5. Allgemeine Bäume und Binärbäume

Bäume - Überblick

- Orientierte Bäume
- Darstellungsarten
- Geordnete Bäume

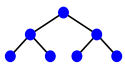
Binäre Bäume: Begriffe und Definitionen

Speicherung von binären Bäumen

- Verkettete Speicherung
- Feldbaum-Realisierung
- Sequentielle Speicherung
- Aufbau von Binärbäumen (Einfügen von Knoten)

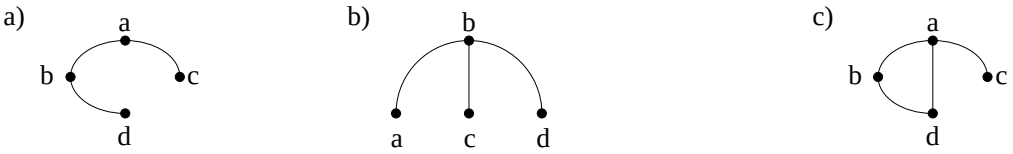
Durchlaufen von binären Bäumen

- Preorder-, Inorder-, Postorder-Traversierung
- Rekursive und iterative Version
- Fädelung



Bäume

Bäume lassen sich als sehr wichtige Klasse von Graphen auffassen

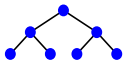


Danach ist ein Baum

- ein azyklischer einfacher, zusammenhängender Graph
- d.h., er enthält keine Schleifen und Zyklen; zwischen jedem Paar von Knoten besteht höchstens eine Kante

Def.: Orientierte Bäume: Sei X eine Basis-Datenstruktur. Eine Menge B von Objekten aus X ist ein *orientierter (Wurzel-) Baum*, falls

1. in B ein ausgezeichnetes Element w - Wurzel von B - existiert
2. die Elemente in B - {w} disjunkt zerlegt werden können in B_1, B_2,..., B_m, wobei jedes B_i ebenfalls ein Baum ist.

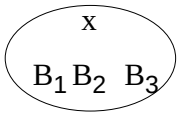


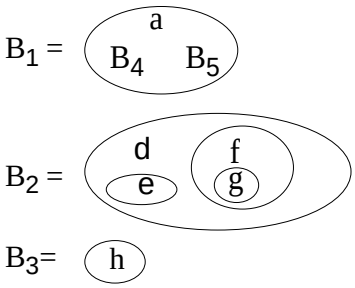
Darstellungsarten für orientierte Bäume

1. Mengendarstellung

Objekte: a, b, c, ...

Bäume: $B_1, B_2, B_3, ...$





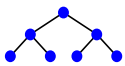
2. Klammerdarstellung

Wurzel = erstes Element innerhalb eines Klammerpaares

$\{x, B_1, B_2, B_3\}$

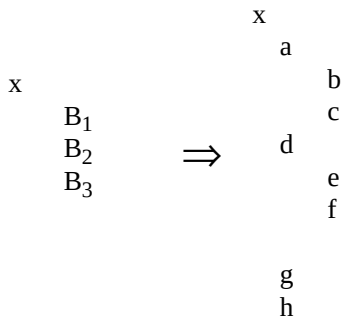
$\{x, \underbrace{\{a, \{b\}, \{c\}\}}_{B1}, \underbrace{\{d, \{e\}, \{f, \{g\}\}\}}_{B2}, \underbrace{\{h\}}_{B3}\}$

$\{a, \{b\}, \{c\}\} \equiv \{a, \{c\}, \{b\}\}$

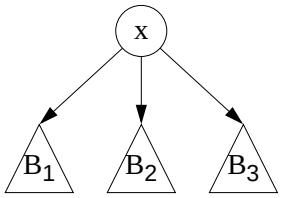


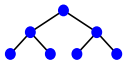
Darstellungsarten für orientierte Bäume (2)

