

SS 2021

Algorithmen und Datenstrukturen

3. Kapitel

Binärbäume

Martin Dietzfelbinger

Mai 2021

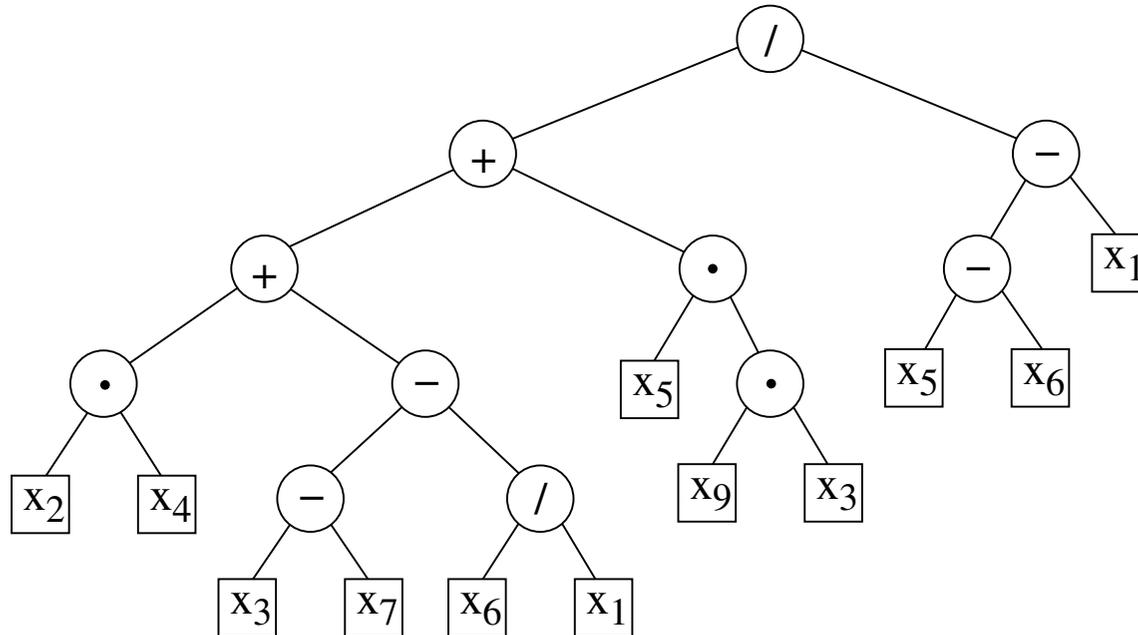
3.1 Binärbäume: Grundlagen

Der Datentyp (und die Datenstruktur) **Binärbaum** tritt überall in der Informatik auf:

- als binärer Suchbaum
- als Repräsentation von binären Ausdrücken (logisch, arithmetisch)
- als Entscheidungsbaum
- als Codierungs-/Decodierungsbaum
- als binärer **Heap** (später)
- ...

Beispiel 1: Arithmetischer Ausdruck mit zweistelligen Operatoren:

$$(((x_2 \cdot x_4) + ((x_3 - x_7) - (x_6/x_1))) + (x_5 \cdot (x_9 \cdot x_3)))/((x_5 - x_6) - x_1)$$

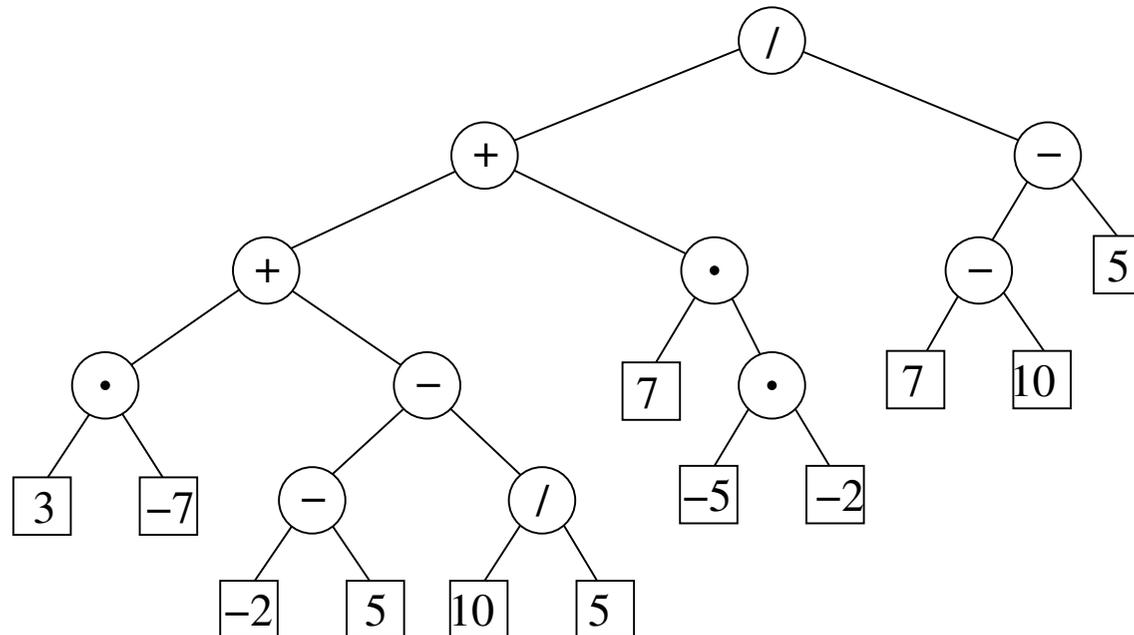


Auswertung:

1) Ordne den Blättern konkrete Werte zu,

z.B. $x_1 = 5$, $x_2 = 3$, $x_3 = -2$, $x_4 = -7$, $x_5 = 7$, $x_6 = 10$, $x_7 = 5$, $x_9 = -5$.

2) Werte aus, von den Blättern startend („bottom-up“).

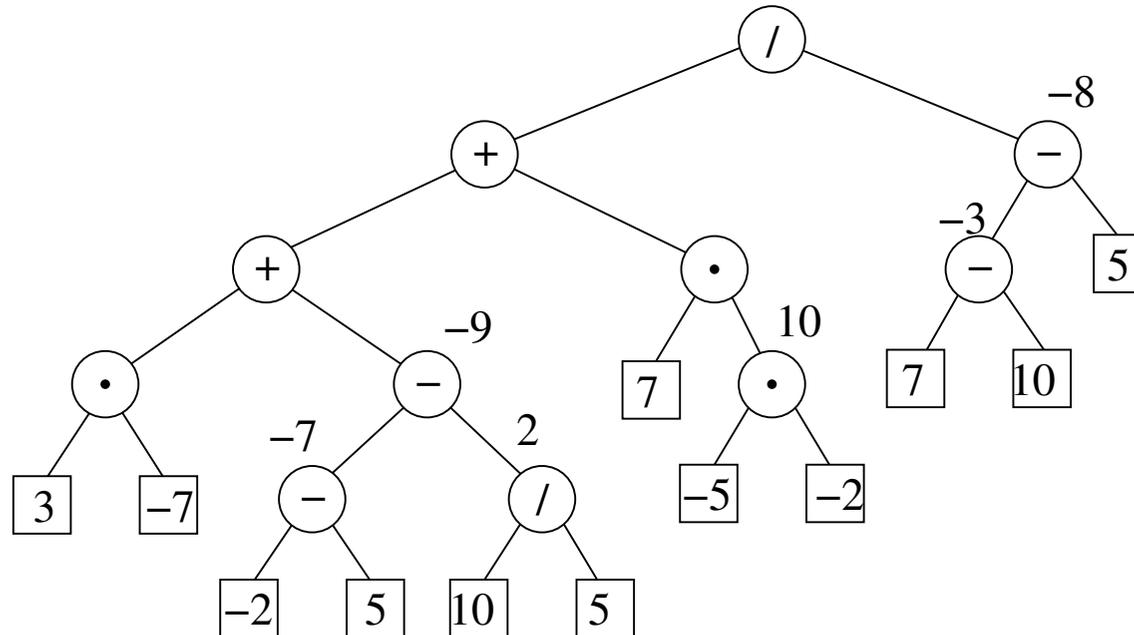


Auswertung:

1) Ordne den Blättern konkrete Werte zu,

z.B. $x_1 = 5$, $x_2 = 3$, $x_3 = -2$, $x_4 = -7$, $x_5 = 7$, $x_6 = 10$, $x_7 = 5$, $x_9 = -5$.

2) Werte aus, von den Blättern startend („bottom-up“).



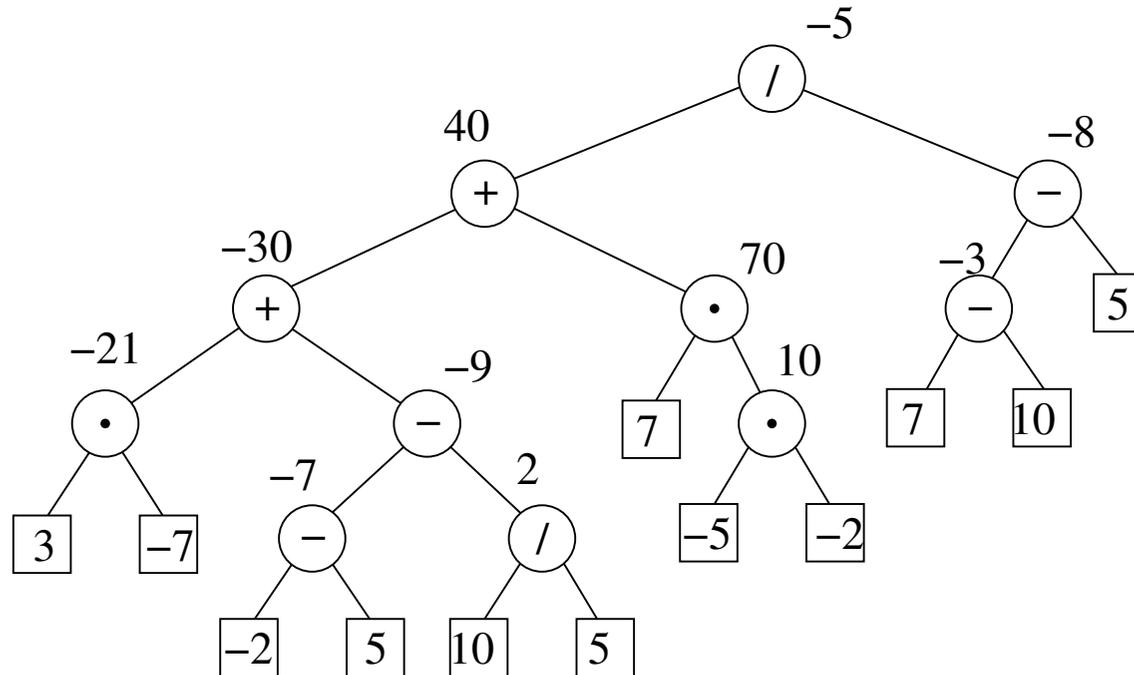
Nach sechs Schritten.

Auswertung:

1) Ordne den Blättern konkrete Werte zu,

z.B. $x_1 = 5$, $x_2 = 3$, $x_3 = -2$, $x_4 = -7$, $x_5 = 7$, $x_6 = 10$, $x_7 = 5$, $x_9 = -5$.

2) Werte aus, von den Blättern startend („bottom-up“).



Fertig. Resultat: -5 , an der Wurzel abzulesen.

Beispiel 2: Codierungs-/Decodierungsbaum

Zeichen aus Alphabet Σ erhalten binäre Codes.

Diese können auch *verschiedene Länge* haben:

„Wenige Bits für häufige Buchstaben, viele Bits für seltene.“

Mini-Beispiel:

A	B	C	D	E	F	G	H	I	K
1100	0110	000	111	10	0011	010	0010	0111	1101

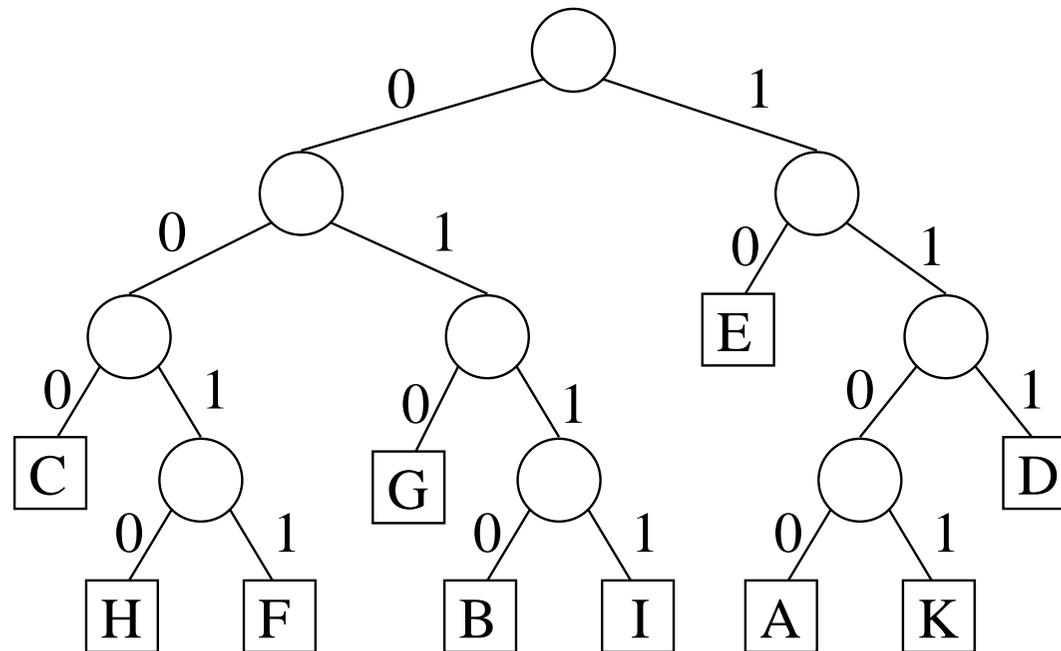
Zur Codierung von Zeichenreihen benutzt man direkt die Tabelle.

