3.4 Darstellung von Bäumen

Darstellung von k-beschränkten Bäumen: Mit wenigen Knoten und/oder kleinem k durchaus adäquat mit folgender Darstellung:

```
type KTree;
type KPtr is access KTree;
type KTree is record
    content : SomeType;
    t1 : KPtr;
        ....
    tk : KPtr;
    end record;
```

Die maximale Anzahl von Söhnen eines Knotens bestimmt die Anzahl der Zeiger für alle Knoten. Evtl. unangemessen großer Speicherplatz wegen vieler null-Zeiger. Bessere Ausdrucksmittel fehlen in Modula-2 (und vielen anderen Sprachen).
Daher: Übliche effiziente Darstellung durch Binärbäumen (siehe später).

Bessere Ausdrucksmittel in Ada:

Darstellung mit "diskriminierten Records"

```
type Ktree (Arity: Natural);    -- Diskriminante identifiziert Anzahl der Unterbäume
type KPtr is access Ktree;
type Subtrees is array (Natural range < >) of Kptr;
type Ktree (Arity: Natural) is record
        content : SomeType;
        children : Subtrees (1..Arity);
end record;
```

Darstellung mit Klassen:

```
type Ktree is tagged record
        content : SomeType;
end record;

type KPtr is access Ktree'Class;
type Subtrees is array (Natural range <>) of Kptr;

type Unary is new Ktree with record    -- Unterklasse fügt (Anzahl der) Unterbäume hinzu
        children : Subtrees (1.. l);
end record;

type Binary is new Ktree with record
        children : Subtrees (1..2);
end record;
```

usw.

Darstellung von Blättern versus interne Knoten

Im Binärbaum können die Hälfte der Knoten Blätter sein; Komponenten für Unterbäume sind in Blättern m.E. nicht sinnvoll.

Darstellung durch variante Records (Modula, Ada):

```
type NodeKind is (Node, Leaf);
type TreeNode (Kind: NodeKind);
type NodePtr is access TreeNode;
type TreeNode (Kind: NodeKind) is record
                content: SomeType;
                case Kind is
                when Node  =>  left: NodePtr;
                               right: NodePtr;
                when Leaf   =>  null;
        end case;
end record;
```

Darstellung durch Klassen (Ada):

type TreeNode wird analog zu KTree im vorherigen Beispiel definiert, z.B. auch mit mehreren "Unterklassen" für interne Knoten. Blätter können dann einfach dazugefügt werden ....

```
type Leaf is new TreeNode with null record;
```


Frage: Wieviele geordnete Bäume mit n Knoten gibt es?
Ihre Anzahl sei A_n.

Erinnerung: Geordneter Baum = ein Baum, bei dem an jedem Knoten die ausgehenden Kanten eine Reihenfolge besitzen.

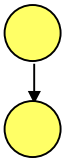
Binärer Baum: Jeder Knoten besitzt zwei Nachfolger.

Binärer geordneter Baum: Jeder Knoten besitzt zwei Nachfolger, die eine Reihenfolge (z.B.: links - rechts) besitzen.

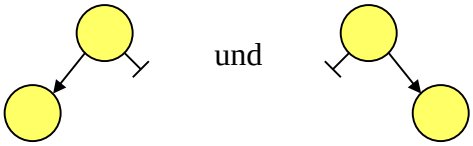
Aus den Übungen wissen wir:
Die Anzahl der geordneten binären Bäume mit n Knoten lautet

$C_n = \frac{1}{n+1}\binom{2n}{n}$ "Catalansche Zahlen".

Hinweis: Geordnete binäre Bäume sind in der regel keine normalen geordneten Bäume, weil jeder Knoten genau zwei ausgehende Kanten besitzen muss. Zum Beispiel: Der Baum