3. Rekursives Einrücken

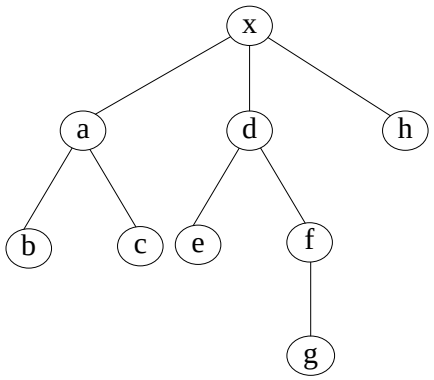


4. Graphendarstellung





Graphendarstellung orientierter Bäume



Stufe (Ebene)

0

1

2

3

Bezeichnungen

Wurzel:

Blätter:

innere Knoten:

Grad K = # Nachfolger von K

Grad (x) =

Grad (g) =

Grad (Baum) = Max (Grad(K_i)) =

Stufe (K_i) = Pfadlänge l von Wurzel nach K_i

Stufe 0:

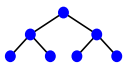
Stufe 1:

Stufe 2:

Stufe 3:

Höhe h =

Gewicht w = # der Blätter =



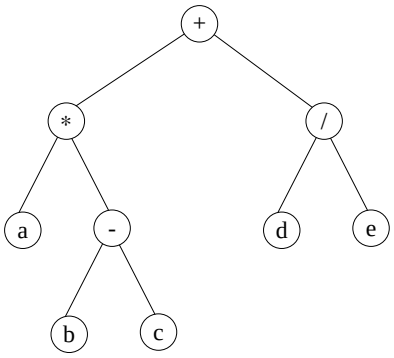
Geordnete Bäume

Def.: Bei einem geordneten Baum bilden die Unterbäume B_i jedes Knotens eine geordnete Menge

Def.: Eine geordnete Menge von geordneten Bäumen heißt Wald

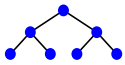
Beispiel: Arithmetischer Ausdruck a * (b - c) + d/e

Graphendarstellung



Klammerdarstellung

{+, { *, { a }, {-, { b }, { c } } }, { /, { d }, { e } } }



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:

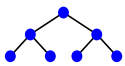
- Die verbleibenden Elemente sind in zwei disjunkte Untermengen zerlegt.
- Jede Untermenge ist selbst wieder ein Binärbaum und heißt linker bzw. rechter Unterbaum des ursprünglichen Baumes

Formale ADT-Spezifikation

Datentyp BINTREE
Basistyp ELEM

Operationen:

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



Axiome:

CREATE	=	$b_{0;}$
EMPTY (CREATE)	=	TRUE;

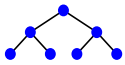
∀ l, r, ∈ BINTREE, ∀ d ∈ ELEM:

EMPTY (BUILD (l, d, r))	=	FALSE;
LEFT (BUILD (l, d, r))	=	l;
ROOT (BUILD (l, d, r))	=	d;
RIGHT (BUILD (l, d, r)	=	r;

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))



Binärbäume (2)

Satz: Die maximale Anzahl von Knoten eines Binärbaumes

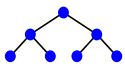
(1) auf Stufe i ist 2^i, $i \geq 0$

(2) der Höhe h ist 2^h-1, $h \geq 1$

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



Binärbäume (3)

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

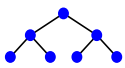
(3) Falls ein innerer Knoten einen rechten Nachfolger auf Stufe k+1 besitzt, dann ist sein linker Teilbaum vollständig mit Blättern auf Stufe k+1

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

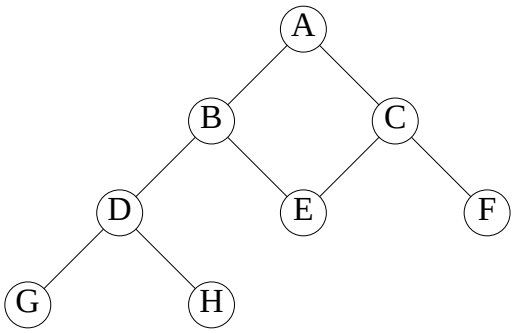
(1) Jedes Blatt im Baum ist auf Stufe k oder k+1 ($k \geq 0$)

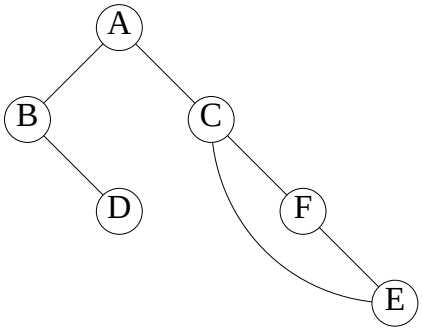
(2) Jeder Knoten auf Stufe < k hat nicht-leere linke und rechte Teilbäume

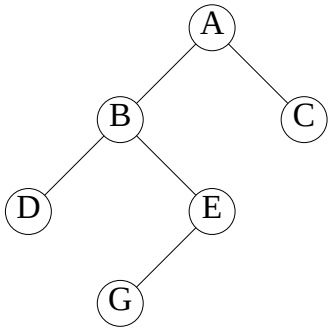
Zwei Binärbäume werden als *ähnlich* bezeichnet, wenn sie dieselbe Struktur besitzen. Sie heißen *äquivalent*, wenn sie ähnlich sind und dieselbe Information enthalten.

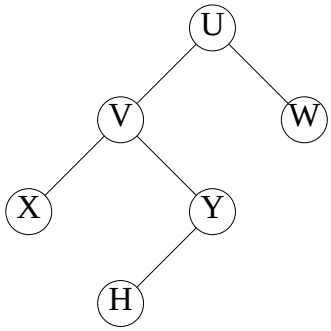


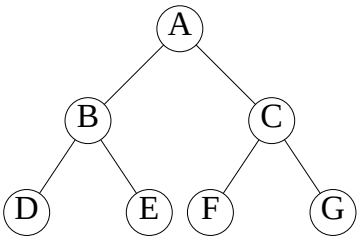
Veranschaulichung der Definitionen

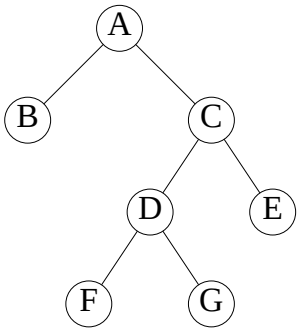


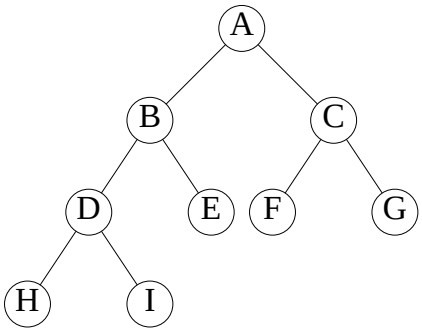


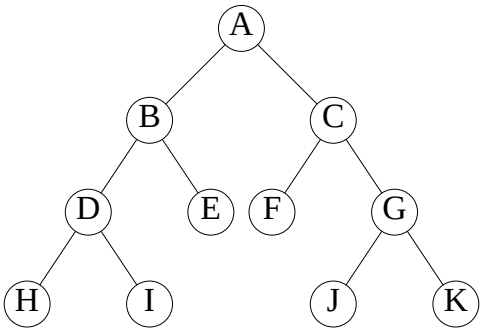






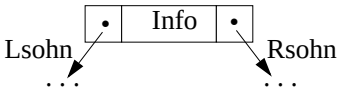






Speicherung von Binärbäumen

1. Verkettete Speicherung



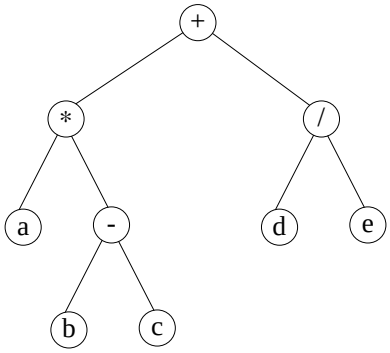
- Freispeicherverwaltung der Struktur wird von der Speicherverwaltung des Programmiersystems übernommen