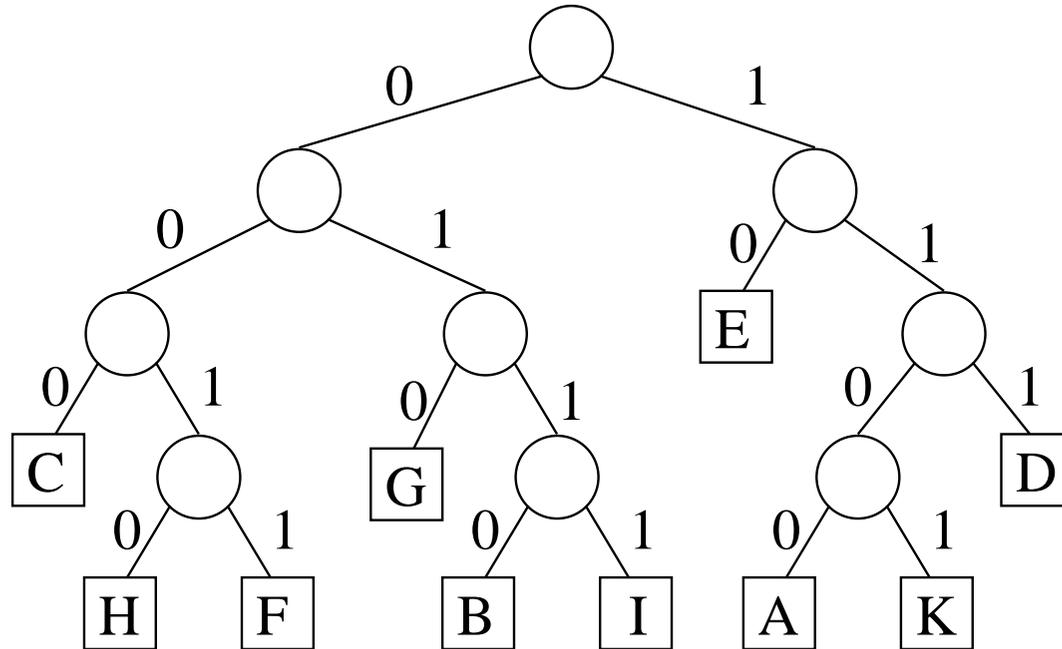
Beispiel:

FEIGE hat Codierung **0011 10 0111 010 10**.

„Präfixfreier“ Code: Kein Codewort ist Anfangsstück eines anderen.
Damit: „eindeutig decodierbar“; Fuge zwischen Codes für Buchstaben muss nicht
markiert werden: Schreibe 001110011101010 statt 0011 10 0111 010 10.

Kompakte Repräsentation des Codes als Binärbaum:





Blätter sind mit Buchstaben markiert; Weg von der Wurzel zum Blatt gibt das Codewort wieder (links: 0, rechts: 1).

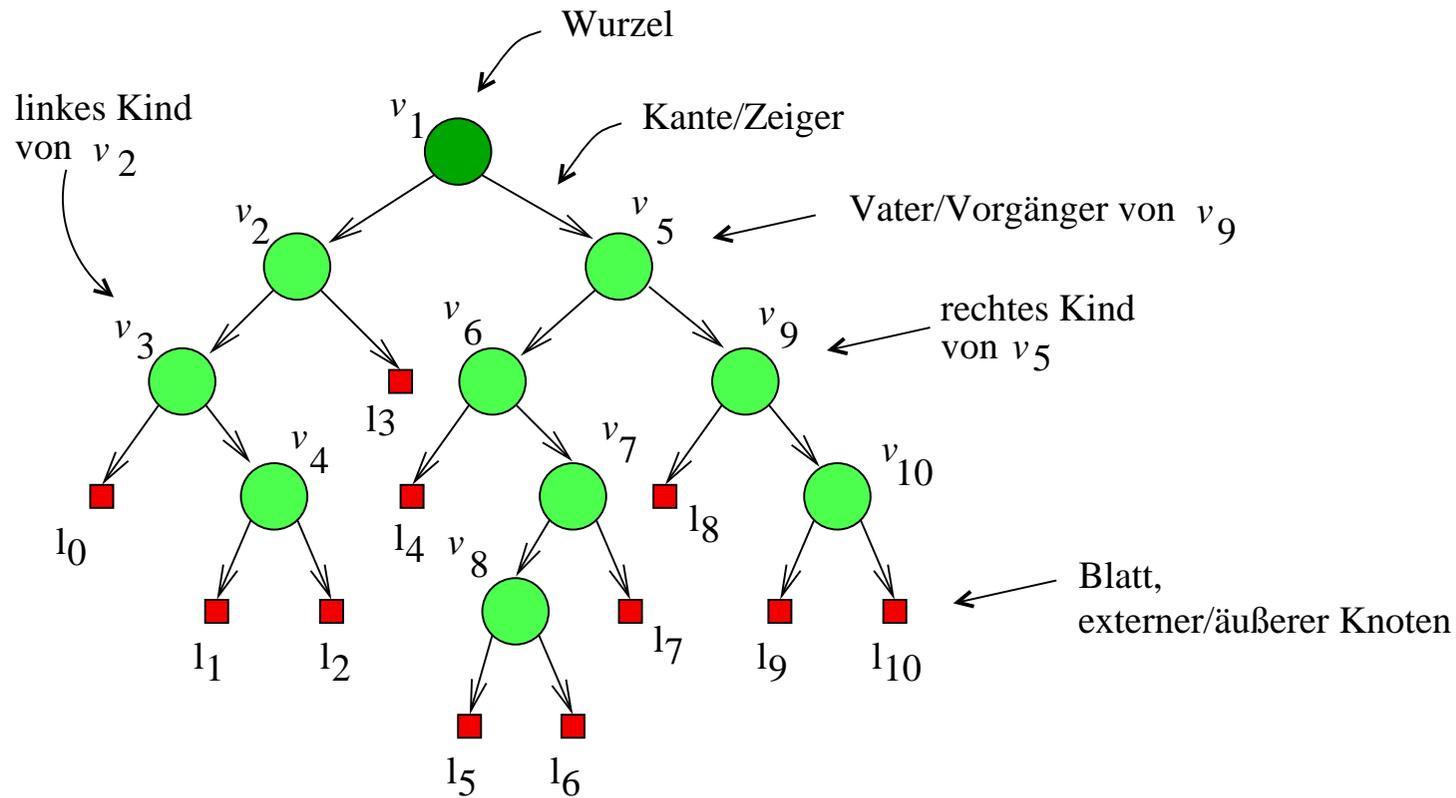
Decodierung: Laufe Weg im Baum, vom Codewort gesteuert, bis zum Blatt.

Wegen Präfixeigenschaft: im Code keine Zwischenräume nötig.

Beispiel: 0000010100011 liefert „CHEF“.

Teil 2: Definitionen

Binärbaum-Terminologie: Überblick



Definition von Binärbäumen: „flach“

Definition 3.1.1

Ein **Binärbaum** T besteht aus einer endlichen Menge V von „inneren“ Knoten, einer dazu disjunkten endlichen Menge L von „äußeren“ Knoten sowie einer **injektiven** Funktion

$$child: V \times \{left, right\} \rightarrow V \cup L,$$

wobei Folgendes gilt:

(i) In $V \cup L$ gibt es genau einen Knoten r , der nicht als Wert $child(v, left)$ oder $child(v, right)$ vorkommt. Dieser Knoten r heißt die **Wurzel**.

Definition von Binärbäumen: „flach“

$child(v, left)$ [$child(v, right)$] heißt das **linke** [**rechte**] **Kind** von $v \in V$.

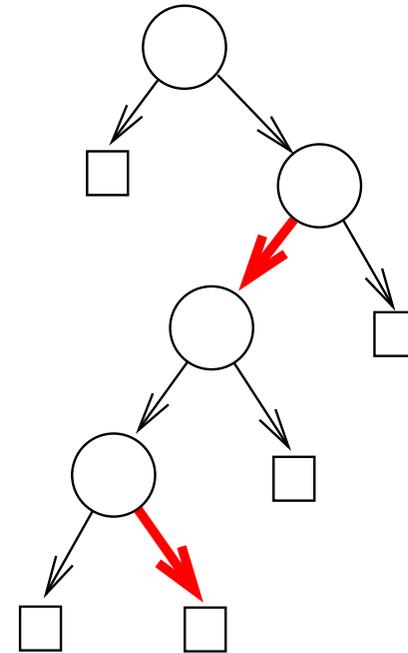
“ $child$ injektiv” heißt:

Für jeden Knoten $w \in V \cup L$ mit $w \neq r$ gibt es **genau einen** Knoten $u \in V$ mit $w = child(u, left)$ oder $w = child(u, right)$ (aber nicht beides).

Dieses u heißt **$p(w)$** , der **Vorgänger**knoten oder **Vater**knoten von w .

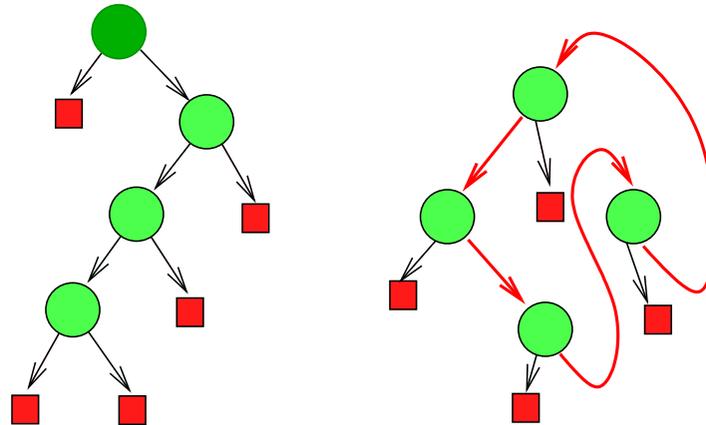
Wir sagen:

Von $p(w)$ nach w verläuft eine **Kante**.



Definition von Binärbäumen: „flach“

Beispiel: Die folgende (zweiteilige) Struktur erfüllt (i), ist aber kein Binärbaum:



(ii) „**Kreisfreiheit**“: Für jeden Knoten $w \in V \cup L$ gilt:

Die Folge $v_0 = w, v_1 = p(w), v_2 = p(v_1), v_3 = p(v_2), \dots$ bricht nach endlich vielen Schritten mit einem $v_d = r$ ab.

(Dadurch ist ein **eindeutiger Weg** von w zur Wurzel festgelegt. Also ist jeder Knoten des Binärbaums von der Wurzel aus entlang einer Kantenfolge („Weg“) erreichbar.)

Definition von Binärbäumen: „flach“

Markierungen (oder **Beschriftungen**) von Knoten und Kanten werden als Funktionen dargestellt: $(D, Z, X$ sind beliebige Mengen.)

Knotenmarkierungen: Funktionen $m_V: V \rightarrow D$ und $m_L: L \rightarrow Z$.

Kantenmarkierungen: Funktion $m_K: V \times \{left, right\} \rightarrow X$.

Beispiel: In einem Codierungsbaum sind die Blätter mit Buchstaben markiert, die beiden Kanten, die aus einem inneren Knoten herausführen, mit 0 und 1.

Definition von Binärbäumen: rekursiv

D sei Menge von (möglichen) Knotenmarkierungen

Z sei Menge von (möglichen) Blattmarkierungen

(Induktive) Definition 3.1.2

(i) Wenn $z \in Z$, dann ist (z) ein (D, Z) -Binärbaum.

// Blatt mit Markierung z . Auch: z (ohne Klammern), \boxed{z}

(ii) Wenn T_1, T_2 (D, Z) -Binärbäume sind und $x \in D$ ist, dann ist auch (T_1, x, T_2) ein (D, Z) -Binärbaum.

// Wurzel mit Markierung x , linker Unterbaum T_1 , rechter Unterbaum T_2 .

(iii)* Nichts sonst ist (D, Z) -Binärbaum.

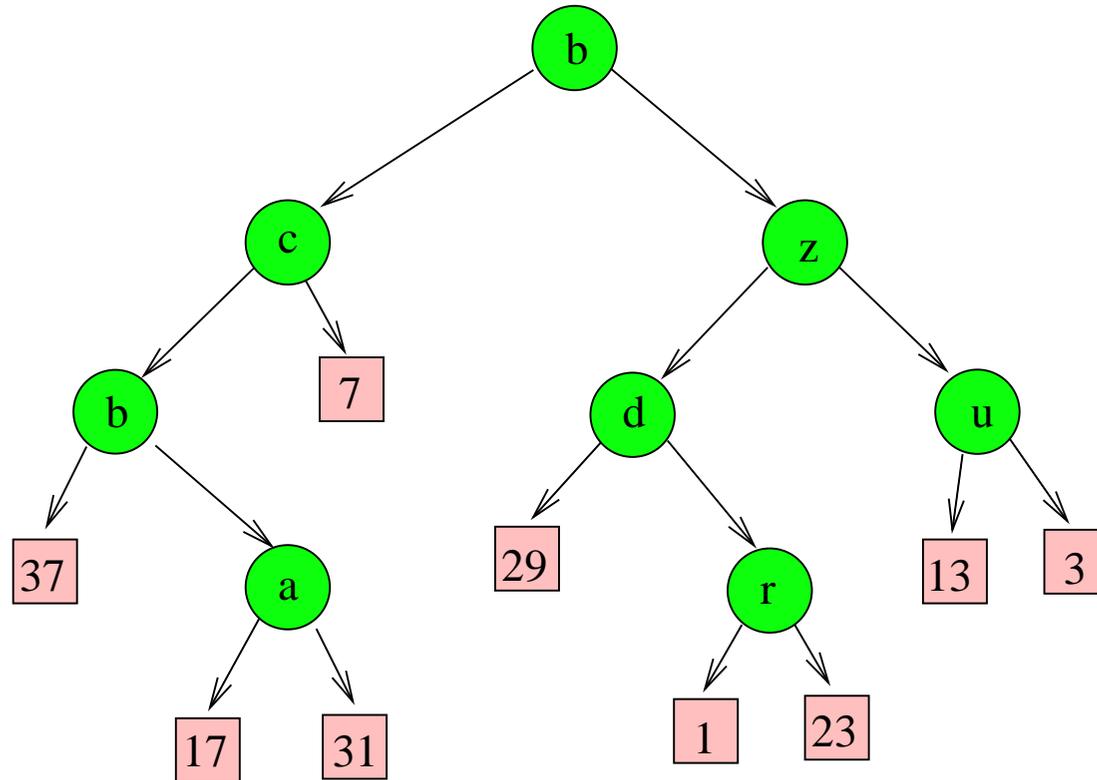
Definition ist äquivalent zur “flachen” Auffassung.

* (iii) wird bei induktiven Definitionen meist weggelassen.

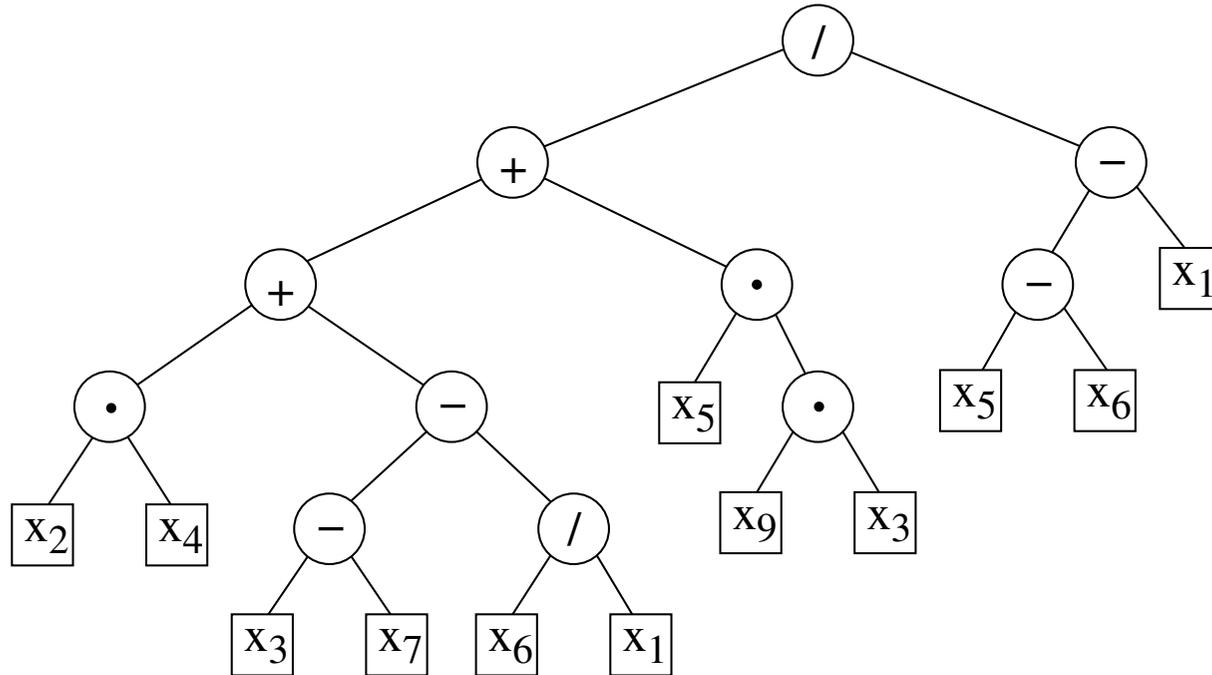
Beispiel: $D = \{a, \dots, z\}$ und $Z = \mathbb{N}$. – Tupeldarstellung (rekursiv):

$((37, b, (17, a, 31)), c, 7), b, ((29, d, (1, r, 23)), z, (13, u, 3))$

Graphische Darstellung („flach“):



Beispiel: **Arithmetischer Ausdruck.** $D = \{+, -, \cdot, /\}$ und $Z = \{x_1, x_2, x_3, \dots\}$.



$$(((x_2 \cdot x_4) + ((x_3 - x_7) - (x_6/x_1))) + (x_5 \cdot (x_9 \cdot x_3)))/(x_5 - x_6 - x_1)$$

Der Ausdruck ist genau die (rekursive) Tupeldarstellung, nur ohne Kommas!

Teil 3: Operationen

Spezifikation des Datentyps „Binärbaum“

Datentyp **Binärbaum** über D : (leere äußere Knoten)

1. Signatur:

Sorten: *Elements*
 Bintrees
 Boolean

Operationen: *empty*: \rightarrow *Bintrees*
 buildTree: *Elements* \times *Bintrees* \times *Bintrees* \rightarrow *Bintrees*
 leftSubtree: *Bintrees* \rightarrow *Bintrees*
 rightSubtree: *Bintrees* \rightarrow *Bintrees*
 data: *Bintrees* \rightarrow *Elements*
 isempty: *Bintrees* \rightarrow *Boolean*

2. Mathematisches Modell: (nächste Folie)

Sorten: *Elements*: (nichtleere) Menge D
Bintrees: $\{T \mid T \text{ ist } (D, \{-\})\text{-Binärbaum}\}$
Boolean: $\{true, false\}$

Op'en: $empty() := \square := (-)$ // „leerer Baum“

$buildTree(x, T_1, T_2) := (T_1, x, T_2)$

$leftSubtree(T) := \begin{cases} T_1, & \text{falls } T = (T_1, x, T_2) \\ \text{undefiniert,} & \text{falls } T = \square \end{cases}$

$rightSubtree(T) := \begin{cases} T_2, & \text{falls } T = (T_1, x, T_2) \\ \text{undefiniert,} & \text{falls } T = \square \end{cases}$

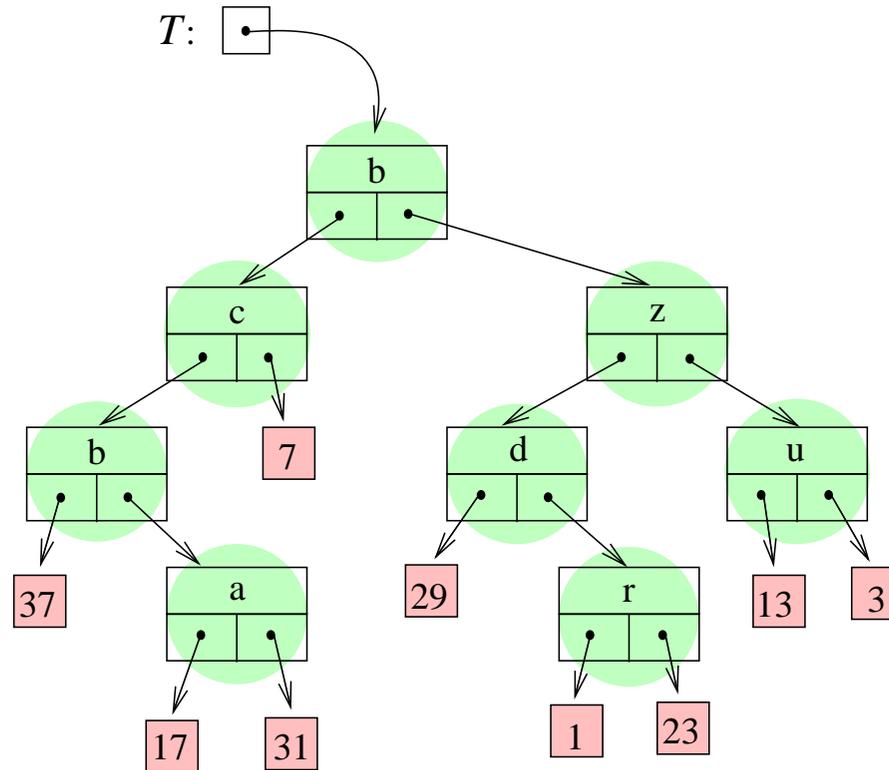
$data(T) := \begin{cases} x, & \text{falls } T = (T_1, x, T_2) \\ \text{undefiniert,} & \text{falls } T = \square \end{cases}$

$isempty(T) := \begin{cases} false, & \text{falls } T = (T_1, x, T_2) \\ true, & \text{falls } T = \square \end{cases}$

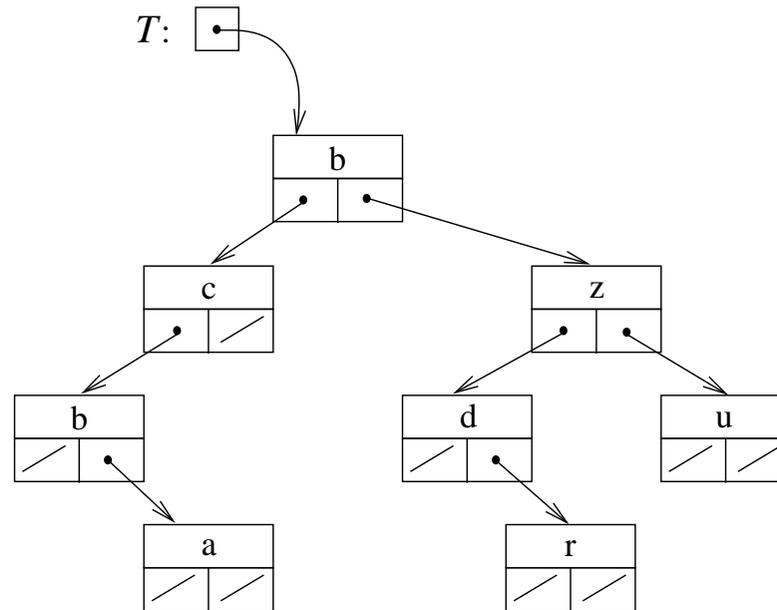
Binärbäume, Implementierung I:

Zeigerstruktur bzw. Referenzstruktur, hier mit externen Knoten, die Inhalt haben.

Für Implementierung in Java vgl. [\[AuP\]](#), Kap. 10 und [\[Saake/Sattler\]](#), Kap. 14.2.



