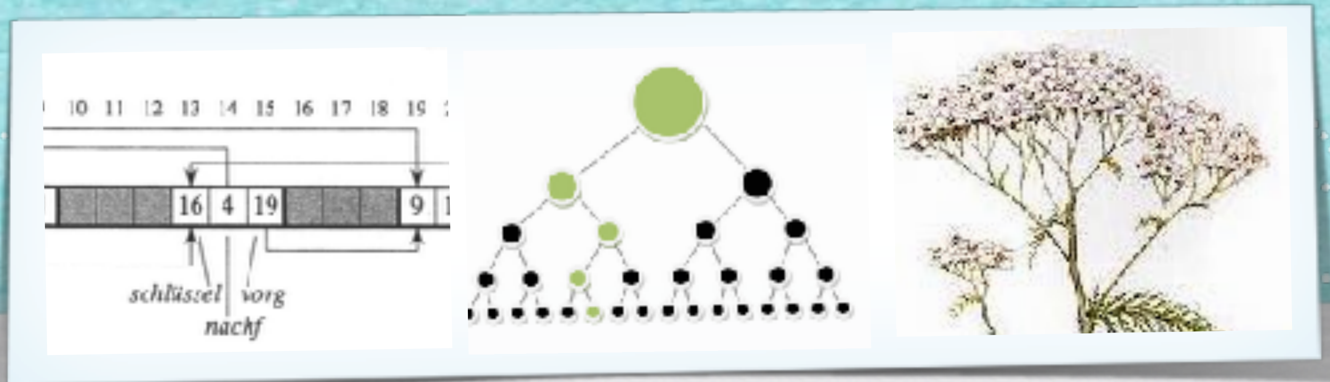
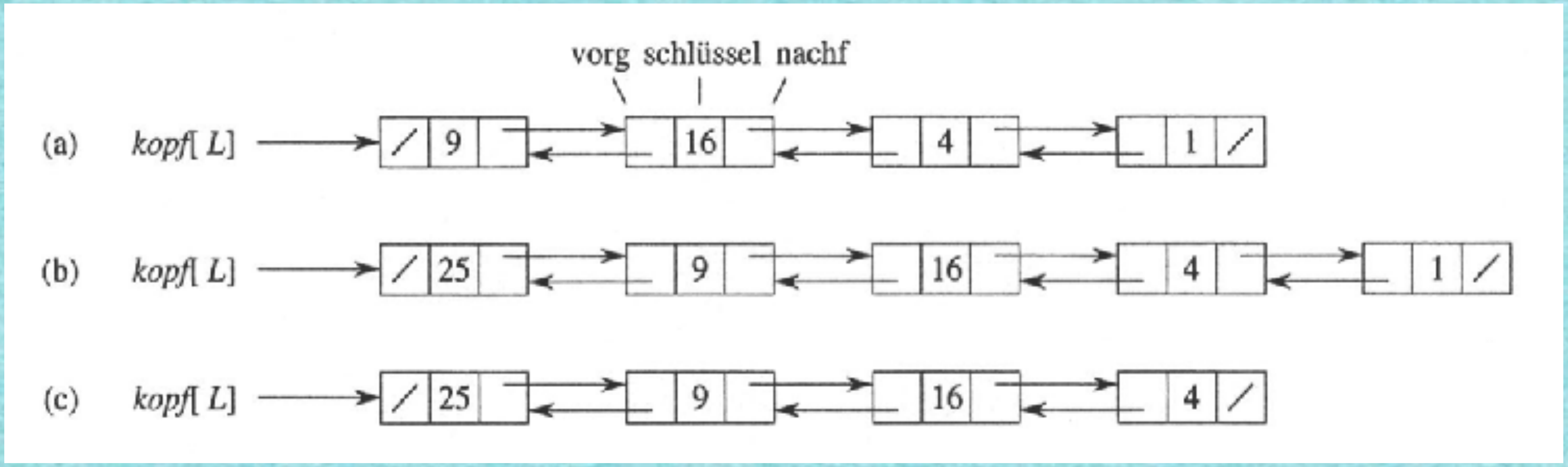


Kapitel 4: Dynamische Datenstrukturen

Algorithmen und Datenstrukturen WS 2019/20

Prof. Dr. Sándor Fekete

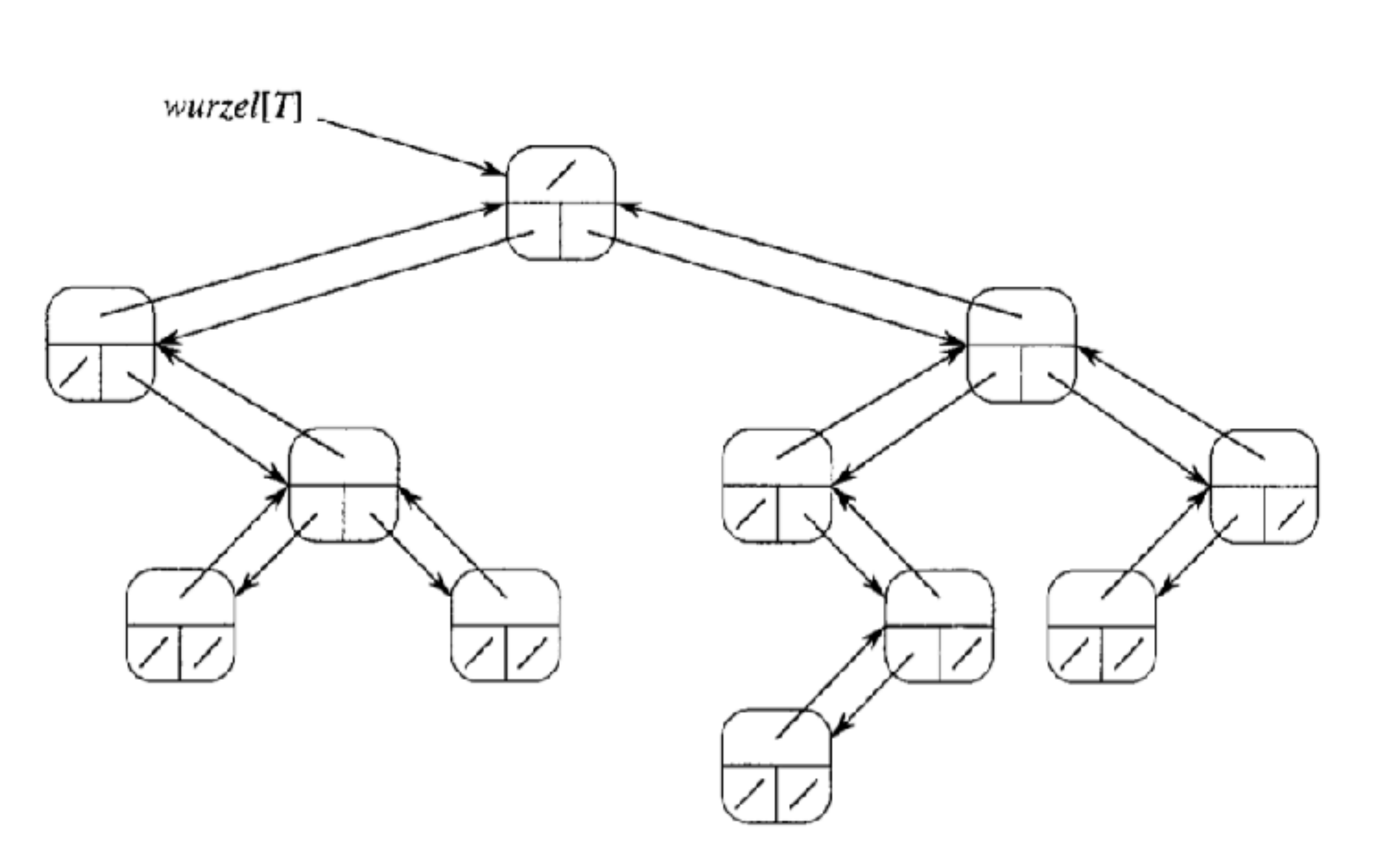
Struktur einer doppelt verketteten Liste



