



Leistungskurs Informatik

Merkteil



Stand: 16.03.2025

Inhaltsverzeichnis

Abkürzungsverzeichnis	3
1. Imperative Programmierung.....	5
1.1. Algorithmen.....	5
1.2. Datentypen	6
1.3. Unterprogramme	19
1.4. Komplexität.....	22
1.5. Heuristiken.....	24
1.6. Grenzen der Algorithmierbarkeit.....	36
2. Softwareentwicklung	38
2.1. Programmiersprachen und -paradigmen	38
2.2. Versionsverwaltung	40
2.3. OOP	42
2.4. Klassisch oder agil.....	47
2.5. Softwarearchitektur.....	51
3. Technische Informatik	52
3.1. Systemsoftware	52
3.2. Zahlen und Codierung.....	53
3.3. Schaltnetze	60
3.4. Rechnerhardware	68
3.5. Netzwerke.....	71
3.6. Kryptologie.....	90
4. Umgang mit Daten	103
4.1. Intro.....	103
4.2. Erstellen der Datenbasis und KI.....	105
4.3. Modellierung	108
4.4. Implementierung und Optimierung	113
4.5. Verarbeitung und mehr.....	117
5. Sprachen und Automaten.....	120
5.1. Sprachen	120
5.2. Automaten.....	123
5.3. Übersetzung formaler Sprachen	128
Stichwortverzeichnis	131

Abkürzungsverzeichnis

ALU	arithmetic logic Unit	elektronisches Rechenwerk
APFS	Apple File System	Dateisystem
ARP	Address Resolution Protocol	Netzwerkprotokoll für Zuordnung von IP-Adresse zu MAC-Adresse
ASCII	American Standard Code for Information Interchange	Zeichencodierung
AST	Abstract Syntax Tree	Syntaxbaum bei der Abarbeitung von Quelltext
BNF	Backus-Naur-Form	Art der Darstellung für formale Sprachen
CIDR	Classless Inter-Domain Routing	flexibles Adressierungssystem für IP-Adressen
CPU	Central Processing Unit	zentrale Verarbeitungseinheit
CSV	comma-separated values	Dateiformat
DB	Datenbasis	Daten und ihre Struktur für eine Datenbank
DBMS	Datenbankmanagementsystem	Software zur Arbeit mit Datenbanken
DBS	Datenbanksystem	Vereinigung aus DB und DBMS
DCL	Data Control Language	Rechteverwaltung in SQL-Datenbanken
DDL	Data Definition Language	Erstellen, Ändern und Löschen von Tabellen in SQL-Datenbanken
DDR	Double Data Rate (Synchronous Dynamic Random Access Memory)	halbleiterbasierter RAM-Typ
DEA	Deterministischer endlicher Automat	Automat für reguläre Sprachen
DHCP	Dynamic Host Configuration Protocol	automatische Zuweisung von IP-Adressen
DML	Data Manipulation Language	Einfügen, Ändern und Löschen von Daten bei SQL-Tabellen
DNF	disjunktive Normalform	oder-Verknüpfung von konjunktiv verknüpften Schalttermen
DNS	Domain Name System	Übersetzung Domainname – IP-Adresse
DQL	Data Query Language	Abfrage und Aufbereitung gesuchter Informationen in SQL
DVD	Digital Video Disc / Digital Versatile Disc	optisches Speichermedium
EXT4	fourth extended filesystem	Dateisystem
FAT	File Allocation Table	Familie von Dateisystemen
FIFO	First In First Out	Grundkonzept Schlange
HA	Halbaddierer	ein besonderes Standardschaltnetz
HDD	Hard Disk Drive	Festplattenlaufwerk
HTTP(S)	Hypertext Transfer Protocol (Secure)	Protokoll zur Übertragung von Webseiten
ICMP	Internet Control Message Protocol	Netzwerkprotokoll für Diagnose- und Fehlerberichte
IEEE	Institute of Electrical and Electronics Engineers	Berufsverband u.a. von Ingenieuren, Technikern und Wissenschaftlern
IMAP	Internet Message Access Protocol	Protokoll zum Organisieren von E-Mails
IP	Internet Protocol	Ansatz zur Adressierung von Netzwerkgeräten

IR	Intermediate Representation	Zwischendarstellung von Quelltext
KNF	konjunktive Normalform	und-Verknüpfung von disjunktiv verknüpften Schaltterminen
KVD	Karnaugh-Veitch-Diagramm	Hilfsmittel für die Vereinfachung von Schaltterminen
LAN	Local Area Network	Lokalbegrenztes Rechnernetzwerk
LIFO	Last In First Out	Grundkonzept Stapel
LLM	Large Language Model	Grundansatz von ChatGPT
MAC	Media Access Control	Ansatz zur Adressierung eines Netzwerkgerätes
MAN	Metropolitan Area Network	Rechnernetzwerk etwa auf Stadtgröße
MUX	Multiplexer	ein besonderes Standardschaltnetz
NAS	Network Attached Storage	Dateiserver innerhalb eines Netzwerks
NEA	Nichtdeterministischer endlicher Automat	Automat für reguläre Sprachen
NTFS	New Technology File System	Dateisystem
OOP	objektorientierte Programmierung	Programmieransatz
PDA	Pushdown automaton	Kellerautomat für kontextfreie Sprachen
POP3	Post Office Protocol Version 3	Protokoll zum E-Mail-Empfang
RAID	Redundant Array of Independent Disks	Technologie, die mehrere Festplatten kombiniert
RIP	Routing Information Protocol	Netzwerkprotokoll um Routing-Informationen zwischen Routern auszutauschen
RTT	Round-Trip Time	Zeit vom Senden eines Echo-Requests bis zum Empfang des Echo-Replies
SAT	Source Address Table	Zuweisungstabelle für Switches
SMTP	Simple Mail Transfer Protocol	Protokoll zum E-Mail-Versand
SQL	Structured Query Language	Abfragesprache für Datenbanken
SSD	Solid-State-Drive	elektronisches Speichermedium
TCL	Transaction Control Language	Transaktionskontrolle für SQL-Datenbank
TCP	Transmission Control Protocol	verbindungsorientiertes Netzwerkprotokoll
TTL	Time To Live	Angabe, wie viele Zwischenstationen ein IP-Paket noch durchlaufen darf
UDP	User Datagram Protocol	verbindungsloses Netzwerkprotokoll
UML	Unified Modeling Language	vereinheitlichte Modellierungssprache
USB	Universal Serial Bus	bit-serielles Übertragungssystem
UTF	Unicode Transformation Format	Zeichencodierungsansatz für Unicode
VA	Volladdierer	ein besonderes Standardschaltnetz
VLAN	Virtual Local Area Network	Logische Segmentierung eines physischen Netzwerks
VLSM	Variable Length Subnet Masking	Ansatz für Subnetze mit unterschiedlichen Maskenlängen
VPN	Virtual Private Network	sichere Verbindung über öffentliche Netzwerke
WAN	Wide Area Network	überregionales Rechnernetzwerk

1. Imperative Programmierung

1.1. Algorithmen

Definition

Als **Algorithmus** bezeichnen wir eine Handlungsvorschrift, die so formuliert werden kann, dass sie von Maschinen abgearbeitet werden kann und zur Lösung von Problemen führt.

Eigenschaften

Wir erwarten von Algorithmen folgende Eigenschaften:

- **Allgemeinheit**

Der Algorithmus löst eine Vielzahl von Problemen der gleichen Art. Die Auswahl eines konkreten Problems erfolgt über die Eingabedaten bzw. Parameter.

- **Determinismus**

Zu jedem Zeitpunkt der Abarbeitung ist der nachfolgende Schritt eindeutig festgelegt.

- **Determiniertheit**

Bei gleichen Voraussetzungen (inkl. Eingaben) liefert der Algorithmus stets dieselben Ausgaben.

- **Endlichkeit / Finitheit**

Die Beschreibung des Algorithmus und die benötigten Ressourcen zur Umsetzung sind endlich.

- **Terminiertheit**

Die Abarbeitung des Algorithmus endet nach endlich vielen Schritten.

1.2. Datentypen

1.2.1. Überblick

Zu Datentypen gehören immer ihr Wertebereich, Speicherplatzbedarf sowie darauf definierte Operationen.

Unterteilung

Wir unterscheiden zwischen elementaren und strukturierten Datentypen.

Zu den **elementaren Datentypen** gehören Wahrheitswerte, Einzelzeichen und Zahlenbereiche wie ganze Zahlen, rationale Zahlen und komplexe Zahlen.

Strukturierte Datentypen sind aus elementaren und/oder anderen strukturierten Datentypen aufgebaut.

Zu strukturierten Datentypen gehören bspw. Zeichenketten, Listen, Dateien, Tupel, Mengen, Graphen, Bäume, Stapel und Schlangen.

Die Datenstruktur des Verbunds bietet uns die Möglichkeit, Daten desselben oder verschiedener Typen zu einem gemeinsamen neuen Datentyp zusammenzufassen.

VariablenTypen

Während die Datentypen allgemein beschreiben, zu welchem Bereich eine Variable gehört, geben die VariablenTypen an, wie dieser Typ in der konkreten Programmiersprache heißt.

In Python können die Datentypen z.B. durch folgende VariablenTypen genutzt werden.

Datentyp	VariablenTyp in Python
ganze Zahl	<code>int</code>
rationale Zahl	<code>float</code>
komplexe Zahl	<code>complex</code>
Wahrheitswert	<code>bool</code>
Zeichenkette	<code>str</code>
Tupel	<code>tuple</code>
Menge	<code>set</code>
Liste	<code>list</code>

Variablenkonzept in Python

In Python enthält jedes Objekt mindestens drei Daten:

- Variablentyp
- Wert
- Referenzzähler

Variablen sind in Python grundsätzlich **Objektreferenzen** (auch **Zeiger** genannt). Wir werden trotzdem statt vom Objekt häufig von der „Variablen“ sprechen.

Der Variablentyp eines Python-Objekts kann über die `type`-Funktion ausgegeben werden.

Durch die `id`-Funktion kann die Adresse des Python-Objekts angezeigt werden.

Unter Verwendung des `sys`-Moduls kann auch der **Referenzzähler** eines Python-Objekts angezeigt werden. Dieser gibt an, wie häufig auf dieses Objekt referenziert (gezeigt) wird. Ist dieser Wert 0, dann löscht der sogenannte Garbage Collector das Objekt und gibt den Speicherplatz damit wieder frei.

Beispiel

```
import sys                                # Einbinden des sys-Moduls

a = [1, 2, 3]
print(sys.getrefcount(a))                  # Ausgabe: 2

b = a
print(sys.getrefcount(a))                  # Ausgabe: 3
```

Achtung: dadurch, dass Variablen in Python Objektreferenzen sind, können Irritationen entstehen.

Beispiel

```
a = [1, 2, 3]
b = a
b[2] = 4
```

Erwartungshaltung: nach Ausführung besitzt `a` den Wert `[1, 2, 3]` und `b` den Wert `[1, 2, 4]`.

Real: `a` und `b` besitzen beide den Wert `[1, 2, 4]`, da das dritte Element des mit `b` verknüpften Objekts auf 4 geändert wird. `a` verknüpft auf dasselbe Objekt und damit ist auch sein drittes Element auf 4 geändert.

Tipp für echte Kopien der Liste: statt `b = a` die Anweisung `b = a[:]` verwenden.

Typinterpretation

Durch Voranstellen des gewünschten Variabtentyps kann versucht werden, eine Variable mit einem anderen Variabtentypen zu interpretieren. Der Variabtentyp der Ausgangsvariable wird dabei nicht automatisch geändert.

Beispiel

```
e = 2.7182828459045
f = int(e)          # e wird als ganze Zahl interpretiert und in
                     # f gespeichert
print(f)           # Ausgabe: 2
```

Ein- und Ausgaben in Python sind grundsätzlich Zeichenketten.

