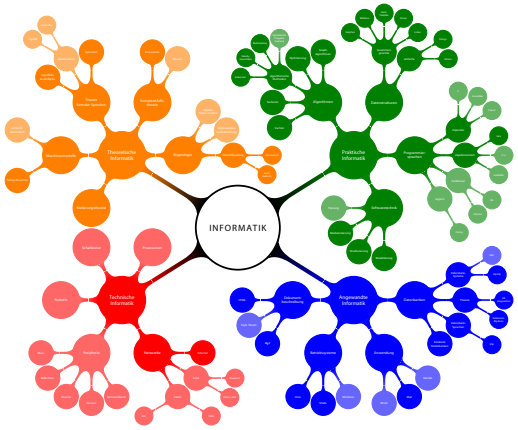
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 27

Bäume

Der Stammbaum

Vorlesung Einführung in die Informatik 1 vom 6. Februar 2014 von Till Tantau



Lernziele von Kapitel 27

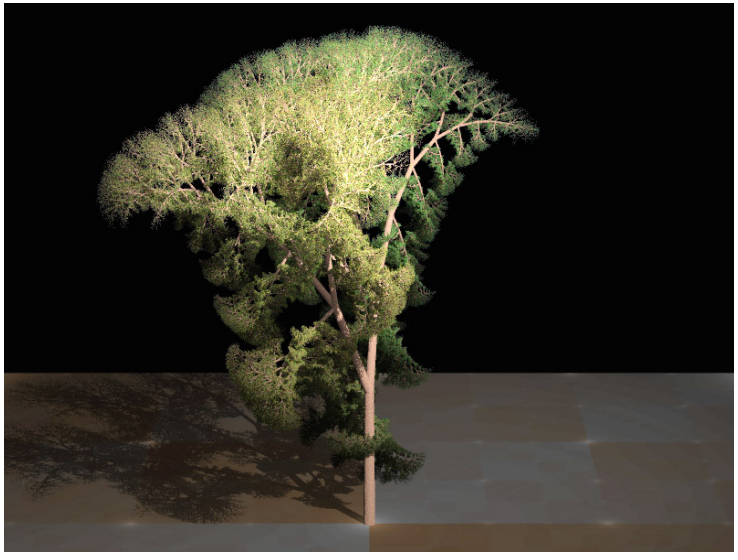
1. Das Konzept des Baumes verstehen
2. Bäume in Java implementieren können
3. Baumtraversierung kennen und implementieren können

Gliederung von Kapitel 27

Was ist ein Baum?

Vertraute Antworten einer Suchmaschine.