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

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

Binärer Suchbaum

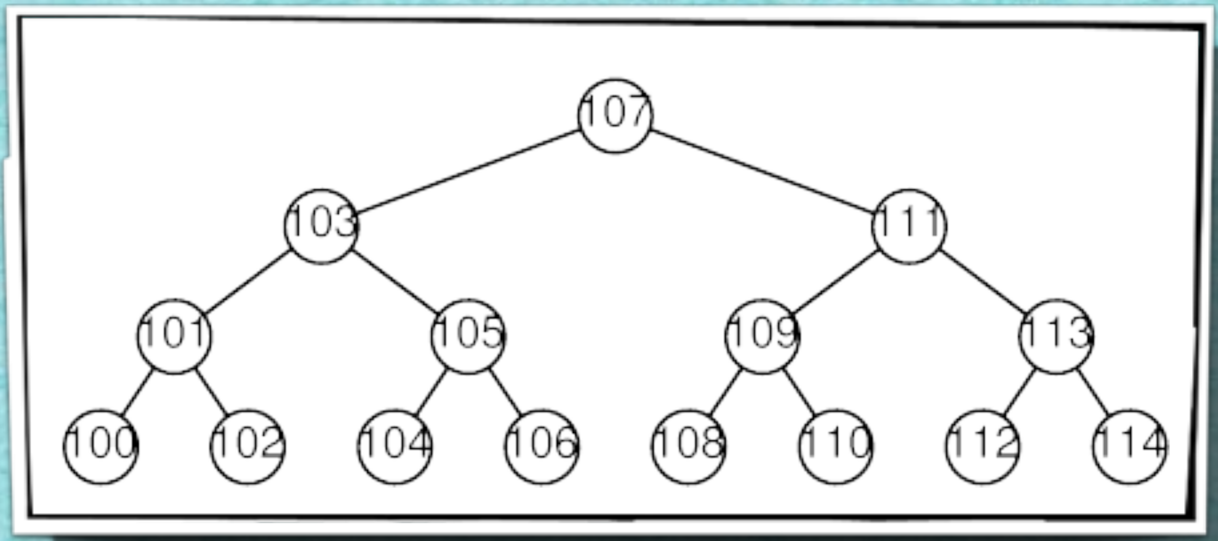


Außerdem wichtig: Struktur der Schlüsselwerte!

4.4 Binäre Suche

Aufgabenstellung:

- ***Rate eine Zahl zwischen 100 und 114!***



Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT, linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {

    1.1. MITTE:= $\left\lfloor \frac{\text{LINKS + RECHTS}}{2} \right\rfloor$

    1.2. IF (S[MITTE]=WERT) THEN

        1.2.1. RETURN MITTE

    1.3. ELSEIF (S[MITTE]>WERT) THEN

        1.3.1. RECHTS:=MITTE-1

    1.4. ELSEIF

        1.4.1. LINKS:=MITTE+1

    }

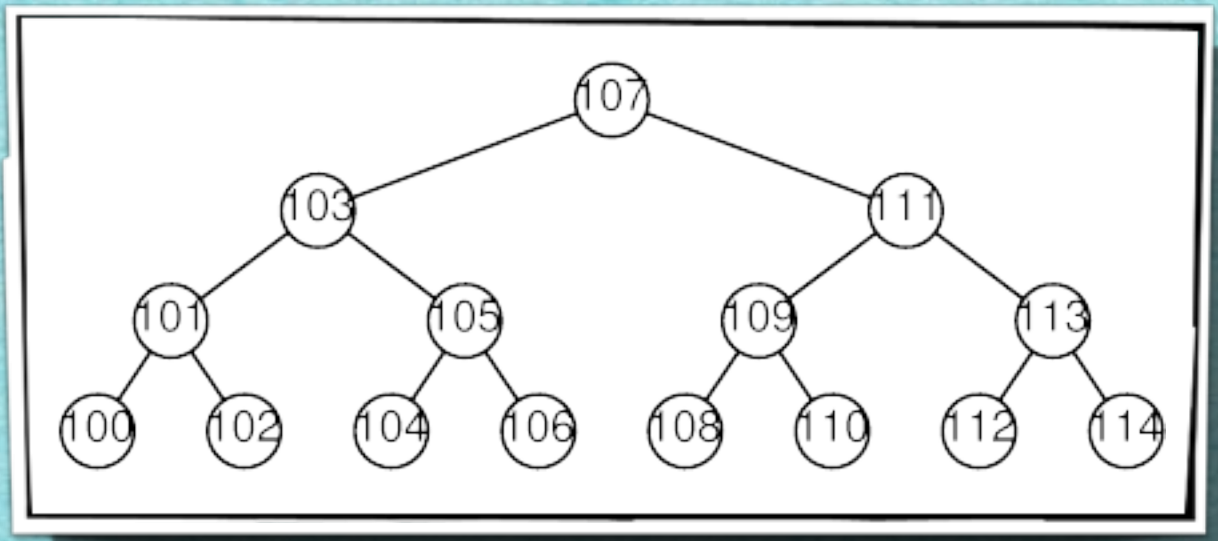
2. RETURN "WERT nicht gefunden!"

Satz 4.2

Die binäre Suche terminiert in O(log(RECHTS-LINKS)) Schritten (für RECHTS>LINKS).

Beweis:

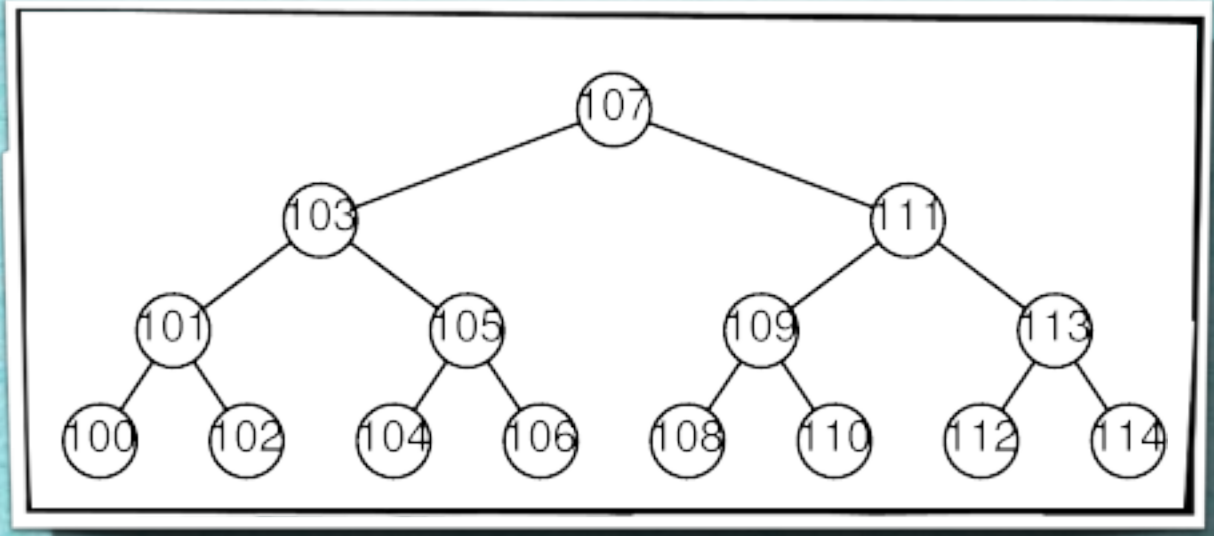
Selbst!



