Binärbäume

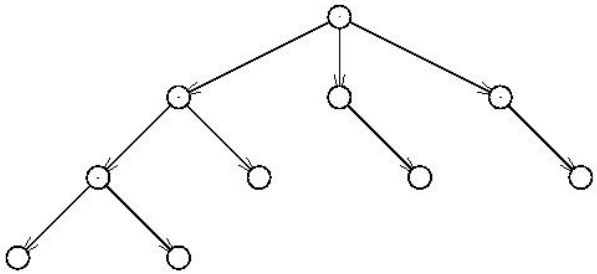
- Grundbegriffe der **Graphentheorie**
- Bäume und Ihre **Anwendungen**
- Unterschiedliche **Darstellungen** von Bäumen und Binärbäumen
- Binärbäume in **Java**
- **Rekursive Traversierung** von Binärbäumen
- **Ebenenweise Traversierung** von Bäumen
- **Anwendungsbeispiel**: Abstrakter Syntax-Baum

Ein Graph besteht aus Knoten und Kanten:

Kopf Kante Ende

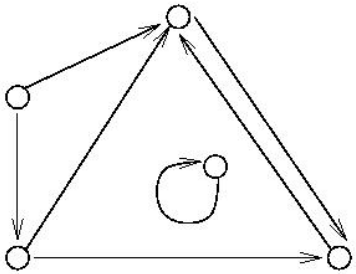
Lineare Liste

innerer Knoten



Baum

Blatt



allgemeiner Graph

Grundbegriffe der Graphentheorie

Ein **Graph** besteht aus einer Menge von **Knoten** und einer Menge von **Kanten**. Eine Kante verbindet zwei Knoten. Man unterscheidet **gerichtete** und **ungerichtete** Graphen.

Ein **Weg** ist eine Folge von aufeinanderfolgenden Kanten (und Knoten).

Ein **Kreis** ist ein geschlossener Weg.

Eine **Liste** ist ein Graph, in dem alle Knoten an einem einzigen Weg, der an dem Kopfknoten beginnt, liegen.

Ein **Baum** ist ein gerichteter Graph, in dem alle Knoten über einen genau einen Weg, der an dem Wurzelknoten beginnt, erreicht werden können.

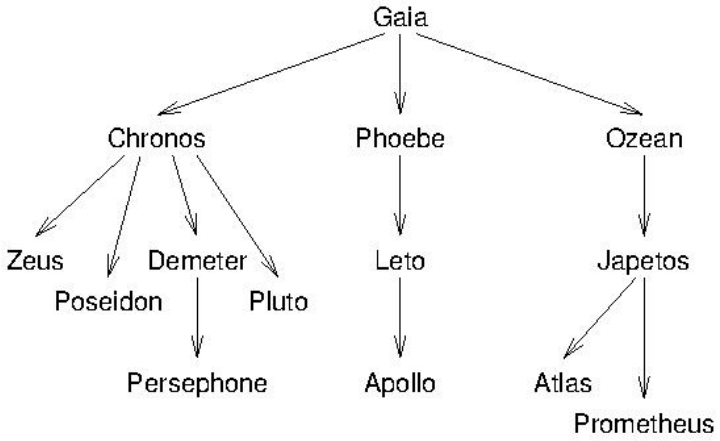
Die in einen Baum unmittelbar nach einem Knoten kommenden Knoten heißen **Kindknoten**, der Vorgängerknoten heißt **Elternknoten**. Knoten ohne Nachfolger heißen auch **Blätter**.

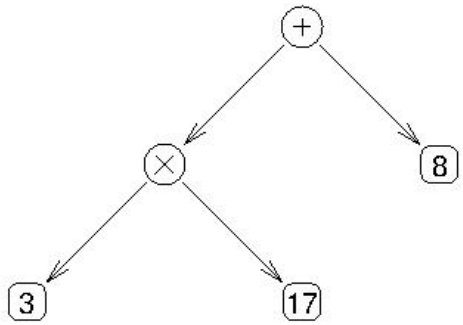
Ein **Binärbaum** ist ein Baum, in dem jeder Knoten maximal zwei Nachfolger hat.

Bäume und ihre Anwendungen

1. Bäume bilden hierarchische Beziehungen ab.

2. Bäume erlauben effizientes Suchen.





Früher musste man noch die Götter kennen

3 * 17 + 8

Unterschiedliche Darstellungen für Bäume

1. Sequentiell (Syntax): 2 + 3 * 4 oder 2 3 + 4 *
2. Sequentiell 1, // 2.1, 2.2,// 3.1, 3.2, 3.3 (vollständiger Baum, z.B. Heapsort)
3. Durch geschickte Array-Struktur (mit Indizes "verkettet")
4. Durch Adjazenz-/Inzidenzmatrix

...