Beispiel

```
01: e = input()
02: n = int(e)
03: print("2*n = " + str(2*n))
```

Erklärung

Zeile 01

Die eingegebene Zeichenkette wird als Objekt gespeichert und mit der Variablen e referenziert.

Zeile 02

Das mit e verbundene Objekt wird als ganze Zahl interpretiert, als neues Objekt angelegt und mit der Variablen n referenziert.

Zeile 03

Das Doppelte des mit n verbundenen Werts wird als Zeichenkette interpretiert. Diese wird hinten an die Zeichenkette "2*n = " angehängt. Die so entstandene Zeichenkette wird in der Konsole ausgegeben.

Veränderbare Variablen

Python unterscheidet zwischen veränderbaren („mutable“) und unveränderlichen („immutable“) Objekten.

Zu den unveränderlichen Objekten zählen solche, die als Variablentyp besitzen: `int`, `float`, `bool`, `complex`, `tuple`, `str`.

Zu den veränderlichen Objekten zählen u.a. `list`-, `set`- und `dict`-Objekte.

Folge

```
01: zk = "Test"  
02: zk[2] = "x"
```

Zeile 02 führt zu der Fehlermeldung „`TypeError: 'str' object does not support item assignment`“ da eine Zeichenkette in Python prinzipiell unveränderlich ist.

Alternative Lösung

```
01: zk = "Test"  
02: zk = zk[:2]+"x"+zk[3:]
```

Erklärung für Zeile 02

Das 0. und 1. Zeichen von `zk` wird mit „`x`“ verbunden. An die so entstandene Zeichenkette wird der Teil von `zk` angehängt, der mit dem Index 3 beginnt. Die so entstehende Zeichenkette wird als Objekt gespeichert und mit der Variablen `zk` referenziert.

1.2.2. Felder und Listen

In Feldern und Listen werden gleichartige Objekte gesammelt. Bei Feldern können alle Elemente direkt adressiert werden, bei Listen nur je ein spezielles Element.

Intern werden beide Datentypen in Python mit dem Variablen Typ `List` umgesetzt.

Implementierung von Feldern

- eckige Klammern, Elemente durch Komma trennen
- Startindex 0
- veränderbarer Typ → Elemente können direkt geändert werden
- Kopie des Felds `m` mit Variablenamen `s` erzeugen: `s = m[:]`
- in Python können Objekte unterschiedlicher Typen im selben Feld zusammengefasst werden

Länge	<code>len</code>	Funktion
Element hinten anhängen	<code>append</code>	Prozedur
Element an Stelle einfügen	<code>insert</code>	Prozedur
Feld hinten anhängen	<code>extend</code>	Prozedur
Feld leeren	<code>clear</code>	Prozedur
Erstes Auftreten eines Elements	<code>index</code>	Funktion
aufsteigendes Sortieren	<code>sort</code>	Prozedur

FIFO vs. LIFO

FIFO	LIFO
First In, First Out	Last In, First Out
Schlange	Stapel
neue Elemente am Ende	neue Elemente am Anfang
Hinzufügen: <code>enqueue</code>	Hinzufügen: <code>push</code>
Entfernen: <code>dequeue</code>	Entfernen: <code>pop</code>
<code>Queue.py</code>	<code>Stack.py</code>
<code>neu = Queue.Queue()</code>	<code>neu = Stack.Stack()</code>

1.2.3. Dateien

Wir unterscheiden in Python zwischen Textdateien und Binärdateien. Bei **Textdateien** werden zeilenweise Zeichenketten gespeichert. Dies führt dazu, dass nur diejenigen Objekte in einer Textdatei gespeichert werden können, die sinnvoll als Zeichenketten darstellbar sind.

Wichtige Befehle im Umgang mit Dateien

Öffnen der Datei Test.txt zum Lesen: `datei = open('Test.txt', 'r')`

Öffnen der Datei Test.txt zum Speichern: `datei = open('Test.txt', 'w')`

Öffnen der Datei Test.txt um Inhalte an den bisherigen Dateiinhalt anzuhängen:

```
datei = open('Test.txt', 'a')
```

Schließen der Datei mit Variablenamen datei: `datei.close()`

Hinzufügen der Zeichenkette zk in die Datei mit dem Variablenamen datei:

```
datei.write(zk)
```

Hinzufügen der Zeichenkette zk in die Datei mit dem Variablenamen datei inkl. anschließendem Zeilenumbruch:

```
datei.write(zk + '\n')
```

Das Auslesen aller Zeilen der Datei datei kann über eine Zählschleife realisiert werden:

```
for eintrag in datei:  
    ...
```

Löschen von Leerzeichen und Zeilenumbruch am Ende der Zeichenkette zk: `zk.rstrip()`

Überprüfung, ob die Datei mit dem Dateinamen dateiname existiert (benötigt Modul os):

```
if os.path.exists(dateiname):
```

Öffnen einer Datei mit utf-8-Kodierung: `datei = open('Test.txt', 'r', encoding='utf-8')`

Erzeugen der Repräsentation des Objekts objekt: `repr(objekt)`

Evaluieren einer als Zeichenkette zk gegebenen Objektrepräsentation: `eval(zk)`

Achtung: beides funktioniert nicht bei allen Datentypen. Bei Schlangen kann ich bspw. die enthaltenen Objekte in einer Datei speichern und die Schlange beim Laden wieder neu aufbauen.

CSV-Dateien

csv-Dateien sind besondere Textdateien. Das Kürzel csv steht für comma-separated values. Dateien dieses Typs können zum Austausch einfach strukturierter Daten (z.B. in Tabellenform) verwendet werden.

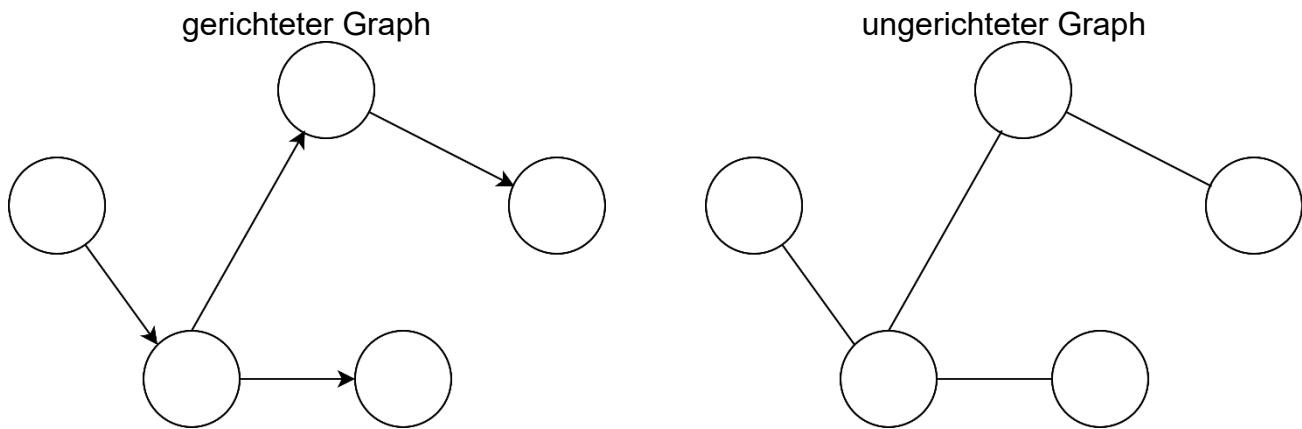
1.2.4. Bäume

Graphen

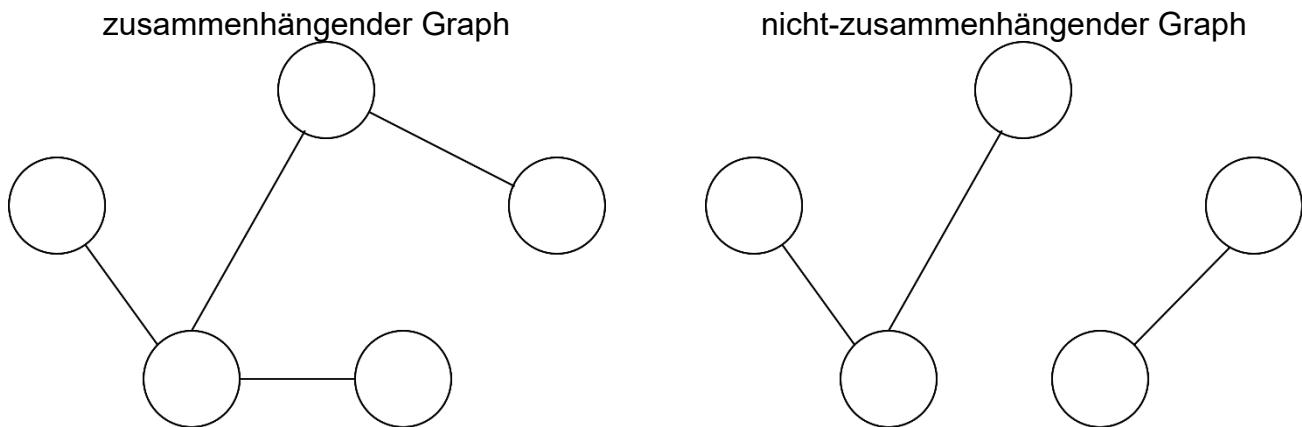
Für den Begriff des **Graphs** gibt es in der Mathematik unterschiedliche Definitionen je nachdem, was mit dem Graph gemacht werden soll. Alle Definitionen eint das Vorhandensein zweier Mengen: einer Menge V der Knoten (engl.: vertex) und eine Menge E der Kanten (engl.: edges).

Graphen spielen in der Informatik ebenfalls eine wichtige Rolle, etwa bei der Ermittlung des kürzesten Weges innerhalb eines Netzes (Rechnernetzes oder auch Streckennetzes), im Bereich der künstlichen Intelligenz oder auch bei der Darstellung von Ordnern als Dateibaum.

Ein Graph heißt **gerichtet**, wenn bei jeder Kante die Richtung angegeben ist. Im Diagramm wird das als Pfeil gekennzeichnet.

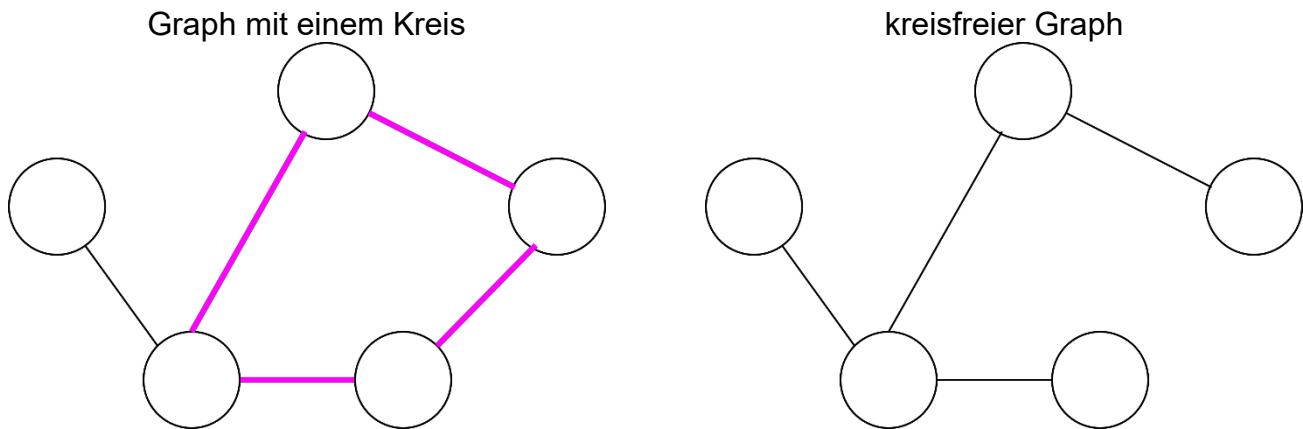


Ein Graph heißt **zusammenhängend**, wenn es zwischen jedem Paar an Knoten unter Nutzung von (ggf. mehreren) Kanten einen Weg gibt.