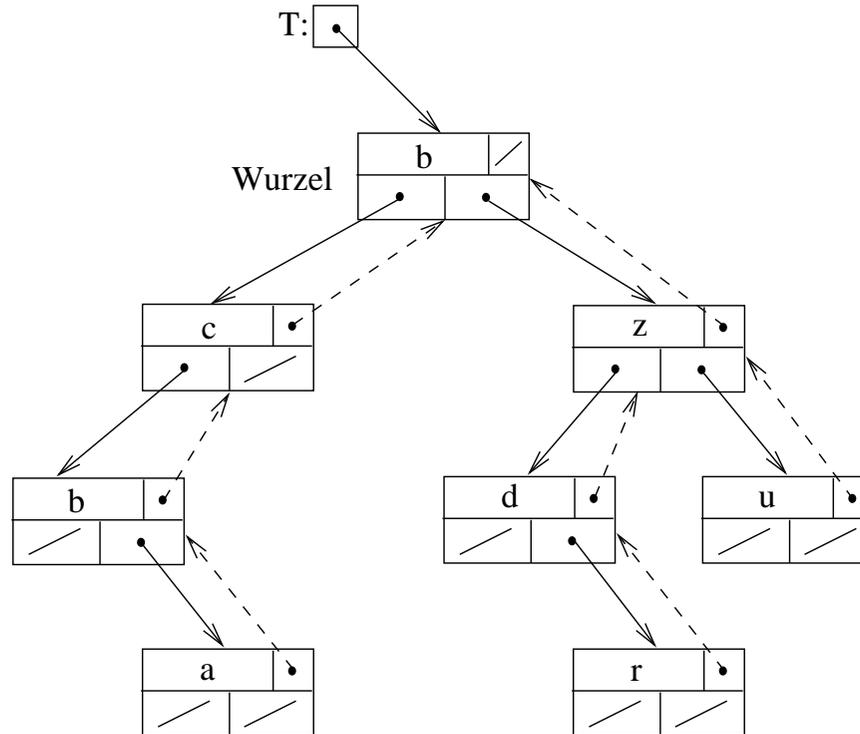
Binärbäume, Implementierung II: Leere externe Knoten



Externe Knoten werden nicht repräsentiert (null-Zeiger/-Referenz). – Oder: Ein externer Knoten ℓ , Referenz auf ℓ statt null (nullNode in [Saake/Sattler], Kap. 14.2.)

Speicherplatzbedarf für n Knoten: n Dateneinträge, $2n$ Zeiger/Referenzen, davon $n + 1$ mal null.

Mögliche Erweiterung, konzeptuell nur in „flacher“ Auffassung: Vorgängerzeiger.



Erleichtert Navigieren im Baum, besonders bei Verwenden von iterativen Verfahren.

Implementierung der Operationen des mathematischen Modells (leere externe Knoten):
Baum wird immer als Zeiger/Referenz auf den Wurzelknoten dargestellt.

empty(): Generiere einen Zeiger/eine Referenz T mit Wert `null`.
(Bzw.: Generiere `nullNode` und generiere Zeiger/Referenz T auf diesen Knoten.)

isempty(T): klar.

data(T): auch klar; nur definiert, wenn T nicht `null` ist.

leftSubtree(T) liefert Zeiger/Referenz $T.left$; nur definiert, wenn T nicht `null` ist.

rightSubtree(T): analog.

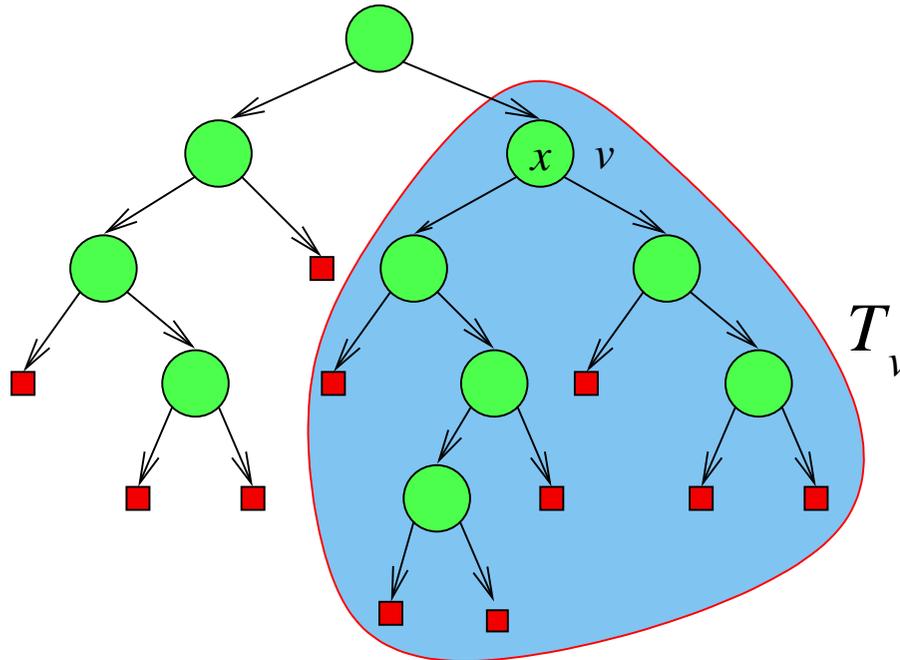
buildTree(x, T₁, T₂) erzeugt einen neuen Knoten (Wurzel) r mit Eintrag x , in die Kindplätze werden die Zeiger/Referenzen auf T_1 und T_2 eingetragen.

Achtung: Der **Benutzer** der *buildTree*-Operation muss sicherstellen, dass die **Knotenmengen** von T_1 und T_2 **disjunkt** sind!

Details: [\[Saake/Sattler\]](#), Kap. 14.2., wenn auch mit leicht anderen Bezeichnungen.

Teil 4: Terminologie

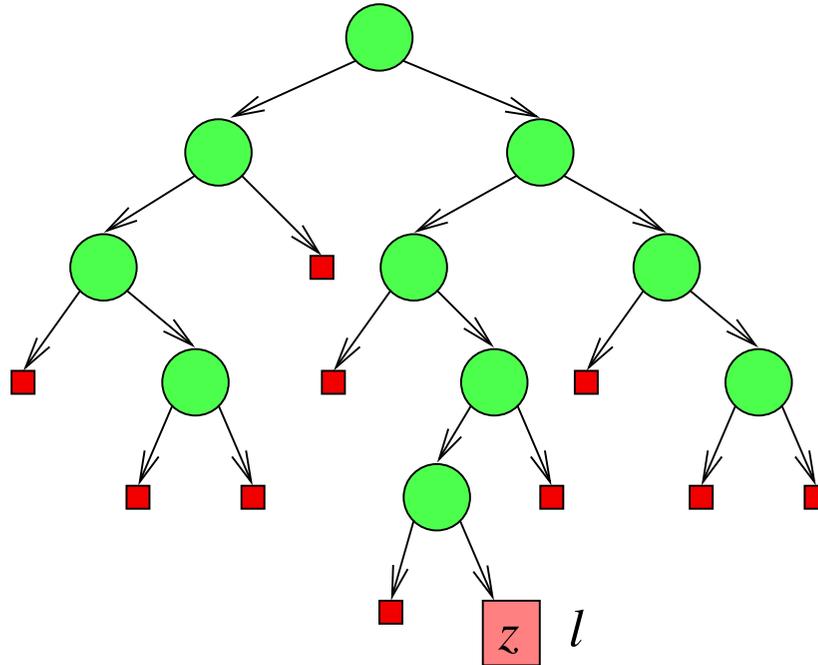
- T_v : Unterbaum mit Wurzel v



T_v ist der Teil von T , der aus den Knoten besteht, deren Weg zur Wurzel durch v verläuft. Die Wurzel von T_v ist v .

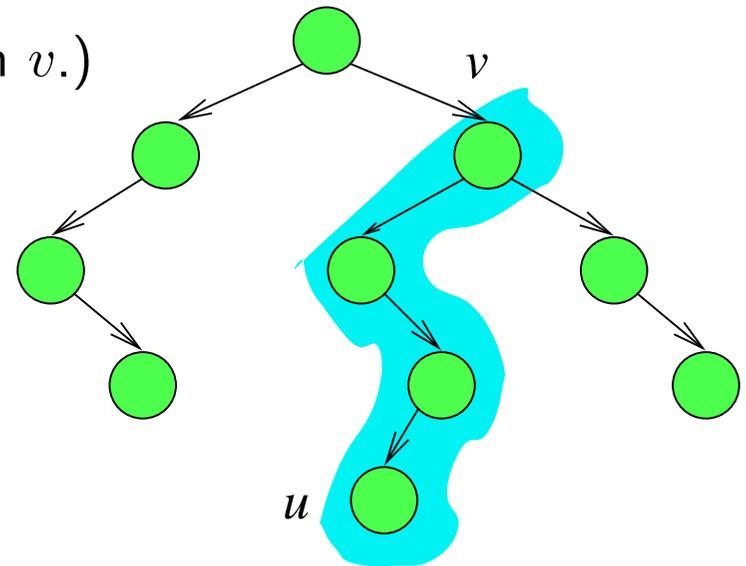
Dabei kann v ein beliebiger innerer oder äußerer Knoten sein.

-
- **(äußerer) Knoten** \equiv **Blatt l** , engl: „leaf“, Plural „leaves“
 - Wurzel, engl: „root“



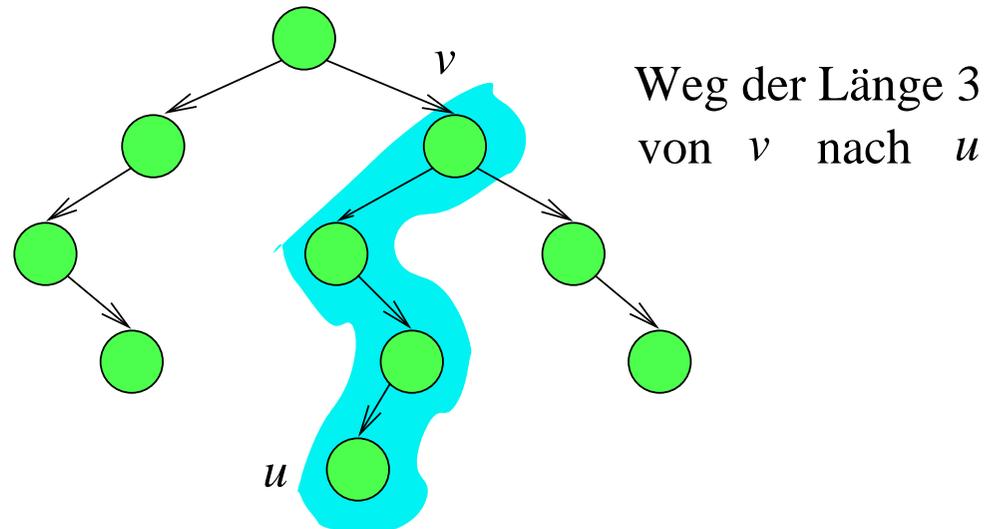
- Blatteintrag: $z = data(l)$

- **Vater/Vorgänger**, engl: „*parent*“, von v : $p(v)$
- linkes **Kind**, engl: „*left child*“, von v : von v : $child(v, left)$
- rechtes **Kind**, engl: „*right child*“, von v : von v : $child(v, right)$
- **Vorfahr** (Vater von Vater von Vater . . . von u),
engl: „*ancestor*“ (Im Bild: v ist Vorfahr von u .)
- **Nachfahr** (Kind von Kind von . . . von v),
engl: „*descendant*“ (Im Bild: u ist Nachfahr von v .)



Achtung: Oft gilt v als (uneigentlicher) Vorfahr/Nachfahr von v .

- **Weg** in T : Knotenfolge oder Kantenfolge, die von v zu einem Nachfahren u führt
- **Länge** eines Wegs (v_0, v_1, \dots, v_s) : Anzahl s der **Kanten** auf dem Weg

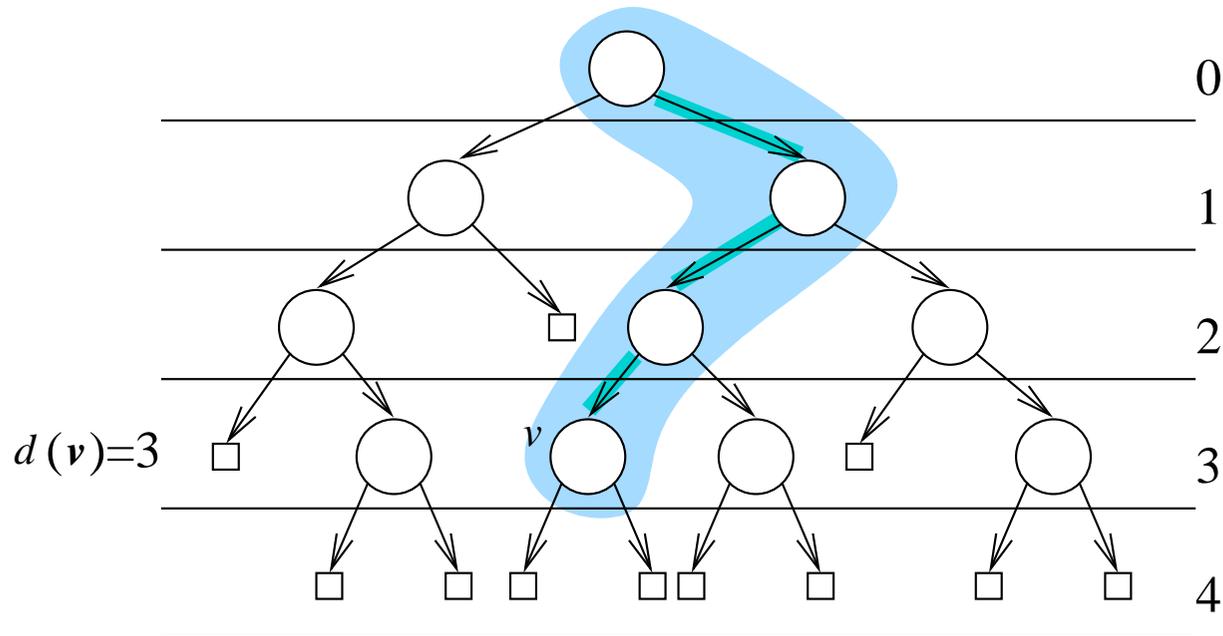


Tiefe/Level eines Knotens

Die Länge (das ist die Kantenzahl) des Wegs (v_0, v_1, \dots, v_d) von der Wurzel $r = v_0$ zum Knoten $v = v_d$ heißt die **Tiefe** oder das **Level** $d(v)$ von Knoten v .