2. Feldbaum-Realisierung

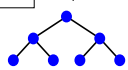
- Simulation einer dynamischen Struktur in einem statischen Feld

Eigenschaften:

- statische Speicherplatzzuordnung
- explizite Freispeicherverwaltung



	Info	Lsohn	Rsohn
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

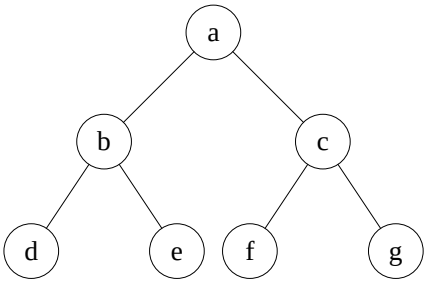


3. Sequentielle Speicherung

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

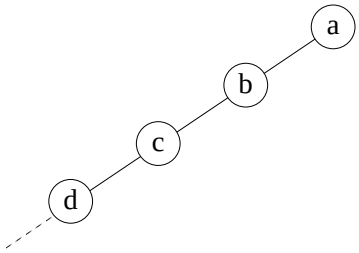
Satz: Ein fast vollständiger Baum mit n Knoten sei sequentiell nach obigem Numerierungsschema gespeichert.
Für jeden Knoten mit Index i, $1 \le i \le n$, gilt:

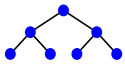
- Vater(i) hat Nummer $\lfloor i/2 \rfloor$ für $i > 1$
- Lsohn(i) hat Nummer 2i für $2i \le n$
- Rsohn(i) hat Nummer 2i+1 für $2i+1 \le n$.



Stufe		Info
0	1	
	2	
	3	
1	4	
	5	
	6	
2	7	

Wie sieht der entartete Fall aus?



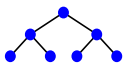


Aufbau von Binärbäumen

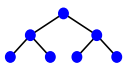
Operationen zum Aufbau eines Binärbaums (Einfügen von Knoten) sowie dem Entfernen von Knoten sind relativ einfach

Java-Realisierung für gekettete Repräsentation

```
class BinaryNode {
   BinaryNode lChild = null;
   BinaryNode rChild = null;
   Object info = null;

   /** Konstruktor */
   BinaryNode(Object info) { this.info = info; }
 }
```

```
public class BinaryTree {
  private BinaryNode root = null;

  public BinaryTree() { root = null; }

  public BinaryTree(BinaryNode root) { this.root = root; }

  public boolean empty() { return root == null; }

  public void build(BinaryTree lTree, BinaryNode elem, BinaryTree rTree)
  throws TreeException {
    if (elem == null) throw new TreeException("Wurzel ist null!");
    BinaryNode tmpRoot = elem;
    if (lTree == null) tmpRoot.lChild = null;
    else tmpRoot.lChild = lTree.getRoot();
    if (rTree == null) tmpRoot.rChild = null;
    else tmpRoot.rChild = rTree.getRoot();
    root = tmpRoot; }

  public BinaryNode getRoot() { return root; }

  public BinaryTree left() throws TreeException {
    if (empty()) throw new TreeException("Wurzel ist null!");
    return new BinaryTree(root.lChild);
  }

  public BinaryTree right() throws TreeException {
    if (empty()) throw new TreeException("Wurzel ist null!");
    return new BinaryTree(root.rChild);
  }
}
```

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

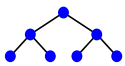
Durchlaufprinzip impliziert sequentielle, lineare Ordnung auf der Menge der Knoten