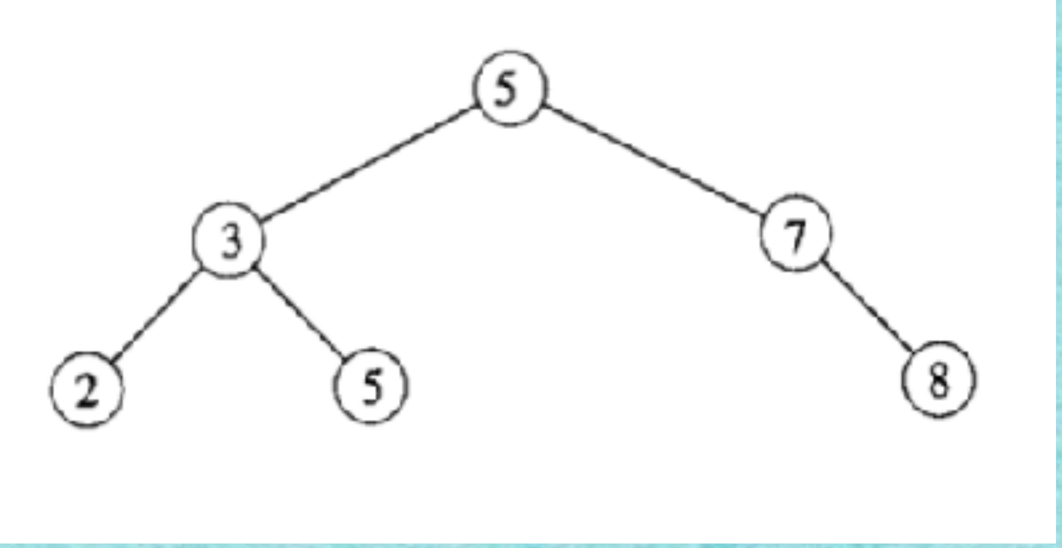
4.5 Binäre Suchbäume

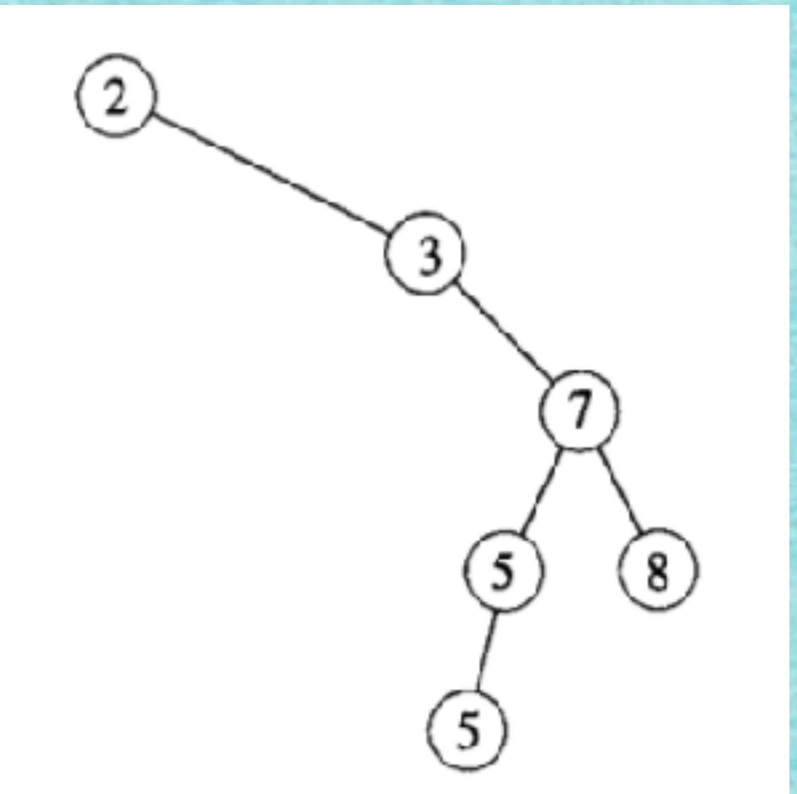
Ideen:

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



Ordnungsstruktur





Linker Teilbaum: Kleinere (bzw. nicht größere) Zahlen
Rechter Teilbaum: Größere Zahlen

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.)

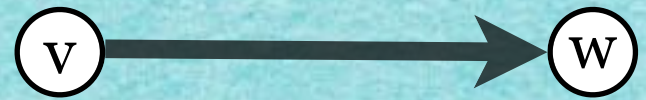
(2) Ein ***gerichteter Baum*** B=(V,T) hat folgende Eigenschaften:

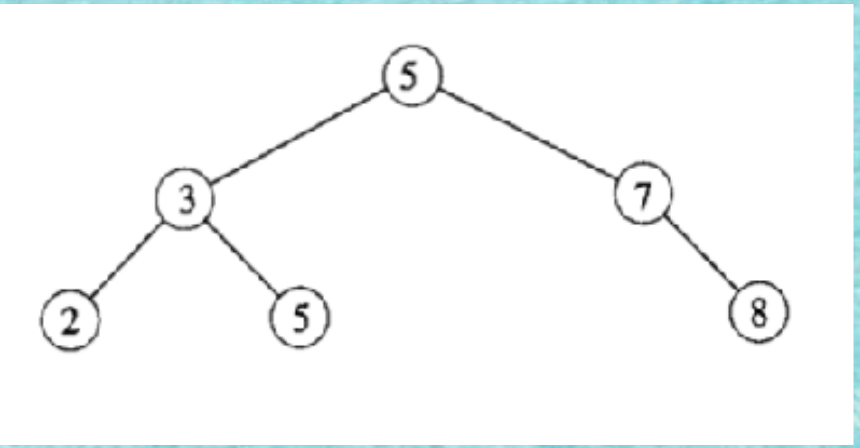
(i) Es gibt einen eindeutigen Knoten w ohne Vorgänger.

(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].





Definition 4.3 (Forts.)

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

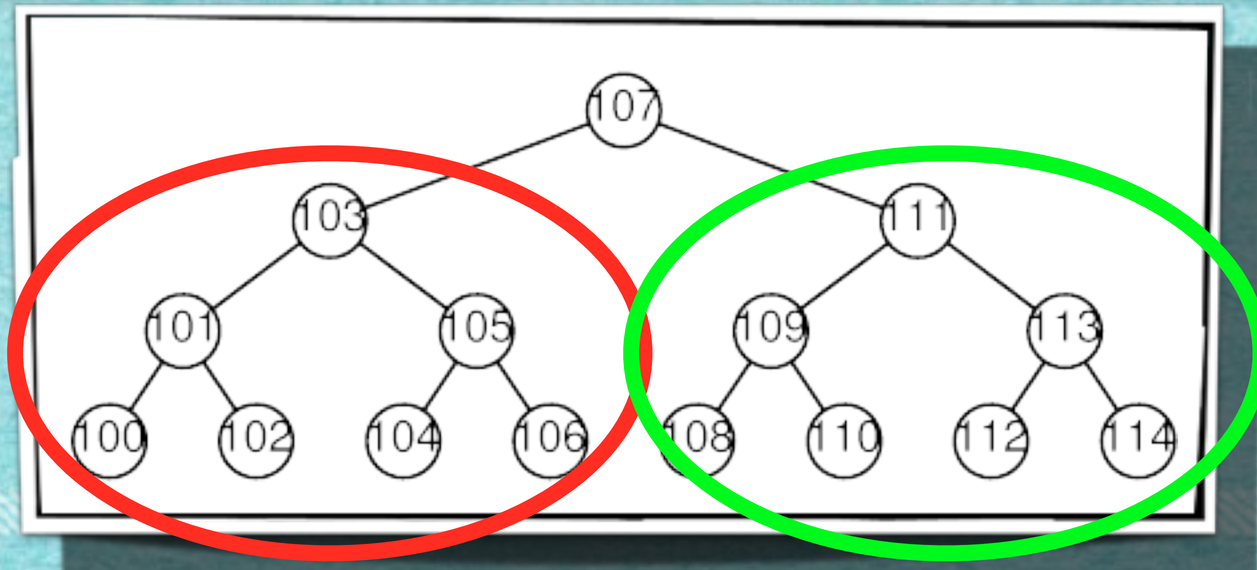
(6) Ein binärer Baum heißt *vollständig*, wenn zusätzlich alle Blätter gleichen Abstand zur Wurzel haben.

(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]≤S[v] für Knoten u im linken Teilbaum von v**
- **S[u]>S[v] für Knoten u im rechten Teilbaum von v**



Dieser Baum ist vollständig; wenn z.B. die Knoten 112 und 114 fehlen, dann ist er voll, aber nicht vollständig.