Die Knoten mit Tiefe l bilden das **Level l** .

Auch externen Knoten wird so eine Tiefe zugeordnet.



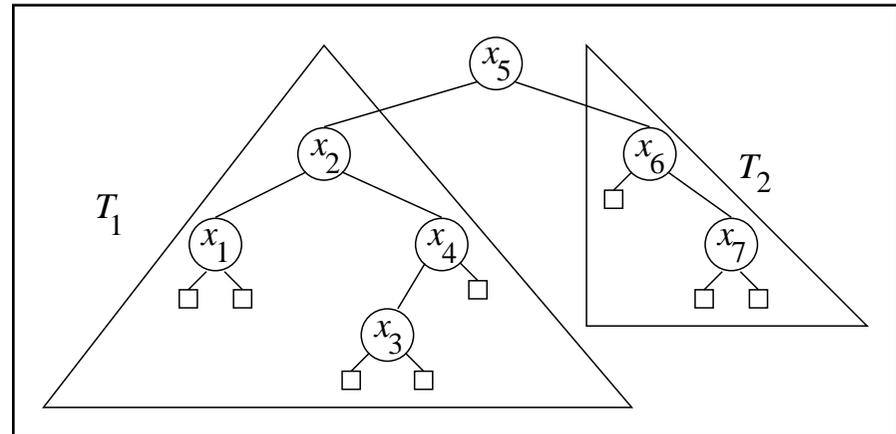
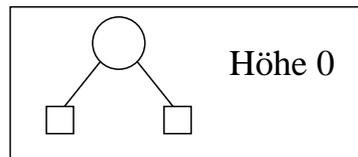
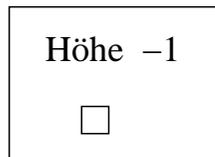
Tiefe (oder synonym: Höhe) $d(T)$ (oder: $h(T)$) eines Binärbaums:

1. Fall: Die externen Knoten sind leer (\square):

$$d(T) := \max(\{-1\} \cup \{d(v) \mid v \text{ innerer Knoten in } T\}).$$

Rekursive Formel: $d(T) = -1$ für $T = \square$ und $d((T_1, x, T_2)) = 1 + \max\{d(T_1), d(T_2)\}$.

Beispiele: Höhe/Tiefe -1 , 0 und 3 .

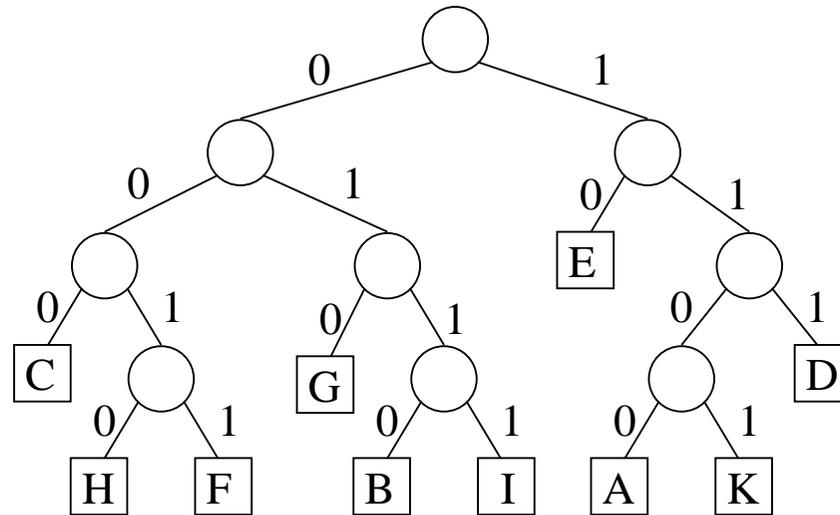


Tiefe (oder synonym: Höhe) $d(T)$ (oder: $h(T)$) eines Binärbaums:

2. Fall: Die externen Knoten sind nicht leer, also von der Form \boxed{z} , $z \in Z$:

$d(T) := \max\{d(v) \mid v \text{ äußerer Knoten in } T\}$.

Rekursive Formel: $d(\square) = 0$; $d((T_1, x, T_2)) = 1 + \max\{d(T_1), d(T_2)\}$.

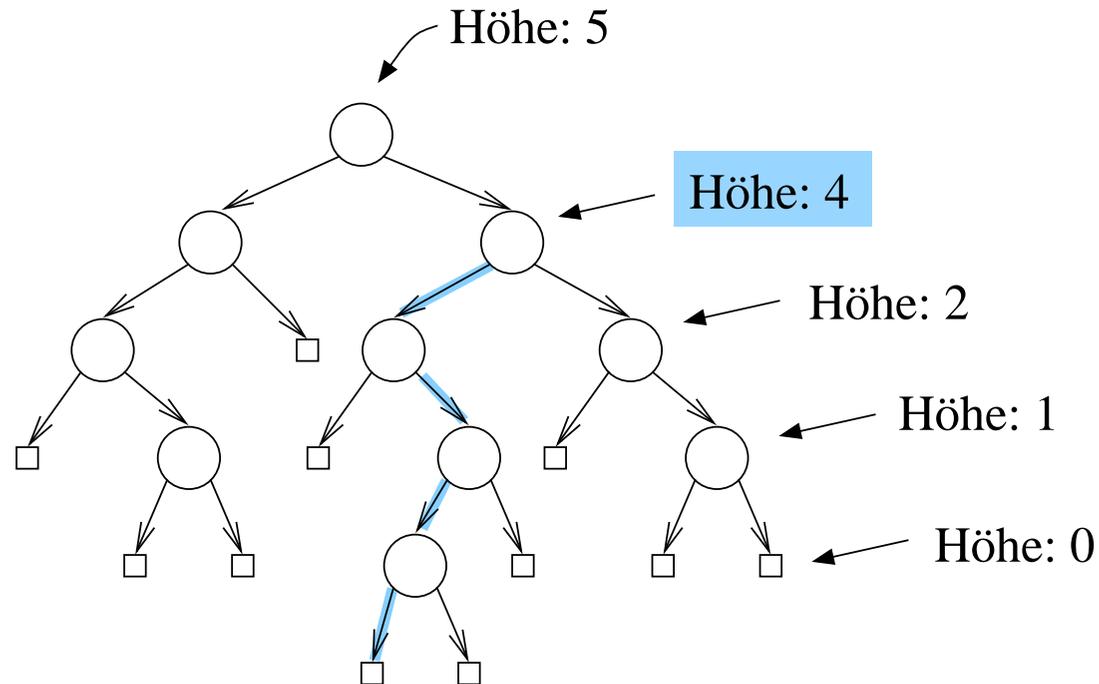


Ein Baum mit Tiefe 4, gleich der maximalen Länge eines Weges zu einem Blatt.

Höhe eines Knotens:

Die **Höhe** $h(v)$ eines Knotens in T ist die **Tiefe** $d(T_v)$ des Teilbaums mit Wurzel v .

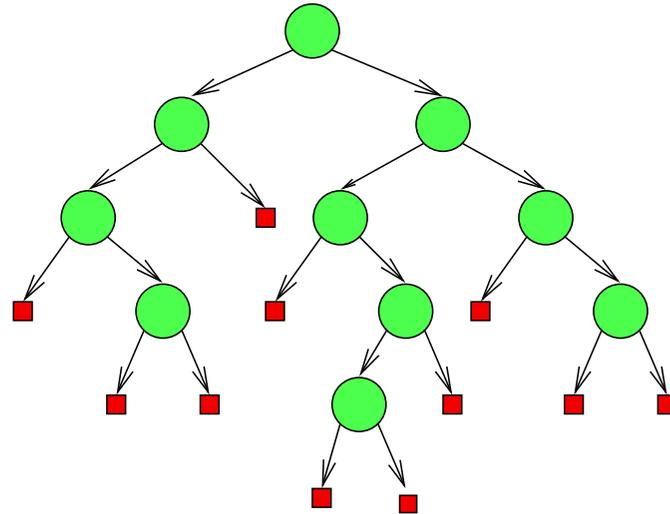
Beispiel:



Hier angegeben: Fall „Blätter nicht leer“.

Teil 5: TIPL und TEPL

3.3 Binärbäume: Eigenschaften



Proposition 3.3.1

Für einen Binärbaum T mit $n \geq 0$ inneren Knoten und **leeren** äußeren Knoten gilt:

- (a) T hat $2n$ Zeiger/Kanten. (Im Beispiel: $n = 10$, $2n = 20$.)
- (b) T hat $n + 1$ äußere Knoten. (Im Beispiel: $n + 1 = 11$.)

(c) Auf Level l liegen höchstens 2^l Knoten, für $l = 0, 1, 2, \dots$

(d) Auf Level $0, 1, \dots, l$ zusammen liegen höchstens

$$1 + 2 + 4 + \dots + 2^l = 2^{l+1} - 1$$

innere Knoten.

Daraus: $\log(n + 1) \leq l + 1$, für das Level l mit dem tiefsten inneren Knoten.

(e)* $\lceil \log(n + 1) \rceil - 1 \leq d(T) \leq n - 1$.

Merke: Die Tiefe eines Binärbaums mit n inneren Knoten ist

mindestens $\log n - 1$ und höchstens $n - 1$.

* $\log n = \log_2 n$.

Beweis von Proposition 3.3.1:

(a) Jeder innere Knoten hat zwei Zeiger/Kanten zu Kindern.

(b) Jeder Knoten außer der Wurzel hat eine eingehende Kante. Nach (a) gibt es $2n$ Kanten, und es gibt $n - 1$ innere Knoten, die nicht die Wurzel sind. Daher muss es $2n - (n - 1) = n + 1$ äußere Knoten geben.

(c) Beweis durch Induktion über l .

$l = 0$: Nur die Wurzel hat Level 0, und $2^0 = 1$.

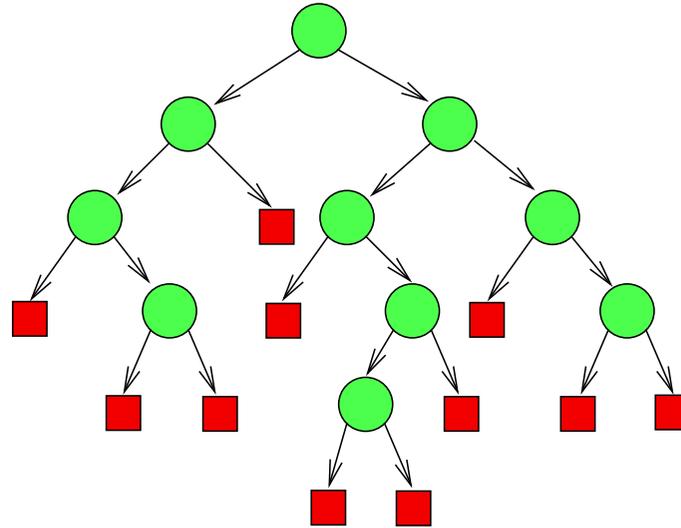
„ $l - 1 \rightarrow l$ “: Nach I.V. liegen auf Level $l - 1$ höchstens 2^{l-1} Knoten. Jeder davon hat höchstens zwei Kinder. Also liegen auf Level l höchstens $2^{l-1} \cdot 2 = 2^l$ Knoten.

(d) Sei l das maximale Level, das einen inneren Knoten enthält. Alle Levels bis l zusammen enthalten n Knoten; nach (c) ist $n \leq 2^0 + 2^1 + \dots + 2^l$. Die Schranke $n \leq 2^{l+1} - 1$ folgt dann mit Hilfe der Summenformel $\sum_{0 \leq j < k} q^j = (q^k - 1)/(q - 1)$ für geometrische Reihen.

