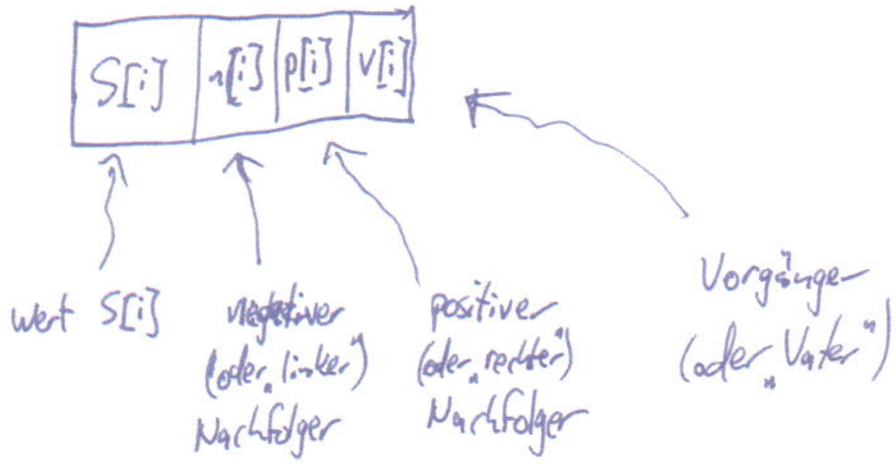
Dabei sieht jedes Knotenobjekt so aus:



(Wie vorher schon steht „/“ für „NIL“, also kein Nachfolger oder Vorgänger.)

Definition 4.3

(1) Ein gerichteter Graph $D=(V,A)$ besteht aus einer endlichen Menge 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 $w \in V$ ohne Vorgänger (d.h. ohne gr. Kante, die auf ihn zeigt).

(ii) Jeder Knoten $v \in V \setminus \{w\}$ ist durch einen eindeutigen Weg von w aus erreichbar; das heißt insbesondere, dass v einen eindeutigen Vorgänger („Vaterknoten“) hat.

(3) Ein binärer Baum ist ein gerichteter Baum, in dem jeder Knoten höchstens zwei Nachfolger („Kindknoten“) hat.

(4) Die Höhe h eines gerichteten Baumes ist die maximale Länge eines gerichteten Weges von der Wurzel.

(5) Ein binärer Baum ist ein gerichteter Baum, in dem jeder Knoten höchstens zwei Nachfolger („Kindknoten") hat. Wir nennen einen davon den linken $\ell(i)$, den anderen den rechten $r(i)$.

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

(7) Ein Knoten ohne Kindknoten heißt Blatt.

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

(9) In einem ~~[illegible]~~ 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 Bäume~~ Binäre Suchbäume als Datenstruktur?!
- Suchen
- Minimum/Maximum bestimmen
- Nachfolger/Vorgänger bestimmen
- Einfügen
- Löschen

Iterative Baumsuche (i, k)

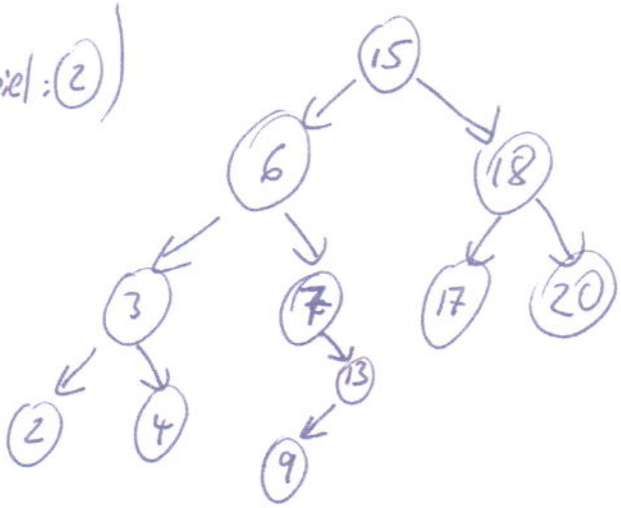
```
WHILE (i ≠ NIL und k ≠ S[i]) DO
    IF k < S[i]
      THEN i := ℓ[i]
      ELSE i := r[i]
RETURN i
```

Minimum:

Baum-Minimum (i)

```
WHILE l[i] ≠ NIL DO
      i := l[i]
RETURN i
```

(Beispiel: 2)



Maximum

Baum-Maximum (i)

```
WHILE ~~l~~[i] ≠ NIL DO
      i := r[i]
RETURN i
```

(Beispiel: 20)

Nachfolger

Baum-Nachfolger (i)

```
IF r[i] ≠ NIL THEN
      RETURN Baum-Minimum (r[i])
j := p[i]
WHILE (j ≠ NIL und := r[j]) DO
      i := j
      j := p[j]
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 einen 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)

```
y := NIL
x := w[T]                        (Wurzel, y Vorgänger von x)
WHILE (x ≠ NIL) DO
    y := x
    IF (S[z] < S[x]) THEN
        x := l[x]
    ELSE
        x := r[x]
p[z] := y
IF (y = NIL) THEN                (Baum war leer)
```

```
   w[T] := z
ELSE IF ( s[z] < s[y]) THEN
   l[y] := z
ELSE
   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: