Algorithmen und Datenstrukturen 1

7. Vorlesung

Peter F. Stadler
Universität Leipzig
Institut für Informatik
studla@informatik.uni-leipzig.de

aufbauend auf den Kursen der letzten Jahre von
E. Rahm, G. Heyer, G. Brewka ,Uwe Quasthoff, Rald Der

Binärbäume

Def.: Ein Binärbaum ist eine endliche Menge von Elementen, die entweder leer ist oder ein ausgezeichnetes Element - die Wurzel des Baumes - besitzt und folgende Eigenschaften aufweist:

- Die verbleibenden Elemente sind in zwei disjunkte Untermengen zerlegt.
- Jede Untermenge ist selbst wieder ein Binärbaum und heißt linker bzw. rechter Unterbaum des ursprünglichen Baumes

Formale ADT-Spezifikation: BINTREE

Datentyp BINTREE, Basistyp ELEM

Operationen:

CREATE:		$\rightarrow$ BINTREE
EMPTY:	BINTREE	$\rightarrow$ {TRUE, FALSE}
BUILD:	BINTREE x ELEM x BINTREE	$\rightarrow$ BINTREE
LEFT:	BINTREE - $\{b_0\}$	$\rightarrow$ BINTREE
ROOT:	BINTREE - $\{b_0\}$	$\rightarrow$ ELEM
RIGHT:	BINTREE - $\{b_0\}$	$\rightarrow$ BINTREE

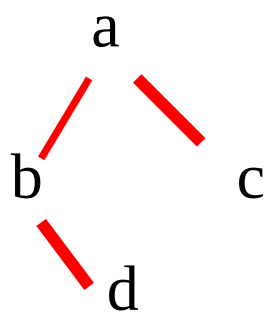
Axiome:

CREATE	$= b_0$;
EMPTY (CREATE)	= TRUE;
$\forall$ l, r $\in$ BINTREE, $\forall$ d $\in$ ELEM:	
EMPTY (BUILD (l, d, r))	= FALSE;
LEFT (BUILD (l, d, r))	= l;
ROOT (BUILD (l, d, r))	= d;
RIGHT (BUILD (l, d, r)	= r;

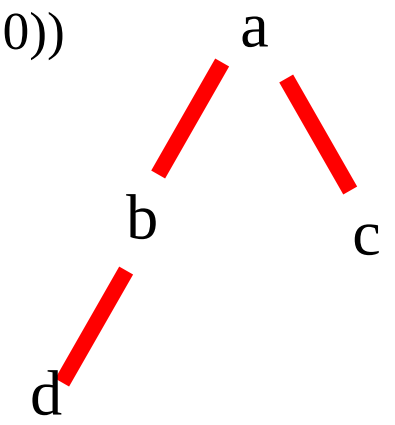
Testfrage

Welche Binärbäume (geordneten Bäume) entstehen durch

BUILD (BUILD (0, b, BUILD (0, d, 0)), a, BUILD (0, c, 0))



BUILD (BUILD (BUILD (0, d, 0), b, 0), a, BUILD (0, c, 0))



Eigenschaften von Binärbäumen I

Satz: Die maximale Anzahl von Knoten eines Binärbaumes

(1) auf Stufe i ist 2^i, $i \geq 0$

(2) der Höhe h ist 2^h-1, $h \geq 0$ (Der leere Baum hat Höhe 0)

Def.: Ein *vollständiger Binärbaum* der Stufe k hat folgende Eigenschaften:

- Jeder Knoten der Stufe k ist ein Blatt.
- Jeder Knoten auf einer Stufe < k hat nicht-leere linke und rechte Unterbäume.

Def.: In einem *strikten Binärbaum* besitzt jeder innere Knoten nicht-leere linke und rechte Unterbäume

Eigenschaften von Binärbäumen II

Def.: Ein *fast vollständiger Binärbaum* ist ein Binärbaum, so daß gilt:

(1) Jedes Blatt im Baum ist auf Stufe k oder k+1 ($k \geq 0$)

(2) Jeder Knoten auf Stufe < k hat nicht-leere linke und rechte Teilbäume

(3) Falls ein innerer Knoten einen rechten Nachfolger auf Stufe k+1 besitzt, dann ist sein linker Teilbaum vollständig mit Blättern auf Stufe k+1

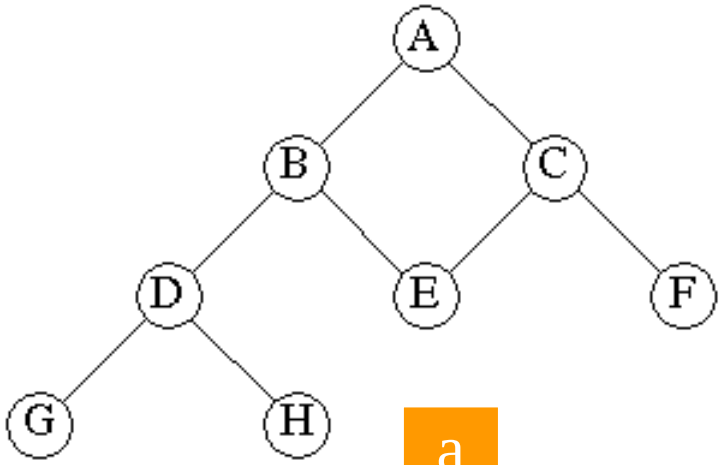
Def.: Ein *ausgeglichener Binärbaum* ist ein Binärbaum, so daß gilt:

(1) Jedes Blatt im Baum ist auf Stufe k oder k+1 ($k \geq 0$)

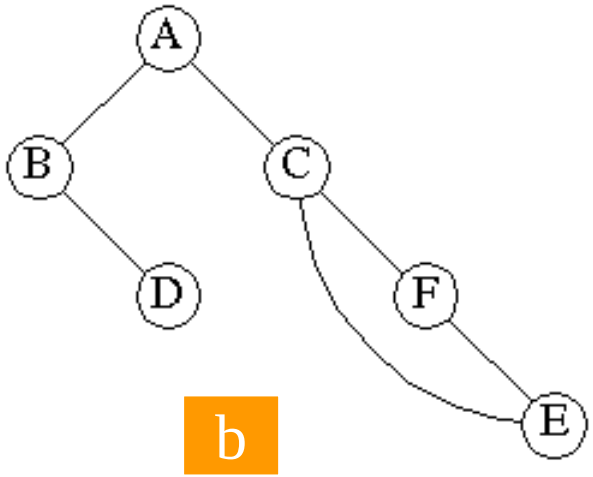
(2) Jeder Knoten auf Stufe < k hat nicht-leere linke und rechte Teilbäume

Zwei Binärbäume werden als *ähnlich* bezeichnet, wenn sie dieselbe Struktur besitzen. Sie heißen *äquivalent*, wenn sie ähnlich sind und dieselbe Information enthalten.

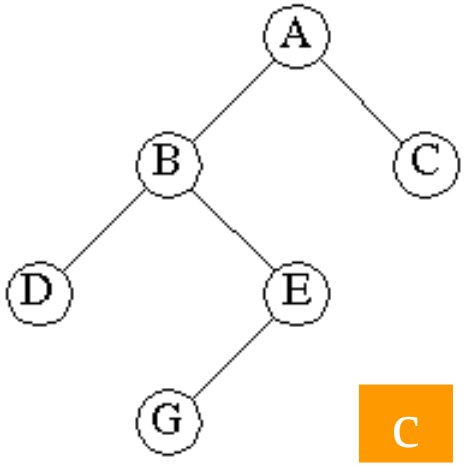
Veranschaulichung der Definitionen



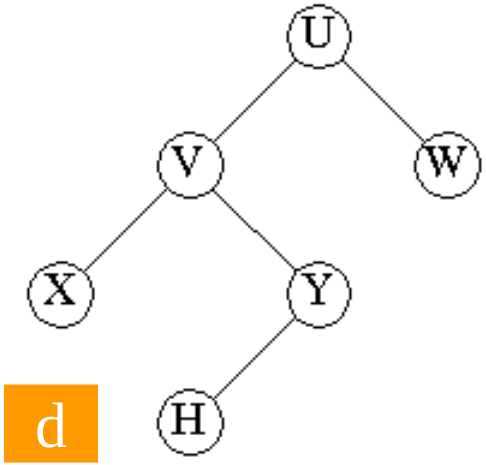
a



b

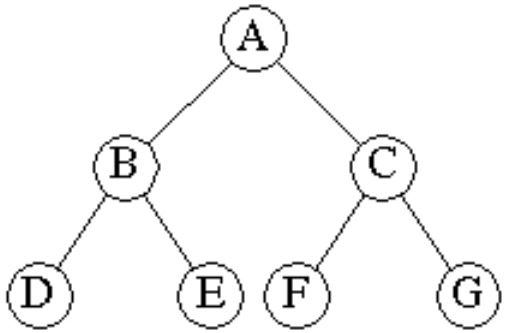


c

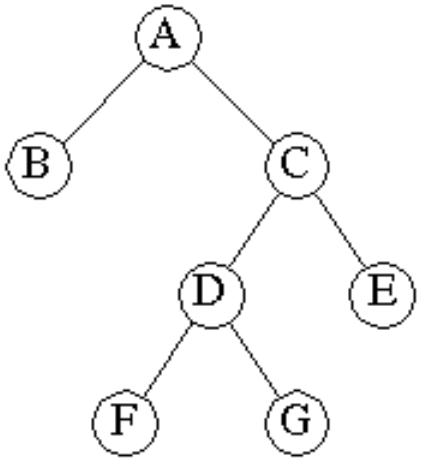


d

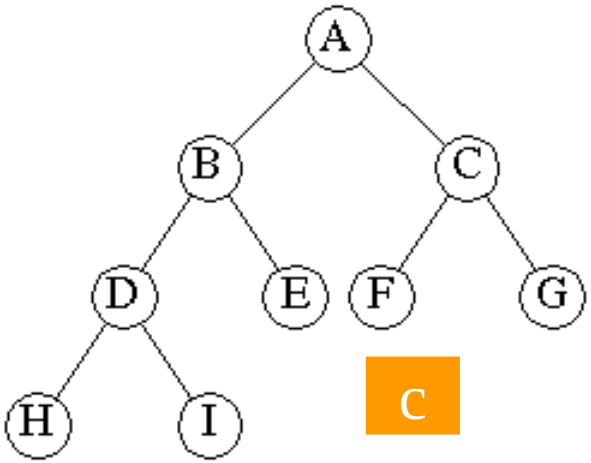
Veranschaulichung der Definitionen



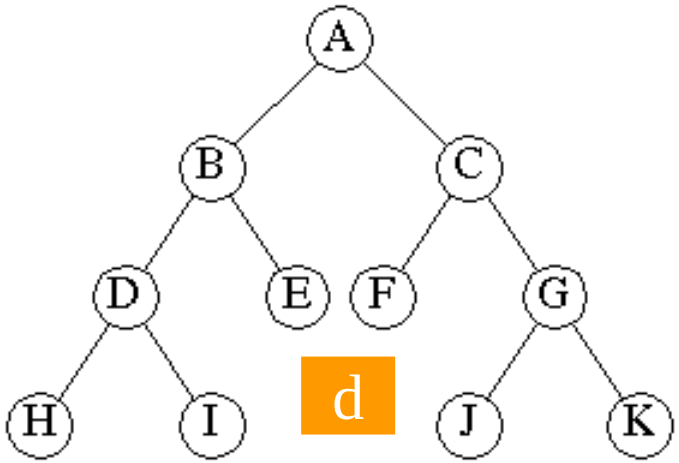
a



b



c



d

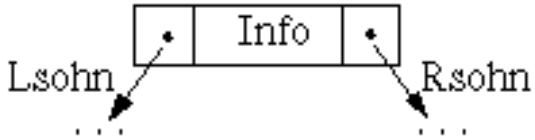
Speicherung in Binärbäumen

Vereinbarung:

Schlüssel in den Knoten des Baumes gespeichert. Dieser hat also so viele Knoten wie Schlüssel.

Für jeden Knoten gilt: Die Schlüssel im li. Teilbaum sind sämtlich kleiner als der in der Wurzel und dieser ist wiederum kleiner als die Schlüssel im re. Teilbaum.

Speicherung von Binärbäumen I

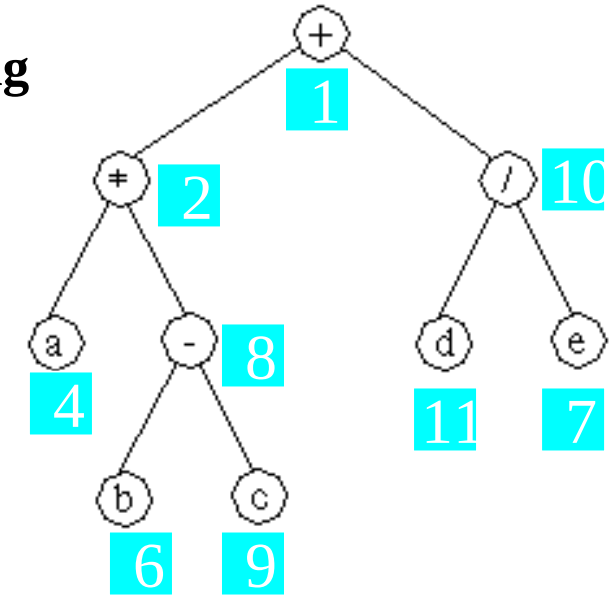


1. Verkettete Speicherung

- Freispeicherverwaltung der Struktur wird von der Speicherverwaltung des Programmiersystems übernommen

2. Feldbaum-Realisierung

- Simulation einer dynamischen Struktur in einem statischen Feld



	Info	Lsohn	Rsohn
1	+	2	10
2	!=	4	8
3	?	5	-1
4	a	-1	-1
5	?	12	-1
6			
7			
8			
9			
10			
11			
12	?	-1	-1

Eigenschaften:

- statische Speicherplatzzuordnung
- explizite Freispeicherverwaltung

Speicherung von Binärbäumen II

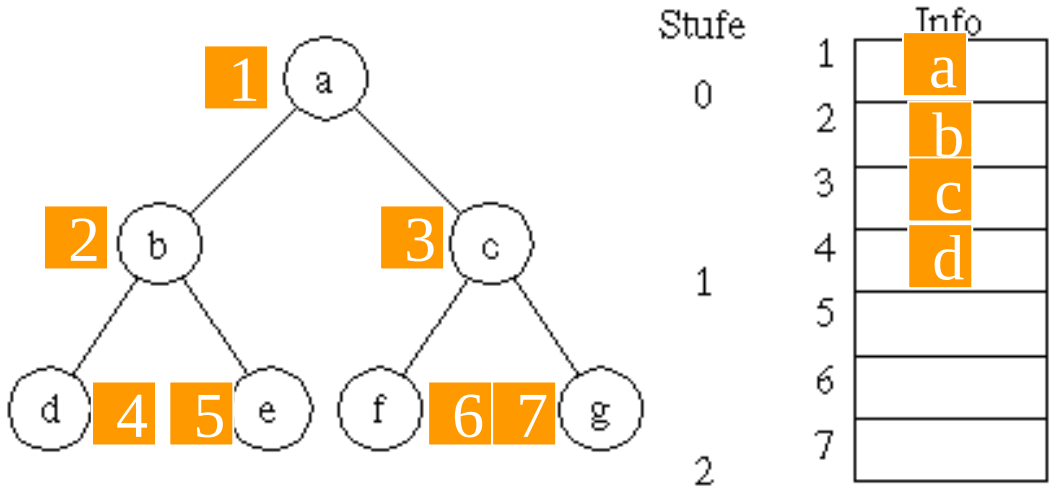
3. Sequentielle Speicherung

Methode kommt ohne explizite Verweise aus. Für fast vollständige oder zumindest ausgeglichene Binärbäume bietet sie eine sehr elegante und effiziente Darstellungsform an.

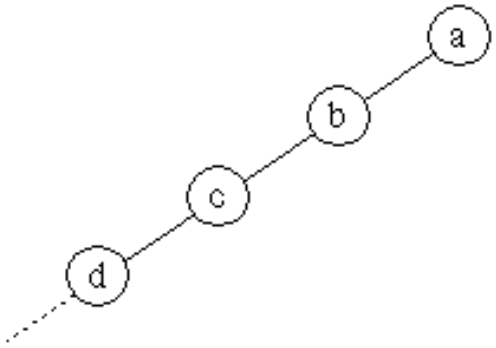
Satz: Ein fast vollständiger Baum mit n Knoten sei sequentiell nach folgendeem Nummerierungsschema gespeichert.

Für jeden Knoten mit Index i, $1 \leq i \leq n$, gilt:

- Vater(i) hat Nummer [i/2] für $i > 1$
- Lsohn(i) hat Nummer 2i für $2i \leq n$
- Rsohn(i) hat Nummer 2i+1 für $2i+1 \leq n$



Wie sieht der entartete Fall aus?



Suchen in Binärbäumen

- Die Suche nach einem Schlüssel x in einem Baum (Teilbaum) läuft nach folgendem rekursiven Schema ab:
- Man inspiziere den Wurzelknoten des Baumes.
- Falls x = Schlüssel des inspizierten Knotens: Suche beendet. Sonst:
- Falls x < Schlüssel des inspizierten Knotens: Setze Suche im linken Teilbaum fort.
- Falls x > Schlüssel des inspizierten Knotens: Setze Suche im rechten Teilbaum fort.

- Maximale Anzahl inspizierter Knoten: Tiefe des Baumes.

- Suche innerhalb der Knoten etwa durch lineares oder binäres Suchen zu realisieren. Da $l \leq m$, ist Aufwand dafür konstant.

Aufbau von Binärbäumen

Operationen zum Aufbau eines Binärbaums (Einfügen von Knoten) sowie dem Entfernen von Knoten sind relativ einfach:

Einfügen: Eingefügt wird immer bei Blättern. Zunächst wird das einzufügende Element gesucht (und nicht gefunden). An der Stelle, wo die Suche endet, wird das Element angefügt.

Löschen (simple Lösung): Das zu löschende Element wird gesucht. Es wird eine Löschmarkierung angebracht, die aussagt, dass das Element gelöscht ist. Trotzdem wird der Knoten wie bisher zur Navigation im Baum genutzt.
Nachteil: Beim Löschen wird kein Speicher frei. Deshalb gelegentlich den ganzen Baum reorganisieren, d.h. ohne gelöschte Elemente neu aufbauen.

Durchlaufen eines Binärbaums

Baumdurchlauf (Traversierung): Verarbeitung aller Baumknoten gemäß vorgegebener Strukturierung

Rekursiv anzuwendende Schritte

1. Verarbeite Wurzel: W
2. Durchlaufe linken UB: L
3. Durchlaufe rechten UB: R

Durchlaufprinzip impliziert sequentielle, lineare Ordnung auf der Menge der Knoten

6 Möglichkeiten:

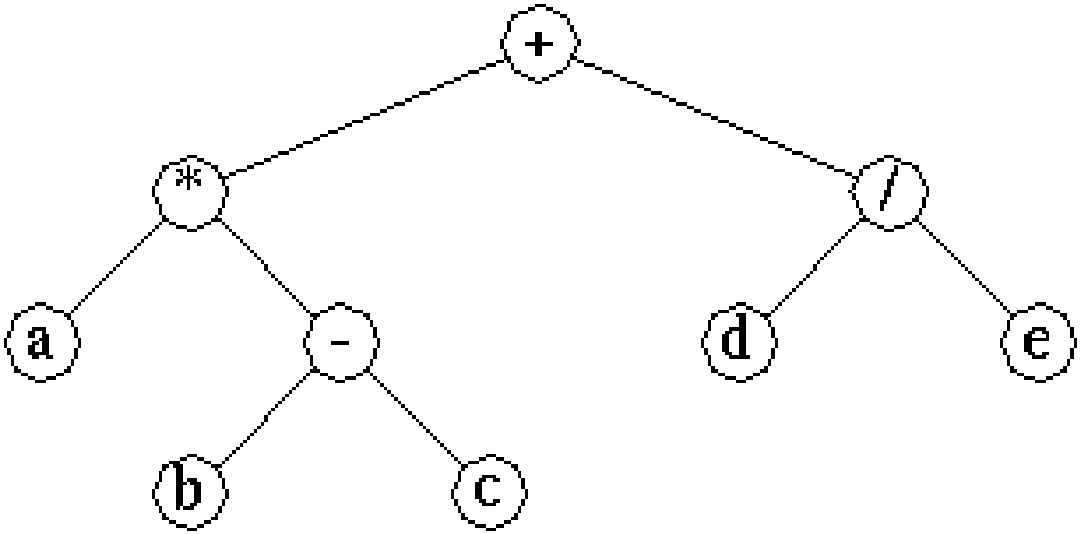
1	2	3	4	5	6
W	L	L	W	R	R
L	W	R	R	W	L
R	R	W	L	L	W