Daraus folgt $n + 1 \leq 2^{l+1}$, also durch Logarithmieren: $\log(n + 1) \leq l + 1$.

(e) In der Situation von (d) gilt $l \geq \log(n + 1) - 1$; weil l ganzzahlig ist, folgt $l \geq \lceil \log(n + 1) \rceil - 1$.

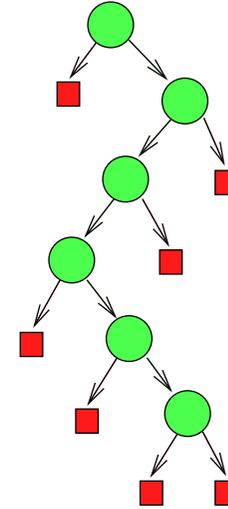
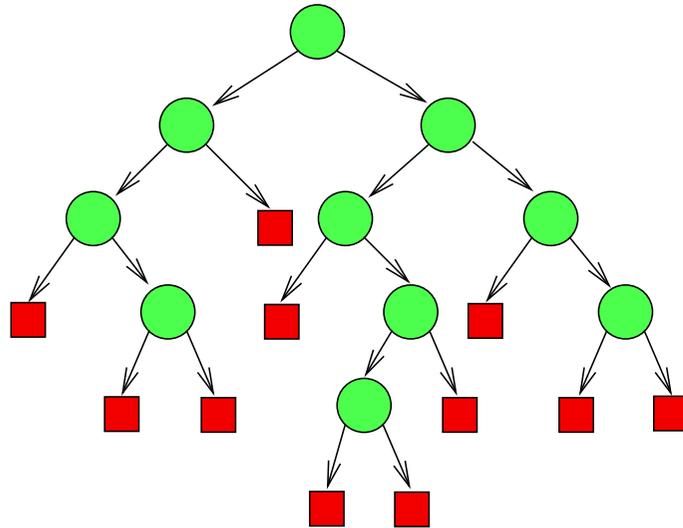
□



Proposition 3.3.2

Für einen Binärbaum T mit $N \geq 1$ **nichtleeren** äußeren Knoten gilt:

- (a) T hat $n = N - 1$ innere Knoten. (Im Beispiel: $N = 11$, $n = 10$.)
- (b) T hat $2N - 2$ Zeiger/Kanten.



(c) $N \leq 2^{d(T)}$.

(Beweis: Induktion über Binärbaume, rekursive Struktur.)

(d) $N - 1 \geq d(T) \geq \lceil \log N \rceil$.

Merke:

Die Tiefe eines Binärbaums mit N **nicht**leeren äußeren Knoten ist $\geq \log N$.

Beweis von 3.3.2:

(a) Dies ist eine Umformulierung von Prop. 3.3.1(b).

(b) Dies ist eine Umformulierung von Prop. 3.3.1(a).

(c) Durch (verallgemeinerte) Induktion über $d = d(T)$ zeigen wir, dass die Anzahl N_T der Blätter von T nicht größer als $2^{d(T)}$ ist.

I.A.: $d = 0$: Dann besteht T nur aus einem Blatt, also $N_T = 1$, und $2^0 = 1$.

Sei nun $d = d(T) > 0$.

I.V.: Behauptung ist richtig für alle Bäume mit Tiefe kleiner als d .

I.-Schritt: T hat eine Wurzel, die innerer Knoten ist, und zwei Unterbäume T_1 und T_2 mit $\max\{d(T_1), d(T_2)\} = d - 1$.

Klar: Baum T hat $N_T = N_{T_1} + N_{T_2}$ Blätter.

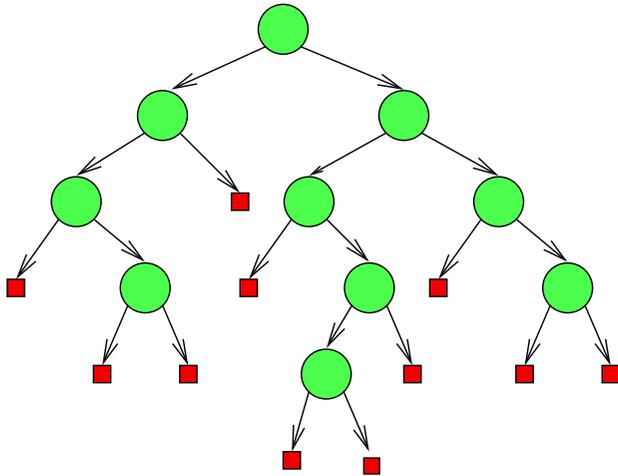
Nach I.V. gilt $N_{T_i} \leq 2^d$, also $N_{T_i} \leq 2^{d-1}$, für $i = 1, 2$.

Also: $N_T = N_{T_1} + N_{T_2} \leq 2 \cdot 2^{d-1} = 2^d$.

(d) Ergebnis (c) bedeutet $N_T \leq 2^{d(T)}$. Logarithmieren liefert $\log(N_T) \leq d(T)$. □

Totale äußere Weglänge

Laufe nacheinander von der Wurzel zu jedem **äußeren** Knoten l .
Wieviele Kanten werden **insgesamt** durchlaufen?



$$2 + 3 \cdot 3 + 5 \cdot 4 + 2 \cdot 5 = 41.$$

Totale äußere Weglänge („total external path length“):

$$\text{TEPL}(T) := \sum_{l \in L} d(l). \quad (\text{Im Beispiel: } 41.)$$

$$\text{Mittlere äußere Weglänge: } \text{TEPL}(T)/(n + 1). \quad (\text{Im Beispiel } \frac{41}{11}.)$$

Proposition 3.3.3

Für jeden Binärbaum mit n inneren Knoten gilt: $\text{TEPL}(T) = \text{TIPL}(T) + 2n$.

(Im Beispiel: $41 = 21 + 2 \cdot 10$.)

Beweis: Induktion über die Knotenanzahl n .

I.A.: $n = 0$: $\text{TEPL}(T) = 0$ (ein Weg der Länge 0) und
 $\text{TIPL}(T) = 0$ (leere Summe, da kein innerer Knoten vorhanden).

Sei nun T Baum mit $n > 0$ Knoten. **I.V.:** Die Behauptung gilt für Bäume mit $n - 1$ Knoten.

I.-Schritt: In T gibt es einen inneren Knoten v , der zwei externe Kinder hat. Sei k die Tiefe von v . Ersetze v und seine beiden Kinder durch neuen externen Knoten ℓ . Dies liefert T' mit $n - 1$ Knoten. Dann:

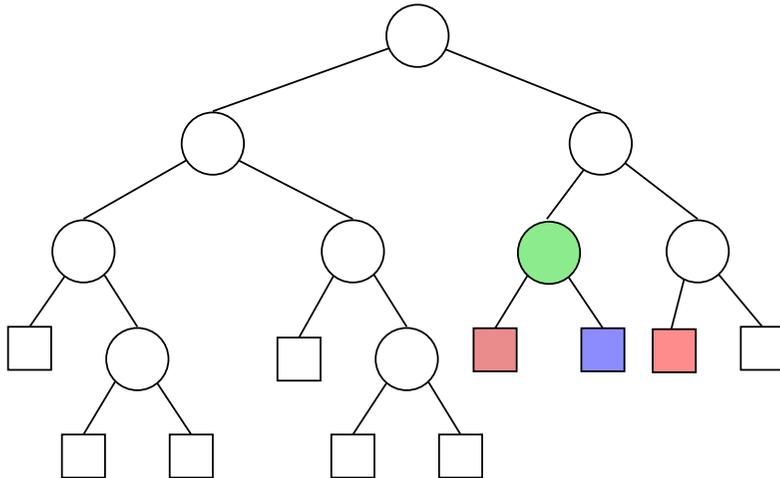
$$\begin{aligned} \text{TEPL}(T') &= \text{TEPL}(T) - 2(k + 1) + k &= \text{TEPL}(T) - k - 2 & \text{[2 Kinder von } v \text{ weg, } \ell \text{ dazu];} \\ \text{TIPL}(T') & &= \text{TIPL}(T) - k & \text{[} v \text{ weg].} \end{aligned}$$

Durch Subtrahieren: $\text{TEPL}(T') - \text{TIPL}(T') = \text{TEPL}(T) - 2 - \text{TIPL}(T)$.

Nach I.V.: $\text{TEPL}(T') - \text{TIPL}(T') = 2(n - 1)$.

Es folgt $\text{TEPL}(T) - \text{TIPL}(T) = 2n$, das ist die I.-Beh. □

Nächstes Ziel: Auch die **mittlere Weglänge** in Binärbäumen wächst mindestens **logarithmisch** mit der Knotenanzahl.



Lemma

Sei T Binärbaum mit N Blättern und **kleinstmöglicher** totaler externer Weglänge. Dann gibt es in T keine zwei Blätter ℓ und ℓ' mit $d(\ell) \geq d(\ell') + 2$.

Beweis hiervon: Sonst könnte man $\text{TEPL}(T)$ durch Umbauen verkleinern, wie im Bild.

Proposition 3.3.5 Binärbaum T hat N äußere Knoten $\Rightarrow \text{TEPL}(T) \geq N \log N$.

Beweis: Wähle T mit kleinstmöglichem Wert $\text{TEPL}(T)$ unter allen Bäumen mit N Blättern.

Sei $d = d(T)$ die Tiefe von T . Nach dem Lemma gibt es in T Blätter auf Level d und vielleicht auch auf Level $d - 1$. Sei $a := \#(\text{Blätter auf Level } d - 1)$ und $b := \#(\text{Blätter auf Level } d)$.

Klar: $a + b = N$. Weiter gilt: $2a + b = 2^d$. (Ersetze Blätter auf Level $d - 1$ durch einen inneren Knoten mit zwei Blättern als Kindern. Dann hat Level d genau 2^d Blätter.) Umstellen ergibt:

$$a = 2^d - N \text{ und } b = 2N - 2^d.$$

Es gilt: (*) $\text{TEPL}(T) = a(d - 1) + bd = (2^d - N)(d - 1) + (2N - 2^d)d = N(d + 1) - 2^d$.

Beobachte: Weil $a \geq 0$, gilt $N \leq 2^d$, und weil $b > 0$, gilt $N > 2^{d-1}$.

Wir diskutieren die Funktion

$$f: [2^{d-1}, 2^d] \rightarrow \mathbb{R}, f(x) = x \log x - (x(d + 1) - 2^d).$$

Wir haben:

$f(2^{d-1}) = 2^{d-1}(d - 1) - (2^{d-1}(d + 1) - 2^d) = 0$ und $f(2^d) = 2^d d - (2^d(d + 1) - 2^d) = 0$.

f ist **konvex** in $[2^{d-1}, 2^d]$ (denn $f'(x) = \log x + \log e - (d + 1)$ und $f''(x) = (\log e)/x > 0$).

Zwischen seinen beiden Nullstellen muss f negativ sein, also gilt $(x(d + 1) - 2^d) \geq x \log x$ für $x \in [2^{d-1}, 2^d]$. Mit (*) folgt $\text{TEPL}(T) = (N(d + 1) - 2^d) \geq N \log N$. \square

Korollar 3.3.6

Für die **mittlere äußere Weglänge** eines Binärbaums T mit N Blättern gilt:

$$\frac{1}{N} \text{TEPL}(T) \geq \log N.$$

Nach Proposition 3.3.3: $\text{TEPL}(T) = \text{TIPL}(T) + 2n$.

Also: $\text{TIPL}(T) \geq \text{TEPL}(T) - 2n \geq (n+1) \log(n+1) - 2n > n(\log n - 2)$.

Korollar 3.3.7

Für die **mittlere innere Weglänge** eines Binärbaums T mit n Knoten gilt:

$$\frac{1}{n} \text{TIPL}(T) > \log n - 2.$$

Teil 6: Baumdurchläufe

3.4 Durchläufe durch Binärbäume

(D, Z) -Binärbäume kann man systematisch durchlaufen, die Knoten nacheinander besuchen und (anwendungsabhängige) Aktionen an den Knoten ausführen.

Abstrakt gefasst: *visit*-Operation.

Organisiere *data*-Teil der Knoten als Objekte einer Klasse;
visit(T) löst Abarbeitung einer Methode von *data*(T) aus.

Möglichkeiten für *visit* bei Knoten v :

- Daten *data*(v) ausgeben;
- laufende Nummer für v vergeben;
- Zeiger auf v an Liste anhängen;
- zu v gehörende Aktion ausführen (z. B. arithmetische Operation)

usw.