<u>MINIMUM(S):</u> **"Suche das Minimum in S"**

Finde in S ein Element von kleinstem Wert. (Annahme: Die Werte lassen sich vollständig vergleichen!)

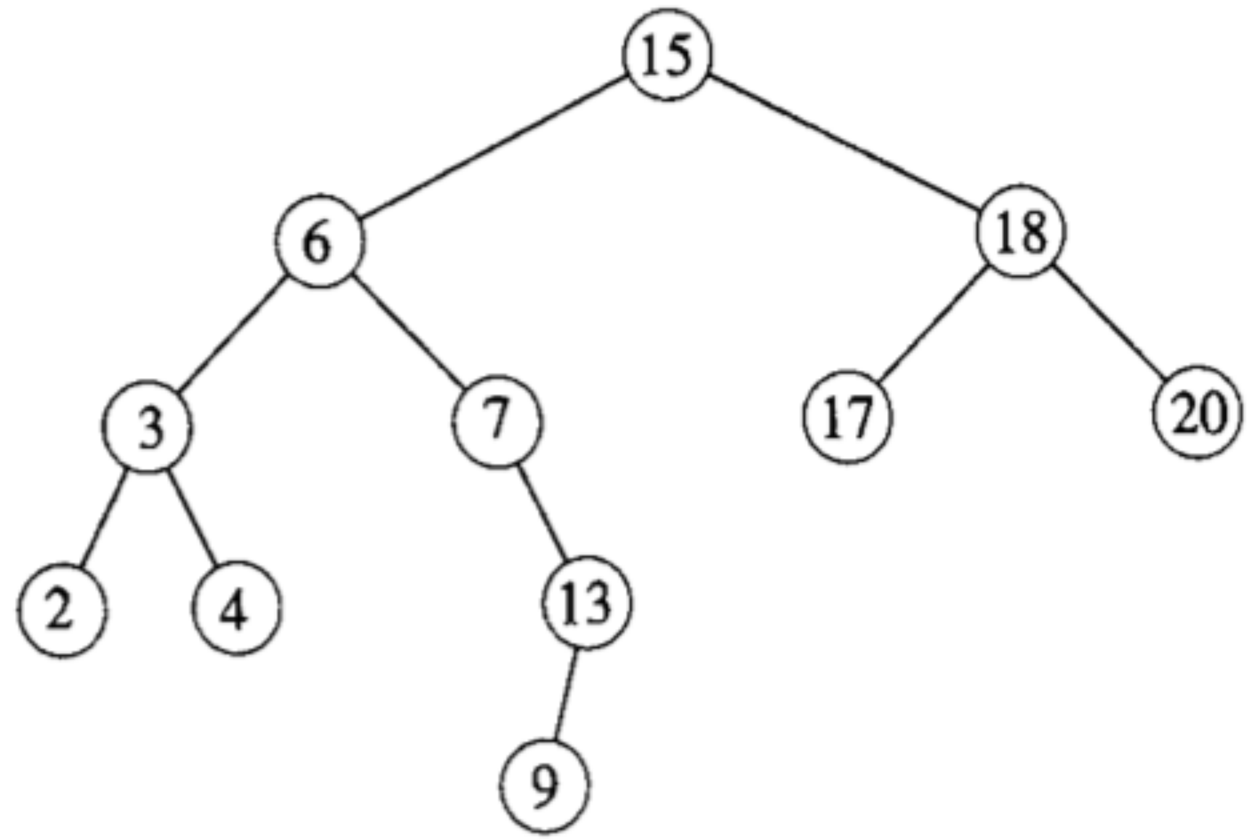
Ausgabe: Zeiger x auf solch ein Element

<u>MAXIMUM(S):</u> **"Suche das Maximum in S"**

Finde in S ein Element von größtem Wert. (Annahme: Die Werte lassen sich vollständig vergleichen!)

Ausgabe: Zeiger x auf solch ein Element

Minimum und Maximum




```
TREE-MINIMUM(x)
while links[x] ≠ NIL
      do x ← links[x]
return x
```

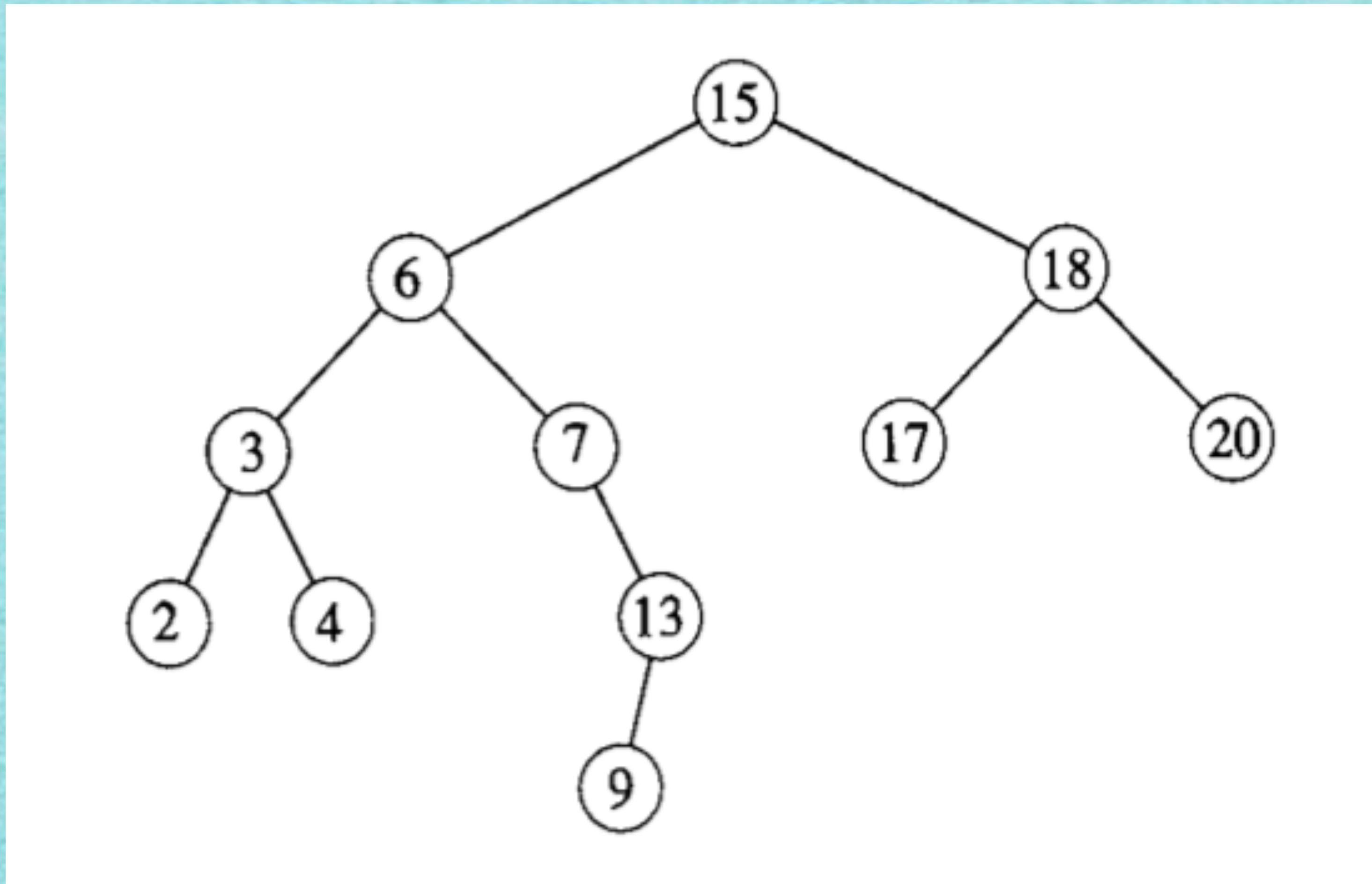
```
TREE-MAXIMUM(x)
while rechts[x] ≠ NIL
      do x ← rechts[x]
return x
```

