10) Elementare Datenstrukturen

Stapel und Warteschlangen

Dynamische Mengen, Element beim Löschen a priori festgelegt.
Unterschiede beim Löschen eines Elements:

- Stack: LIFO
- Warteschlangen: FIFO

Stapel

Realisierung mittels Datenfeld
Einfügen meist als PUSH bezeichnet,
löschen als POP (ohne Element als Argument).

$$S = [17, 7, 4, 8] \xrightarrow[\text{PUSH(S,10)}]{\text{PUSH(S,3)}} S = [17, 7, 4, 8, 3, 10] \xrightarrow{\text{POP(S)}} S = [17, 7, 4, 8, 3]$$

POP auf [ ]: underflow
PUSH auf vollen Array: overflow

Operationen für Stapel

Funktion $t(S)$: Position des letzten Elements

Algorithm EMPTY(S)

```
1: if t(S)=0 then
2:     return TRUE
3: else
4:     return FALSE
5: end if
```

Algorithm PUSH(S, x)

```
1: t(s) := t(S) + 1
2: S[t(S)] := x
```

Algorithm POP(S)

```
1: if EMPTY(S) then
2:     error "underflow"
3: else
4:     t(S) := t(S) − 1
5:     return S[t(S) + 1]
6: end if
```

Warteschlangen

Einfügen wird ENQUEUE genannt, Löschen DEQUEUE.
DEQUEUE benötigt kein Element als Argument.
Warteschlange Q hat einen Kopf $k(Q)$ und ein Ende $e(Q)$.

$Q = [\_, \_, \_, \_, \_, 17, 7, 4, 8, \_, \_\_]$
$k(Q) = 6,\ e(Q) = 10$

Nach ENQUEUE(Q,3), ENQUEUE(Q,10), ENQUEUE(Q,9):

$Q = [9, \_, \_, \_, \_, 17, 7, 4, 8, 3, 10]$
$k(Q) = 6,\ e(Q) = 2$

DEQUEUE(Q) gibt 17 aus. Danach:

$Q = [9, \_, \_, \_, \_, \_\_, 7, 4, 8, 3, 10]$
$k(Q) = 7,\ e(Q) = 2$

Operationen für Warteschlangen

Algorithm ENQUEUE(Q, x)

```
1: Q[e(Q)] := x
2: if e(Q) = |Q| then
3:     e(Q) := 1
4: else
5:     e(Q) := e(Q) + 1
6: end if
```

Algorithm DEQUEUE(Q)

```
1: x := Q[k(Q)]
2: if k(Q) = |Q| then
3:     k(Q) := 1
4: else
5:     k(Q) := k(Q) + 1
6: end if
7: return x
```

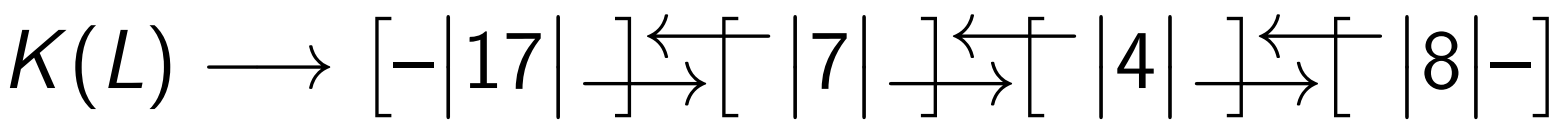
Verkettete Listen

Datenfeld: Ordnung induziert durch Indizes (Positionen)

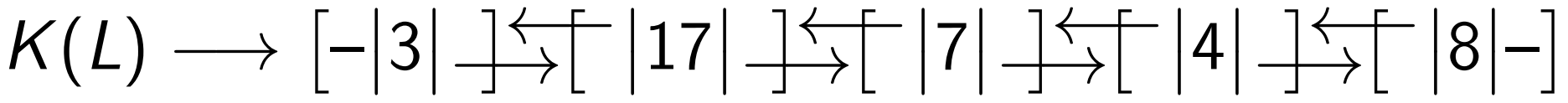
Liste: Ordnung gegeben durch Zeiger
Elemente x haben Schlüssel $S(x)$ und einen Nachfolger $N(x)$,
bei doppelt verketteten Listen auch einen Vorgänger $V(x)$.
$N(x)$ und $V(x)$ sind Zeiger.