3 Strategien verbleiben aufgrund Konvention:
linker UB vor rechtem UB

1. Vorordnung (preorder): WLR
2. Zwischenordnung (inorder): LWR
3. Nachordnung (postorder): LRW

Durchlaufmöglichkeiten I

Referenzbeispiel



WLR: + * a - b c / d e

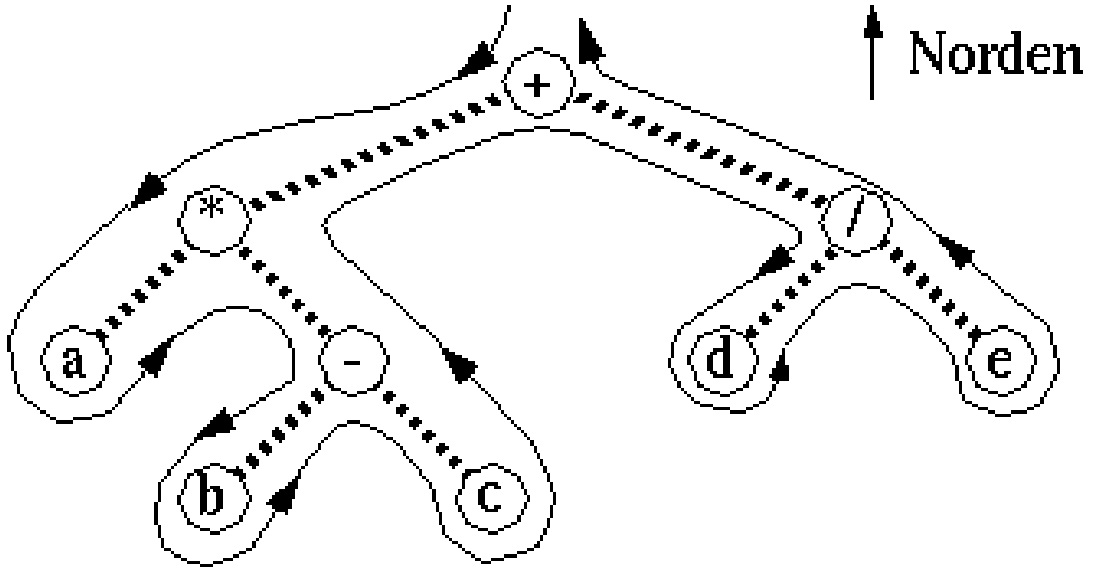
LWR: a * b - c + d / e

LRW:

Durchlaufmöglichkeiten II

Anschauliche Darstellung

Ausgabe bei Passage des Turms (Knoten) im Fall



WLR: in Richtung Süden + * a - b c / d e

LWR: an seiner Südseite a * b - c + d / e

LRW: in Richtung Norden a b c - * d e / +

Rekursive Realisierung

Rekursive Version für Inorder-Traversierung (LWR)

```
DurchlaufInOrder(Baum)
   Fall Baum leer, dann fertig
   sonst
       DurchlaufInOrder(LinkeTochter(Baum))
       Drucke Inhalt des aktuellen Knotens
       DurchlaufInOrder(RechteTochter(Baum))
```

Iterative Realisierung

Iterative Version: LWR

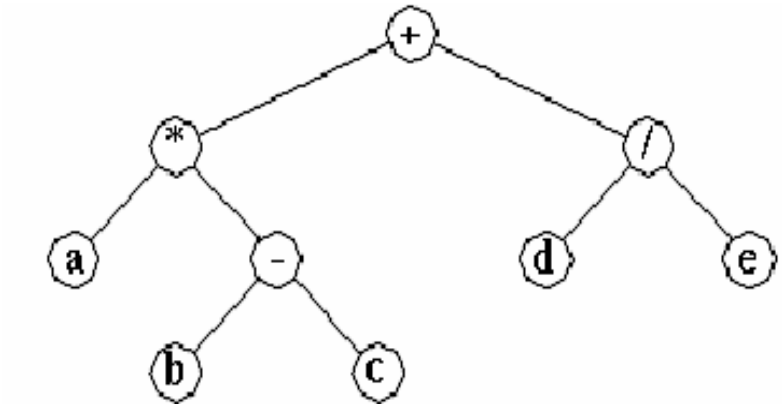
Ziel: effizientere Ausführung durch eigene Stapelverwaltung

Vorgehensweise:

- Nimm, solange wie möglich, linke Abzweigung und speichere die Knoteninhalte des zurückgelegten Weges auf einem Stapel (Aktion 1).

- Wenn es links nicht mehr weitergeht, wird der oberste Knoten des Stapels ausgegeben und vom Stapel entfernt. Der Durchlauf wird mit dem rechten Unterbaum des entfernten Knotens fortgesetzt (Aktion 2).

Iterative Version: Durchlaufbeispiel



Stapel	cuurent	Aktion	Ausgabe
+	+	1	-
+ *	*	1	-
+ * a	a	1	-
+ *	O	2	a
+	O	2	*
+ -	-	1	-
+ - b	b	1	-
+ -	O	2	b