<u>SEARCH(S,k):</u> **“Suche in S nach k”**

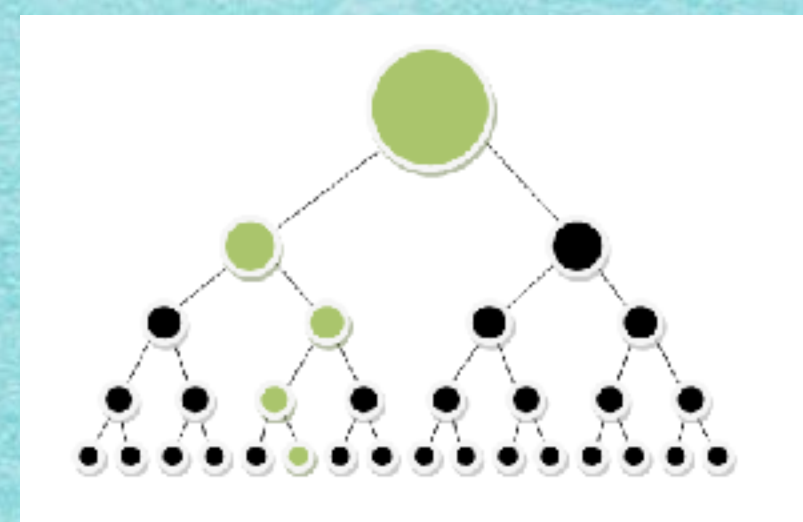
Durchsuche die Menge S nach einem Element von Wert k.

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

Suche im Suchbaum




```
ITERATIVE-TREE-SEARCH(x, k)
while x ≠ NIL und k ≠ schlüssel[x]
        do if k < schlüssel[x]
                then x ← links[x]
                else   x ← rechts[x]
return x
```

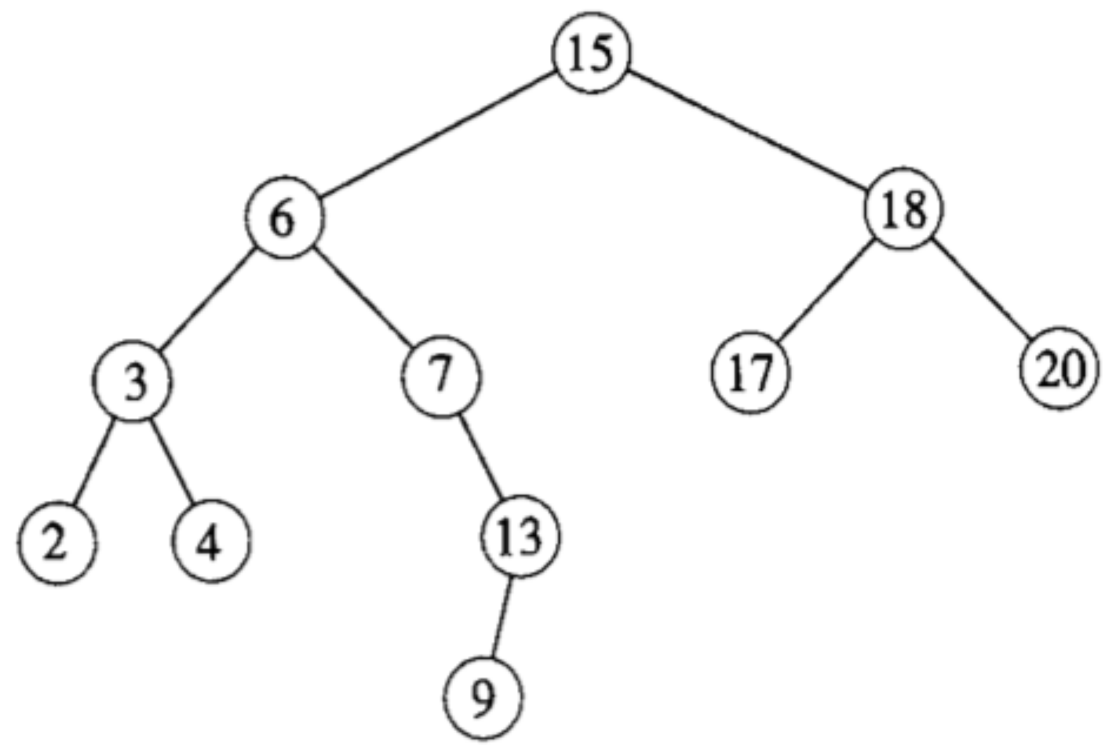


<u>SUCCESSOR(S,x):</u>
“Finde das nächstgrößere Element”

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



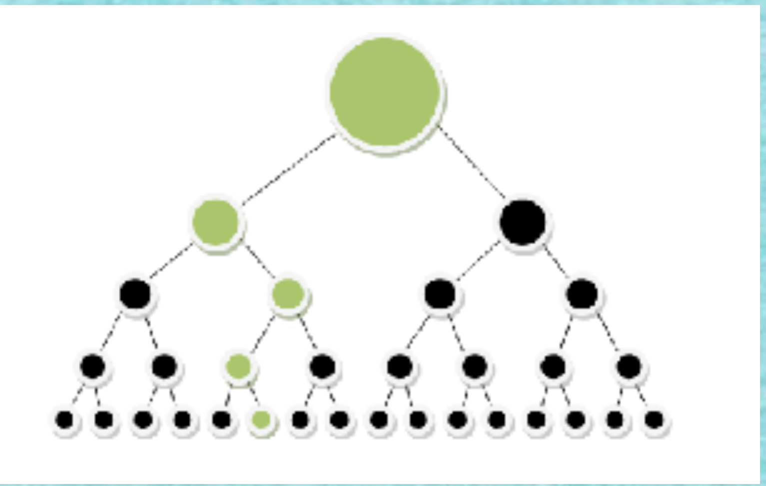

```
TREE-SUCCESSOR(x)
if rechts[x] ≠ NIL
    then return TREE-MINIMUM(rechts[x])
y ← p[x]
while y ≠ NIL und x = rechts[y]
      do x ← y
         y ← p[y]
return y
```

Satz 4.4

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

Beweis:

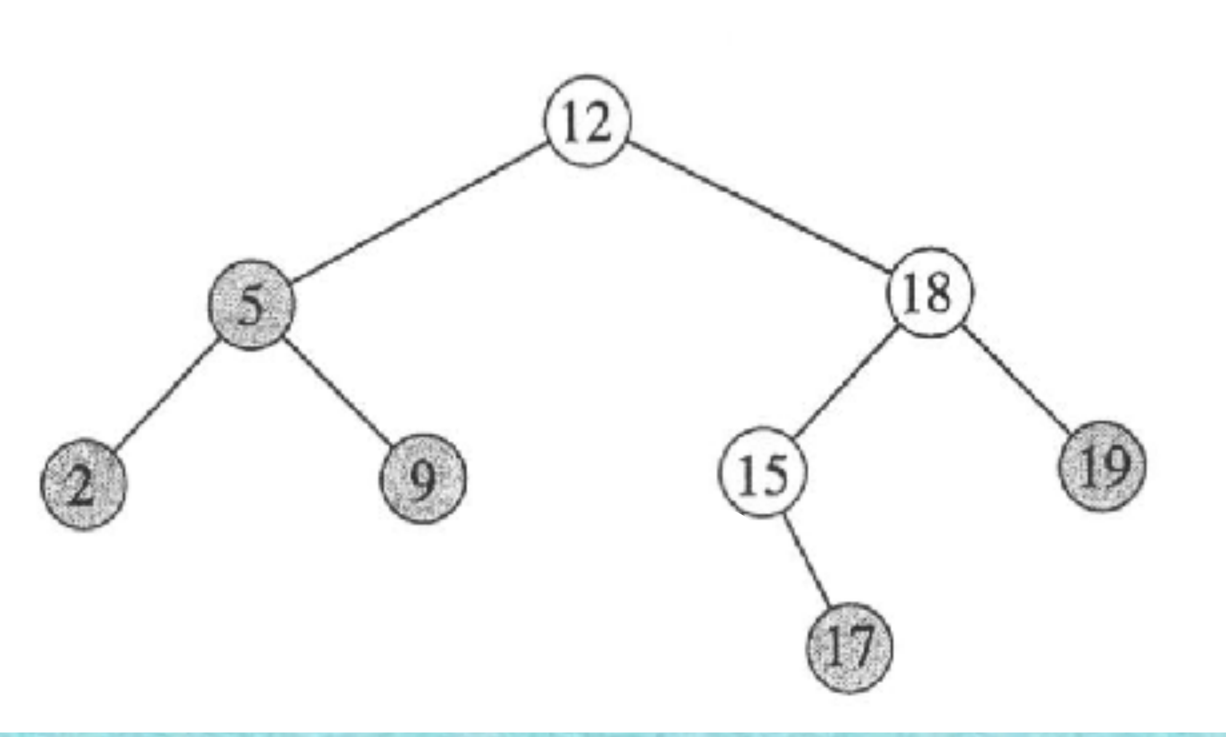
Klar, der Baum wird nur einmal vertikal durchlaufen!