Für das letzte Element setzt man $N(x)$=NIL.
Der Zeiger $K(L)$ zeigt auf das erste Element der Liste L.

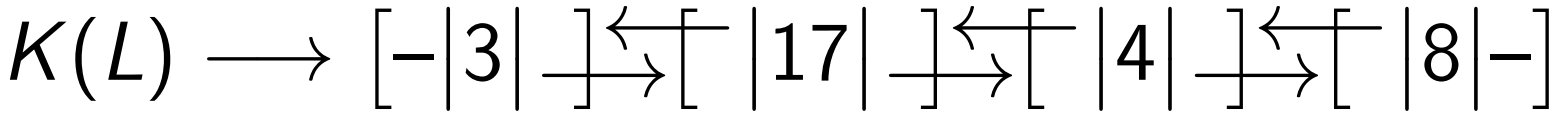
Eine Liste heißt sortiert, wenn die Schlüssel in sortierter Reihenfolge auftreten, andernfalls unsortiert.
Eine Liste heißt zyklisch, wenn $N(x)$ des letzten Elements auf das erste Element a zeigt und $V(a)$ auf x zeigt.



Nach Anwendung von LISTE-EINFÜGEN($L, 3$)
(eig. LISTE-EINFÜGEN(L, x) mit $S(x) = 3$):



Nach Anwendung von LISTE-LÖSCHEN($L, 7$)
(eig. LISTE-EINFÜGEN(L, x) mit $S(x) = 7$):



Operationen für Listen

SUCHE(L, n) liefert das Element x mit $S(x) = n$
und NIL, falls es kein solches Element gibt.

Algorithm SUCHE(L, n)

```
1: x := K(L)
2: while x ≠NIL and S(x) ≠ n do
3:     x := N(x)
4: end while
5: return x
```

Algorithm LISTE-EINFÜGEN(L, x)

```
1: N(x) := K(L)
2: if K(L) ≠NIL then
3:     V(K(L)) := x
4: end if
5: K(L) := x
6: V(x) :=NIL
```

Algorithm LISTE-LÖSCHEN(L, x)

```
1: if V(x) ≠NIL then
2:     N(V(x)) := N(x)
3: else
4:     K(L) := N(x)
5: end if
6: if N(x) ≠NIL then
7:     V(N(x)) := V(x)
8: end if
```

Methoden für Listen auch auf Bäume anwendbar:

- Binärbäume: $K(T)$ zeigt auf die Wurzel. Doppelte Verkettung entlang jeder Kante.
- Andere Bäume: $K(T)$ zeigt auf die Wurzel.
  - Doppelte Verkettung zwischen jedem Knoten und seinem am weitesten links liegenden Nachfolgeknoten.
  - Einfache Verkettung von jedem Knoten zu seinem nächsten Geschwisterknoten (von links nach rechts) bzw seinem Elternknoten, falls es keine Geschwisterknoten weiter rechts gibt.

Binäre Suchbäume

Knoten werden mittels Schlüsseln identifiziert.
Der Einfachheit halber unterscheiden wir nicht zwischen Knoten und deren Schlüsseln.
Für jeden Knoten gibt es wieder Zeiger zu anderen Knoten.

Definition

Ein binärer Suchbaum ist ein Binärbaum, der im Speicher mit Hilfe von doppelten Verkettungen abgelegt wird und die Suchbaumeigenschaft erfüllt:

- *Für alle Knoten y im linken Teilbaum von x gilt $y < x$;*
- *für alle Knoten z im rechten Teilbaum von x gilt $z > x$,*