1.2. Datentypen

Ein Graph heißt **Kreis**, wenn von einem Knoten ausgehend eine Folge aus paarweise verschiedenen Kanten so existiert, dass der Knoten wieder erreicht wird. Ein Graph heißt **kreisfrei**, wenn kein Kreis in ihm existiert.



Ein Graph wird als **gewichteter Graph** bezeichnet, wenn jeder Kante ein Kantengewicht (i.d.R. eine nichtnegative rationale Zahl) zugewiesen wird. Gewichtete Graphen werden z.B. zur Suche kürzester Wege bei der Navigationssoftware verwendet.

Darstellung von Graphen

Beispiel:

$$G = (V, E) \text{ mit } V = \{a; b; c; d\} \text{ und } E = \{aa; ab; ac; bd; bc; cd; dd\}$$

Adjazenzliste

Für jeden Knoten des Graphs wird eine Liste allerjenigen Knoten gebildet, die mindestens eine Kante mit dem Knoten gemeinsam haben.

$$[(a|[a; b; c]); (b|[a; c; d]); (c|[a; b; d]); (d|[b; c; d])]$$

Adjazenzmatrix

Es wird eine Matrix (rechteckiges Zahlenschema) erstellt, die in der Zelle, die aus Zeile i und Spalte j angibt, wie viele Kanten zwischen dem Knoten i und Knoten j existieren. Bei ungerichteten Graphen entsteht dabei eine symmetrische Matrix.

	a	b	c	d
a	1	1	1	0
b	1	0	1	1
c	1	1	0	1
d	0	1	1	1

Inzidenzmatrix

Es wird eine Matrix erstellt, die in der Zelle, die aus Zeile i und Spalte j angibt, ob der Knoten i an der Kante j beteiligt ist. Bei gewichteten Graphen wird das Kantengewicht anstelle der üblichen Zahl 1 eingetragen.

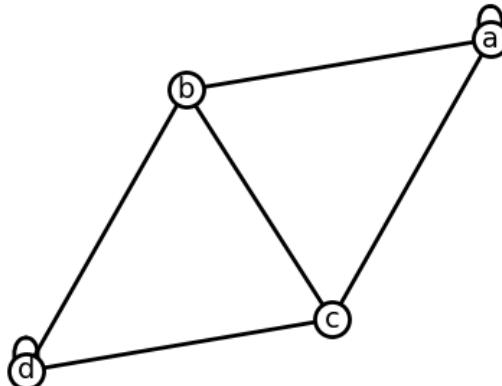
$G = (V, E)$ mit $V = \{a; b; c; d\}$ und $E = \{aa; ab; ac; bd; bc; cd; dd\}$

	aa	ab	ac	bd	bc	cd	dd	
a	1	1	1	0	0	0	0	
b	0	1	0	1	1	0	0	
c	0	0	1	0	1	1	0	
d	0	0	0	1	0	1	1	

Bemerkung: je nach genutzter Definition der Inzidenzmatrix kann bei einer Schleife (also z.B. der Kante aa) auch eine 2 notiert werden.

Darstellung als Graphendiagramm

Jeder Knoten wird üblicherweise als Kreis, jede Kante als geradlinige Verbindung zwischen den Kreisen dargestellt. Es wird üblicherweise versucht, den Graph so darzustellen, dass sich keine Kanten überschneiden. Für Fans: ist eine solche Darstellung möglich, nennt man den Graph **planar**.



Mögliche Darstellung des Diagramms in Python mit networkx

```
import networkx
import matplotlib.pyplot as plt

def main():
    # Graph erzeugen
    G = networkx.Graph()
    V = ["a", "b", "c", "d"]
    for v in V:
        G.add_node(v)
    E = [("a", "a"), ("a", "b"), ("a", "c"), ("b", "d"), ("b", "c"), ("c", "d"), ("d", "d")]
    for (v1,v2) in E:
        G.add_edge(v1,v2)
```

1.2. Datentypen

```
# Knoten, Kanten und Adjazenzlisten ausgeben
print(G.nodes())
print(G.edges())
print(G.adj)

# Graph zeichnen
# pos = {1: (x1,y1), 2: (x2,y2), ...}
options = {
    "font_size": 14,
    "node_size": 300,
    "node_color": "white",
    "edgecolors": "black",
    "linewidths": 2,
    "width": 2
}
#networkx.draw_networkx(G, pos, **options)
networkx.draw_networkx(G, **options)
ax = plt.gca()
ax.margins(0.20)
plt.axis("off")
plt.show()

if __name__ == '__main__':
    main()
```

Auch mit Matrizen kann in Python gerechnet werden. Es empfiehlt sich dabei das Modul numpy zu verwenden. In den folgenden Kästen werden unterschiedliche mögliche Umsetzungen der Matrizen angegeben.

Lösung ohne numpy

```
def add_matrices(A, B):
    """
    Addition der Matrizen A und B
    """
    return [[A[i][j] + B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

def multiply_matrices(A, B):
    """
    Multiplikation der Matrizen A und B
    """
    result = [[0] * len(B[0]) for _ in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                result[i][j] += A[i][k] * B[k][j]
    return result
```

1.2. Datentypen

```
def transpose_matrix(A):
    """
    Transponieren der Matrix A
    """
    return [[A[j][i] for j in range(len(A))] for i in range(len(A[0]))]

def determinant_2x2(A):
    """
    Determinante der 2x2-Matrix A
    """
    return A[0][0] * A[1][1] - A[0][1] * A[1][0]

def main():
    A = [[1,2],[3,4]]          # Angabe zeilenweise
    B = [[5,6],[7,8]]
    print(f"Matrix A: {A}")
    print(f"Matrix B: {B}")

    Summe = add_matrices(A,B)
    Produkt = multiply_matrices(A,B)
    Atrans = transpose_matrix(A)
    detB = determinant_2x2(B)
    print(f"Summe von A und B: {Summe}")
    print(f"Produkt A*B: {Produkt}")
    print(f"A transponiert: {Atrans}")
    print(f"det(B): {detB}")

if __name__ == '__main__':
    main()
```

Lösung mit numpy

```
import numpy as np

def main():
    A = np.array([[1,2],[3,4]])
    B = np.array([[5,6],[7,8]])
    print(f"Matrix A: {A}")
    print(f"Matrix B: {B}")

    Summe = A+B
    Produkt = np.dot(A,B)    # oder auch A @ B
    Atrans = A.T
    detB = np.linalg.det(B)
    print(f"Summe von A und B: {Summe}")
    print(f"Produkt A*B: {Produkt}")
    print(f"A transponiert: {Atrans}")
    print(f"det(B): {detB}")

if __name__ == '__main__':
    main()
```

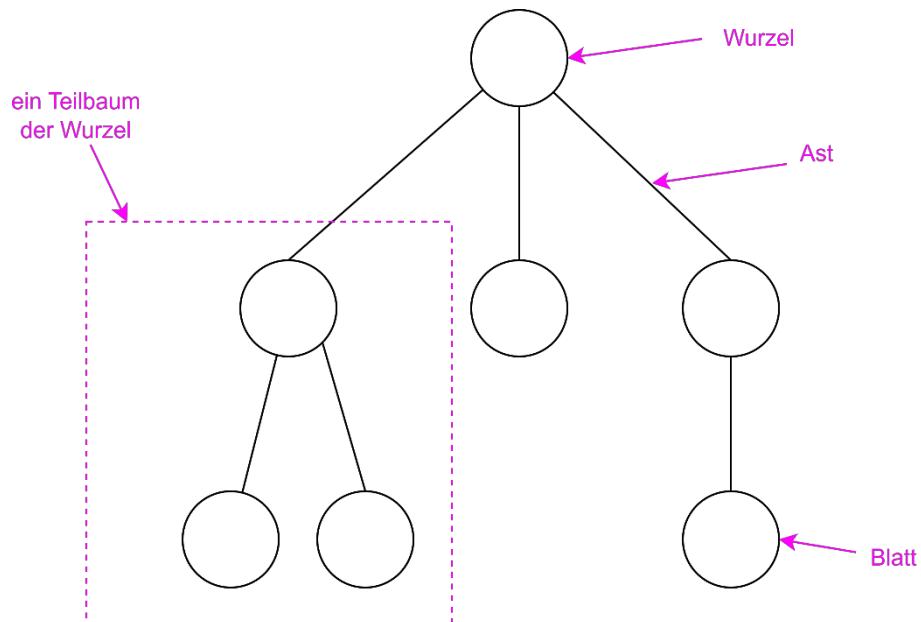
Baum

Ein Graph heißt **Baum**, wenn er zusammenhängend und kreisfrei ist.

Bäume werden in der Informatik auch als Datentyp verwendet. Dabei existiert ein ausgezeichneter Knoten, der als **Wurzel** des Baums bezeichnet wird. Die Kanten eines Baums werden in der Informatik auch als **Äste** bezeichnet.

Der Begriff des Baums kann dann rekursiv definiert werden: ein Baum ist entweder leer oder er besteht aus einer Wurzel und einer Anzahl an Teilbäumen, die mit ihm über einen Ast verbunden sind.

Bäume, die nur aus einer Wurzel bestehen, werden als **Blätter** bezeichnet.



Der abgebildete Baum besitzt sieben Knoten, sechs Äste und vier Blätter.

1.2. Datentypen

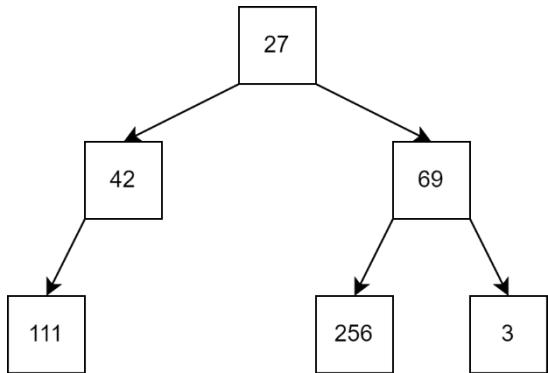
Ein Baum heißt **binärer Baum**, wenn jeder Knoten maximal zwei Teilbäume besitzt.

Implementierung in Python

Die einfachste Möglichkeit einen binären Baum in Python umzusetzen, ist die Verwendung der folgenden selbstgeschriebenen Klasse (mehr zu Klassen: siehe 2.3.).

```
class BinBaum:  
    def __init__(self,value=None):  
        self.left = None  
        self.right = None  
        self.value = value
```

Beispiel



```
# Wurzeln erzeugen  
Baum1 = BinBaum.BinBaum(27)  
Baum2 = BinBaum.BinBaum(42)  
Baum3 = BinBaum.BinBaum(69)  
Baum4 = BinBaum.BinBaum(111)  
Baum5 = BinBaum.BinBaum(256)  
Baum6 = BinBaum.BinBaum(3)  
  
# Zusammenhänge einarbeiten  
Baum1.left = Baum2  
Baum1.right = Baum3  
Baum2.left = Baum4  
Baum3.left = Baum5  
Baum3.right = Baum6
```

Bemerkung zur Darstellung: die Knoten können wie hier gezeigt auch als Rechtecke dargestellt werden. Auf die Kennzeichnung der Pfeilrichtung kann verzichtet werden solange die Wurzel des Baums erkennbar wird.

1.3. Unterprogramme

Definition

Als **Unterprogramm** bezeichnen wir eine eigenständige Programmeinheit zur Lösung eines Teilproblems. Ein Unterprogramm fasst die Anweisungen zu einer neuen Einheit zusammen, die mit einem Namen aufgerufen wird.

Warum?

- Vermeidung von Dopplungen innerhalb des Codes
- Heuristik „Divide and Conquer“ (siehe 1.5.) → möglichst effiziente Lösung des Teilproblems mit denen der anderen Teilprobleme zusammenführen
- Erhöhen der Übersichtlichkeit im Programm

Arten

Funktion

Bei dieser Art von Methoden muss stets mindestens ein Rückgabewert erzeugt werden.