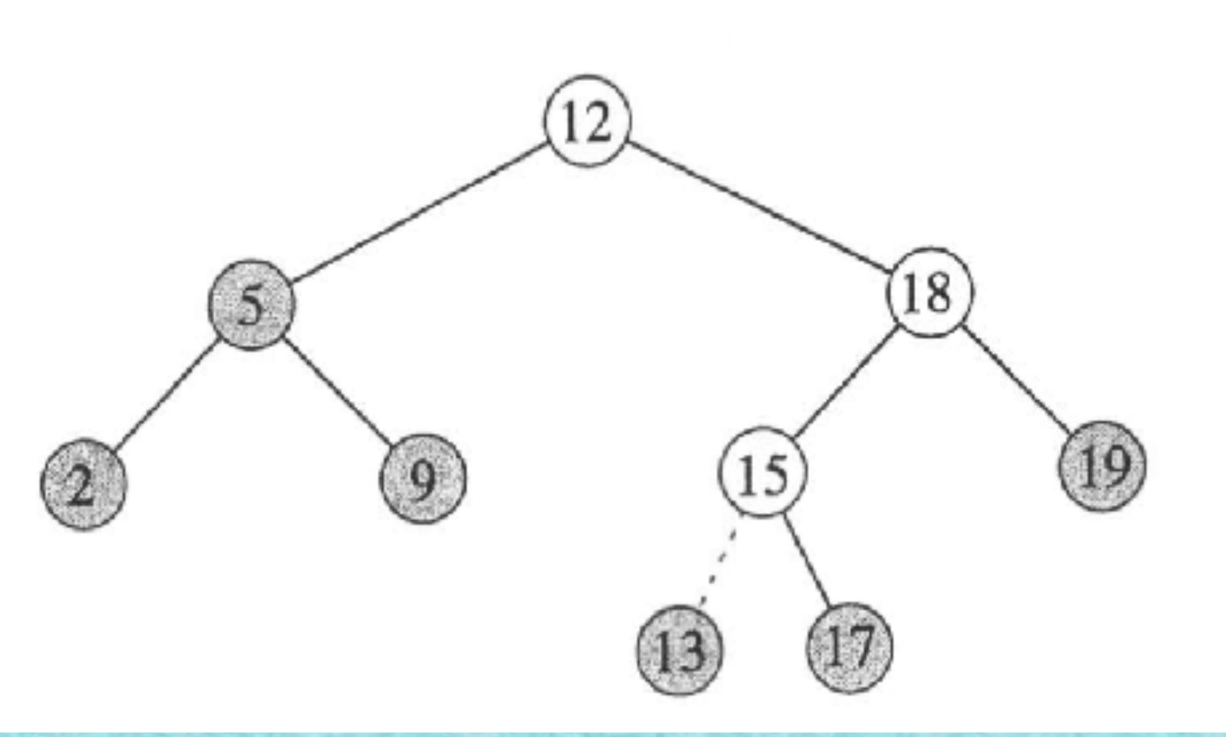
<u>INSERT(S,x):</u> **“Füge x in S ein”**

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

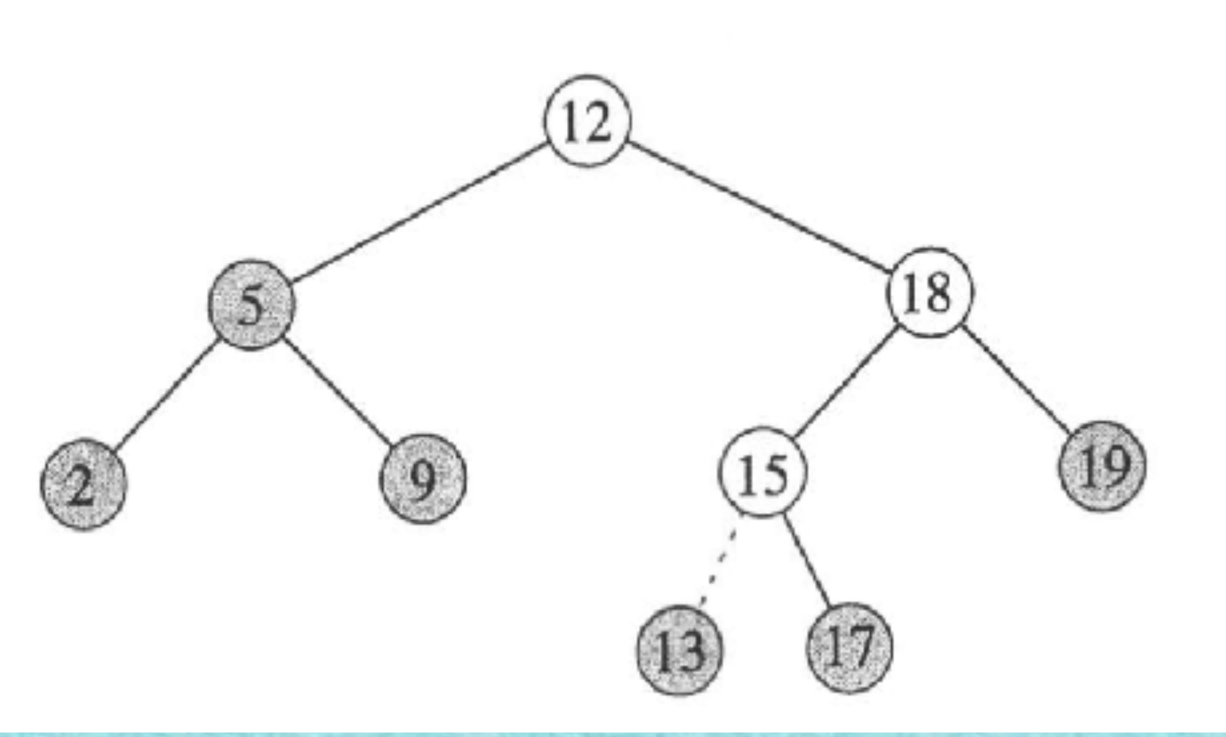
Einfügen im Suchbaum



Einfügen im Suchbaum

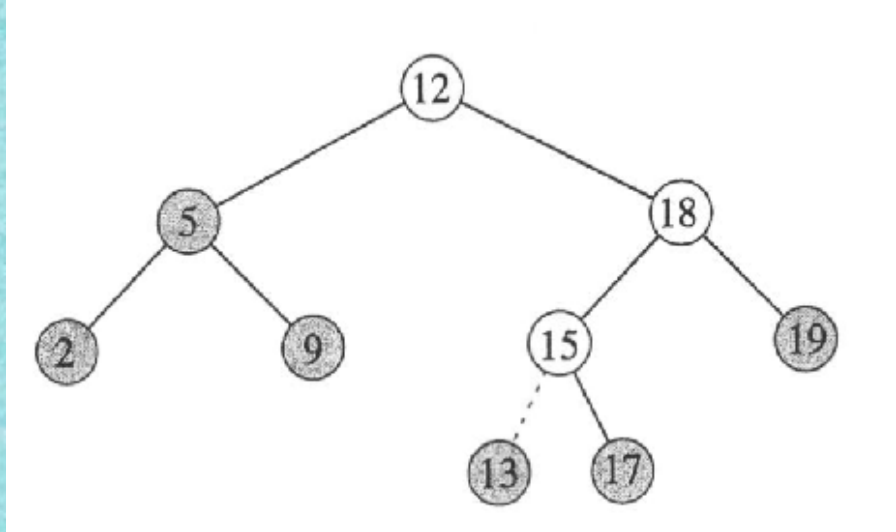


Einfügen im Suchbaum



Füge 13 ein!