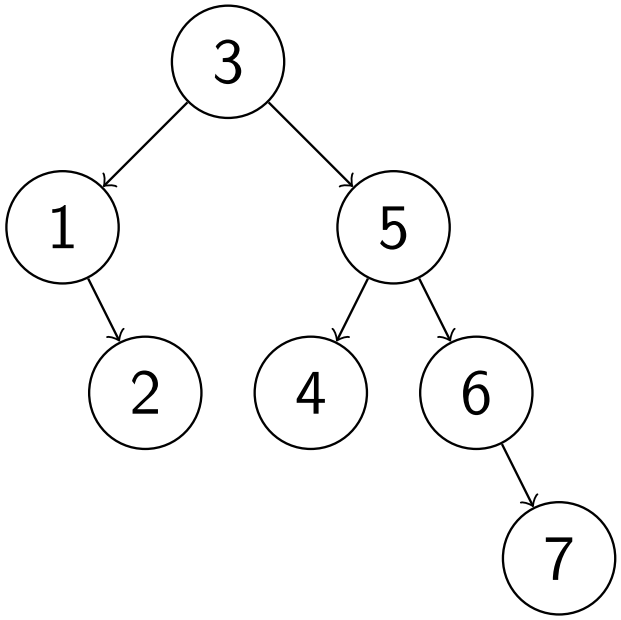
Daten können einfach sortiert extrahiert werden:
Inordertraversierung des Baums
L(i), R(i): Zeiger zum linken bzw rechten Nachfolger
T bezeichne den Zeiger zur Wurzel, $S(T)$ deren Eintrag (Schlüssel)

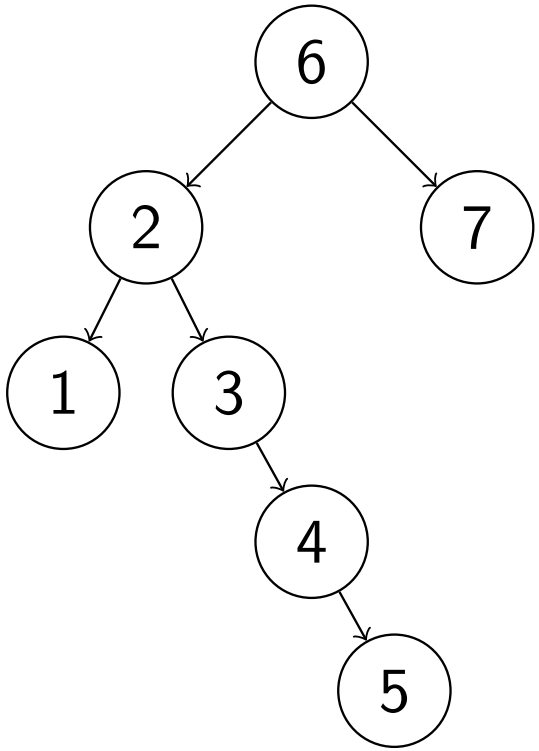
Algorithm INORDER(T)

```
1: x := S(T)
2: if T ≠NIL then
3:     INORDER(L(x))
4:     print x
5:     INORDER(R(x))
6: end if
```

Beispiele von binären Suchbäumen

Die binären Suchbäume zu $[3, 1, 5, 6, 2, 4, 7]$ und $[6, 2, 1, 7, 3, 4, 6]$:





Suchen in binären Suchbäumen

Übergeben wird ein Zeiger T zur Wurzel und ein Schlüssel s, $S(T)$ sei der Schlüssel der Wurzel bzw die Wurzel

Algorithm SUCHE(T, s)

```
1: x := S(T)
2: if T =NIL or x = s then
3:     return x
4: end if
5: if s < x then
6:     return SUCHE(L(x), s)
7: else
8:     return SUCHE(R(x), s)
9: end if
```

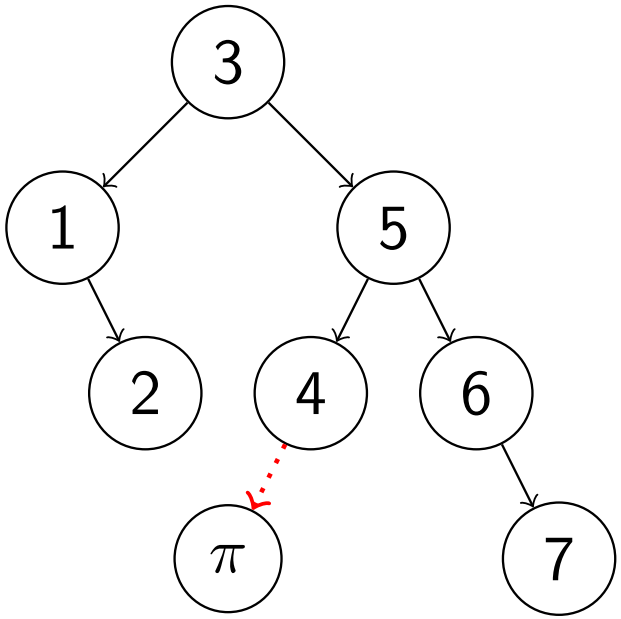
Laufzeit: $\mathcal{O}(h)$, wobei h die Länge des längsten Pfades von der Wurzel zu einem Blatt ist.