Prozedur

Bei dieser Art von Methoden wird kein (direkter) Rückgabewert erzeugt.

Beispiele

Funktion	Prozedur
<pre># Implementierung der Methode def Summe(a, b): return a + b # Aufruf im Programm print(Summe(19, 9))</pre>	<pre># Implementierung der Methode def Anzeige(n, text): for i in range(n): print(text) # Aufruf im Programm Anzeige(3, "Python")</pre>

Parameterart

- **Werteparameter (Call-by-Value)**: es wird eine **Kopie** der zugehörigen Variable übermittelt; Effekt: bei Änderungen innerhalb des Unterprogramms wird der vorherige Wert der Variable im aufrufenden Programmabschnitt **nicht** geändert
- **Referenzparameter (Call-by-Reference)**: es wird die **Speicheradresse** der zugehörigen Variable übermittelt; Effekt: wird innerhalb des Unterprogramms der Variablenwert verändert, so gilt dieser veränderte Wert **auch** nach Abarbeitung des Unterprogramms im aufrufenden Programmabschnitt

Python verwendet aufgrund seines Variablenkonzepts Referenzparameter.

Formale und aktuelle Parameter

Beispiel

```
def Anzeige(n, text):
    for i in range(n):
        print(text)

Anzeige(3, "Python")
```

Bei der Implementierung der Methode (= des Unterprogramms) werden die Parameter n und text genutzt. Sie werden als **formale Parameter** bezeichnet.

Wird die Methode aufgerufen, dann ersetzen die in der Klammer an der passenden Stelle angegebenen Werte diese formalen Parameter. Die ersetzenden Werte werden von uns als **aktuelle Parameter** bezeichnet.

Globale und lokale Variablen

Sollen globale, also während des gesamten Moduls geltende, Variablen innerhalb einer Methode genutzt werden, muss dies in Python mit dem Schlüsselwort `global` angekündigt werden.

Beispiel

```
def Testmethode(a):
    global b
    b = 2*a

b = 27
Testmethode(22)
print(b)
```

In der Prozedur `Testmethode` wird die global existierende Variable `b` anstelle einer neuen lokalen Variable `b` verwendet. Dies führt dazu, dass dieses Python-Skript am Ende folgende Ausgabe erzeugt: 44

Unterstützung des Programmierers

Im Kopf der Methode kann angegeben werden, welche Variablentypen die formalen Parameter haben sollen. Dies dient als Empfehlung und Orientierung für den Programmierer und führt **nicht** automatisch zu einer Fehlermeldung im Debugger, wenn dagegen verstößen wird.

Beispiel

```
def Test(a: int, b: float) -> str:
```

1.3. Unterprogramme

Verpflichtende Vorgabe für Schüler des Leistungskurses

Durch mehrzeilige Kommentare muss für jede selbstgeschriebene Methode mindestens angegeben werden, was die Methode bewirkt (Ausnahmen folgen in 2.3.).

```
def Test(a: int, b: float) -> str:  
    """  
        Die Met (a: int, b: float) -> str  
    """  
    return Die Methode berechnet das Produkt aus a und b und gibt dies als  
    Zeichenkette zurück.  
print(Test())
```

1.4. Komplexität

Effizienz von Algorithmen

Die Laufzeit, die ein Verfahren auf einem Rechner benötigt, hängt von verschiedenen Parametern ab. Um die Effizienz eines Verfahrens zu beurteilen, versuchen wir, die rechnerabhängigen Größen (etwa Anzahl der Prozessorkerne, Arbeitsspeicher, ...) zu vernachlässigen.

Als **Laufzeitkomplexität** betrachten wir die Anzahl der Basisoperationen in Abhängigkeit der Eingabegröße.

Bei Sortierverfahren und Suchverfahren verstehen wir die Anzahl der vorhandenen Objekte, die sortiert bzw. durchsucht werden, als Eingabegröße.

Was zu den Basisoperationen zählt, unterscheidet sich je nach Quelle.

Möglich wären:

- Anzahl an arithmetischen Operationen (Addition, Subtraktion, ...)
- Anzahl an logischen Operationen (AND, OR, NOT, ...)
- Anzahl an Vergleichen (if ..., for ..., while ...)
- Anzahl an Zuweisungen

Landau-Notation

Zum besseren Vergleichen der Laufzeitkomplexität von Verfahren bietet es sich an, Klassen von Verfahren ähnlicher Laufzeit zu bilden. Dafür verwenden wir die Landau-Notation, benannt nach Edmund Landau.

Es gilt: $g \in \mathcal{O}(f) :\Leftrightarrow \exists n_0 > 0 \exists c > 0 \forall n \geq n_0: g(n) \leq c \cdot f(n)$

Achtung:

- die Landau-Notation betrachtet asymptotisches Verhalten für große Werte von n
- die rechnerabhängige Konstante c kann sehr groß sein
- der Indexwert n_0 ab dem die Folgeglieder von $c \cdot f(n)$ beschränkt werden kann sehr groß sein

Komplexitätsklassen

Wir sollten die folgenden Komplexitätsklassen kennen.

Landau-N.	Name	Beispiel(e)
$\mathcal{O}(1)$	konstant	Prüfung einer Binärzahl der Länge n auf Teilbarkeit durch 2
$\mathcal{O}(\log n)$	logarithmisch	Binäre Suche im sortierten Feld der Länge n
$\mathcal{O}(n)$	linear	Lineare Suche im unsortierten Feld der Länge n
$\mathcal{O}(n \log n)$	superlinear	Merge-Sort, Quick-Sort für Feld der Länge n
$\mathcal{O}(n^2)$	quadratisch	Bubble-Sort, Selection-Sort, Insertion-Sort für Feld der Länge n
$\mathcal{O}(n^3)$	kubisch	Multiplikation zweier Matrizen des Formats $n \times n$
$\mathcal{O}(n^k)$	polynomial	Finden der kürzesten Route nach Dijkstra-Verfahren
$\mathcal{O}(2^n)$	exponentiell	Optimierungsprobleme (z.B. Suche nach einem optimalen Stundenplan für das JKG)

Zeitmessung in Python

Für die Zeitmessung kann in Python das Modul `time` eingebunden werden.

Die Methode `time()` liefert dann die seit 01.01.1970 00:00:00 UTC abgelaufene Anzahl an Sekunden (als rationale Zahl).

Durch Regression z.B. in einer Tabellenkalkulation kann anhand verschiedener Messwerte experimentell/empirisch versucht werden, die Laufzeitkomplexität eines Verfahrens zu ermitteln.

Theoretische Ermittlung

Für die Komplexität gilt:

- **Summenregel:** Die Komplexität einer Sequenz von Befehlen ist die Summe der Komplexitäten der Befehle (dann größte Schranke nutzen).
Bsp.: $\mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(n) = \mathcal{O}(1 + 1 + n) = \mathcal{O}(n)$
- **Produktregel:** Die Komplexität einer Schleife ist das Produkt aus der Komplexität des Schleifenkörpers mit der Anzahl der Schleifendurchläufe.
- **Mastertheorem:** bei rekursiven Verfahren, welche die **Rekurrenzgleichung** $T(n) = a \cdot T\left(\frac{n}{b}\right) + g(n)$ mit $a \geq 1$ zu lösenden Teilproblemen und Zerlegung in $b > 1$ Teilprobleme sowie Kosten für Zerlegung und Zusammensetzung der Teillösungen $g(n) = \mathcal{O}(n^k)$ erfüllen, gilt

$$T(n) = \begin{cases} \mathcal{O}(n^k) & \text{falls } a < b^k \\ \mathcal{O}(n^k \cdot \log n) & \text{falls } a = b^k \\ \mathcal{O}(n^{\log_b a}) & \text{falls } a > b^k \end{cases}$$

Bsp.: Quick-Sort $a = 2, b = 2, g(n) = \mathcal{O}(n) \Rightarrow a = b^1 \Rightarrow T(n) = \mathcal{O}(n \cdot \log n)$

Binäre Suche $a = 1, b = 2, g(n) = \mathcal{O}(1) \Rightarrow a = b^0 \Rightarrow T(n) = \mathcal{O}(\log n)$

Bemerkungen: Wir verwenden eine Vereinfachung des Mastertheorems. Genau genommen ist zwischen \mathcal{O}, Θ und Ω in ihrer Landau-Notation zu unterscheiden. Während Rekurrenzgleichungen auch die Anfangsbedingung betrachten, wird bei Rekurrenzgleichungen nur auf den rekursiven Aufruf zur Berechnung fokussiert.

1.5. Heuristiken

1.5.1. Rekursion

Erinnerung Mathematik: Eine Bildungsvorschrift einer Zahlenfolge wird rekursiv genannt, wenn die Berechnung eines Folgegliedes auf vorherige Folgeglieder zurückgeführt wird.

Beispiel: Die Folge der Fibonacci-Zahlen kann über die Rekursionsformel $f_n = f_{n-1} + f_{n-2}$ errechnet werden. So gilt bspw. $f_2 = f_1 + f_0$ oder auch $f_{27} = f_{26} + f_{25}$. Zur Berechnung benötigen wir dann noch die Abbruchbedingung(en), hier $f_0 = 0, f_1 = 1$.

Übertragen auf Methoden:

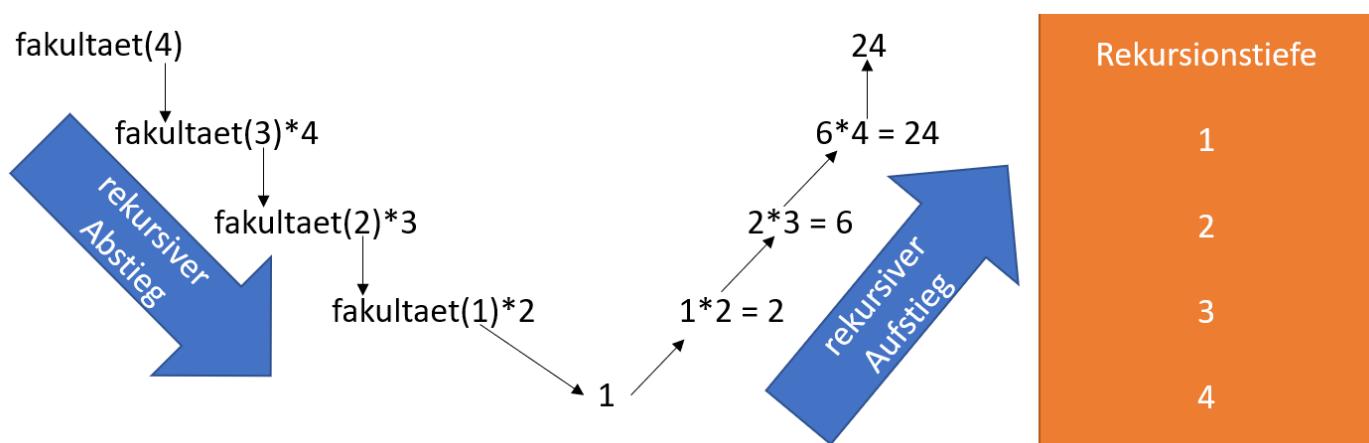
Wir nennen eine Methode **rekursiv**, wenn sie sich (direkt oder indirekt) selbst aufruft.

Beispiel

```
def fakultaet(n: int) -> int:
    if n == 1:
        return 1
    else:
        return fakultaet(n-1)*n
```

Im Sonst-Zweig der hier enthaltenen zweiseitigen Verzweigung ruft sich die Funktion fakultaet selbst auf. Durch die Reduktion von n um 1 verringern wir den betrachteten Wert bei jedem Schritt, sodass für positive natürliche Zahlen die Abarbeitung ein Ende finden wird.