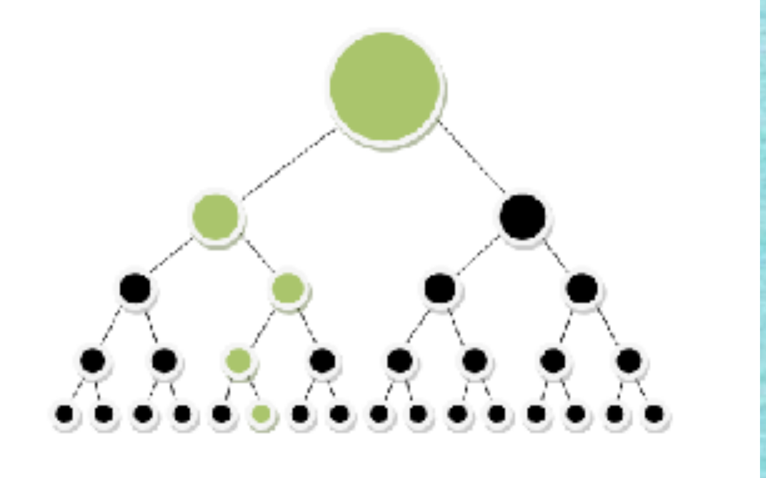
Einfügen im Suchbaum




```
TREE-INSERT(T, z)
y ← NIL
x ← wurzel[T]
while x ≠ NIL
        do y ← x
           if schlüssel[z] < schlüssel[x]
              then x ← links[x]
              else x ← rechts[x]
p[z] ← y
if y = NIL
   then wurzel[T] ← z                          ▷ Baum T war leer
   else if schlüssel[z] < schlüssel[y]
           then links[y] ← z
           else rechts[y] ← z
```

<u>Satz 4.5</u>
Einfügen benötigt O(h) für einen binären Suchbaum der Höhe h.

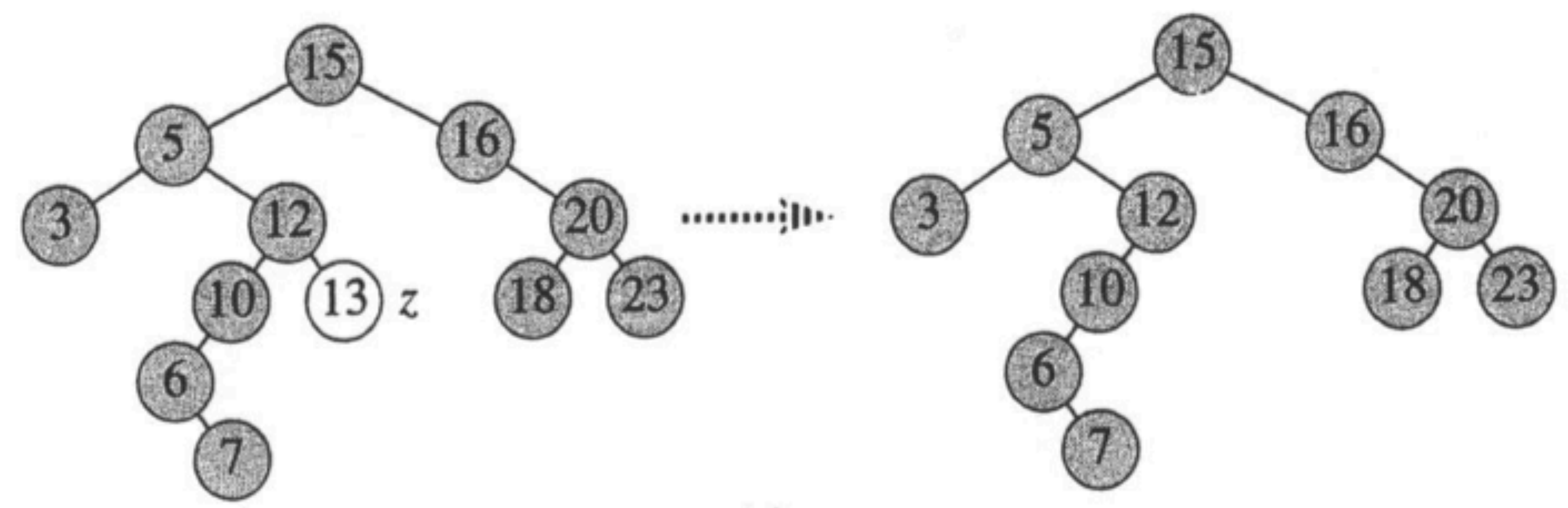
<u>Beweis:</u>
Klar, der Baum wird nur vertikal abwärts durchlaufen!



<u>DELETE(S,x):</u> **“Entferne x aus S”**

Lösche das unter der Adresse x stehende Element aus der Menge S.

Löschen im Suchbaum



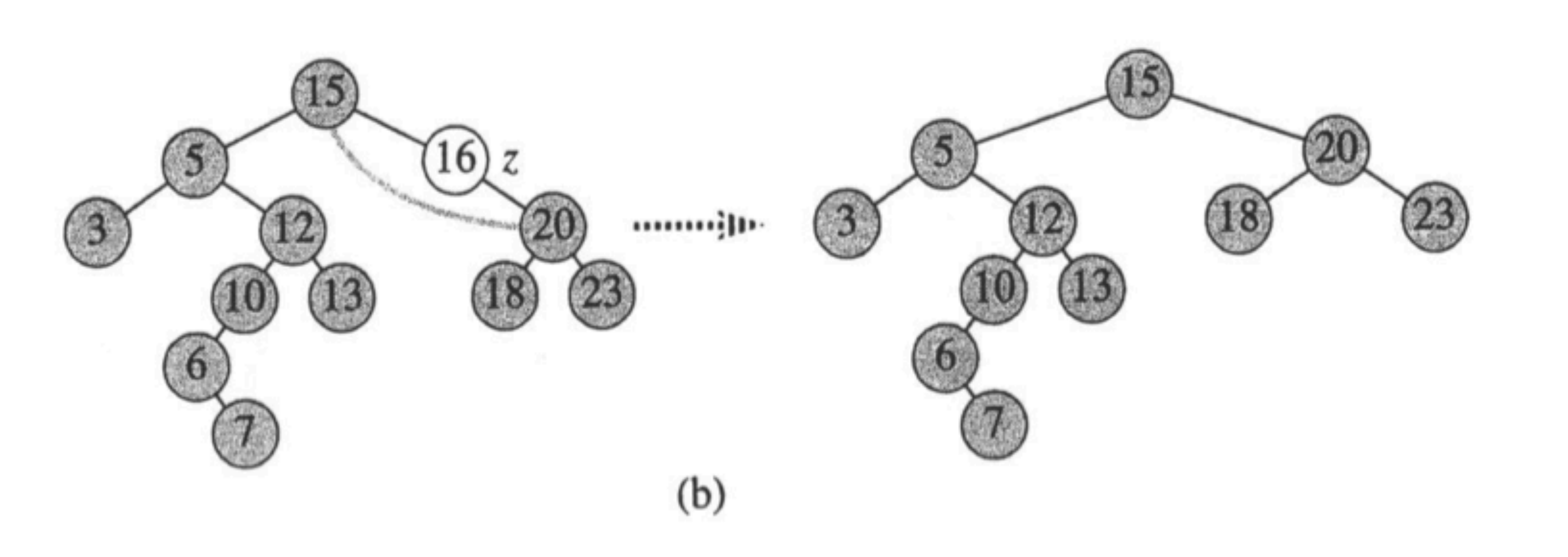
(a)

Lösche 13!