Einfügen und Löschen in binären Suchbäumen

Einfügen eines Knotens v mit L(v)=R(v)=NIL in einem Baum T

Algorithm EINFÜGEN(T, v)

```
 1: x := S(T)
 2: if x =NIL then
 3:     Sei w ein neuer Knoten
 4:     w := v
 5:     L(w):=NIL
 6:     R(w):=NIL
 7:     return w
 8: else if v = x then
 9:     print "Schlüssel existiert bereits!"
10: else if v < x then
11:     L(x):=EINFÜGEN(L(x),v)
12:     return T
13: else if v > x then
14:     R(x):=EINFÜGEN(R(x),v)
15:     return T
16: end if
```


3
1
5
2
4
6
π
7

Strategie beim Löschen:

- Entferne Element v aus dem Baum.
- Verschiebe geeignetes Element in den freien Platz:
  - Falls v einen nichtleeren linken Teilbaum hatte, nimm das Maximum von dort;
  - andernfalls das Minimum des rechten Teilbaums, falls dieser nichtleer ist.

Algorithm FINDE_MIN(T)

```
1: x := S(T)
2: y :=R(x)
3: while L(x) ≠NIL do
4:       T :=L(x)
5:       x := S(T)
6: end while
7: return (x, y)
```

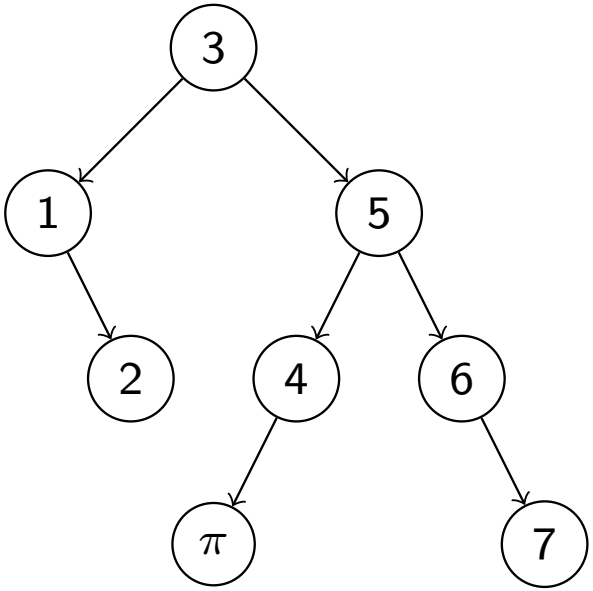
FINDE_MAX analog.

Algorithm LÖSCHE_WURZEL(T)

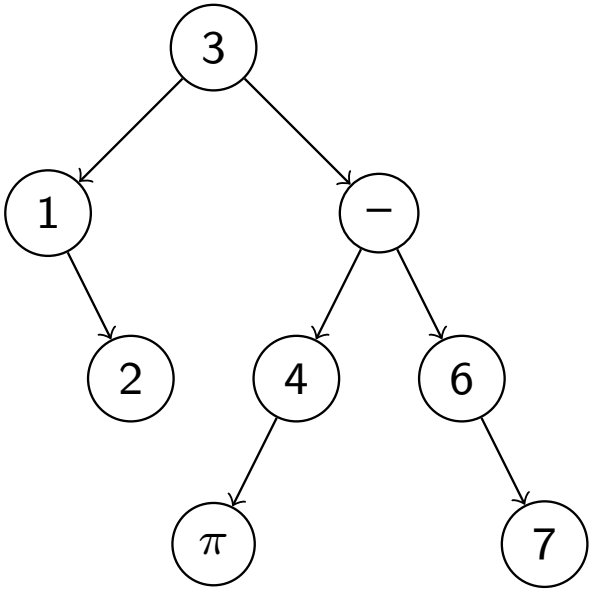
```
 1: x := S(T)
 2: if L(x) ≠NIL then
 3:     (M, T_L) :=FINDE_MAX(L(x))
 4:     x := M
 5:     L(x) := T_L
 6:     return T
 7: else if R(x) ≠NIL then
 8:     (m, T_R) :=FINDE_MIN(R(X))
 9:     x := m
10:     L(x) := T_R
11:     return T
12: else
13:     return NIL
14: end if
```