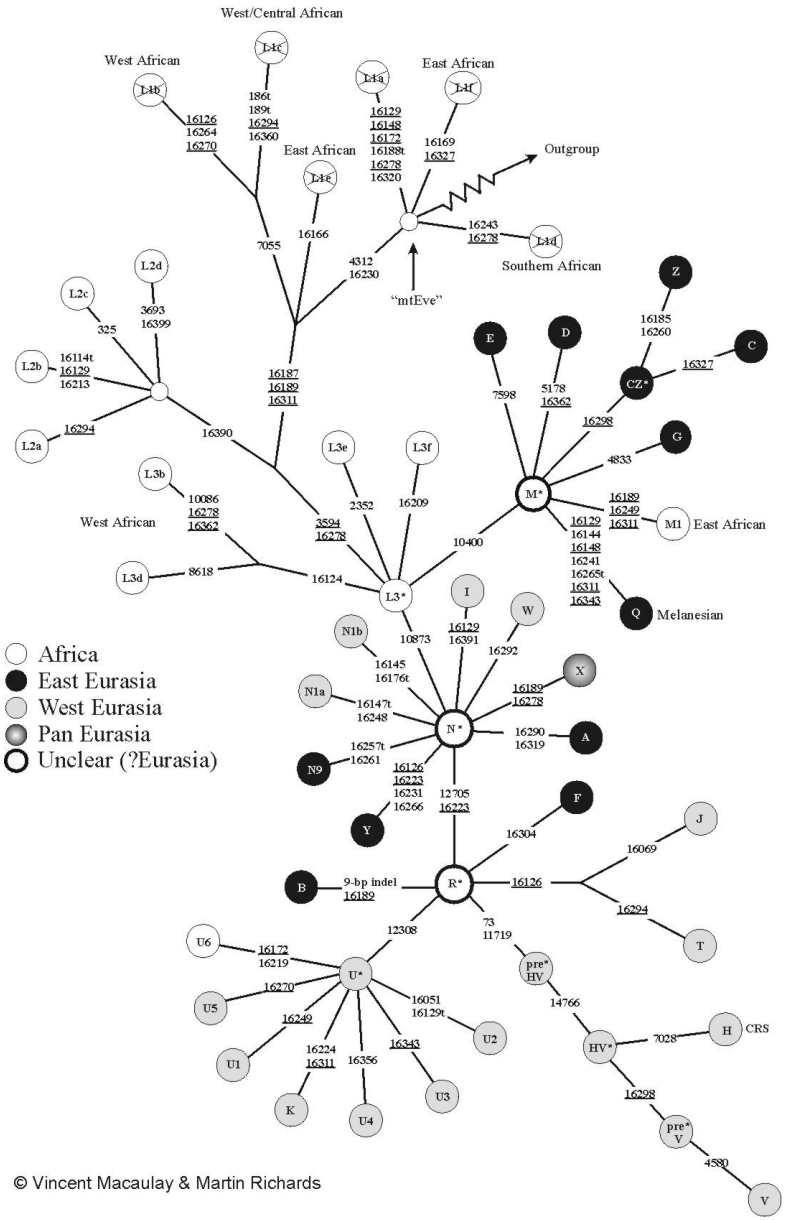
Caspar David Friedrich, public domain



Unknown author, public domain

Was ist ein Baum?

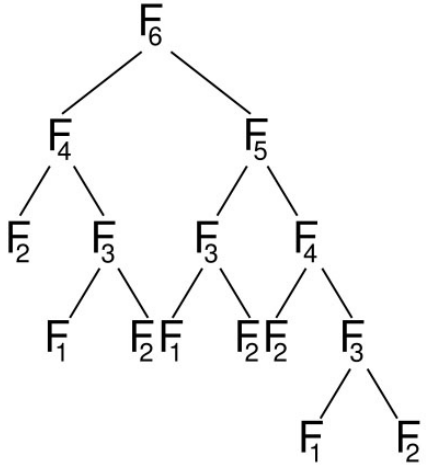
Einem Biologen vertraute Antwort einer Suchmaschine.



Was ist ein Baum?

Einer Informatikerin vertraute Antworten einer Suchmaschine.

Unknown author, public domain



Unknown author, public domain

»Bäume« sind hierarchische Strukturen.

Bäume in der Informatik

Der Begriff *Baum* steht in der Informatik allgemein für eine *hierarchische Struktur*.

Beispiele (Dinge, die als Bäume modelliert werden können)

- Genetische Stammbäume
- Dateibäume
- Menüs
- Verwaltungsstrukturen in Behörden und Firmen

Erste Motivation von Bäumen.

Baumartige Strukturen kommen so häufig vor, dass es sich lohnt,

1. ihre Eigenschaften allgemein zu studieren und
2. ihre Implementation zu beherrschen.

Zweite Motivation: Ungelöste Probleme

Betrachten Sie eine der folgenden Situationen:

- Eine Telekom-Firma möchte die Telefonnummern der deutschen Bevölkerung verwalten.
- Eine Biotech-Firma möchte ihre Produktdatenbank verwalten.
- Eine Fachschaft möchte ihre Mitgliederliste verwalten.

Probleme

- Sortierte Arrays sind als Datenstrukturen wenig geeignet, da ständig neue Einträge hinzukommen und alte gelöscht werden.
  Einfügen und Löschen dauern bei Arrays zu lange.
- Sortierte Listen sind als Datenstrukturen wenig geeignet, da ständig Einträge gesucht werden müssen.
  Suchen dauert bei Listen zu lange.

Zweite Motivation von Bäumen.

