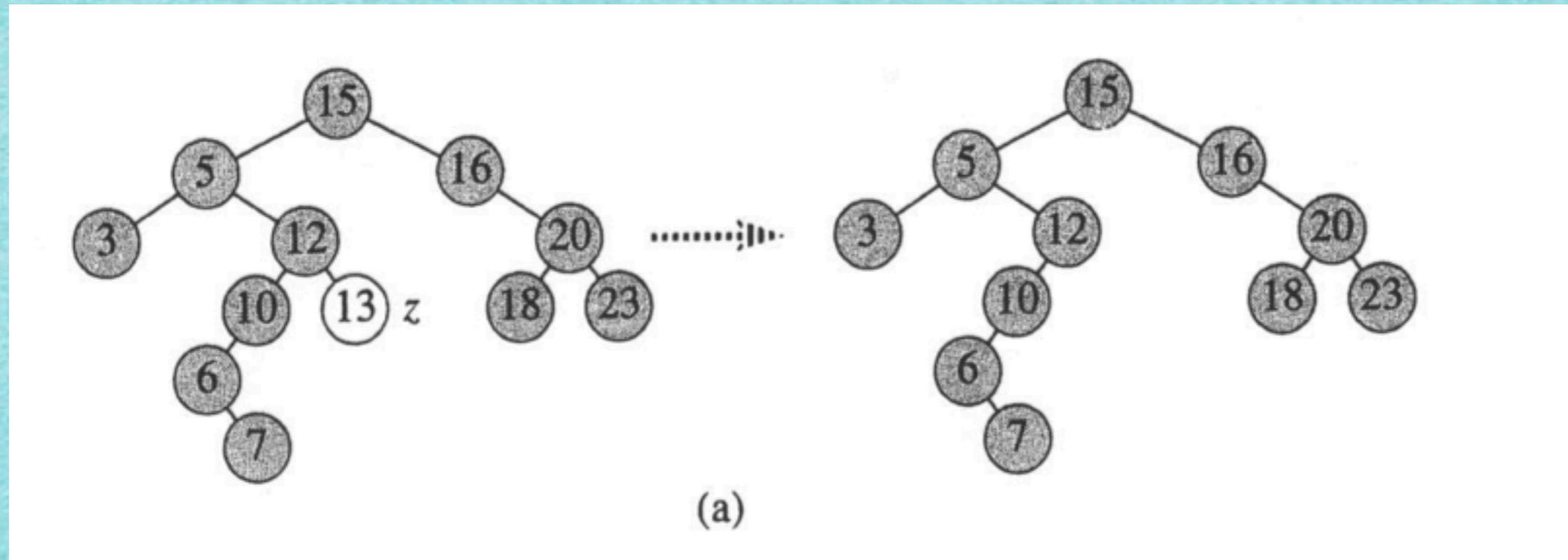
Einfügen benötigt $O(h)$ für einen binären Suchbaum der Höhe h .

Beweis:

Klar, der Baum wird nur vertikal abwärts durchlaufen!



Löschen im Suchbaum

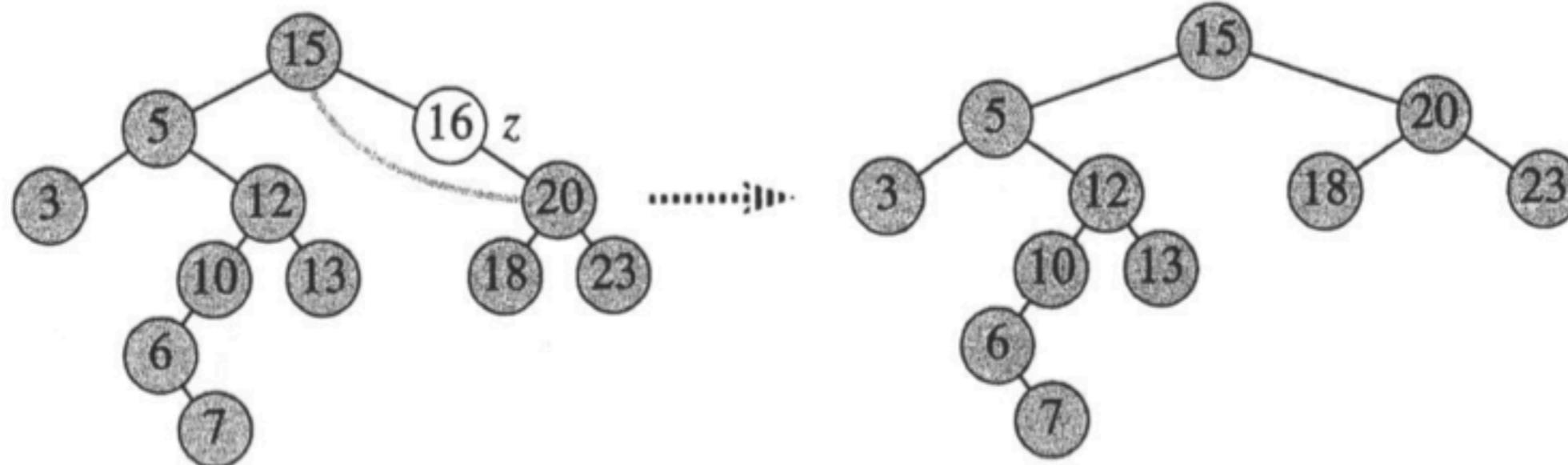


Lösche 13!