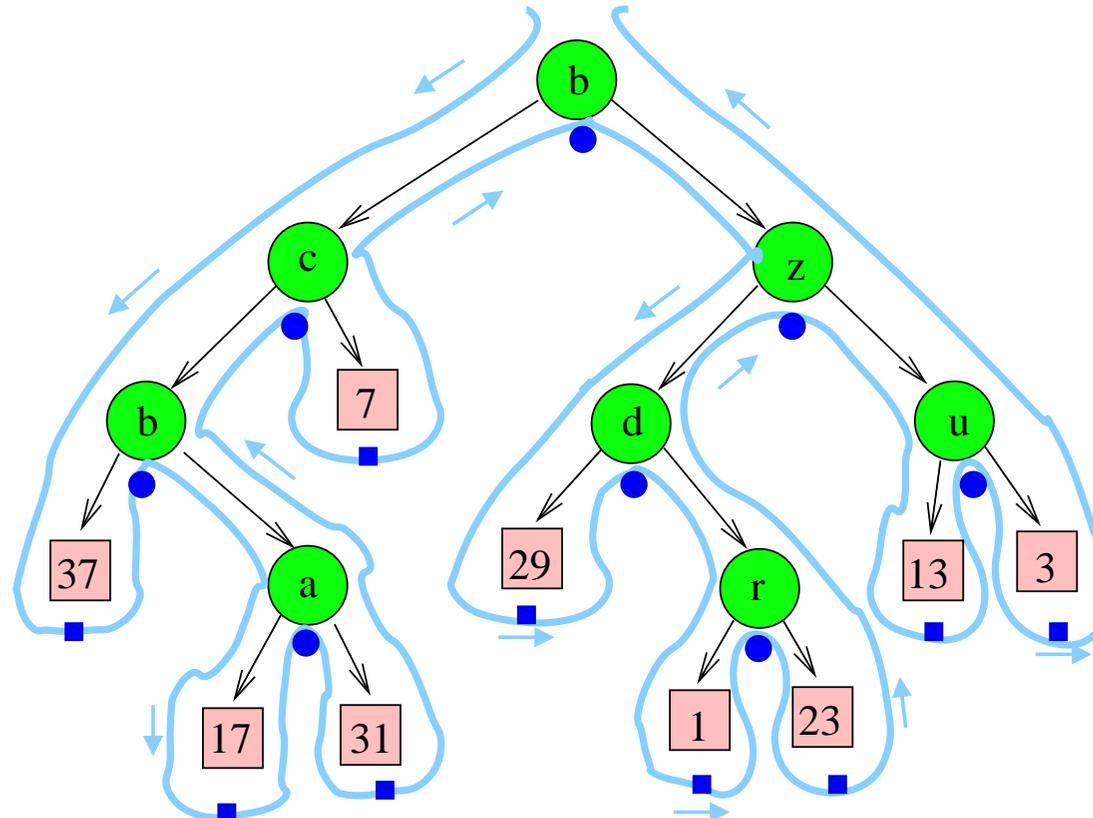
Inorder-Durchlauf durch T

falls $T = [z]$:
 $\text{e-visit}(z)$; // besuche äußeren Knoten

falls $T = (T_1, x, T_2)$:
 Inorder-Durchlauf durch T_1 ;
 $\text{i-visit}(x)$; // besuche inneren Knoten
 Inorder-Durchlauf durch T_2 .

Inorder: „Erst den linken Unterbaum, dann die Wurzel,
dann den rechten Unterbaum.“

Inorder-Durchlauf durch T



Ausgabe: 37, b, 17, a, 31, c, 7, b, 29, d, 1, r, 23, z, 13, u, 3.

Beobachte: Man besucht abwechselnd externe und interne Knoten.

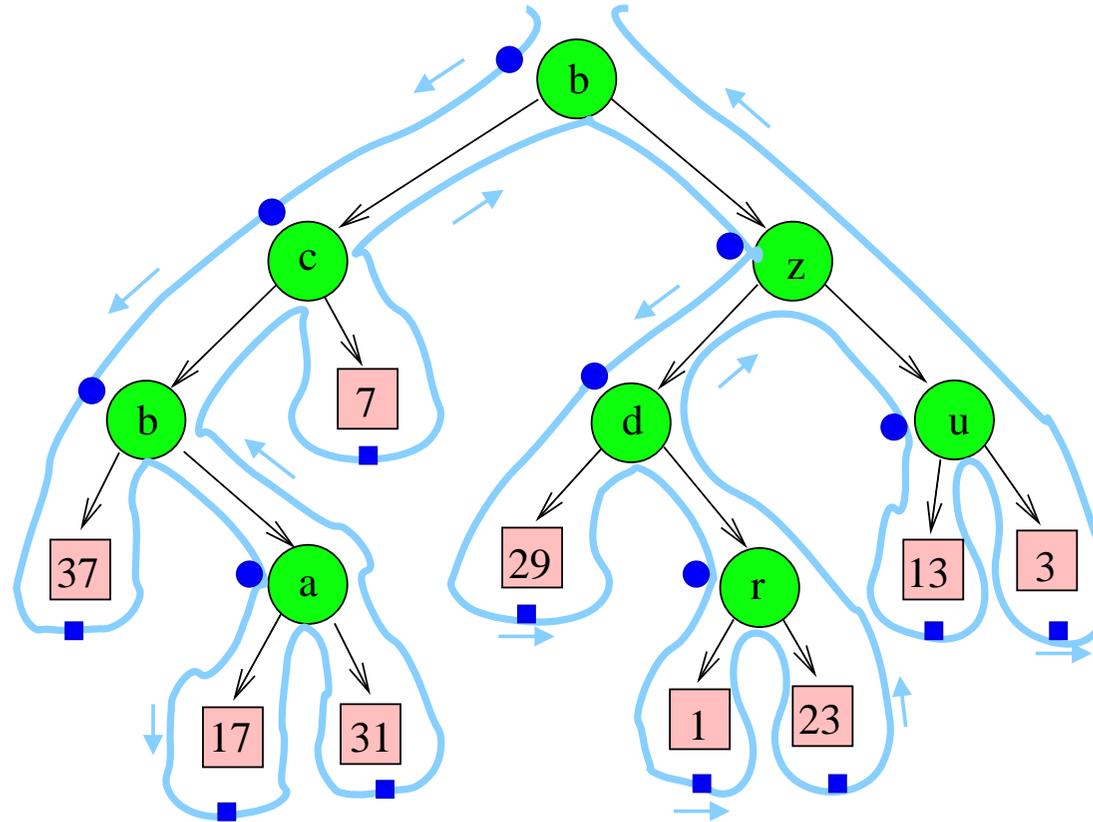
Präorder-Durchlauf durch T

falls $T = \boxed{z}$:
e-visit(z) ;

falls $T = (T_1, x, T_2)$:
i-visit(x) ;
Präorder-Durchlauf durch T_1 ;
Präorder-Durchlauf durch T_2 ;

Präorder: „Erst die Wurzel, dann den linken Unterbaum,
dann den rechten Unterbaum.“

Präorder-Durchlauf durch T



Ausgabe: b, c, b, 37, a, 17, 31, 7, z, d, 29, r, 1, 23, u, 13, 3

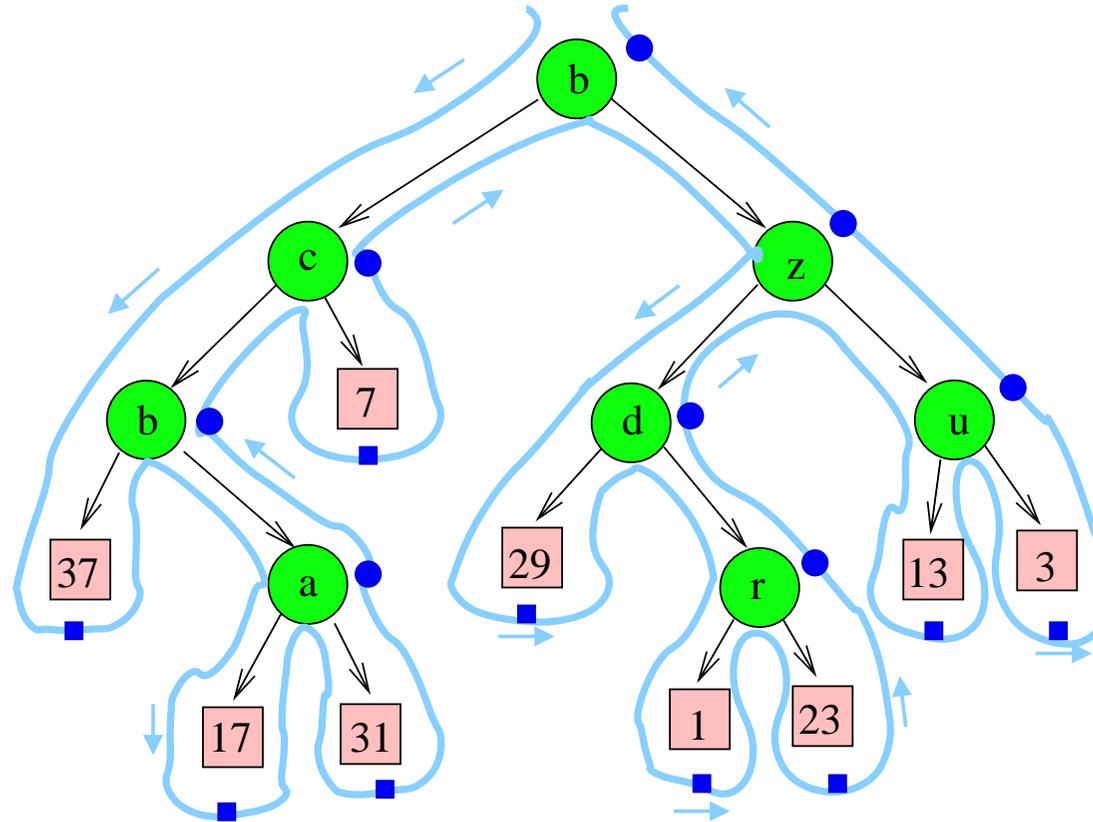
Postorder-Durchlauf durch T

falls $T = \boxed{z}$:
e-visit(z) ;

falls $T = (T_1, x, T_2)$:
Postorder-Durchlauf durch T_1 ;
Postorder-Durchlauf durch T_2 ;
i-visit(x) ;

Postorder: „Erst den linken Unterbaum, dann den rechten Unterbaum,
zuletzt die Wurzel.“

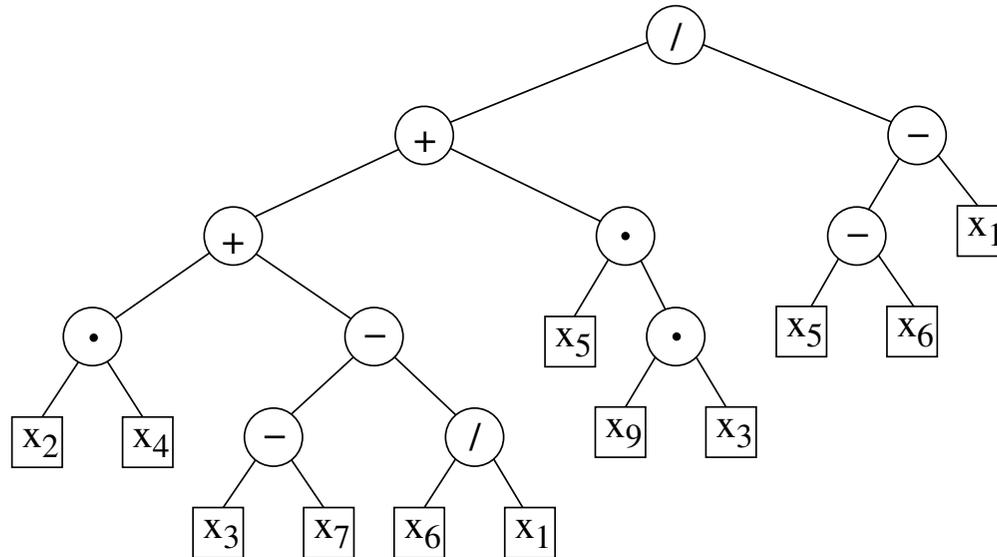
Postorder-Durchlauf durch T



Ausgabe: 37, 17, 31, a, b, 7, c, 29, 1, 23, r, d, 13, 3, u, z, b.