(a) Keine Kinder:

Einfach entfernen

Löschen im Suchbaum

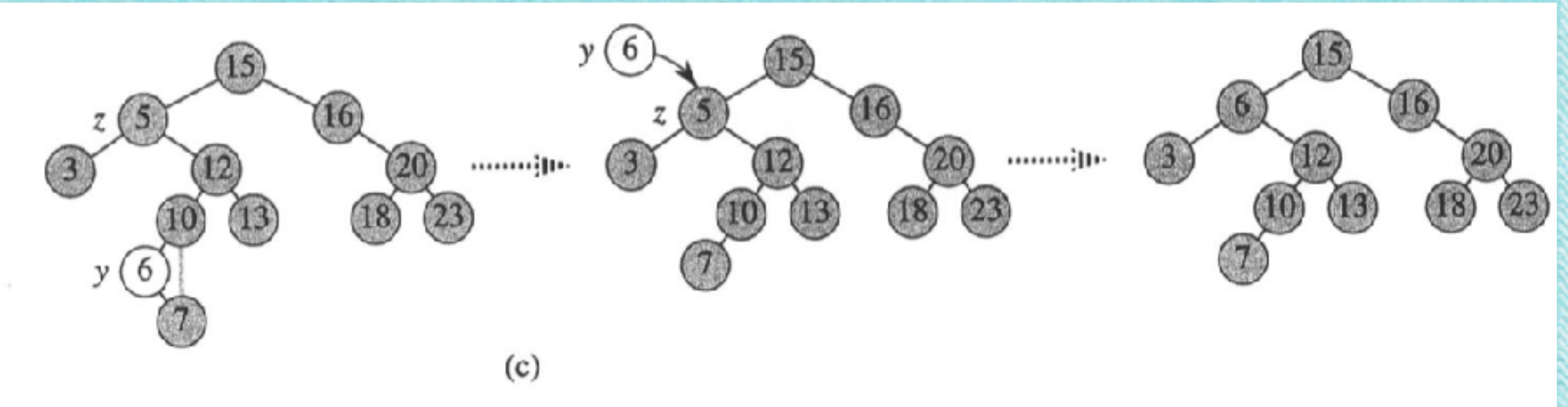


Lösche 16!

(b) Ein Kind:

"Ausschneiden"

Löschen im Suchbaum



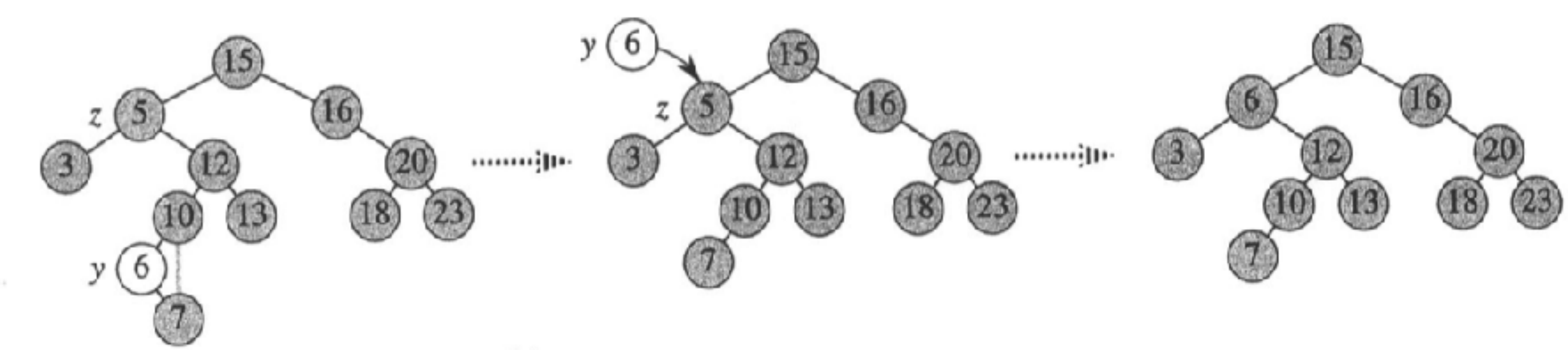
Lösche 5!

(c) Zwei Kinder:

"Nachfolger verpflanzen"

Löschen im Suchbaum

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



(c)

<u>Satz 4.6</u>
Löschen benötigt O(h) für einen binären Suchbaum der Höhe h.

<u>Beweis:</u>
Klar, der Baum wird i.W. nur einmal durchlaufen!

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

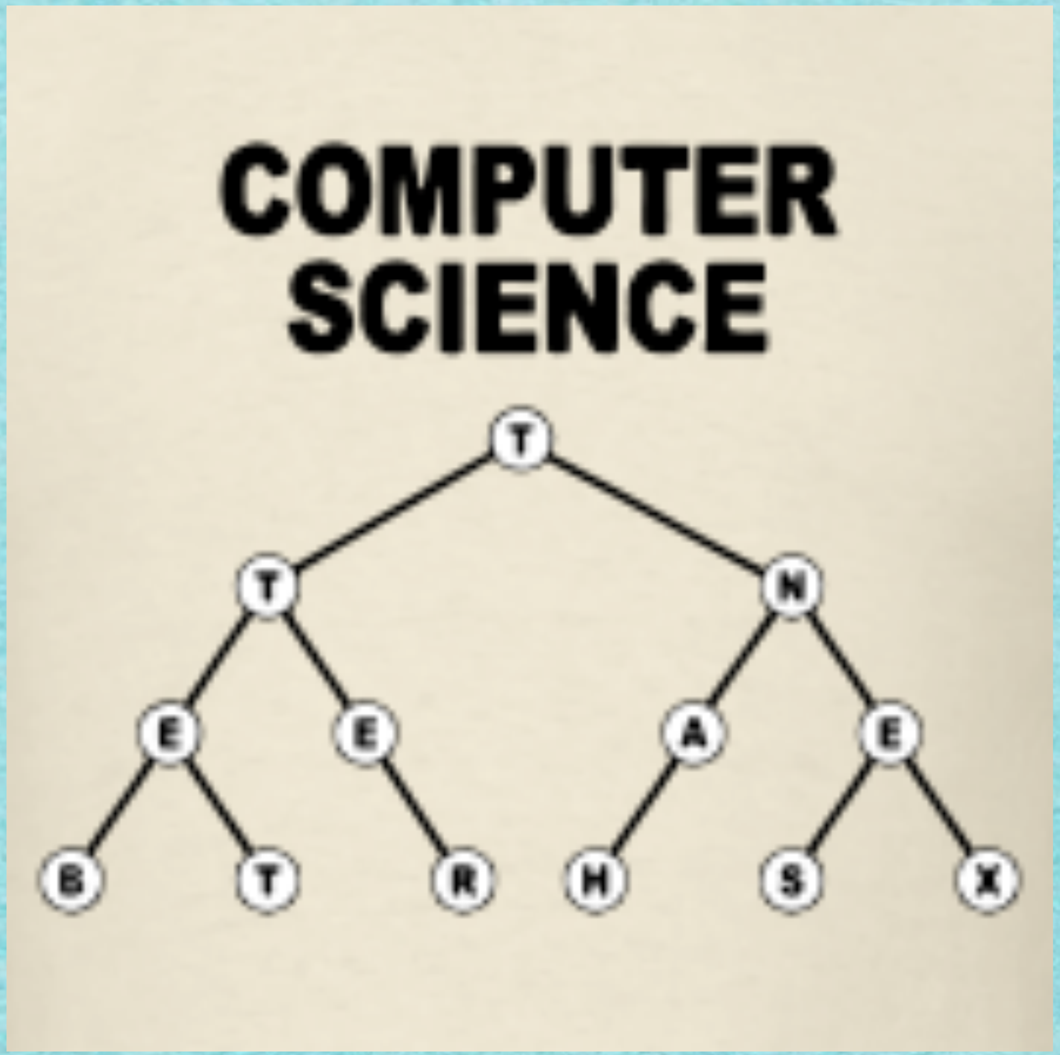
4.5 Binäre Suchbäume

Schnell:

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

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

4.5 Binäre Suchbäume



Mehr demnächst!

s.fekete@tu-bs.de