Rekursive Lösungen zu finden ist häufig einfacher als iterative (im Beispiel also etwa eine Formel, die bei Eingabe von n das Ergebnis ohne Berechnung vorheriger Folgeglieder ermittelt), jedoch rechnerisch im Regelfall wesentlich ineffizienter (es müssen Zwischenschritte und derzeit geltende Parameterbelegungen zwischengespeichert werden – das wird schnell aufwendig).



In Python kann die maximale Rekursionstiefe im Modul sys über die Methoden `sys.getrecursionlimit()` und `sys.setrecursionlimit(n)` abgefragt und geändert werden.

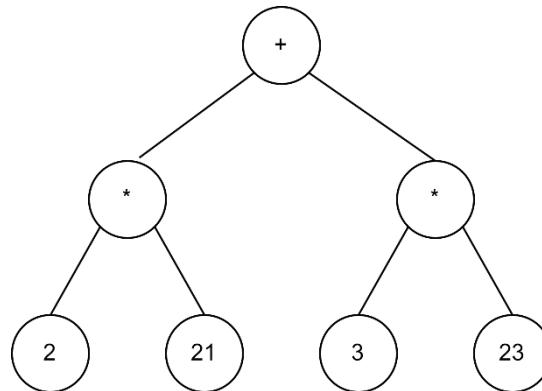
1.5.2. Traversierungen

Als **Traversierung** bezeichnen wir ein Verfahren, bei dem jeder Knoten eines Graphen besucht wird. Ziel dabei ist es, jeden Knoten und möglichst jede Kante nur genau einmal zu besuchen.

Für allgemeine Bäume gibt es Breiten- und Tiefensuche. Diese werden in 1.5.4. näher beschrieben.

Für binäre Bäume gibt es spezielle Arten der Traversierung: Preorder, Inorder und Postorder.

Beispiel



Bei der **Preorder-Traversierung** wird zuerst die Wurzel notiert, dann der linke Teilbaum des binären Baums, dann der rechte Teilbaum.

Im Beispiel ergibt sich die folgende Rückgabe: + * 2 21 * 3 23

Bei der **Inorder-Traversierung** wird zuerst der linke Teilbaum ausgewertet, dann die Wurzel notiert, dann der rechte Teilbaum.

Im Beispiel ergibt sich die folgende Rückgabe: 2 * 21 + 3 * 23

Die Inorder-Traversierung ähnelt in der Nutzung für Rechenbäume der für uns gewöhnlichen Notation von Rechentermen.

Bei der **Postorder-Traversierung** wird zuerst der linke Teilbaum, dann der rechte Teilbaum ausgewertet und dann die Wurzel notiert.

Im Beispiel ergibt sich die folgende Rückgabe: 2 21 * 3 23 * +

Die Postorder-Traversierung für Rechenbäume wurde früher bei Taschenrechnern verwendet, da sie ohne Klammern und Vorrangautomatik auskommt. Die Abarbeitung eines solchen Ausdrucks ist unter Nutzung eines Stapels sehr gut machbar.

Offensichtlich sind diese drei Arten der Traversierung binärer Bäume gut durch Rekursion umsetzbar.

1.5.3. Divide and Conquer und Sortierverfahren

Divide and Conquer-Verfahren bezeichnen Verfahren, bei denen das Problem in Teilprobleme zerlegt wird, bis diese gelöst sind. Aus den Teillösungen wird dann eine Lösung für das Gesamtproblem erzeugt.

Beispiele

- Fallunterscheidungen beim Lösen mathematischer Gleichungen (z.B. Betragsgleichungen oder quadratische Gleichungen)
- Sortierverfahren Quick-Sort und Merge-Sort
- Binäre Suche in einem Feld

Sortierverfahren allgemein

Gerade in Zeiten von Big Data, wo viele Daten erhoben und ausgewertet werden, ist die Fähigkeit essenziell die vorhandenen Daten effizient verarbeiten zu können. Dazu gehört das Suchen nach gewissen Mustern ebenso wie das Sortieren anhand gegebener Kriterien.

Im Rahmen des schulischen Kontexts können wir davon ausgehen, dass wir die zu sortierenden Daten allesamt in einem Feld gleichartiger Datentypen vorliegen haben und wir diese nach einem Attribut, das wir **Sortierschlüssel** nennen wollen, auf- oder absteigend sortieren sollen.

Beispiel: Sortieren aller Kursmitglieder unseres Kurses aufsteigend nach Abstand zum nächsten Geburtstag oder nach ihrer Körperlänge

Für Fans mathematischer Notationen:

geg.: n Objekte a_1, \dots, a_n mit ihren Schlüsselwerten k_1, \dots, k_n anhand derer die Objekte sortiert werden sollen

ges.: Eine Anordnung $(a_{i1}| \dots |a_{in})$ so, dass $(i1| \dots |in)$ eine Permutation von $(1| \dots |n)$ ist und $k_{i1} \leq k_{i2} \leq \dots \leq k_{in}$ (bzw. alles \geq) gilt.

Stabilität eines Sortierverfahrens

Angenommen, wir sortieren die folgende Liste aufsteigend nach der notierten Zahl.

$[(\text{Franziska}|9), (\text{Dean}|4), (\text{Antonio}|7), (\text{Vlad}|4), (\text{Kevin}|5), (\text{Johanna}|7)]$

Wir nennen ein Sortierverfahren **stabil**, wenn es die Reihenfolge von Elementen mit gleichen Schlüsselwerten stets beibehält. Bezogen auf unsere Liste heißt das: Nach dem Sortieren muss das Element $(\text{Dean}|4)$ vor $(\text{Vlad}|4)$ und das Element $(\text{Antonio}|7)$ vor $(\text{Johanna}|7)$ bleiben, wenn das Verfahren stabil sein will.

Besitzt das Verfahren diese Eigenschaft nicht, so nennen wir es **instabil**.

In situ / ex situ

Wir bezeichnen ein Sortierverfahren als **in situ-Verfahren**, wenn kein weiterer Speicherplatz zusätzlich zum zu sortierenden Feld benötigt wird (wobei davon ausgegangen wird, dass die zu sortierenden Feldelemente die Indizes 1 bis n belegen und das Feldelement mit Index 0 für Vertauschungen genutzt werden kann).

Besitzt das Sortierverfahren diese Eigenschaft nicht, so nennen wir es **ex situ-Verfahren**.

Ausgewählte Sortierverfahren

Es gibt eine große Menge unterschiedlicher Sortierverfahren. Beispiele sind

- Bubble-Sort
- Insertion-Sort
- Selection-Sort
- Quick-Sort
- Merge-Sort
- Heap-Sort
- Random-Sort

Bubble-Sort

Grundprinzip: vertausche die relative Reihenfolge **benachbarter** Elemente, so dass kleine Elemente nach vorne und größere Elemente nach hinten wandern.

Namensgebend: größere (Luft-)Blasen steigen in einem Getränk schneller nach oben als kleinere

Alternativer Name: Sortieren durch (Nachbar-)Vertauschen

Beispielablauf

	Erster Durchlauf	Zweiter Durchlauf	Dritter Durchlauf	Vierter Durchlauf
Ausgangsfeld	[57, 11, 93, 12, 44]	[11, 57, 12, 44, 93]	[11, 12, 44, 57, 93]	[11, 12, 44, 57, 93]
Einzelne Schritte (Vergleiche)	[11, 57, 93, 12, 44]	[11, 57, 12, 44, 93]	[11, 12, 44, 57, 93]	
	[11, 57, 93, 12, 44]	[11, 12, 57, 44, 93]		
	[11, 57, 12, 93, 44]			
Ergebnisfeld	[11, 57, 12, 44, 93]	[11, 12, 44, 57, 93]	[11, 12, 44, 57, 93]	[11, 12, 44, 57, 93]

Insertion-Sort

Grundprinzip: ein bereits vorsortiertes (Teil-)Feld wird in jedem Durchlauf um ein Element erweitert. Dieses wird durch Vergleiche und Vertauschungen an die korrekte Stelle eingefügt

Namensgebend: in jedem Schritt wird ein Element in eine bereits sortierte Gruppe von Elementen eingefügt

Alternativer Name: Sortieren durch Einfügen

Beispielablauf

	Erster Durchlauf	Zweiter Durchlauf	Dritter Durchlauf	Vierter Durchlauf
Ausgangsfeld	[57, 11, 93, 12, 44]	[11, 57, 12, 44, 93]	[11, 12, 57, 44, 93]	[11, 12, 44, 57, 93]
Einzelne Schritte (vorsortiertes Feld, betrachtetes Element, Vergleich)	[57, 11, 93, 12, 44]	[11, 57, 12, 44, 93] h=12	[11, 12, 57, 44, 93] h=44	[11, 12, 44, 57, 93] h=93
		[11, 57, 57, 44, 93] h=12	[11, 12, 57, 57, 93] h=44	
Ergebnisfeld	[11, 57, 12, 44, 93]	[11, 12, 57, 44, 93]	[11, 12, 44, 57, 93]	[11, 12, 44, 57, 93]

Selection-Sort

Grundprinzip: das größte/kleinste Element des noch unsortierten Teilstücks wird ausgewählt und mit dem Schluss/Anfang dieses Teilstücks vertauscht

Namensgebend: in jedem Schritt wird das größte/kleinste Element ausgewählt

Alternativer Name: Sortieren durch Auswählen

Beispielablauf

	Erster Durchlauf	Zweiter Durchlauf	Dritter Durchlauf	Vierter Durchlauf
Ausgangsfeld	[57, 11, 93, 12, 44]	[57, 11, 44, 12, 93]	[12, 11, 44, 57, 93]	[12, 11, 44, 57, 93]
Einzelne Schritte (vorsortiertes Feld, Vergleiche)	[57, 11, 93, 12, 44] s=1	[57, 11, 44, 12, 93] s=1	[12, 11, 44, 57, 93] s=1	[12, 11, 44, 57, 93] s=1
	[57, 11, 93, 12, 44] s=3	[57, 11, 44, 12, 93] s=1	[12, 11, 44, 57, 93] s=3	
	[57, 11, 93, 12, 44] s=3	[57, 11, 44, 12, 93] s=1		
	[57, 11, 93, 12, 44] s=3			
Ergebnisfeld	[57, 11, 44, 12, 93]	[12, 11, 44, 57, 93]	[12, 11, 44, 57, 93]	[11, 12, 44, 57, 93]

Quick-Sort

Grundprinzip: das Ausgangsfeld wird anhand eines Schlüsselementes („**Pivotelement**“) in zwei Teilstäler aufgeteilt so, dass das eine Teilstäler genau diejenigen Elemente enthält, die kleiner-gleich dem Schlüsselement sind. Auf die Teilstäler wird dann Quick-Sort (rekursiv) angewendet

Namensgebend: Verfahren zählt zu den schnelleren Verfahren

Alternativer Name: Sortieren durch Partitionierung

Beispielablauf

Ausgangsfeld: [57, 11, 93, 12, 44]

Wahl des Pivotelements 57: [57, 11, 93, 12, 44] → [57, 11, 12, 44] und [93]

 Wahl des Pivotelements 57: [57, 11, 12, 44] → [11, 12, 44] und [57]

 Wahl des Pivotelements 11: [11, 12, 44] → [11] und [12, 44]

 Wahl des Pivotelements 12: [12, 44] → [12] und [44]

 Teilergebnisfeld [12, 44]

 Teilergebnisfeld [11, 12, 44]

 Teilergebnisfeld [11, 12, 44, 57]

Ergebnisfeld: [11, 12, 44, 57, 93]

Merge-Sort

Grundprinzip: das Ausgangsfeld wird solange in je zwei Teilstäler aufgeteilt, bis diese kleinstmöglich sind. Beim Zusammenführen („**mergen**“) zweier bereits vorsortierter Teilstäler werden diese der Größe nach sortiert.

