

- (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. Wir nennen einen davon den linken  $l(i)$ , den anderen den rechten  $r(i)$ .
- (5) Ein binärer Baum heißt voll, wenn jeder Knoten zwei oder keinen Kindknoten hat.
- (6) Ein Knoten ohne Kindknoten heißt Blatt.
- (7) Der Teilbaum eines Knotens ist durch die Menge der <sup>von</sup>erreichbaren Knoten und der dabei verwendeten Kanten definiert; der linke Teilbaum ist der Teilbaum von  $l[i]$ .
- (8) In einem ~~Suchbaum~~ binären Suchbaum hat jeder Knoten  $i$  einen Schlüsselwert  $S[i]$ , und es gilt:  
 - Wenn  $j$  ein Knoten im linken Teilbaum ist, gilt  $S[j] \leq S[i]$ .  
 - Wenn  $j$  ein Knoten im rechten Teilbaum ist, gilt  $S[i] \leq S[j]$ .

~~Binäre Suchbäume~~ Binäre Suchbäume als Datenstruktur?!

12.12.07

- Suchen
- Minimum/Maximum bestimmen
- Nachfolger/Vorgänger bestimmen
- Einfügen
- Löschen

Iterative Baumsuche  $(i, k)$

```

1 WHILE  $(i \neq \text{NIL} \text{ und } k \neq S[i])$  DO
2   IF  $k < S[i]$ 
3     THEN  $i := l[i]$ 
4     ELSE  $i := r[i]$ 
5 RETURN  $i$ 

```

Minimum:

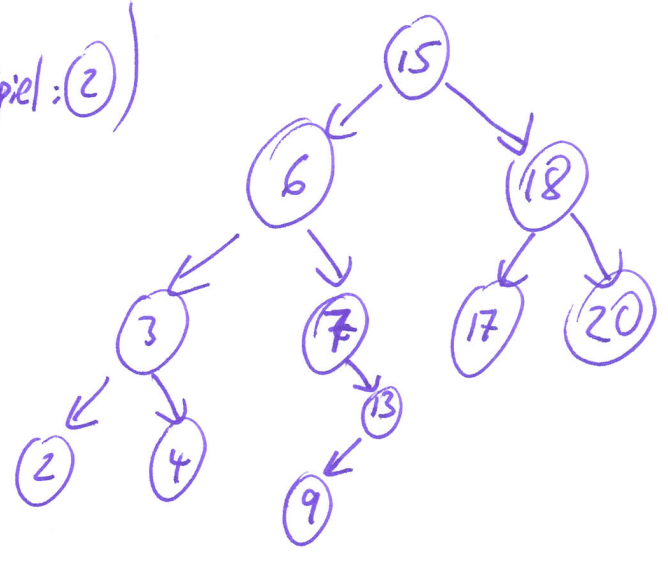
Baum-Minimum(i)

```

1 WHILE l[i] ≠ NIL DO
2   i := l[i]
3 RETURN i

```

(Beispiel: 2)



Maximum

Baum-Maximum(i)

```

1 WHILE r[i] ≠ NIL DO
2   i := r[i]
3 RETURN i

```

(Beispiel: 20)

Nachfolger

Baum-Nachfolger(i)

```

1 IF r[i] ≠ NIL THEN
2   RETURN Baum-Minimum(r[i])
3 j := p[i]
4 WHILE (j ≠ NIL und i := r[j]) DO
5   i := j
6   j := p[j]
7 RETURN j

```

(Beispiel: Nachfolger von 15 ist 17  
 → Min. rechter Teilbaum  
 Nachfolger von 13 ist  
 letzter Vorfahre, dessen linkes  
 Kind auch Vorfahre ist,  
 also 15)

## Satz 4.4

Suche, Minimum, Maximum, Nachfolger, Vorgänger

Können in einem Binären Suchbaum der Höhe  $h$  in Zeit  $O(h)$  ausgeführt werden.