In speziellen Bäume, genannt *Suchbäume*, kann man

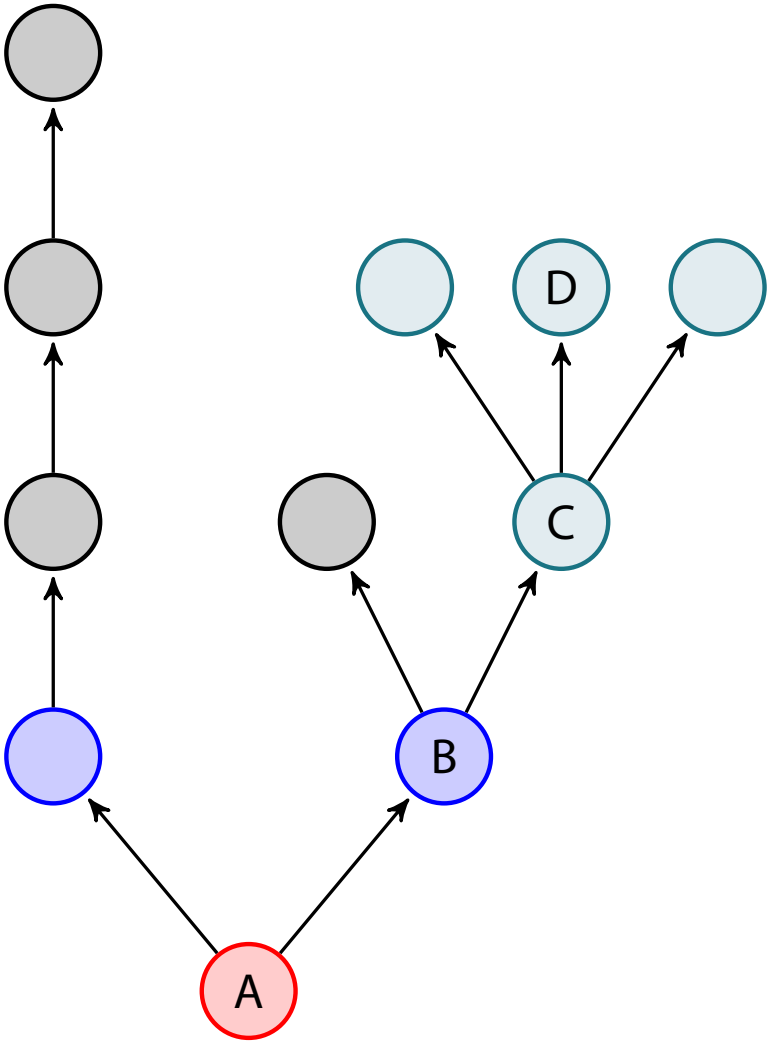
- in Zeit $O(\log n)$ Elemente einfügen,
- in Zeit $O(\log n)$ Daten löschen,
- in Zeit $O(\log n)$ Daten suchen.

Suchbäume sind also ein guter Ausgleich der Eigenschaften von Listen und Arrays.

Woraus bestehen Bäume in der Informatik?

- Informatik-Bäume beginnen an einer *Wurzel*.
- Die Wurzel hat eine Reihe von *Kindern*.
- Jedes Kind kann *wieder Kinder* haben.
- Die Wurzel und alle ihre Kinder und Kindeskinder heißen *Knoten*.
- Knoten ohne Kinder heißen *Blätter*.

Beispiel eines Baumes.



Höhe/Tiefe 4

Höhe/Tiefe 3

Unterbaum mit Wurzel C

Kinder von A

Wurzel

Wichtige Begriffe zu Bäumen

Knoten	Die Elemente des Baumes
Elternknoten	Knoten, von denen Pfeile zu Kindern ausgehen.
Kinder	Knoten, die durch Pfeile mit einem Elternknoten verbunden sind
Enkelkinder	Kinder von Kindern
Nachfahren	Kinder, Enkelkinder, Urenkelkinder, usw.
Vorfahren	Elternknoten, Großelternknoten, usw.
Wurzel	Ursprung des Baumes; einziger Knoten ohne Elternknoten
Blatt	Knoten ohne Kinder
innerer Knoten	Knoten mit Kindern
äußerer Knoten	Andere Bezeichnung für Blätter
Tiefe	Bei Knoten, die Entfernung zur Wurzel; bei Bäumen, die maximale Tiefe
Höhe	Andere Bezeichnung für Tiefe (!)
Unterbaum	Baum, der entsteht, wenn man nur einen Knoten und seine Nachfahren betrachtet
Teilbaum	Baum, der entsteht, wenn man einige Knoten weglässt

5-Minuten-Aufgabe

Zur Übung

Schlagen Sie Definitionen der folgenden Begriffe vor:

- Geschwisterknoten,
- Schicht,
- Level,
- Einzelkind,
- Onkel,
- Tante.

Varianten von Bäumen

Wo stehen die Daten?

1. In allen Knoten (Normalfall).
2. Nur in den Blättern.
3. Zusätzlich oder ausschließlich an den Kanten.

Wie viele Kinder gibt es?

1. Beliebig viele Kinder an allen Knoten.
2. Genau zwei oder null Kinder (Binärbaum).
3. Höchstens zwei Kindern (auch Binärbaum genannt, grrr).
4. Genau zwei, drei oder null Kinder (2-3-Baum).