(a) Keine Kinder:

Einfach entfernen

Löschen im Suchbaum



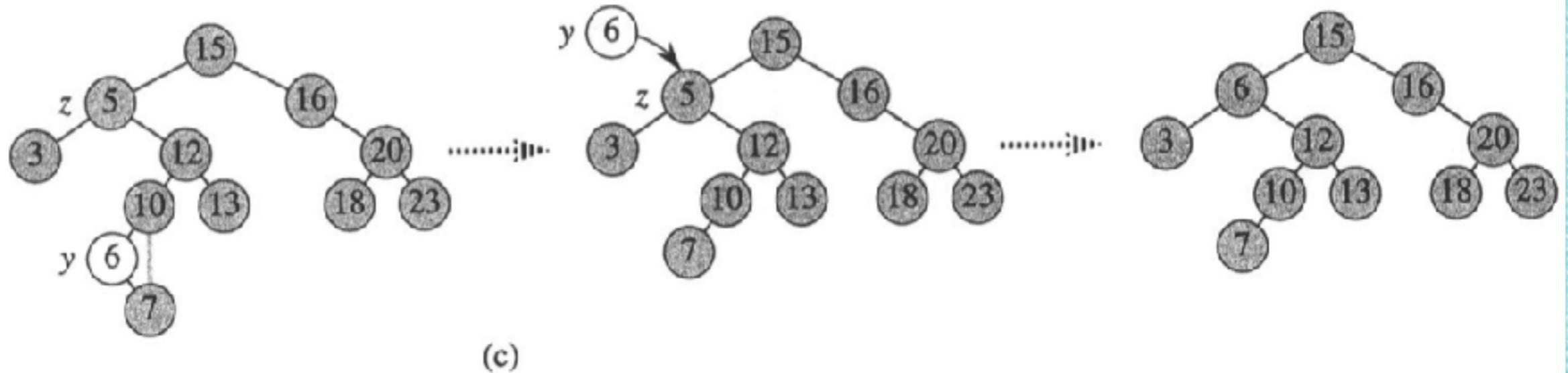
(b)

Lösche 16!

(b) Ein Kind:

“Ausschneiden”

Löschen im Suchbaum



Lösche 5!

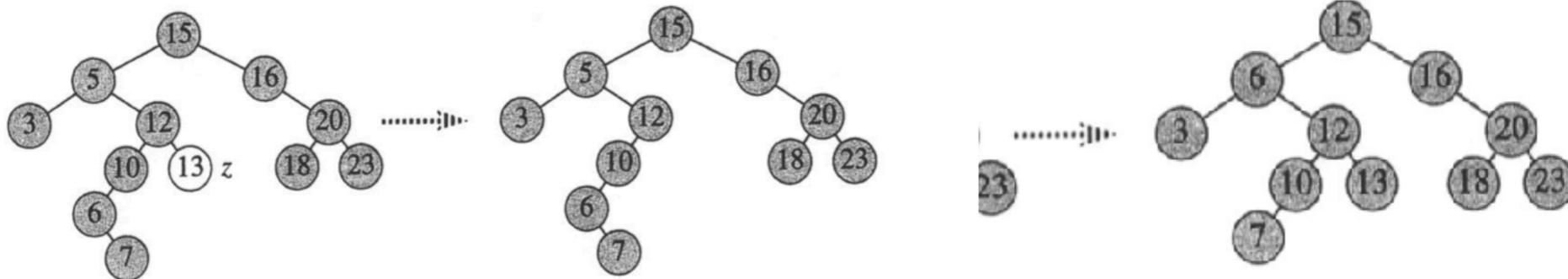
(c) Zwei Kinder:

“Nachfolger verpflanzen”

Löschen im Suchbaum

TREE-DELETE(T, z)

```
1  if  $links[z] = NIL$  oder  $rechts[z] = NIL$ 
2    then  $y \leftarrow z$ 
3    else  $y \leftarrow TREE-SUCCESSOR(z)$ 
4  if  $links[y] \neq NIL$ 
5    then  $x \leftarrow links[y]$ 
6    else  $x \leftarrow rechts[y]$ 
7  if  $x \neq NIL$ 
8    then  $p[x] \leftarrow p[y]$ 
9  if  $p[y] = NIL$ 
10   then  $wurzel[T] \leftarrow x$ 
11   else if  $y = links[p[y]]$ 
12         then  $links[p[y]] \leftarrow x$ 
13         else  $rechts[p[y]] \leftarrow x$ 
14  if  $y \neq z$ 
15   then  $schlüssel[z] \leftarrow schlüssel[y]$ 
16         kopiere die Satellitendaten von  $y$  in  $z$ 
17  return  $y$ 
```



4.5 Binäre Suchbäume

Satz 4.6

Löschen benötigt $O(h)$ für einen binären Suchbaum der Höhe h .

Beweis:

Klar, der Baum wird i.W. nur einmal durchlaufen!

4.5 Binäre Suchbäume

4.5 Binäre Suchbäume

Schnell:

4.5 Binäre Suchbäume

Schnell:

- $O(\log n)$: *logarithmische Zeit*

4.5 Binäre Suchbäume

Schnell:

- $O(\log n)$: *logarithmische Zeit*
- $O(h)$: *Tiefe des Baumes*

4.5 Binäre Suchbäume

Schnell:

- $O(\log n)$: *logarithmische Zeit*
- $O(h)$: *Tiefe des Baumes*

4.5 Binäre Suchbäume

Schnell:

- $O(\log n)$: *logarithmische Zeit*
- $O(h)$: *Tiefe des Baumes*

Also: Wie können wir die Tiefe des Baumes auf $O(\log n)$ beschränken?

Mehr demnächst!

s.fekete@tu-bs.de