Beweis:

Klar, der Baum wird nur vertikal durchlaufen!

### Einfügen

Gegeben: binärer Suchbaum  $T$ , Knoten  $z$  mit  $S[z]$ ,  
 $l[z] = NIL$ ,  $r[z] = NIL$

Aufgabe:  $z$  passend in  $T$  einfügen

#### Baum-Einfügen ( $T, z$ )

```
1  y := NIL
2  x := w[T]                                (Wurzel, y Vorgänger von x)
3  WHILE (x ≠ NIL) DO
4      y := x
5      IF [S[z] < S[x]] THEN
6          x := l[x]
7      ELSE
8          x := r[x]
9  p[z] := y
10 IF (y = NIL) THEN                        (Baum war leer)
```

```
11 w[T] := z
12 ELSE IF (s[z] < s[y]) THEN
13   l[y] := z
14 ELSE
15   r[y] := z
```

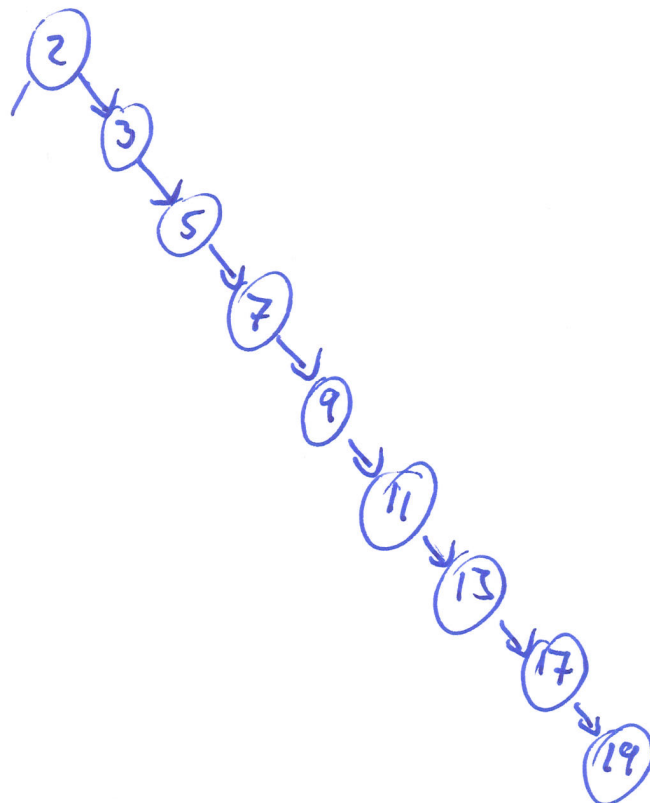
### Satz 4.5

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

Wie groß kann  $h$  werden?!

Betrachte sequentielles Einfügen:

2, 3, 5, 7, 9, 11, 13, 17, 19:



Das kann also zu einem degenerierten Baum führen: einer Liste der Höhe  $h = n$ !

Abhilfe?!  $\rightarrow$  Später! (Erfordert Umbau des Baumes!)

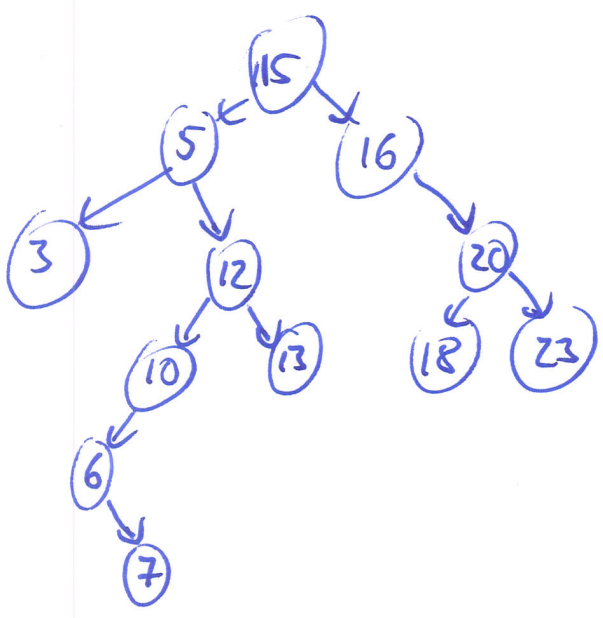
Vorher:

**LÖSCHEN**

Gegeben: Suchbaum  $T$ , Knoten  $z$

Aufgabe:  $z$  aus Baum löschen, dabei Sucheigenschaft erhalten!

