Übung: Suchbäume

Definition *"Binärbaum"*

Ein Baum heisst ***Binärbaum***, wenn jeder Knoten *höchstens zwei* Sohnknoten besitzt.
In diesem Fall spricht man vom *linken* bzw. *rechten* Sohnknoten.
Hier betrachten wir nur Bäume, in denen jeder Knoten *genau zwei* (ggf. leere) Teilbäume enthält

```
> restart;
```

Definition *"Suchbaum"*

Wenn je zwei Knoteninhalte verglichen werden können (z.B. alle Knoten reelle Zahlen enthalten),
heisst ein Binärbaum ***Suchbaum***, wenn für jeden Knoten gilt:

- die Inhalte des linken Teilbaums (falls es einen gibt) sind kleiner,
- der Inhalte des rechten Teilbaums (falls es einen gibt), sind größer

als der Inhalt des Knoten selbst.

```
> LEER := 'LEER';
```

$$LEER := LEER$$

```
> d := [1, [LEER, LEER]];
  b := [2, [d, LEER]];
  e := [5, [LEER, LEER]];
  c := [4, [LEER, e]];
  a := [3, [b, c]];
```

$$d := [1, [LEER, LEER]]$$

$$b := [2, [[1, [LEER, LEER]], LEER]]$$

$$e := [5, [LEER, LEER]]$$

$$c := [4, [LEER, [5, [LEER, LEER]]]]$$

$$a := [3, [[2, [[1, [LEER, LEER]], LEER]], [4, [LEER, [5, [LEER, LEER]]]]]]$$

```
>
```

Suchen im Baum

- Parameter: Suchbaum `b`, Zahl `k`.
- Ergebnis: Teilbaum von `b` mit Wurzelinhalt `k`, falls `k` im Baum vorkommt, `FAIL` sonst.

```
> finde := proc(b, k::numeric)
    if b=LEER
      then FAIL
    elif b[1]=k
      then b
    elif b[1]>k
      then finde(b[2][1], k)
    else
      finde(b[2][2], k)
    fi
  end;
```

finde := **proc**(*b*, *k*::*numeric*)

if $b = LEER$ **then** *FAIL*

elif $b[1] = k$ **then** b

elif $k < b[1]$ **then** finde($b[2][1]$, k)

else finde($b[2][2]$, k)

end if

end proc

```
> finde(a, 5);
```

[5, [*LEER*, *LEER*]]

```
> finde(a, 7);
```

FAIL

Aufwand zum Suchen:

- Wir zählen die Anzahl der (rekursiven) Aufrufe der Funktion `finde()`.
- Die ist, je nach Position des gesuchten Knotens im Baum 1..tiefe(b).
- Interessanter ist der Zusammenhang zwischen der Anzahl der Knoten und dem Aufwand.
  Dazu betrachten wir als Beispiel Bäume mit $2^t - 1$ Knoten für natürliches t.
  So ein Baum hat maximal Tiefe $2^t - 1$, minimal Tiefe t.
  In letzterem Fall heisst der Baum *balanciert*, und der Aufwand zum Finden eines Eintrags wächst nur **logarithmisch** mit der Anzahl der Knoten.
- Fazit: in Suchbäumen lässt sich effizient suchen, wenn sie möglicht gut balanciert sind.

```
>
```

Einfügen in den Baum:

Das Einfügen in einen Suchbaum geschieht ganz analog.

Allerdings muss beim Einfügen der Baum selbst verändert werden.
Daher dürfen die Prozedur den entsprechenden Parameter nicht auswerten, da ihm dann kein Wert mehr zugewiesen werden kann.
Dazu gibt es in Maple den Deklarator `evaln`:

```
> insert := proc(baum::evaln,k::numeric)
    if eval(baum) = LEER
       then baum := [k, [LEER,LEER]];
    else
       if eval(baum)[1] > k
       then
          insert(baum[2,1],k);
       else
          insert(baum[2,2],k);
       end if;
    end if;
    eval(baum);
  end;
```

insert := **proc**(*baum*::*evaln*, *k*::*numeric*)

if eval(*baum*) = *LEER* **then** *baum* := [*k*, [*LEER*, *LEER*]]

else if $k <$ eval(*baum*)[1] **then** insert(*baum*[2, 1], *k*) **else** insert(*baum*[2, 2], *k*) **end if**

end if;

eval(*baum*)

end proc

Ein kleines Beispiel:

```
> l := LEER;
> insert(l,5);
> insert(l,4);
```

l := *LEER*

[5, [*LEER*, *LEER*]]

[5, [[4, [*LEER*, *LEER*]], *LEER*]]

Ein größeres Beispiel:

```
> zufall := rand(1..100);
```

zufall := **proc**()

local *t*;

global *_seed*;

_seed := irem(427419669081∗*_seed*, 999999999989); *t* := *_seed*; irem(*t*, 100) + 1

end proc

```
> r := LEER:
> for i from 1 to 10
  do
    insert(r,zufall(i)):
  end do:
> r;
```

[82, [[71, [[64, [[39, [[22, [[10, [*LEER*, *LEER*]], *LEER*]], *LEER*]], [69, [*LEER*, *LEER*]]]],
[77, [*LEER*, *LEER*]]]], [98, [[86, [*LEER*, *LEER*]], *LEER*]]]

```
>
```