Algorithm LÖSCHEN(T, v)

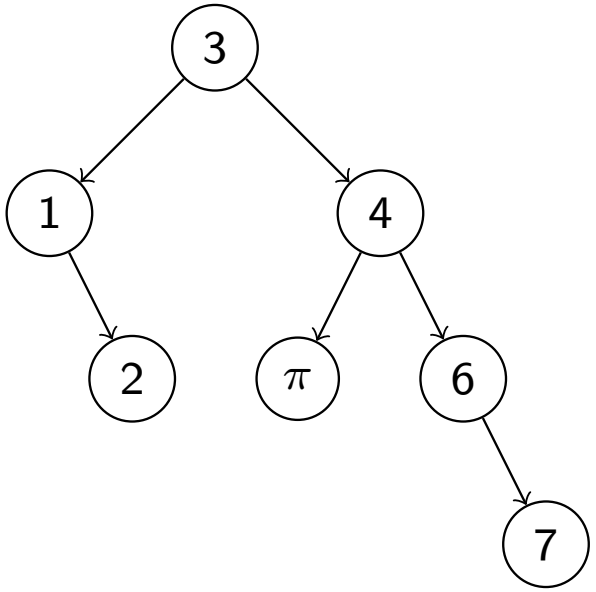
```
 1: x := S(T)
 2: if x =NIL then
 3:     print "nicht gefunden"
 4: else if x = v then
 5:     return LÖSCHE_WURZEL(T)
 6: else if v < x then
 7:     L(x) :=LÖSCHEN(L(x), v)
 8:     return T
 9: else
10:     R(x) :=LÖSCHEN(R(x), v)
11:     return T
12: end if
```



Nach dem Löschen von 5:



Das Maximum des linken Teilbaums der Lücke füllt die Lücke:



Analyse

- Die Prozeduren SUCHE, EINFÜGEN und LÖSCHEN haben jeweils Laufzeit $\mathcal{O}(h)$.
- Wie hoch ist ein binärer Suchbaum?
- Worst-Case: n Elemente in sortierter Reihenfolge eingefügt. $\leadsto$ Pfad, $h = \Theta(n)$.
- Best-Case: vollständig balancierter Baum $\leadsto$ $h = \lfloor \log_2 n \rfloor$
- Average-Case?

Annahmen:

- Gleichverteilung: Jede Permutation der n Schlüssel tritt mit Wahrscheinlichkeit $\frac{1}{n!}$ auf.
- Baum durch n Einfüge-Operationen erzeugt, danach statisch.

Ähnlich wie bei Heaps definieren wir die Höhe $h(v)$ eines Knotens v als die Anzahl der Kanten des Pfades von v zur Wurzel.

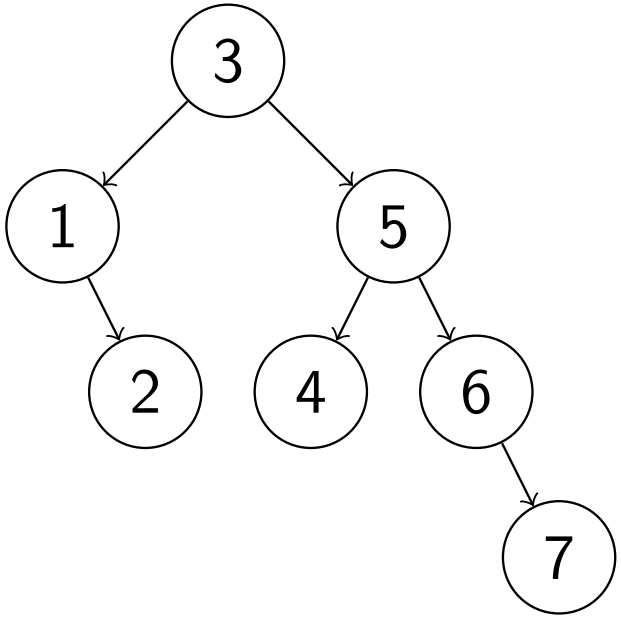
Um einen Knoten v in einem binären Suchbaum zu finden, sind $h(v) + 1$ Schlüsselvergleiche nötig.