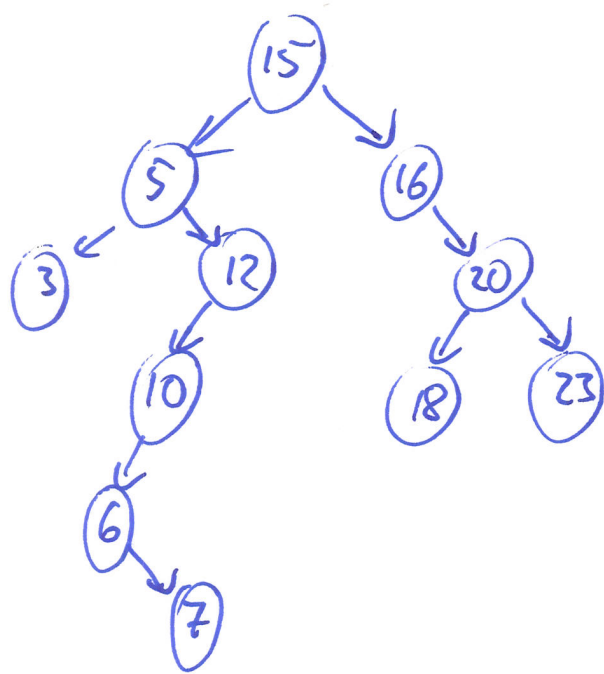
Vorbetrachtung:



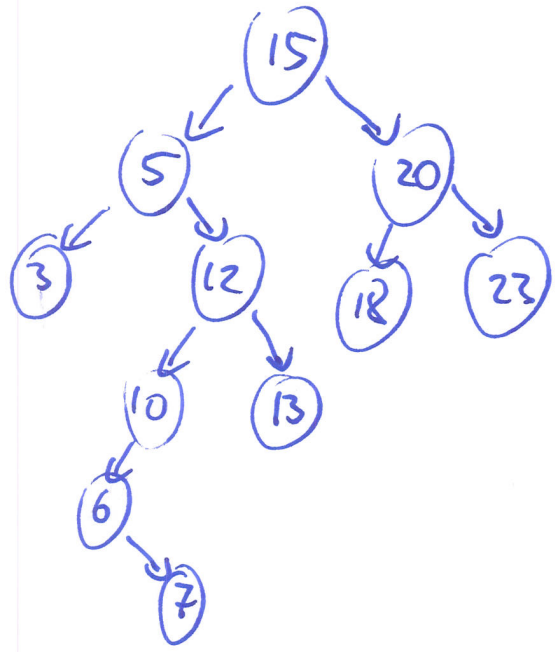
- Lösche:
- (a) 13
  - (b) 16
  - (c) 5



(9) z hat keinen Nachfolger : löschen!



(6) z hat einen Nachfolger : ausschneiden, d.h. Nachfolger „auffrischen“ lassen!



(c) z hat zwei Nachfolger:

Wähle einen anderen Knoten aus, der an die Stelle von z rückt!

Wichtig: (i) Suchbaumeigenschaft (Def. 4.3 (8)) muss erhalten bleiben!

(ii) ~~Wahl~~ Zahl der ~~Nachfolger~~ Kinder muss noch passen

Beobachtung: Der Nachfolger von z ist ein geeigneter Kandidat!