n. Durch verkettete Knotenobjekte

Binärbäume in Java

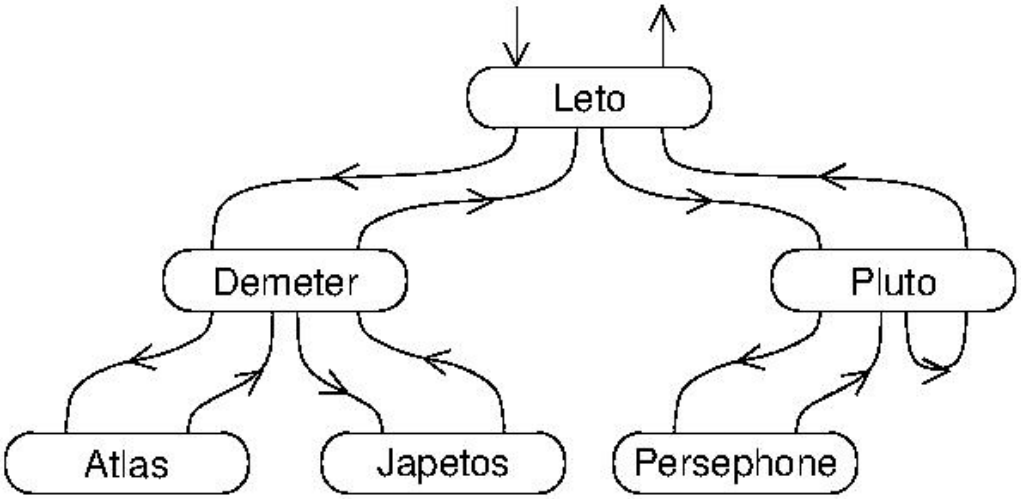
```
/**
 * TreeNode - Allgemeiner Knoten fuer Binaerbaeume
 *
 * WARNUNG: Diese Klasse zeigt unterschiedliche Moeglichkeiten im
 *   Umgang mit Binaerbauemen.
 *   Ein solcher Mischmasch ist in der Praxis SCHLECHTER PROGRAMMIERSTIL!
 */
public abstract class TreeNode {
    protected TreeNode left;
    protected TreeNode right;

    public TreeNode() {
        this(null, null);
    }

    public TreeNode(TreeNode left, TreeNode right) {
        this.left = left;
        this.right = right;
    }
```

Ein Baum ist eine rekursive Datenstruktur.

Baumalgorithmen sind oft rekursiv.

Wie besucht man alle Knoten in einem Baum ?

1. Rekursiv den Baum durchlaufen (rekursive Datenstruktur)
2. Jeder (innere) Knoten wird 3x besucht (nur 1 Aktion!).



Rekursive Traversierung

```
public static void preOrder(TreeNode n) {
    if (n == null) return;
    System.out.println(n);
    preOrder(n.left);
    preOrder(n.right);
}

public static void inOrder(TreeNode n) {
    if (n == null) return;
    inOrder(n.left);
    System.out.println(n);
    inOrder(n.right);
}

public static void postOrder(TreeNode n) {
    if (n == null) return;
    postOrder(n.left);
    postOrder(n.right);
    System.out.println(n);
}
```

Rekursive Traversierung statisch oder objektorientiert

Zwei Auffassungen: Baumknoten sind Daten -> Klassenfunktion
Baumknoten sind Objekte -> Methode

Zwei Lösungen: In Klassenfunktion, wenn Parameter == null
In Methode kein Aufruf für null-Objekt

Die objektorientierte Lösung erlaubt für verschiedenen Unterklassen verschiedene Lösungen!

```
public static void postOrder(TreeNode n) { // gehoert in Klasse Tree
    if (n == null) return;
    postOrder(n.left);
    postOrder(n.right);
    System.out.println(n);
}
```

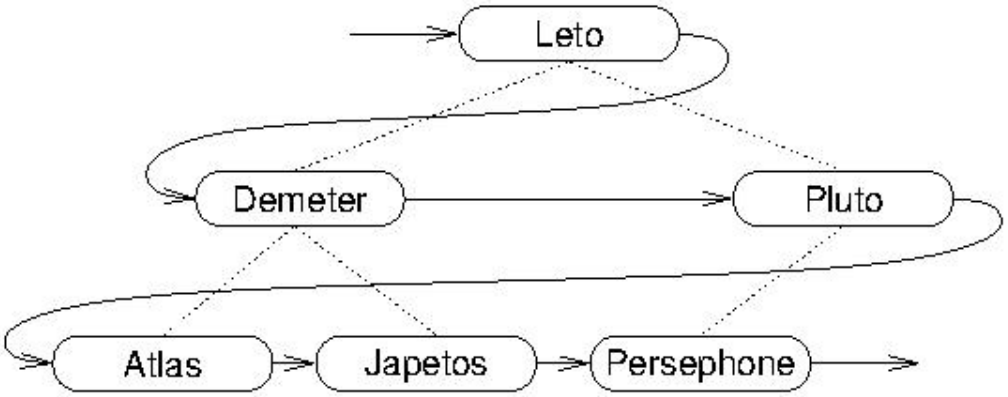
```
public void postOrder() {                    //gehoert in Klasse TreeNode
    if (left != null) left.postOrder() ;
    if (right != null)  right.postOrder();
    System.out.println(this);
}
```

Einfache rekursive Algorithmen

```
/**
 * Gibt die Anzahl aller Knoten ab root zurück
 */
public static int anzahl(TreeNode root) {
   if (root == null)
       return 0;
    else
        return 1 + anzahl(root.left) + anzahl(root.right);
}

/**
 * Gibt die Hoehe des Baums ab diesem Knoten zurück.
 * Hoehe = Laenge des laengsten Weges.
 */
public int hoehe() {
    int l = 0, r = 0;
    if (left != null) l = left.hoehe() + 1;
    if (right != null) r = right.hoehe() + 1;
    return Math.max(l, r);
}
```

Machmal ist die Reihenfolge egal.