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

6 Möglichkeiten:

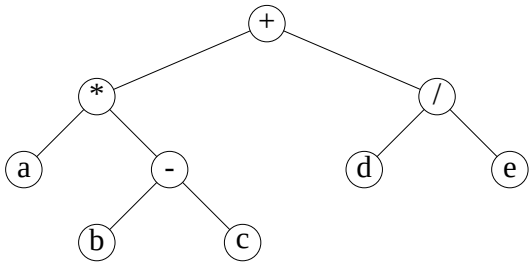
3 Strategien verbleiben aufgrund Konvention: linker UB vor rechtem UB

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



Durchlaufmöglichkeiten

Referenzbeispiel

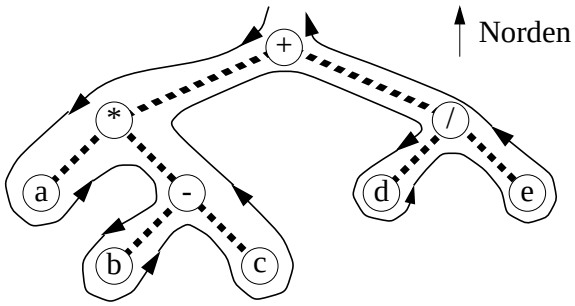


WLR:

LWR:

LRW:

Anschauliche Darstellung

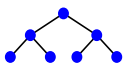


Ausgabe bei Passage des Turms (Knoten)

WLR: in Richtung Süden

LWR: an seiner Südseite

LRW: in Richtung Norden



Rekursive und iterative Realisierung

Rekursive Version für Inorder-Traversierung (LWR)

```
public void printInOrder (BinaryNode current) {
  if (current != null) {
    printInOrder(current.lChild);
    System.out.println(current.info);
    printInOrder(current.rChild);
  }
}
```

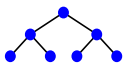
Iterative Version: LWR

- Ziel: effizientere Ausführung durch eigene Stapelverwaltung
- Vorgehensweise: Nimm, solange wie möglich, linke Abzweigung und speichere den zurückgelegten Weg auf einem Stapel.

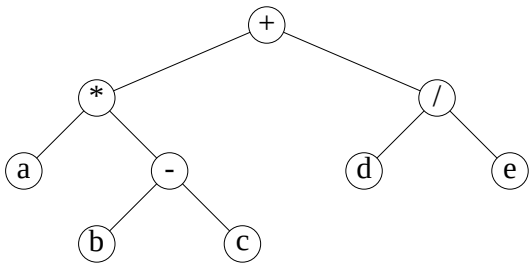
```
Aktion 1:   stack.push(current);
            current = current.lChild;
```

- 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.

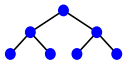
```
Aktion 2:   System.out.println(((BinaryNode) stack.top()).info);
            current = ((BinaryNode) stack.top()).rChild;
            stack.pop();
```



Iterative Version: Durchlaufbeispiel



Stapel S	current	Aktion	Ausgabe



Gefädelte Binärbäume

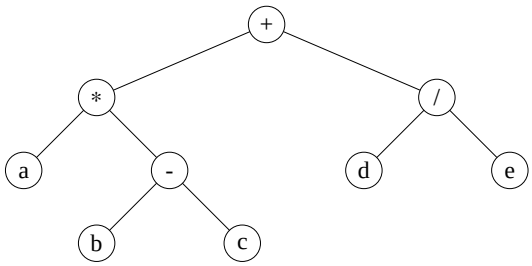
Weitere Verbesserung von iterativen Durchlaufalgorithmen

Methode benutzt einen “Faden”, der die Baumknoten in der Folge der Durchlaufordnung verknüpft.