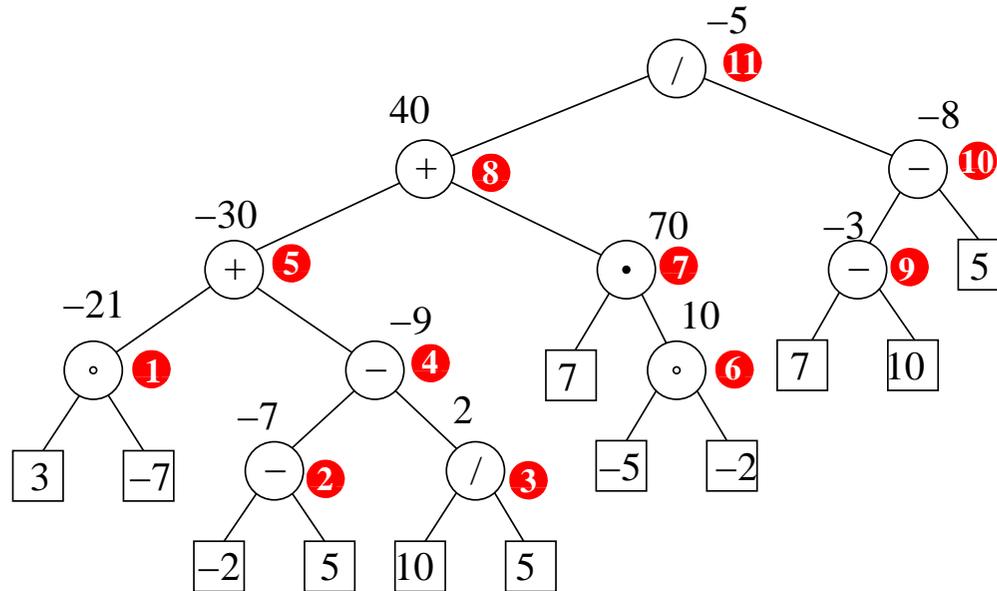
Beispiel: Arithmetischer Ausdruck als Baum

$D = \{+, -, \cdot, /\}$ und $Z = \{x_1, x_2, x_3, \dots\}$.



Beispiel: Arithmetischer Ausdruck als Baum

$D = \{+, -, \cdot, /\}$ und $Z = \{x_1, x_2, x_3, \dots\}$.



Ordne den Blättern konkrete Werte zu, z.B. $x_1 = 5$, $x_2 = 3$, $x_3 = -2$, $x_4 = -7$, $x_5 = 7$, $x_6 = 10$, $x_7 = 5$, $x_9 = -5$. Dann ordne jedem Knoten einen Wert zu. (Ergebnis des Unterbaums.) Geeignete Reihenfolge: Postorder-Durchlauf. **(Rote Kreise.)**

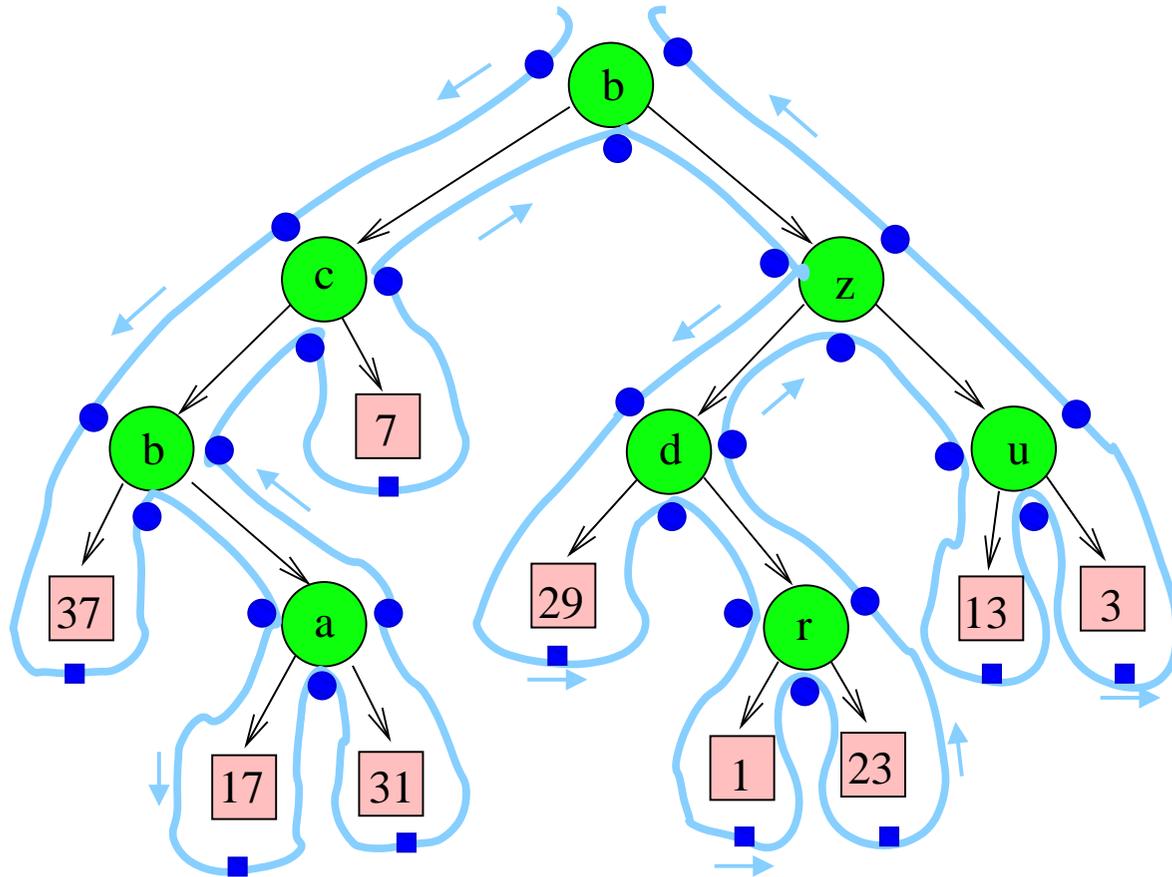
Auch möglich: Kombi-Durchlauf:

falls $T = \boxed{z}$:
 e-visit(z) ;

falls $T = (T_1, x, T_2)$:
 preord-i-visit(x) ;
 Kombi-Durchlauf durch T_1 ;
 inord-i-visit(x) ;
 Kombi-Durchlauf durch T_2 ;
 postord-i-visit(x) ;

Ermöglicht Datentransport von der Wurzel nach unten in den Baum hinein und wieder zurück.

Kombi-Durchlauf durch T



Beispiel für Anwendung eines Kombi-Durchlaufs:

Gegeben: Binärbaum T als verzeigerte Struktur (Folie 16).

Aufgabe: Generiere die rekursive Darstellung von T , aufgeschrieben als ein String (Folien 11–12).

Zeichen sollen sequenziell in die Ausgabe geschrieben werden („**print**“).

Dazu muss man nur die vier Besuchsmethoden geeignet spezifizieren:

```
e-visit( $z$ ):           print("(" ,  $z$  , ")").  
preord-i-visit( $x$ ):    print("(").  
inord-i-visit( $x$ ):     print(", " ,  $x$  , ", ").  
postord-i-visit( $x$ ):  print(")").
```

Man sollte dies an einem Beispiel testen.

Zeitanalyse für Baumdurchlauf

Die Baumdurchläufe können mittels rekursiver Prozeduren implementiert werden.

Beispiel: Inorder-Durchlauf. Prozedur: `inorder(T)`

Angewendet auf einen Baum T mit n inneren Knoten.

Für jeden inneren und äußeren Knoten wird die Prozedur `inorder(.)` einmal aufgerufen, insgesamt $(2n + 1)$ -mal.

Kosten pro Aufruf: $O(1)$ plus Kosten für i-visit/e-visit-Operation.

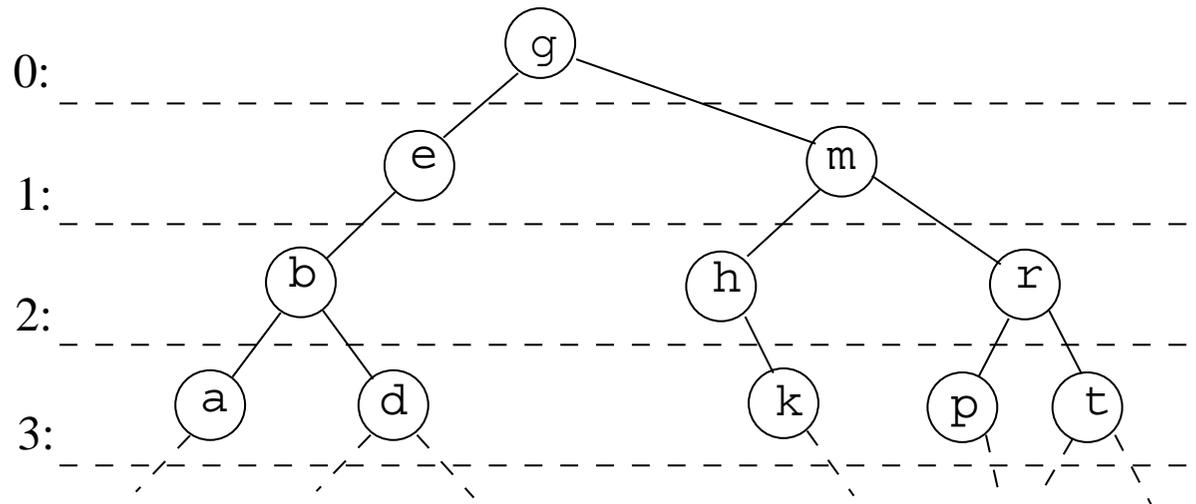
Insgesamt: $(2n + 1) \cdot (O(1) + C_{\text{visit}})$,

wobei C_{visit} eine Schranke für die Kosten der visit-Operationen ist.

Behauptung 3.4.1

Baumdurchläufe haben lineare Laufzeit.

Levelorder-Durchlauf



Wunschausgabe: g em bhr adkpt , „levelweise von links nach rechts“.

Datenstruktur:

Queue Q, die Zeiger/Referenzen auf Binärbaumknoten speichern kann.

Prozedur Leveldurchlauf (T : Baum mit leeren äußeren Knoten)

Eingabe: v : node; // Wurzel von T

```
(1)   Q: Queue für node;
(2)   count  $\leftarrow$  0;
(3)   if  $v \neq \text{NULL}$  then // entdecke Wurzel
(4)       count++;  $v.lnum \leftarrow$  count;
(5)        $v.level \leftarrow$  0; enqueue( $v$ );
(6)   while not Q.isempty do
(7)        $w \leftarrow$  Q.first; Q.dequeue;
(8)       visit( $w$ ); // bearbeite  $w$ 
(9)        $p \leftarrow$   $w.leftchild$ ;
(10)  if  $p \neq \text{NULL}$  then
(11)      count++;  $p.lnum \leftarrow$  count; // entdecke  $p$ 
(12)       $p.level \leftarrow$   $w.level + 1$ ; enqueue( $p$ );
(13)   $p \leftarrow$   $w.rightchild$ ;
(14)  if  $p \neq \text{NULL}$  then
(15)      count++;  $p.lnum \leftarrow$  count; // entdecke  $p$ 
(16)       $p.level \leftarrow$   $w.level + 1$ ; enqueue( $p$ );
```

Behauptung 3.4.2 „Lineare Zeit, Korrektheit“

(a) Levelorder-Durchlauf benötigt Zeit $n \cdot t_{\text{visit}} + O(n)$,
wobei t_{visit} die Zeit für `visit` angibt.

(b) Der Levelorder-Durchlauf ordnet jedem Binärbaumknoten v (in `v.level`)
korrekt seine Tiefe zu.

Beweis: Man zeigt durch Induktion über das Level von v :

Jeder innere Knoten v des Binärbaums wird genau einmal in die Queue **eingefügt**,
genau einmal **entnommen** und im Schleifenrumpf (7)–(16) **bearbeitet**.

Dabei wird die korrekte Levelnummer vergeben.

Weil Queue-Operationen Zeit $O(1)$ benötigen, ist der restliche Zeitbedarf $O(n)$. \square

Ergänzung: (Angeordnete) Bäume mit beliebiger Anzahl von Kindern

Anstelle von Binärbäumen kann man auch Bäume betrachten, bei denen ein (innerer) Knoten v eine beliebige Anzahl $d_v \geq 0$ von Kindern haben kann. Wenn $d_v = 0$ ist, ist v ein Blatt. Dabei hat ein Knoten v nicht ein „linkes“ und ein „rechtes“ Kind, sondern Kind 1, \dots , Kind d_v (ohne die Möglichkeit eines „leeren“ Kindplatzes).

Induktive Definition von „Mehrwegbaum mit Knotenmarkierungen aus D “:

- (i) Für $x \in D$ ist (x) ein Mw- D -Baum.
- (ii) Wenn $d \geq 1$ ist und T_1, \dots, T_d Mw- D -Bäume sind und $x \in D$ ist, dann ist auch (x, T_1, \dots, T_d) ein Mw- D -Baum.

Implementierung solcher Bäume: Ein Knoten enthält einen Dateneintrag aus D und eine (lineare) Liste oder ein Array von Zeigern auf Kindknoten. In [\[AuP\]](#), Kap. 10, wird skizziert, wie sich solche Bäume mit Hilfe von Binärbaumknoten darstellen lassen.

Für solche Mehrwegbäume sind Präorder- und Postorder-Durchlauf sowie Levelorder-Durchlauf sinnvoll (und benötigen lineare Zeit).

Uns werden solche Bäume später als **Mehrweg-Suchbäume** und im Kontext von **Graphdurchläufen** („Breitensuchbaum“, „Tiefensuchbaum“) begegnen.

ENDE

3. Kapitel