Namensgebend: wesentlicher Teil des Verfahrens ist das Verschmelzen der Teilstäler

Beispielablauf

Ausgangsfeld: [57, 11, 93, 12, 44]

Aufteilung in [57, 11, 93] und [12, 44] – Anwenden von Merge-Sort auf die Teilstäler

 Aufteilung von [57, 11, 93] in [57, 11] und [93]

 Aufteilung von [57, 11] in [57] und [11]

 Verschmelzen von [57] und [11] zu [11, 57]

 Verschmelzen von [11, 57] und [93] zu [11, 57, 93]

 Aufteilung von [12, 44] in [12] und [44]

 Verschmelzen von [12] und [44] zu [12, 44]

Verschmelzen von [11, 57, 93] und [12, 44] zu [11, 12, 44, 57, 93]

Ergebnisfeld: [11, 12, 44, 57, 93]

1.5.4. Backtracking

Backtracking ist ein Problemlöseverfahren, welches das Versuch-und-Irrtum-Prinzip nach dem Ansatz „wenn möglich vorwärts, sonst rückwärts“ anwendet.

Voraussetzungen

- Problem lässt sich durch eine Folge von Entscheidungen lösen
- immer nur endlich viele Entscheidungen zur Wahl
- es ist erkennbar, ob eine mögliche Entscheidung falsch ist

Genereller Ansatz

Solange Lösung noch nicht gefunden ist:

falls es noch nicht probierte Entscheidungen gibt:

falls die Entscheidung nicht erkennbar falsch ist:

verlängere die Entscheidungsfolge um die Entscheidung

sonst:

entferne die letzte Entscheidung aus der Entscheidungsfolge

Beispiele

- Labyrinth
 - Entscheidung: welche Richtung nutzen bei Weggabelung
 - Entscheidungsfolge: Richtungsentscheidungen an Gabelungen
 - erkennbar falsch: ein bereits begangenes Feld wird gewählt oder Weg führt in Sackgasse
 - Lösung gefunden: aktuelles Feld ist Ausgang
- Sudoku
 - Entscheidung: welche Ziffer wird in aktuelles Feld eingetragen
 - Entscheidungsfolge: Folge aus Tripeln von Feldkoordinaten und eingetragener Ziffer
 - erkennbar falsch: Verstoß gegen Sudokuregeln (Ziffer mehrfach in Zeile, Spalte oder Block)
 - Lösung gefunden: kein offenes Feld und kein Verstoß gegen Sudokuregeln

Tiefen- und Breitensuche in Bäumen

Neben den Traversierungsarten Preorder, Inorder und Postorder für binäre Bäume gibt es weitere Traversierungsarten für binäre und allgemeine Bäume. Dazu zählen Tiefen- und Breitensuche.

Tiefensuche

Im Baum wird zuerst jeder Pfad vollständig in die Tiefe beschritten, bevor abzweigende Pfade beschritten werden. Das kann durch Backtracking realisiert werden.

Breitensuche

Im Baum werden zuerst alle Knoten derselben Tiefe besucht, ehe deren Tochternknoten besucht werden. Dies ist umsetzbar, indem man mit der Wurzel startend erst alle Tochternknoten in eine Schlange einfügt und dann stets das erste Element der Schlange besucht.

1.5.5. Suchverfahren

Als **Suchverfahren** bezeichnen wir ein Verfahren, dass in einer gegebenen Datenstruktur prüft, ob ein gegebenes Objekt in dieser Struktur enthalten ist (und ggf. den Speicherort bzw. Index zurückgibt).

In diesem Abschnitt beschränken wir unsere Suche auf Zahlen oder Zeichenketten in einem Feld.

Lineare Suche

Einfache Lösung: ich sehe mir alle Feldelemente mit steigendem Index an, ob sie mit dem gesuchten Objekt übereinstimmen.

Dabei ist einem Feld der Länge n im besten Fall nur ein Element zu überprüfen (wenn das erste Element direkt dem gesuchten Objekt entspricht), im schlimmsten Fall sind alle n Elemente zu überprüfen (wenn das gesuchte Element das letzte Element ist oder nicht enthalten ist). Die Komplexität dieses Verfahrens ist daher linear.

Mögliche Implementierung in Python

```
def lineare_suche(liste, objekt):
    """
    Führt eine lineare Suche in der Liste durch.

    :param liste: Liste von Zahlen, in der gesucht werden soll
    :param objekt: Das zu suchende Objekt
    :return: Index des Objekts, wenn gefunden, sonst -1
    """

    for index, element in enumerate(liste):
        if element == objekt:
            return index # Objekt gefunden, Index zurückgeben
    return -1 # Objekt nicht gefunden

def main():
    # Beispieldaten und Suchobjekt
    zahlen_liste = [13, 27, 42, 99, 111]
    zu_suchen = 69

    # Aufruf der linearen Suche
    ergebnis = lineare_suche(zahlen_liste, zu_suchen)

    # Ausgabe des Ergebnisses
    if ergebnis != -1:
        print(f"Das Objekt {zu_suchen} wurde an Position {ergebnis} gefunden.")
    else:
        print(f"Das Objekt {zu_suchen} wurde nicht in der Liste gefunden.")

if __name__ == '__main__':
    main()
```

Binäre Suche

Wenn das Feld bereits sortiert vorliegt, kann die aus Mathematik bekannte Intervallhalbierung als Idee genutzt werden. Ich vergleiche das gesuchte Objekt mit der Mitte des Felds. Ist das gesuchte Objekt kleiner, suche ich in der linken Teilhälfte des Feldes weiter, ist das gesuchte Objekt größer, so suche ich in der rechten Teilhälfte des Feldes weiter.

Beispiel – Suche nach 99 im Feld [13, 27, 42, 99, 111]:

- Betrachtetes Feld: [13, 27, 42, 99, 111] – Mitte: 42 – Vergleich $99 > 42 \rightarrow$ rechtes Teilstück
- Betrachtetes (Teil-)Feld: [99, 111] – Mitte: 99 – Vergleich $99 = 99 \rightarrow$ Objekt gefunden

Beispiel – Suche nach 27 im Feld [13, 27, 42, 99, 111]:

- Betrachtetes Feld: [13, 27, 42, 99, 111] – Mitte: 42 – Vergleich $27 < 42 \rightarrow$ linkes Teilstück
- Betrachtetes (Teil-)Feld: [13, 27] – Mitte: 13 – Vergleich $13 < 27 \rightarrow$ rechtes Teilstück
- Betrachtetes (Teil-)Feld: [27] – Mitte: 27 – Vergleich $27 = 27 \rightarrow$ Objekt gefunden

Beispiel – Suche nach 69 im Feld [13, 27, 42, 99, 111]:

- Betrachtetes Feld: [13, 27, 42, 99, 111] – Mitte 42 – Vergleich $69 > 42 \rightarrow$ rechtes Teilstück
- Betrachtetes (Teil-)Feld: [99, 111] – Mitte 99 – Vergleich $69 < 99 \rightarrow$ linkes Teilstück
- Betrachtetes (Teil-)Feld: [] \rightarrow Objekt nicht im Feld enthalten

Durch die Halbierung des Felds bei jedem Schritt ergibt sich (siehe Mastertheorem) eine logarithmische Laufzeit. Hierbei wird natürlich nicht beachtet, dass zum Sortieren des Feldes ebenfalls Zeit nötig ist, sondern davon ausgegangen, dass das Feld (z.B. beim Erstellen) bereits sortiert gespeichert vorliegt.

Mögliche Implementierung in Python

```
def binaere_suche(liste, objekt):
    """
    Führt eine binäre Suche in einer sortierten Liste durch.
    :param liste: Sortierte Liste von Zahlen, in der gesucht werden soll
    :param objekt: Das zu suchende Objekt
    :return: Index des Objekts, wenn gefunden, sonst -1
    """

    links = 0
    rechts = len(liste) - 1

    while links <= rechts:
        mitte = (links + rechts) // 2
        if liste[mitte] == objekt:
            return mitte # Objekt gefunden, Index zurückgeben
        elif liste[mitte] < objekt:
            links = mitte + 1 # Im rechten Teil weitersuchen
        else:
            rechts = mitte - 1 # Im linken Teil weitersuchen
    return -1 # Objekt nicht gefunden
```

Suche mit Hash-Tabellen

Die bereits gute Komplexität der Binären Suche kann ggf. noch verbessert werden durch Verwendung von Hash-Tabellen.

Eine **Hash-Funktion** weist jedem Objekt einen Wert fester Größe zu. Betrachten wir das am Beispiel der Suche nach Namen in einer Liste. Als Liste betrachten wir die Kursteilnehmer des Abijahrgangs 2026: [Armin, Daniel, Darius, Hannes, Hermann, Jakob, Jannik, Jonathan, Lorenz, Lukas, Martin, Quentin, Stefan].

Ich lege fest, dass es 16 mögliche Hash-Werte geben soll. Dies kann ich z.B. darüber erreichen, dass ich jedem Vornamen die Summe seiner ASCII-Werte zuweise und das Ergebnis modulo 16 berechne.

So erhalte ich z.B. für Jannik: $74 + 97 + 110 + 110 + 105 + 107 = 603$ und damit bei Division durch 16 den Rest 11. Damit wird Jannik die Zahl 11 zugewiesen.

Lukas erhält nach demselben Verfahren die Zahl 0, Stefan die Zahl 1 usw.

Anstelle eines Feldes speichere ich mir nun 16 mögliche Felder (je nachdem, welchen Rest das Ergebnis bei der Division durch 16 besitzt).

Suche ich nun ein Objekt im Feld, so berechne ich auch für dieses Element den Hash-Wert und suche dann nur in dem zu diesem Wert gehörenden Teifeld.

Achtung: nach Schubfachprinzip wird es spätestens beim 17ten Element zu einer Kollision kommen, d.h. zwei unterschiedliche Objekte erhalten denselben Hash-Wert. In der Kursliste besitzt z.B. auch Martin den Wert 11. Es gibt unterschiedliche Wege, mit Kollisionen umzugehen, die sich auf die Laufzeit unterschiedlich auswirken. Ich könnte z.B. Martin in das nächste „freie“ Teifeld legen, muss dann auf der Suche nach Martin aber nicht nur das Feld mit der 11, sondern auch darauffolgende Felder untersuchen. Es ist daher sinnvoll vorher darüber nachzudenken, in wie viele Teifelder ich das gesamte Feld aufteilen möchte. Ist die Anzahl zu klein, wird es schnell zu Kollisionen kommen, ist sie zu groß, halte ich mir unnötigerweise viele leere Felder im Speicher vor.

Hash-Verfahren werden wir in der Behandlung des Themas Kryptografie wiederfinden.

Für Interessierte: der obige Kurs würde sich wie folgt auf die 16 Teifelder verteilen.

```
[['Lukas'], ['Stefan'], [], ['Jonathan'], ['Quentin'], [], [], ['Jakob'],
['Armin'], ['Darius'], ['Hermann'], ['Lorenz'], ['Martin', 'Jannik'], [],
['Daniel', 'Hannes'], [], []]
```

Mögliche Implementierung in Python

```

def hashfkt(obj, anz):
    """
    Ermittelt den Hash-Wert des Objekts obj
    """

    summe = 0
    for i in obj:
        summe += ord(i)
    return summe%anz

def LineareSuche(feld, element):
    """
    Gibt aus, ob element im Objekt feld enthalten ist
    """

    for i in range(len(feld)):
        if feld[i]==element:
            return True
    return False

# Anzahl der gewünschten zur Verfügung stehenden Elemente im Feld
AnzElemente = int(input("Wie viele Spalten soll die Hash-Tabelle erhalten? "))

# Datensammlung als leere Liste vorbereiten mit passender Anzahl an Elementen
Datensammlung = []
for _ in range(AnzElemente):
    Datensammlung.append([])

# Datensammlung füllen als Import aus hashnamen.txt
datei = open("hashnamen.txt", "r")
for zeile in datei:
    Datensammlung[hashfkt(zeile.rstrip(), AnzElemente)].append(zeile.rstrip())
datei.close()

# Ausgabe der Datensammlung in der Hash-Tabelle
print(Datensammlung)

# Eingabe eines zu suchenden Elements
elt = input("Bitte geben Sie das zu suchende Objekt ein: ")
# Ausgabe des Hash-Werts dieses Elements
hwert = hashfkt(elt, AnzElemente)
print(elt + " besitzt den Hash-Wert "+str(hwert)+".")

# Anzeige der Elemente mit demselben Hash-Wert
print("Liste aller Elemente mit diesem Hash-Wert: "+str(Datensammlung[hwert]))
# Jetzt Lineare Suche in dieser Sammlung
if LineareSuche(Datensammlung[hwert], elt):
    print(elt + " ist in dieser Liste enthalten.")
else:
    print(elt + " ist in dieser Liste leider nicht enthalten.")