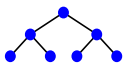
- Zwei Typen von Fäden
- *Rechtsfaden* verbindet jeden Knoten mit seinem Nachfolgerknoten in Durchlaufordnung
- *Linksfaden* stellt Verbindung zum Vorgängerknoten in Durchlaufordnung her

Lösung 1. Explizite Speicherung von 2 Fäden

```
class ThreadedBinNode {
  ThreadedBinNode lChild = null;
  ThreadedBinNode rChild = null;
  Object info = null;
  ThreadedBinNode lThread = null;
  ThreadedBinNode rThread = null;
  /** Konstruktor */
  ThreadedBinNode(Object info) {
    this.info = info; } }
```



Beispiel: Zwischenordnung



Gefädelte Binärbäume (2)

Lösung 2. Vermeidung von Redundanz

Eine zweite Art der Fädelung kommt ohne zusätzliche Zeiger aus und erfordert daher geringeren Speicherplatzaufwand. Die Wartungs- und Durchlauf-Algorithmen werden lediglich geringfügig komplexer.

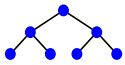
Beobachtung 1: Binärbaum mit n Knoten hat n+1 freie Zeiger (null)

Beobachtung 2: für die Zwischenordnung können Fadenzeiger in inneren Knoten durch Folgen von Baumzeigern ersetzt werden

Idee: Benutze freie Zeiger und Baumzeiger für Fädelung

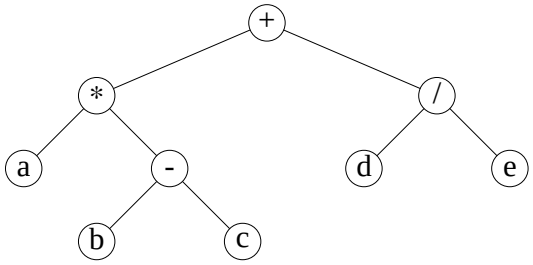
- pro Knoten zusätzliche Boolesche Variablen Lfaden, Rfaden:

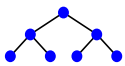
```
class OptThreadedBinNode {
    OptThreadedBinNode lChild = null;
    OptThreadedBinNode rChild = null;
    Object info = null;
    boolean lThread = null;  // true, wenn lChild Fadenzeiger
    boolean rThread = null;  // true, wenn rChild Fadenzeiger
    /** Konstruktor */
    OptThreadedBinNode(Object info) { this.info = info; } }
```



Prozedur für Traversierung in Zwischenordnung mit optimierter Fädelung

```
public class ThreadedBinTree {
   private OptThreadedBinNode root = null;
   ...
// Baumdurchlauf iterativ über Fädelung
   public void printInOrder () {
     OptThreadedBinNode current = null;
     OptThreadedBinNode previous = null;
    if (root == null) return;  // Baum leer
     current = root;
   do {
       while ((! current.lThread) && (current.lChild != null))
        current = current.lChild;  // verzweige nach links so lange wie mgl.
       System.out.println(current.info);
       previous = current;
       current = current.rChild;
       while ((previous.rThread) && (current != null)) {
         System.out.println(current.info);
         previous = current;
         current = current.rChild;
       }
     } while (current != null);
   }
 }
```





Zusammenfassung

Definitionen

- Baum, orientierter Baum (Wurzel-Baum), geordneter Baum, Binärbaum
- vollständiger, fast vollständiger, strikter, ausgeglichener, ähnlicher, äquivalenter Binärbaum
- Höhe, Grad, Stufe / Pfadlänge, Gewicht

Speichrung von Binärbäumen

- verkettete Speicherung
- Feldbaum-Realisierung
- sequentielle Speicherung

Baum-Traversierung

- Preorder (WLR): Vorordnung
- Inorder (LWR): Zwischenordnung
- Postorder (LRW): Nachordnung

Gefädelte Binärbäume: Unterstützung der (iterativen) Baum-Traversierung durch Links/Rechts-Zeiger auf Vorgänger/Nachfolger in Traversierungsreihenfolge