Teil I
XML

> **XML**
> XML steht für “eXtensible Markup Language“. XML ist eine universelle, erweiterbare Sprache, mit der man konkrete Auszeichnungssprachen erzeugen kann.

.

Mit XML können Dokumente zur Informationsdarstellung gebildet werden. Die Struktur von XML-Dokumenten kann genau festgelegt werden, um Sprachen für bestimmte Anwendungsbereiche zu entwickeln.

1 Wohlgeformt

> **Wohlgeformt**
> Ein XML-Dokument, das alle Regeln von XML erfüllt, heißt wohlgeformt.

.

Beispiel-XML-Dokument:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Kurs SYSTEM "kurs.dtd">
<Kurs name="Infokurs">
    <Schueler id="87" geschlecht="m">
      <Vorname>Tobias</Vorname>
      <Name>Schwarz</Name>
      <Kurssprecher/>
    </Schueler>
</Kurs>
```

.

Ein XML-Dokument beginnt mit einem Prolog, dieser kennzeichnet das Dokument als XML-Dokument.

```
<?xml version="1.0" encoding="utf-8"?>
```

.

Eine XML-Datei besteht aus Elementen. Diese können sowohl untergeordnete Elemente (in diesem Fall Name und Vorname) als auch Attribute (id und geschlecht) halten. Sogar leere Elemente sind möglich (Kurssprecher).

```
<Schüler id="87" geschlecht="m">
  <Name>Schwarz</Name>
  <Vorname>Tobias</Vorname>
  <Kurssprecher/>
</Schüler>
```

.

Eine Dokumenttypdefinition kann wie folgt eingebunden werden:

```
<!DOCTYPE Kurs SYSTEM "kurs.dtd">
```

2 Dokumenttypdefinitionen (DTD)

> **Valide**
> Ein XML-Dokument, das alle Festlegungen einer DTD erfüllt, heißt gültig bzw. valide bzgl. dieser DTD.

Die DTD für das oben angegebene Beispiel wäre folgendes:

```
<!ELEMENT Kurs (Schueler*) >
<!ATTLIST Kurs
  name CDATA #REQUIRED
>
<!ELEMENT Schueler (Vorname, Name, Kurssprecher?) >
<!ATTLIST Schueler
  id CDATA #REQUIRED
  geschlecht CDATA #REQUIRED
>
<!ELEMENT Vorname (#PCDATA) >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Kurssprecher EMPTY >
```

.
In Attributlisten muss bei CDATA spezifiziert werden, ob das Attribut in in jedem Element vorkommen muss (REQUIRED) oder weggelassen werden kann (IMPLIED).

Tabelle 1: Zusatzsymbole

()	Klammern zur Bildung von Elementgruppen
,	Trennzeichen innerhalb einer Sequenz von Elementen
|	Trennzeichen zwischen sich ausschließenden Alternativen
*	Element(gruppe) kann beliebig oft (auch gar nicht) vorkommen
+	Element(gruppe) muss mindestens einmal vorkommen, kann mehrfach vorkommen
?	Element(gruppe) kann einmal oder kein mal vorkommen

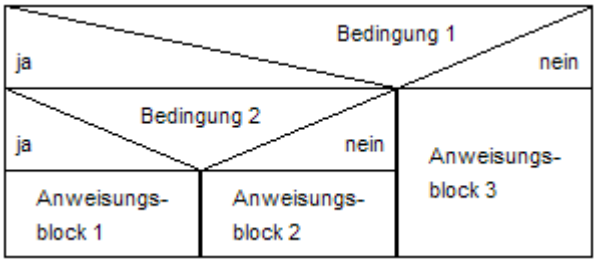
Teil II
Algorithmen

Algorithmusbegriff
Ein Algorithmus ist eine Verarbeitungsvorschrift zur Lösung eines Problems, die so präzise formuliert ist, dass sie auch von einer Maschine abgearbeitet werden kann.

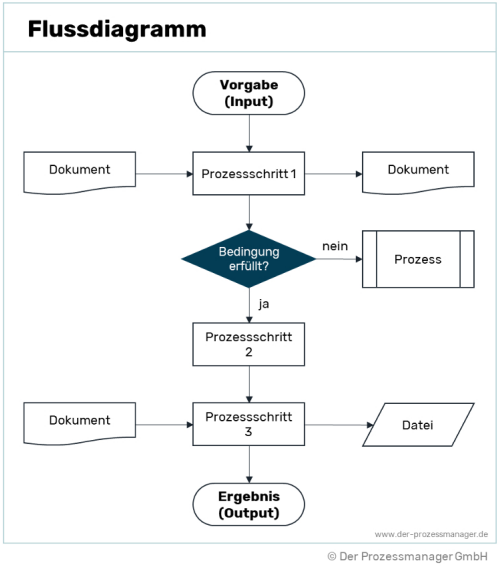
Ansprüche:

- Ausführbar
  Jeder Einzelschritt soll vom "Prozessor" ausführbar sein.
- Eindeutig
  Die Abfolge der Schritte sollte eindeutig sein.
- Endlich
  Die Anweisungen des Algorithmus sollten nicht unendlich sein.
- Allgemein
  Der Algorithmus sollte nicht nur ein bestimmtes Problem, sondern auch ähnliche Probleme lösen können.

3 Struktogramm und Flussdiagramm



Dies ist ein nassi-shneiderman Diagramm, auch Struktogramm genannt.



Dies ist ein Flussdiagramm.

4 Anweisungen

Algorithmen bestehen aus zwei Arten von Anweisungen:

- Elementaranweisungen
  Basisaktionen des Prozessors
- Kontrollanweisungen
  Anweisungen, die die Ablauflogik festlegen (Wiederholungen, Fallunterscheidungen, Sequenzbildung)

5 Spezifikation

Spezifikation
Eine Spezifikation besteht aus einer Vorbedingung, die einen Ausgangszustand beschreibt, sowie einer Nachbedingung, die einen Endzustand beschreibt.

terminierend
Ein Algorithmus heißt terminierend bzgl. einer Spezifikation, wenn er bei jedem Ausgangszustand, der die Vorbedingung erfüllt, nach endlich vielen Verarbeitungsschritten zu einem Ende kommt.

korrekt
Ein Algorithmus heißt (total) korrekt bzgl. einer Spezifikation, wenn er terminierend ist und jeden Ausgangszustand, der die Vorbedingung erfüllt, in einen Endzustand überführt, der die Nachbedingung erfüllt

.

Die Korrektheit und Termination von Algorithmen kann auf zwei Arten erforscht werden: Einerseits durch Beweis und andererseits durch Testfälle.

6 Laufzeitverhalten

Problemgröße
Eine (Kosten-) Funktion f wächst schneller als eine (Kosten-) Funktion g, wenn der Quotient f(n)/g(n) mit wachsendem n gegen unendlich strebt.
Eine (Kosten-) Funktion f wächst langsamer als eine (Kosten-) Funktion g, wenn der Quotient f(n)/g(n) mit wachsendem n gegen 0 strebt.
Eine (Kosten-) Funktion f wächst genauso schnell wie eine (Kosten-) Funktion g, wenn der Quotient f(n)/g(n) mit wachsendem n gegen eine Konstante c mit c>0 strebt.

7 Sortieralgorithmen

7.1 Selection Sort

> **Selection Sort**
> Bei Selection Sort wird jeweils das kleinste Element der Liste ausgewählt und an die neue Liste angehängt.

.

Der Algorithmus ist folgender:

```
ALGORITHMUS selectionsort
Übergabe: Liste L
unsortierter Bereich ist die gesamte Liste L
der sortierte Bereich ist zunächst leer
SOLANGE der unsortierte Bereich noch Elemente hat:
    suche das kleinste Element im unsortierten Bereich
    entferne es im unsortierten Bereich
    füge es im sortierten Bereich an letzter Stelle an
Rückgabe: sortierter Bereich
```

.

Das Laufzeitverhalten ist folgendes:

	Best Case	Worst Case	Average Case
$O(n)$	n^2	n^2	n^2

7.2 Insertion Sort

> **Insertion Sort**
> Bei Insertion Sort wird das jeweils erste Element der alten Liste genommen und in die richtige Position in der neuen Liste gesetzt.

Der Algorithmus ist folgender:

```
ALGORITHMUS insertionsort
Übergabe: Liste
sortierter Bereich besteht aus dem ersten Element der Liste
unsortierter Bereich ist die Restliste (ohne das erste Element)
SOLANGE der unsortierte Bereich Elemente hat:
    entferne das erste Element aus dem unsortierten Bereich
    füge es an der richtigen Stelle im sortierten Bereich ein
Rückgabe: sortierter Bereich
```

.

	Best Case	Worst Case	Average Case
$O(n)$	n	n^2	n^2

7.3 Bubble Sort

> **Bubble Sort**
> Bei Bubble Sort werden von Anfang bis Ende Elemente der Liste vertauscht, bis die Länge des unsortierten Bereiches 0 ist. Diese Länge des unsortierten Bereiches wird mit jedem Durchlauf verringert.

.

Der Algorithmus ist folgender:

```
ALGORITHMUS bubblesort
Übergabe: Liste
unsortierter Bereich ist zunächst die gesamte Liste
SOLANGE der unsortierte Bereich noch mehr als ein Element hat:
    durchlaufe den unsortierten Bereich von links nach rechts
    wenn dabei zwei benachbarte Elemente in der falschen Reihenfolge vorliegen:
        vertausche die beiden Elemente
    verkürze den unsortierten Bereich durch Weglassen des letzten Elements
Rückgabe: überarbeitete Liste
```

.

Das Laufzeitverhalten ist folgendes:

	Best Case	Worst Case	Average Case
$O(n)$	n	n^2	n^2

7.4 Quicksort

> **Bubble Sort**
> Bei Bubble Sort werden von Anfang bis Ende Elemente der Liste vertauscht, bis die Länge des unsortierten Bereiches 0 ist. Diese Länge des unsortierten Bereiches wird mit jedem Durchlauf verringert.

```
ALGORITHMUS quicksort
Übergabe: Liste L
wenn die Liste L mehr als ein Element hat:
    # zerlegen
    wähle als Pivotelement p das erste Element der Liste aus
    erzeuge Teillisten K und G aus der Restliste (L ohne p) mit:
    - alle Elemente aus K sind kleiner als das Pivotelement p
    - alle Elemente aus G sind größergleich als das Pivotelement p
    # Quicksort auf die verkleinerten Listen anwenden
    KSortiert = quicksort(K)
    GSortiert = quicksort(G)
    # zusammensetzen
    LSortiert = KSortiert + [p] + GSortiert
sonst:
    LSortiert = L
Rückgabe: LSortiert
```

	Best Case	Worst Case	Average Case
$O(n)$	$n * log(n)$	$n * log(n)$	n^2

8 Graphen

8.1 Algorithmus von Dijkstra

```
ALGORITHMUS # von Dijkstra
Übergabedaten: Graph, startKnoten, zielKnoten
# Vorbereitung des Graphen
für alle knoten des Graphen:
    setze abstand auf 'u'
    setze herkunft auf None
setze abstand von startKnoten.abstand auf 0
füge startKnoten in eine Liste zuVerarbeiten ein
# Verarbeitung der Knoten
SOLANGE die Liste zuVerarbeiten nicht leer ist:
    bestimme einen Knoten minKnoten aus zuVerarbeiten mit minimalem Abstand
    entferne minKnoten aus zuVerarbeiten
    für alle nachbarKnoten von minKnoten:
        gewicht = Gewicht der Kante von minKnoten zu nachbarKnoten
        neuerAbstand = (abstand von minKnoten) + gewicht
        WENN abstand von nachbarKnoten == 'u':
            füge nachbarKnoten in die Liste zuVerarbeiten ein (z.B. am Listenende)
            setze abstand von nachbarKnoten auf neuerAbstand
            setze herkunft von nachbarKnoten auf minKnoten
        SONST:
            WENN nachbarKnoten in zuVerarbeiten liegt:
                WENN abstand von nachbarKnoten > neuerAbstand:
                    setze abstand von nachbarKnoten auf neuerAbstand
                    setze herkunft von nachbarKnoten auf minKnoten
weglaenge = abstand von zielKnoten
# Erzeugung des Weges zum zielKnoten
weg = []
knoten = zielKnoten
SOLANGE knoten != startKnoten und herkunft von knoten != None:
    weg = [(herkunft von knoten, knoten)] + weg
    knoten = herkunft von knoten
Rückgabedaten: (weg, weglaenge)
```

8.2 Travelling-Salesman-Problem

> **Integration der naheliegendsten Knoten**
> Die naheliegende Lösung wäre, den jeweils naheliegendendsten Knoten für die Rundreise zu integrieren