Denn: (i) Er ist größer als jeder andere Knoten im linken Teilbaum von z

UND

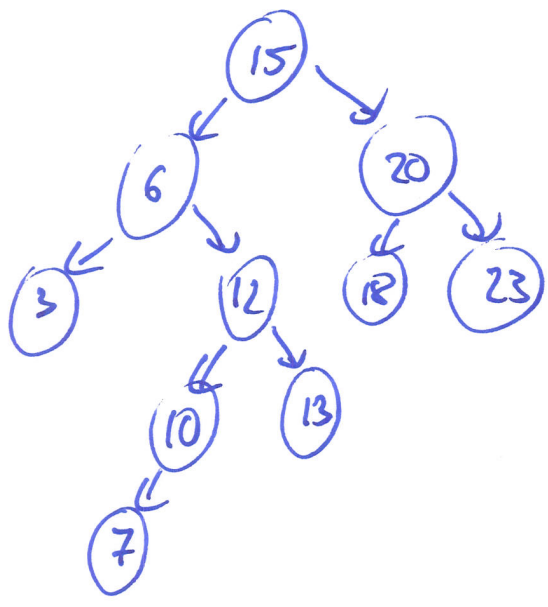
(ii) Er ist kleiner als jeder andere Knoten im rechten Teilbaum von z

das ist kritisch!



(ii) Er hat bislang nur ~~einem~~ ein ~~Wgc~~ Kind, kann also einfach ausgeschnitten werden.

Also hier:



Zusammengefasst:

Baum-Löschen (T, z)

- 1 IF (l[z] = NIL ODER r[z] = NIL) THEN
- 2     y := z
- 3 ELSE
- 4     y := Baum-Nachfolger(z)
- 5 IF (l[y] ≠ NIL) THEN
- 6     x := l[y]
- 7 ELSE
- 8     x := r[y]
- 9 IF (x ≠ NIL) THEN
- 10     p[x] := p[y]



```

11 IF ( p[y] = NIL ) THEN
12     w[T] := x
13 ELSE IF ( y = l [ p[y] ] ) THEN
14     l [ p[y] ] := x
15 ELSE
16     r [ p[y] ] := x
17 IF ( y ≠ z ) THEN
18     S[z] := S[y]
19     l [z] := l [y]
20     r [z] := r [y]
21     p [z] := p [z]
22 RETURN y

```

Also: Falls nur ein oder kein Kind von z existiert, schneide z aus (Zeilen 1-2); sonst (Zeile 3) bestimme Nachfolger von z (Zeile 4).

Dieser hat nur ein Kind.

Falls dieses das linke ist, wähle dieses (Zeile 5/6), sonst das rechte (Zeile 7/8).

In der Folge (Zeile 9) ~~setzt~~ <sup>schneidet</sup> man y aus (Zeilen 10-16); dabei muss man aufpassen, weil x = NIL oder y die Wurzel sein kann.

Schließlich werden für den Fall (c) die Informationen für y umgespeichert und der Speicherplatz für y freigegeben.

Man sieht wieder:

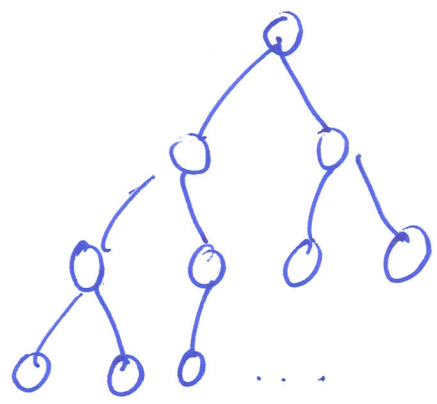
Satz 4.6

Löschen kann für einen binären Suchbaum der Höhe h in Zeit  $O(h)$  vorgenommen werden.

Bew.: klar.

Problematik: Wie kann man verhindern, dass die Höhe des Suchbaumes zu groß wird?!

Ideal wäre,



d.h. ein "balanciertere" Form als im degenerierten Fall!