```

1.6. Grenzen der Algorithmierbarkeit

Grundsatzfrage

David Hilbert und Wilhelm Ackermann stellten sich 1928 im Werk „Grundzüge der theoretischen Logik“ die Frage, ob ein Algorithmus existiert, der entscheiden kann, ob eine mathematische Aussage beweisbar ist oder nicht.

Dies wird als **Entscheidungsproblem** bezeichnet und auf die Berechenbarkeit von Funktionen zurückgeführt.

Berechenbarkeit

Ansatz Kurt Gödel

Eine Funktion heißt berechenbar, wenn sie rekursiv ist.

Ansatz Alan Turing

Eine Funktion heißt berechenbar, wenn sie durch eine Turing-Maschine umgesetzt werden kann, die ihre Werte berechnet.

Ansatz Alonzo Church

Eine Funktion heißt berechenbar, wenn sie durch einen Lambda-Term ausgedrückt werden kann, der ihre Werte berechnet.

Erkenntnis

Alan Turing beweist durch die Nichtentscheidbarkeit des Halteproblems, dass die automatisierte Bestimmung logischer Feststellungen nicht möglich ist.

Das **Halteproblem** fragt dabei, ob es eine Turing-Maschine gibt, die entscheiden kann, ob eine Maschine anhält oder nicht.

P-NP-Problem

In den 1950er Jahren formulierten Kurt Gödel und John Forbes Nash (unabhängig voneinander) das P-NP-Problem. Dabei geht es darum, ob die Menge P aller Probleme, die in polynomialer Laufzeit lösbar sind und die Menge NP aller Probleme, die von einer nichtdeterministischen Turingmaschine in polynomialer Laufzeit gelöst werden können, identisch ist.

Kenntnisstand bis jetzt

- alle Probleme in P sind auch in NP enthalten (kurz: $P \subseteq NP$)
- Einführung von Mengen NP-vollständiger und NP-schwerer Probleme
- Beispiele für NP-vollständige Probleme: Problem des Handlungsreisenden (Travelling-Salesman-Problem), Rucksackproblem, Problem der Färbung von Graphen, Knacken beliebiger asymmetrischer Verschlüsselungsverfahren, Proteinfaltung in der Biologie
- Prämie von 1 Million Dollar ausgesetzt für die Lösung des Problems (eines der „Millennium-Probleme“)

2. Softwareentwicklung

2.1. Programmiersprachen und -paradigmen

Auszug der historischen Entwicklung

- 19. Jhd.: Lochstreifen bei Webstühlen, Analytical Engine (Charles Babbage, Ada Lovelace)
- 1930er/1940er: Lambda-Kalkül (Alonzo Church, Stephen Kleene) – universelle Programmiersprache, aber noch nicht auf Rechner umgesetzt
- 1954: John Backus – erste tatsächlich umgesetzte Programmierhochsprache FORTRAN
- 1970er: Pascal, C, Prolog
- 1991: Python

Klassifizierung von Programmiersprachen

- Nach Generation
 - Einteilung nach Abstraktionsniveau
 - Niedere Programmiersprachen
 - Hardwarenah, Befehlssatz und Datentypen auf verwendeten Rechner abgestimmt
 - Maschinencode, Assemblercode
 - Höhere Programmiersprachen
 - Abstrakter, maschinenunabhängig
 - Werden entweder in Maschinencode übersetzt (compilierte Sprachen) oder interpretiert (Skriptsprachen)
- Nach Paradigma (= grundsätzlichem Ansatz)

Imperatives Programmierparadigma

Programme imperativer Programmiersprachen bestehen aus Anweisungen/Befehlen die beschreiben, wie das Programm seine Ergebnisse erzeugt.

Beispiele: Fortran, Pascal, C, Python

Deklaratives Programmierparadigma

Programme deklarativer Programmiersprachen bestehen aus Bedingungen, welche die Ausgabe des Programms erfüllen muss.

Das deklarative Programmierparadigma kann in funktionale und logische Programmierung unterteilt werden.

Funktionales Programmierparadigma

Das Programm wird als Funktion aufgefasst. Die Aufgabe wird dann durch das selbstständige Anwenden von Funktionsersetzung und Auswertung gelöst.

Dabei sollen die Funktionen so umgesetzt werden, dass sie von Zuständen des Systems unabhängig werden (also insbesondere bei gleichen Eingaben immer dieselben Ausgaben erzeugen).

Beispiele: Haskell, DrScheme

Auch in Python können Aspekte der funktionalen Programmierung umgesetzt werden, z.B. das Lambda-Kalkül.

Die Python-Anweisung

```
quadrat = lambda x : x**2
```

ist semantisch (=inhaltlich) identisch zur folgenden Methodenimplementierung

```
def quadrat(x):
    return x**2
```

Vorteile der funktionalen Programmierung

- Es können auch Funktionen als Parameter übergeben werden. Dies kann zu kleinerem Quellcode führen.
- Dadurch dass alles über Funktionen beschrieben werden soll, kann eine wesentlich genauere (automatische) Fehlersuche betrieben werden.

Logisches Programmierparadigma

Das Programm wird als Sammlung von Fakten und Regeln aufgefasst. Der Benutzer stellt dann an das Programm Anfragen und erhält darauf Antworten.

Beispiel: Prolog

Nutzung z.B. zum Lösen von Logicals und Kryptogrammen, aber auch bei der Traversierung in Bäumen

2.2. Versionsverwaltung

Warum Versionsverwaltung?

- Änderungen nachvollziehbar machen (Autor, Zeitpunkt, Gründe, konkrete Änderungen)
- Änderungen rückgängig machen
- Parallele Entwicklung an Teilprozessen möglich (vgl. Divide and Conquer-Heuristik)

Git vs. GitHub

Git	GitHub
Lokale (portabel nutzbare) Software zur Versionsverwaltung	Kostenfreier Onlinedienst zur Softwareentwicklung und Versionsverwaltung auf Git-Basis
April 2005: Linus Torvalds	Start Februar 2008, Übernahme 2018 durch Microsoft
https://git-scm.com/	https://github.com

Grundbegriffe

repository: Ordner mit allen Projektdateien und verstecktem Ordner .git

push / pull: Hoch- bzw. Herunterladen des aktuellen Codes ins/aus dem Repository

staged area: Änderungen die gesichert werden sollen

commit: Zusammenfassung von Änderungsschritten und änderndem Nutzer zu einem Paket

branch: Entwicklungszweig

merge: Zusammenführen von Änderungen verschiedener Branches zum Haupt-Branch

2.2. Versionsverwaltung

Ausgewählte Befehle

git init

Erstellen des lokalen Repositories

git status

aktuellen Zustand des Repositories anzeigen (offene Commits, staged areas)

git add --all

alle Änderungen zur staged area bringen

git commit -m "Nachricht"

Commit mit Betreff „Nachricht“ durchführen

git log

historische Entwicklung aller Commits des Branches anzeigen lassen inkl. Hash-Werten, Autor, Datum und Bemerkungen des Commits

git config --global user.email "meine@email.com"

git config --global user.name "MeinName"

E-Mail und Nutzernname des globalen git-Nutzers einstellen (alternativ: local für nur dieses Repository und system wenn es für den ganzen PC gelten soll)

git revert idhash

Commit mit Hash-Wert idhash rückgängig machen

git reset idhash

Branch auf Stand des Commits mit Hash-Wert idhash zurücksetzen

2.3. OOP

KOAM – Klassen, Objekte, Attribute, Methoden

Aus der Sekundarstufe I sollten wir uns mit den Grundbegriffen der objektorientierten Modellierung, kurz OOM, bereits auskennen. Zu ihnen gehören: Klassen, Objekte, Attribute und Methoden.

Die Klassen dienen gewissermaßen als Bauplan für ähnliche Objekte. Sie besitzen alle dieselben Attribute (ggf. aber mit unterschiedlichen Attributwerten) und Methoden, die diese Objekte umsetzen können.

Beispiel: das Zeichen **@** ist ein Objekt der Textverarbeitungsklasse „ZEICHEN“. Es besitzt damit alle Attribute dieser Klasse. Zu denen gehören z.B. Zeichenwert, Schriftart, Schriftgröße und Schriftfarbe. Für jedes Zeichen kann die Schriftfarbe innerhalb der Textverarbeitung geändert werden. Dabei wird eine zugehörige Methode aufgerufen, die wir z.B. Umfaerben nennen können. Diese erhält die neue Schriftfarbe als Parameter beim Aufruf mitgeliefert.

Passende UML-Karten für diesen Ausschnitt der Betrachtung

Klassenkarte	Objektkarte
<p>ZEICHEN</p> <p>Zeichenwert: Einzelzeichen Schriftart: Zeichenkette Schriftgröße: rationale Zahl Schriftfarbe: Farbe</p> <p>Umfaerben(neueFarbe: Farbe)</p>	<p>at:ZEICHEN</p> <p>Zeichenwert = "@" Schriftart = "Consolas" Schriftgröße = 18.0 Schriftfarbe = #FF0000</p>

Bemerkung: **UML** ist das Kürzel für Unified Modeling Language, also eine vereinheitlichte Modellierungssprache. Innerhalb dieser Sprache gibt es verschiedene Diagrammarten und Vorgaben bez. der Darstellung der beteiligten Elemente.

UML-Diagramme können wir in draw.io anfertigen.

Implementierung in Python

Für jede Klasse sollten wir eine eigene Python-Datei schreiben, die wir über den import-Befehl in unseren Skripten einbinden können. Jede Klasse muss dabei mindestens die **Konstruktor**-Methode mit dem fest vorgegebenen Namen `__init__` besitzen. Diese Methode wird aufgerufen, wenn ein Objekt dieser Klasse erzeugt wird.

Methoden, die auf Attribute des Objekts zugreifen sollen, benötigen in der Implementierung der Klasse in Python eine Variable `self`, die beim Aufruf von außen nicht mit angegeben werden muss.

Bemerkung: der Name dieser Variable kann an sich frei gewählt werden, es empfiehlt sich aber den Namen `self` beizubehalten.

2.3. OOP

Obiges Beispiel könnte in Python wie folgt umgesetzt werden (wobei die Schriftfarbe von rot auf magenta geändert wird während der Laufzeit).

Zeichen.py	Zeichenobjekt.py
<pre>class Zeichen(): def __init__(self, wert, art, groesse, farbe): self.Zeichenwert = wert self.Schriftart = art self.Schriftgroesse = groesse self.Schriftfarbe = farbe def Umfaerben(self, neueFarbe): self.Schriftfarbe = neueFarbe</pre>	<pre>import Zeichen atzeichen = Zeichen.Zeichen("@", "Consolas", 18, "#ff0000") atzeichen.Umfaerben("#F00FF")</pre>

Kapselung

Unter der **Kapselung** von Daten verstehen wir in der objektorientierten Programmierung, kurz OOP, den Ansatz, Daten und Informationen die nicht als Schnittstelle nach außen zwingend benötigt werden, vor dem direkten Zugriff zu schützen.

Üblicherweise gibt es dafür drei Sichtbarkeitstypen: public, protected und private.