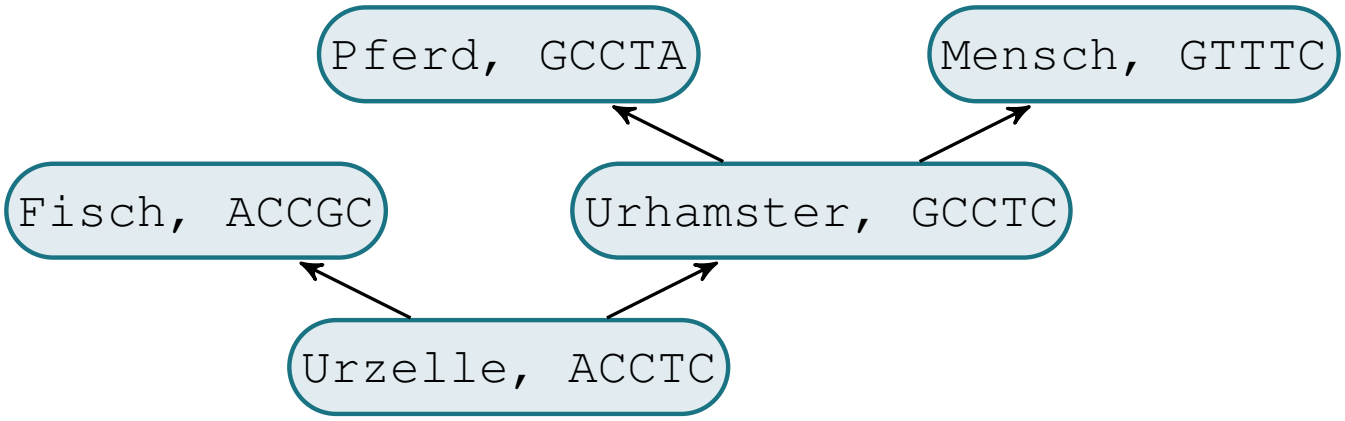
Wie implementiert man Bäume in Java?

Wie bei Listen benutzt man mehrere Klassen, um Bäume zu implementieren:

1. Eine *Knotenklasse*. Diese entspricht den »Zellen« bei Listen.
2. Eine *eigentliche Baumklasse*, die einen Verweis auf die Wurzel enthält.
3. Optional eine Klasse für die *Werte*, die an den Knoten stehen.
   Beispiel: Bei einem phylogenetischen Baum wäre diese Klasse `Species`

Implementation phylogenetischer Bäume

Die `Species`-Klasse.




```
class Species {
  String speciesName;
  String genome;

  // Konstruktor
  Species (String name, String genome)
  {
    this.speciesName = name;
    this.genome      = genome;
  }
}
```

Implementation phylogenetischer Bäume

Die Knoten-Klasse

Implementation mit beliebig vielen Kindern:

```
class Node {
  // Die Spezies, die der Knoten speichert
  Species species;
  // Die ganze Brut von Kindern...
  Node[] children;
}
```

Implementation mit höchstens zwei Kindern:

```
class Node {
  Species species;
  Node leftChild, rightChild;
}
```

Bemerkungen:

- Zur Vereinfachung betrachten wie im Folgenden nur binäre Bäume.
- Man kann zusätzliche ein `parent` Attribut einführen. Dies entspricht dem `prev`-Attribut bei doppelt verketteten Listen.

Implementation phylogenetischer Bäume

Die eigentliche Baumklasse


```
class PhylogeneticTree {

  // Attribut, das auf die Wurzel verweist
  Node root;

  // Default-Konstruktor
  PhylogeneticTree () {
    this.root = null;
  }

  // Methoden
  boolean isEmpty () {
    return this.root == null;
  }
  ...
}
```

Konstruktion neuer Bäume

- Ein neu erzeugter Baum ist erstmal leer.
- Dann kann man nach und nach Elemente einfügen.
- Alternativ kann man aber auch neue Bäume erzeugen, indem man zwei bestehende zu einem neuen zusammenfasst.

```
class PhylogeneticTree {
  // Konstruktor, der einen Baum aufbaut
  // mit bereits konstruierten linken und rechten
  // Teilbäumen. Die Wurzel wird neu erzeugt.
  PhylogeneticTree (Species s,
                    PhylogeneticTree left,
                    PhylogeneticTree right) {
    this.root = new Node ();
    this.root.species = s;
    if (left != null) {
      this.root.leftChild = left.root;
    }
    if (right != null) {
      this.root.rightChild = right.root;
    }
  }
}
```

Zur Übung

Visualisieren Sie die Objekte und Verweise, die durch folgenden Code erzeugt werden:

```
Species start  = new Species("first cell", "CCCT");
Species human  = new Species("human",      "ACGT");
Species monkey = new Species("monkey",     "ACCT");
Species fish   =
  new Species("Hommingberger Gepardenforelle", "CTCT");

PhylogeneticTree myTree =
  new PhylogeneticTree (start,
    new PhylogeneticTree (monkey,
      null,
      new PhylogeneticTree (human, null, null)
    ),
    new PhylogeneticTree (fish, null, null));
```