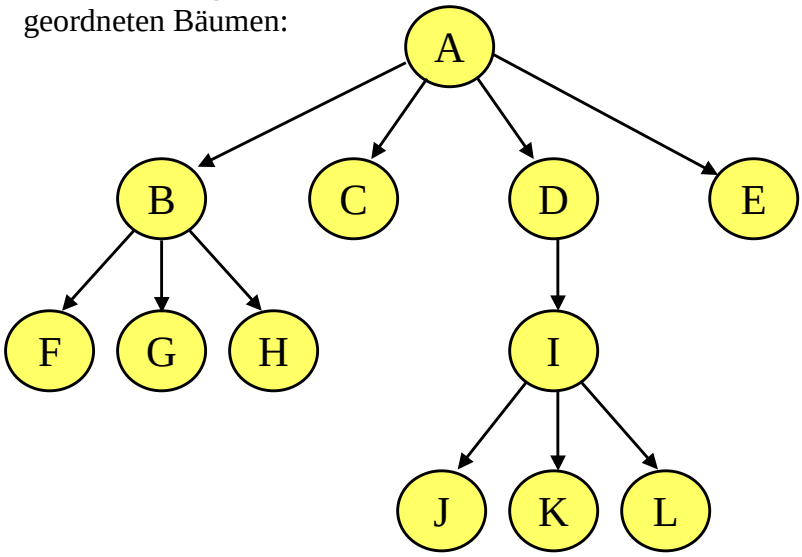


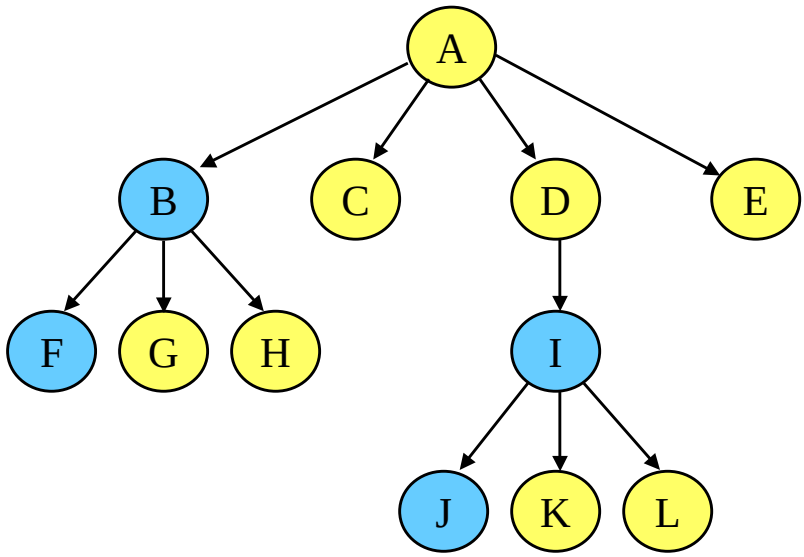
ist ein geordneter Baum mit zwei Knoten, er ist jedoch kein geordneter binärer Baum, da es zwei Möglichkeiten zu seiner Darstellung gäbe:

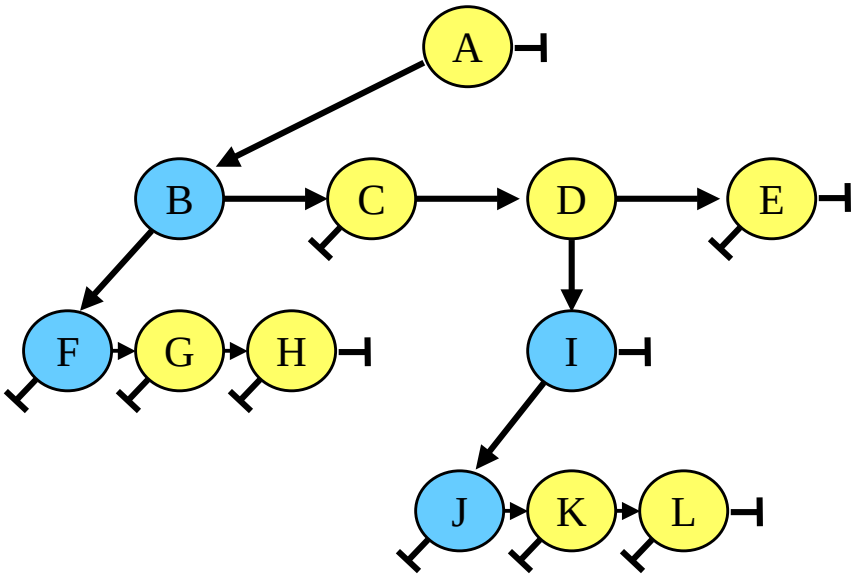


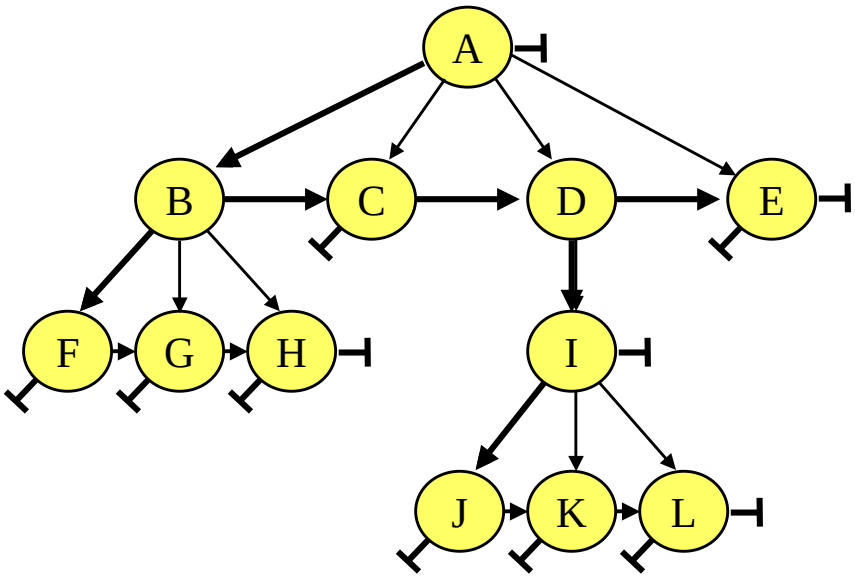
Anders ausgedrückt: Verschiedene geordnete binäre Bäume können als "normale" geordnete Bäume gleich sein.

Binarisierung von geordneten Bäumen:




A
B
C
D
E
F
G
H
I
J
K
L


A
B
C
D
E
F
G
H
I
J
K
L



Binarisierung von Bäumen:

Ein beliebiger Baum mit n Knoten wird in einen binären Baum mit n Knoten umgewandelt, wobei der erste Knoten nur einen linken Nachfolger und keinen rechten Nachfolger hat. Daraus folgt:

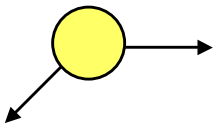
Die Anzahl A_n aller geordneten Bäume mit n Knoten ist gleich der Anzahl der binären geordneten Bäume C_{n-1} mit n-1 Knoten, d.h.:

$$A_n = \frac{1}{n}\binom{2n-2}{n-1} \quad \text{für alle } n \geq 1.$$

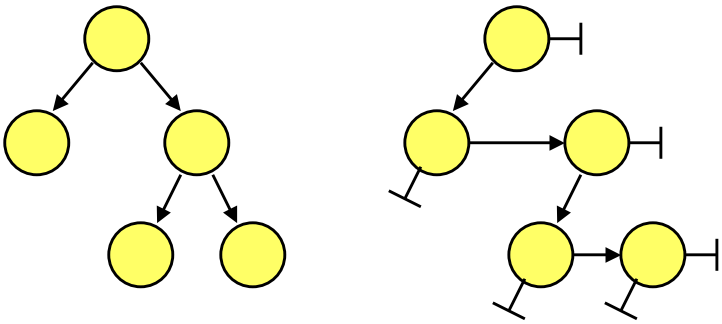
Normierter Binärbaum:

```
type B;
type B_Ptr is access B;
type B is record
        node : inhalt;
        First_Child : B_Ptr;
        Next_Sibling : B_Ptr;
end record;
```

```
type B;
type B_Ptr is access B;
type B is record
        node : inhalt;
        left : B_Ptr;
        right : B_Ptr;
end record;
```



Beispiel: Binarisierung eines 2-beschränkten Baums:



Durchlauf durch Bäume ("Traversieren")

inorder	linker Unterbaum, Knoten, rechter Unterbaum
preorder	Knoten, linker Unterbaum, rechter Unterbaum
postorder	linker Unterbaum, rechter Unterbaum, Knoten

```
procedure Inoder (p: B_Ptr) is
begin
      if  p /= null  then
            Inorder (p.left); "bearbeite Knoten p"; Inoder (p.right);
      end if;
end Inorder;
```


Inorder-Traversierung: nichtrekursive Implementierung

```
procedure Inorder (q : BPtr) is     -- Stack s ist eine globale Variable.
p : BPtr;
begin   p := q;
        loop
              while p /= null loop
                    Push (s, p); p := p.left;
              end loop;
              if not Empty(s) then
                    p := Top(s); Pop (s);
                    Integer_Text_IO.Put-Line(p.node); p := p.right;
              end if;
              exit when Empty(s) and (p = null);
        end loop;
end Inorder;
```