Ebenenweise Traversierung



Anwendung: besondere Gestaltung der Ausgabe, Suchalgorithmen

Algorithmus:
- rekursiv geht nicht, da der Weg quer zum Baum liegt !
- man braucht aber eine Datenstruktur zum "Vormerken": Queue !

Ebenenweise Traversierung mit Queue

```
public void levelOrder() {
    // Initialisierung der Warteschlange mit dem Wurzelknoten:
    Queue<Node> q = new ListQueue<Node>();
    q.put(this);
    // alle Teilbäume aus q bearbeiten:
    while (!q.isEmpty()) {
        // 1. nächsten Knoten holen:
        Node p = q.get();
        // 2. Knoten bearbeiten:
        System.out.println(p);
        // 3. linkes Kind vormerken:
        if (p.left != null) q.put(p.left);
        // 4. rechtes Kind vormerken:
        if (p.right != null) q.put(p.right);
    }
}
```

"Rekursive" Traversierung mit Stack

Der Unterschied von rekursiver Traversierung und ebenenweiser Traversierung ist der Unterschied von Stack ≅ rekursiv und Queue ≅ ebenenweise.

```
public void preOrder2() {
    // Initialisierung des Stapels mit dem Wurzelknoten:
    Stack<Node> s = new ListStack<Node>();
    s.push(this);
    while (!s.isEmpty()) {
        // 1. nächsten Knoten holen:
        Node p = s.pop();
        // 2. Knoten bearbeiten:
        System.out.println(p);
        // 3. rechtes Kind vormerken:
        if (p.right != null) s.push(p.right); // gemogelt ?!
        // 4. linkes Kind vormerken:
        if (p.left != null) s.push(p.left);

    }
}
```

Ein objektorientiertes Anwendungsbeispiel

Wir haben eine gemeinsame (abstrakte) Oberklasse `Node` und konkrete Klassen für die Rechenoperationen. Damit lassen sich Abstrakte Syntaxbäume rekursiv auswerten.

Der Aufbau des Baumes für 3 * 4 + 11 * 88 geschieht im Normalfall durch einen *Parser*. Als Beispiel aber auch durch direkte Konstruktoraufrufe:

```
public class TreeTest {
    public static void main (String[] args) {
    Node e = // Aufgabe eines Parsers (Compilerbau)
        new PlusNode(
        new MultNode( new NumNode(3), new NumNode(4)),
        new MultNode( new NumNode(17), new NumNode(88)));

    System.out.println("Postorder-Ausgabe: " + e.postOrder());
    System.out.println("Ergebnis: " + e.value());
}
```

Realisierung des Syntaxbaums (1)

```
public interface Node {
    public double value();
    public String postOrder();
}

public class AbstractBinOp implements Node {
    private Node left, right;

    public AbstractBinOp(Node l, Node r) {
        left = l; right = r;
    }

    protected abstract char operator();

    public String postOrder() {
        StringBuilder b = new StringBuilder();
        b.append(left.postOrder());
        b.append(' ');
        b.append(right.postOrder());
        b.append(' ');
        b.append(operator());
        return b.toString();
    }
}
```

Realisierung des Syntaxbaums (2)

```
public class PlusNode extends AbstractBinOp {
    public PlusNode(Node l, Node r) {
        super(l, r);
    }
    public double value() {
        return left.value() + right.value();
    }
    protected char operator() {
        return '+';
    }
}

public class NumNode implements Node {
    private double value;
    public NumNode(double value) {
        this.value = value;
    }
    public double value() {
        return value;
    }
    public String postOrder() {
        return String.valueOf(value);
    }
}
```

Verschiedene Formen der Traversierung von Ausdrucksbäumen:

Ausdruck: 3 * 4 + 17 * 88

1. **Preorder**: `(+(* 3 4)(* 17 88) )` **Prefix** (poln. Notation - keine Klammern)
   (Lisp/Scheme, Funktionsschreibweise)

2. **Inorder**: `((3 * 4) + (17 * 18))` **Infix** (algebr. Notation - Klammern)
   (algebr. Ausdrücke)

3. **Postorder**: `((3 4 *)(17 18 *)+)` **Postfix** (UPN - keine Klammern)
   (Forth, UPN, virtuelle Stackmaschine)

4. **Levelorder** + * * 3 4 17 88 macht hier keinen Sinn.

bei dem Sortierverfahren Heapsort sind die Baumknoten in Levelorder in einem Array gespeichert !

Bäume werden in Compilern / Interpretern / Datenbanken genutzt, um einfach mit der Struktur der Daten umgehen zu können (vgl. auch Maple).