Einfügen und Löschen in Bäumen

- Das Einfügen und Löschen einzelner Knoten in der Mitte eines Baumes ist *fummelig*.
- Einfacher ist es, ganze Teilbäume zu löschen oder einzufügen.

```
class PhylogeneticTree {
  ...

  void addTreeAsLeftChild (Node             where,
                           PhylogeneticTree what) {
    where.leftChild = what.root;
  }

  void removeLeftSubtree (Node where) {
    where.leftChild = null;
  }
}
```

Traversierung von Bäumen

- *Traversieren* bedeutet, dass man alle Knoten eines Baumes abläuft.
- An jedem Knoten kann man nun etwas »tun«, beispielsweise, die Spezies ausdrucken oder verändern.
- Die *Reihenfolge*, in der die Knoten traversiert werden, ist bei Bäumen nicht ganz klar.
- Es gibt drei wichtige Reihenfolgen
  1. In-Order:
     Erst den linken Teilbaum, dann der Knoten, dann der rechte Teilbaum.
  2. Pre-Order:
     Erst der Knoten, dann der linke Teilbaum, dann der rechte Teilbaum.
  3. Post-Order:
     Erst der linke Teilbaum, dann der rechte Teilbaum, dann der Knoten.

Beispiel einer Traversierung

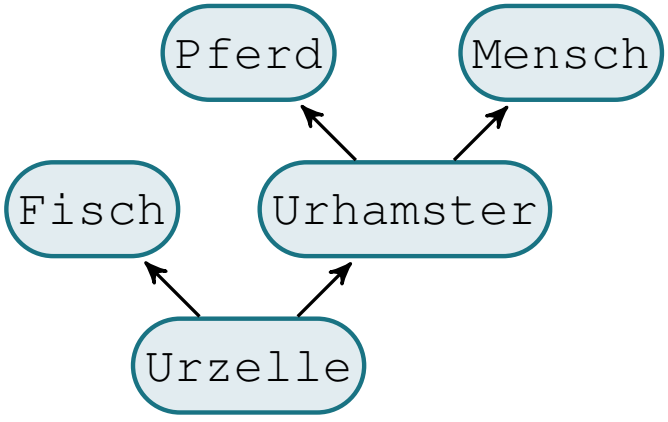

```
class PhylogeneticTree {
  ...
  int countNodesInSubtree (Node node) {
    // Rekursive Post-Order-Traversierung
    if (node == null) {
      return 0;
    }
    else {
      return countNodesInSubtree (node.leftChild) +
             countNodesInSubtree (node.rightChild) +
             1;
    }
  }

  int countNodeInTree () {
    return countNodeInSubtree (this.root);
  }
}
```

5-Minuten-Aufgabe

Zur Übung

Was geben die Methoden bei Eingabe `Urzelle` aus?




```
void printAllSpeciesPreOrder (Node node) {
  if (node != null) {
    System.out.println (node.species.speciesName);
    printAllSpeciesPreOrder (node.leftChild);
    printAllSpeciesPreOrder (node.rightChild);
} }
void printAllSpeciesInOrder (Node node) {
  if (node != null) {
    printAllSpeciesInOrder (node.leftChild);
    System.out.println (node.species.speciesName);
    printAllSpeciesInOrder (node.rightChild);
} }
```

Bäume in der Informatik

- Unter *Bäumen* versteht man in der Informatik alle Arten hierarchischer Strukturen.
- Bäume bilden eine *Datenstruktur*, die ähnlich wie Listen aufgebaut ist:

```
class Tree {
  Node root;
}

class Node {
  SomeType data;

  Node leftChild;
  Node rightChild;
}
```

- Einfügen und Löschen in Bäumen ist »fummelig«.

Zusammenfassung

Die drei Arten der Traversierung eines Baumes


```
void doSomethingPreOrder (Node node) {
  if (node != null) {
    do_something_for (node);
    doSomethingPreOrder (node.leftChild);
    doSomethingPreOrder (node.rightChild);
} }

void doSomethingPostOrder (Node node) {
  if (node != null) {
    doSomethingPostOrder (node.leftChild);
    doSomethingPostOrder (node.rightChild);
    do_something_for (node);
} }

void doSomethingInOrder (Node node) {
  if (node != null) {
    doSomethingInOrder (node.leftChild);
    do_something_for(node);
    doSomethingInOrder (node.rightChild);
} }
```