Definition

Sei T ein binärer Suchbaum. Dann ist die Pfadlänge von T definiert durch

$$L(T) = \sum_{v \in V} (h(v) + 1).$$

Beispiel:



Hier gilt

$$h(3) = 0,\ h(1) = h(5) = 1,\ h(2) = h(4) = h(6) = 2,\ h(7) = 3$$

Daher

$$L(T) = 1 + 2 + 2 + 3 + 3 + 3 + 4 = 18, \qquad \frac{L(T)}{|T|} = \frac{18}{7} \approx 2,57.$$

Satz

Die mittlere Höhe eines binären Suchbaums beträgt $\Theta(\log n)$.

Beweis: Sei T ein binärer Suchbaum, T_0 der leere Baum und T_L und T_R der linke bzw rechte Teilbaum der Wurzel von T. Dann gilt

$$\begin{aligned} L(T) &= L(T_L) + |T_L| + L(T_R) + |T_R| + 1 \\ &= L(T_L) + L(T_R) + |T|, \qquad L(T_0) = 0. \end{aligned}$$

Die Permutation des T erzeugenden Datenfelds sei zufällig gewählt und es gelte $|T| = n$. Dann gilt

- Das erste Element a_1 ist der Schlüssel der Wurzel. Sei $a_1 = k$.
- Der linke Teilbaum enthält alle Schlüssel, die kleiner als k sind, also $|T_L| = k - 1$ und folglich $|T_R| = n - k$.
- $\mathbb{P}\{a_1 = k\} = \frac{1}{n}$.

Die Zufallsvariable L_n bezeichne die Pfadlänge von T. Dann gilt

$$\mathbb{E}L_0 = 0 \text{ und } \mathbb{E}L_n = \frac{1}{n}\sum_{k=1}^{n}(\mathbb{E}L_{k-1} + \mathbb{E}L_{n-k} + n)$$

$$= n + \frac{1}{n}\left(\sum_{k=1}^{n}\mathbb{E}L_{k-1} + \sum_{k=1}^{n}\mathbb{E}L_{n-k}\right)$$

$$= n + \frac{2}{n}\sum_{k=0}^{n-1}\mathbb{E}L_k$$

Daraus folgt

$$n\mathbb{E}L_n = n^2 + 2\sum_{k=0}^{n-1}\mathbb{E}L_k$$

$$(n-1)\mathbb{E}L_{n-1} = (n-1)^2 + 2\sum_{k=0}^{n-2}\mathbb{E}L_k$$

$$n\mathbb{E}L_n - (n-1)\mathbb{E}L_{n-1} = 2n - 1 + 2\mathbb{E}L_{n-1}$$

$$\Longrightarrow \qquad \mathbb{E}L_n = \frac{2n-1}{n} + \frac{n+1}{n}\mathbb{E}L_{n-1}$$

$$\mathbb{E}L_n = \frac{2n-1}{n} + \frac{n+1}{n}\mathbb{E}L_{n-1}$$

ist eine lineare Rekursion erster Ordnung mit Anfangsbedingung $\mathbb{E}L_0 = 0$.
Deren Lösung lautet

$$\begin{aligned}\mathbb{E}L_n &= \sum_{i=1}^{n} \frac{2i-1}{i} \prod_{j=i+1}^{n} \frac{j+1}{j} = (n+1)\sum_{i=1}^{n} \frac{2i-1}{i(i+1)} \\ &= (n+1)\sum_{i=1}^{n}\left(\frac{3}{i+1} - \frac{1}{i}\right) = (n+1)\left(\frac{3}{n+1} + 3(H_n - 1) - H_n\right) \\ &= 2(n+1)H_n - 3n.\end{aligned}$$

Somit erhalten wir für die mittlere Höhe eines binären Suchbaums der Größe n die Gleichung

$$\frac{\mathbb{E}L_n}{n} \sim 2\log n.$$

□