Attribute und Methoden, die als **public** gekennzeichnet sind, sind nach außen sichtbar und direkt nutzbar. In der UML-Klassenkarte kennzeichnen wir ein solches Attribut bzw. eine solche Methode durch ein vorangestelltes „+“ vor dem Namen. Im Python-Quelltext sind Attribute und Methoden prinzipiell als public angenommen, wenn sie nicht wie gleich beschrieben gekennzeichnet werden.

Attribute und Methoden, die als **protected** gekennzeichnet sind, sind nach außen nur für Objekte der eigenen Klasse und den aus ihnen abgeleiteten Klassen zugreifbar. Wir kennzeichnen dies in der Klassenkarte durch ein vorangestelltes „#“ vor dem Namen. Um ein Attribut oder eine Methode in Python als protected zu kennzeichnen, muss der Name mit einem Unterstrich (_) beginnen.

Attribute und Methoden, die als **private** gekennzeichnet sind, sind nur für Objekte der eigenen Klasse zugreifbar. Die Kennzeichnung in der UML-Klassenkarte geschieht durch ein vorangestelltes „-“ vor dem Namen. Um ein Attribut oder eine Methode in Python als private zu kennzeichnen, muss der Name mit zwei Unterstrichen (__) beginnen.

Vorteil: da die Implementierung einer Klasse anderen Klassen nicht bekannt sein sollte, kann die Implementierung bei privaten Attributen und Methoden geändert werden, ohne die Zusammenarbeit mit anderen Klassen zu beeinträchtigen.

Vorgabe für Leistungskursler: Attribute werden in der Regel als private gekapselt, Methoden werden genau dann public gemacht, wenn sie von außen benötigt werden. Um den Attributwert auszulesen bzw. zu ändern sind dann get/set-Methoden nötig. Diese sind die in 1.3. angekündigte Ausnahme bez. der Kommentierung, da für diese Methoden eine Kommentierung nicht erforderlich ist (solang aus dem Namen „get“, „set“ und der Name der Variable erkennbar ist).

2.3. OOP

Der Vorgabe folgend müsste obiges Beispiel wie folgt angepasst werden.

Klassenkarte	Python-Implementierung														
<table border="1"> <thead> <tr> <th>ZEICHEN</th></tr> </thead> <tbody> <tr> <td>- Zeichenwert: Einzelzeichen</td></tr> <tr> <td>- Schriftart: Zeichenkette</td></tr> <tr> <td>- Schriftgröße: rationale Zahl</td></tr> <tr> <td>- Schriftfarbe: Farbe</td></tr> <tr> <td>+ __init__(wert, art, groesse, farbe)</td></tr> <tr> <td>+ setFarbe(neueFarbe)</td></tr> <tr> <td>+ getFarbe()</td></tr> <tr> <td>+ setGroesse(neueGroesse)</td></tr> <tr> <td>+ getGroesse()</td></tr> <tr> <td>+ setArt(neueArt)</td></tr> <tr> <td>+ getArt()</td></tr> <tr> <td>+ setWert(neuerWert)</td></tr> <tr> <td>+ getWert()</td></tr> </tbody> </table>	ZEICHEN	- Zeichenwert: Einzelzeichen	- Schriftart: Zeichenkette	- Schriftgröße: rationale Zahl	- Schriftfarbe: Farbe	+ __init__(wert, art, groesse, farbe)	+ setFarbe(neueFarbe)	+ getFarbe()	+ setGroesse(neueGroesse)	+ getGroesse()	+ setArt(neueArt)	+ getArt()	+ setWert(neuerWert)	+ getWert()	<pre>class Zeichen(): def __init__(self, wert, art, groesse, farbe): """ Konstruktoraufruf für Zeichenklasse """ self.__Zeichenwert = wert self.__Schriftart = art self.__Schriftgroesse = groesse self.__Schriftfarbe = farbe def setFarbe(self, neueFarbe): self.__Schriftfarbe = neueFarbe def getFarbe(self): return self.__Schriftfarbe def setGroesse(self, neueGroesse): self.__Schriftgroesse = neueGroesse def getGroesse(self): return self.__Schriftgroesse def setArt(self, neueArt): self.__Schriftart = neueArt def getArt(self): return self.__Schriftart def setWert(self, neuerWert): self.__Zeichenwert = neuerWert def getWert(self): return self.__Zeichenwert</pre>
ZEICHEN															
- Zeichenwert: Einzelzeichen															
- Schriftart: Zeichenkette															
- Schriftgröße: rationale Zahl															
- Schriftfarbe: Farbe															
+ __init__(wert, art, groesse, farbe)															
+ setFarbe(neueFarbe)															
+ getFarbe()															
+ setGroesse(neueGroesse)															
+ getGroesse()															
+ setArt(neueArt)															
+ getArt()															
+ setWert(neuerWert)															
+ getWert()															

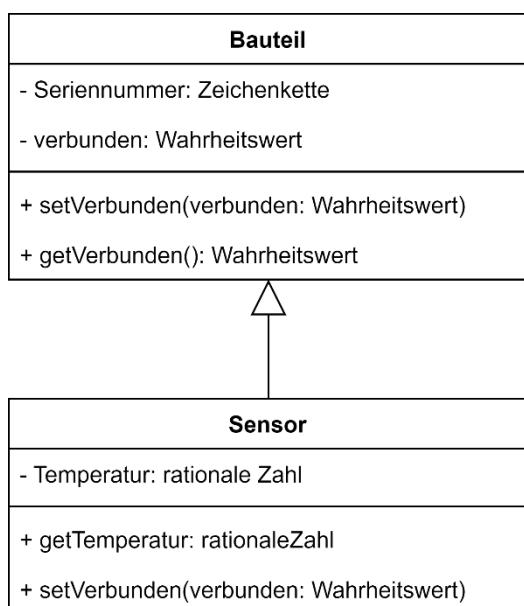
Vereinbarung: Auf die Angabe des Konstruktors in der Klassenkarte darf verzichtet werden.

Vererbung und Polymorphie

Häufig finden sich Situationen wieder, in denen generelle Klassen (z.B. Bauteile) spezialisiert nochmal unterschieden werden sollen (z.B. in Sensoren und Aktoren). Dies kann über das Konzept der Vererbung umgesetzt werden.

Bei einer **Vererbung** erhält die vererbte Klasse von ihrer Elternklasse alle Attribute und Methoden.

Beispiel

UML-Klassendiagramm	Implementierung
<p>Bauteil</p> <ul style="list-style-type: none"> - Seriennummer: Zeichenkette - verbunden: Wahrheitswert <p>+ setVerbunden(verbunden: Wahrheitswert)</p> <p>+ getVerbunden(): Wahrheitswert</p>  <pre> classDiagram class Bauteil { -Seriennummer: Zeichenkette -verbunden: Wahrheitswert +setVerbunden(verbunden: Wahrheitswert) +getVerbunden(): Wahrheitswert } class Sensor { -Temperatur: rationale Zahl +getTemperatur: rationaleZahl +setVerbunden(verbunden: Wahrheitswert) } Sensor < -- Bauteil </pre>	<pre> class Bauteil(): def __init__(self, Seriennummer, verbunden): """ Konstruktor der Bauteil-Klasse """ self.__Seriennummer = Seriennummer self.__verbunden = verbunden def setVerbunden(self, verbunden): self.__verbunden = verbunden def getVerbunden(self): return self.__verbunden class Sensor(Bauteil): def __init__(self, Seriennummer, verbunden, Temperatur): """ Konstruktor der Sensorklasse, abgeleitet von Bauteil """ self.__Temperatur = Temperatur super().__init__(Seriennummer, verbunden) def getTemperatur(self): return self.__Temperatur def setVerbunden(self, verbunden): super().setVerbunden(verbunden) print("Der neue Status ist " + str(self.getVerbunden())) Bauteil0815 = Bauteil("SN0815", False) TempSensor = Sensor("Celius42", True, 18.9) TempSensor.setVerbunden(False) </pre>

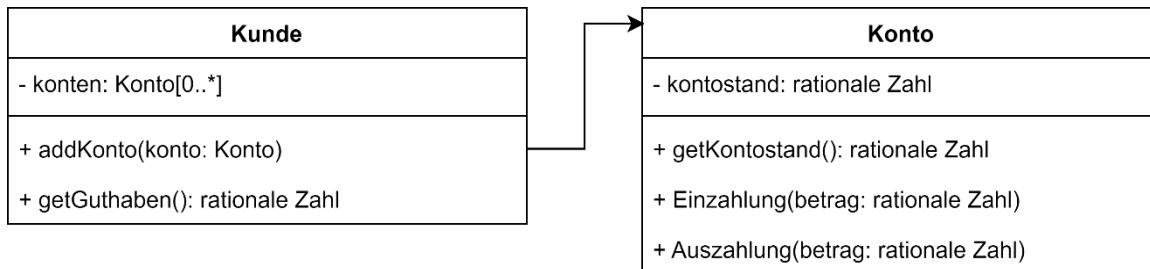
Eine Methode kann in der Basisklasse anders implementiert sein als in einer von ihr abgeleiteten Klasse. Wir sprechen dann von **Polymorphie**. Im obigen Beispiel ruft die `setVerbunden`-Methode des Sensors über `super().setVerbunden` die gleichgenannte Methode der Basisklasse auf und gibt dann zusätzlich noch etwas aus. Die Methode `setVerbunden` ist damit polymorph vorhanden.

Assoziation und Aggregation

Klassen können miteinander auch anders in Beziehung stehen als über Vererbung. Solche Beziehungen nennen wir **Assoziationen**. Assoziationen werden über Pfeile im UML-Klassendiagramm gekennzeichnet.

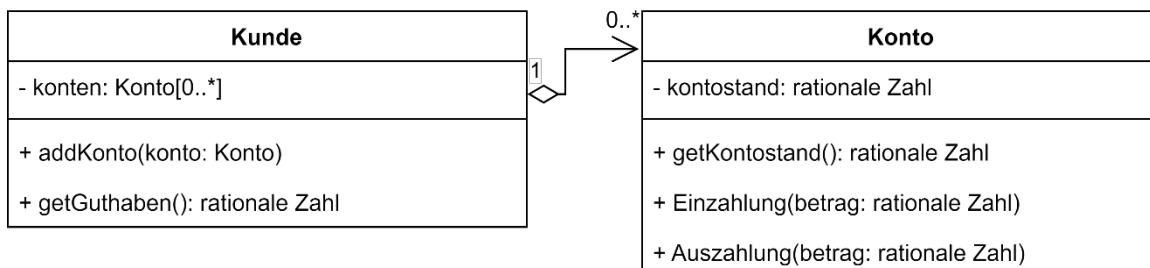
Beispiel

Bei der Finanzverwaltung sollen für jeden Kunden seine Konten gespeichert werden. Zum Hinzufügen eines Kontos bedarf die Klasse des Kunden zeitweise Zugriff auf die Klasse Konto.



Auf die Klasse Konto muss aber nicht nur zeitweise, sondern dauerhaft zugegriffen werden – hier, weil die Konten dauerhaft in einem Attribut des Kunden eingespeichert sind.

Assoziationen, die dauerhaft bestehen bleiben, werden als **Aggregationen** bezeichnet. Im UML-Klassendiagramm wird dafür eine Raute an den Pfeil gesetzt und die Kardinalität ähnlich zum ER-Diagramm gekennzeichnet.



Kardinalität am Beispiel: Jeder Kunde kann 0 bis unendlich viele Konten haben (daher auf Kontoseite die Notation $0..*$) und jedes Konto gehört zu genau einem Kunden (daher auf der Kundenseite die Notation 1).

Das Wissen über Assoziationen und Aggregationen ist u.a. wichtig, um keine Programmfehler zu erzeugen. Wird beispielsweise ein Konto im Speicher gelöscht, dann muss darauf geachtet werden, es auch beim Kunden zu entfernen, da andernfalls auf einen freigegebenen, vielleicht schon anders neugrenutzten, Speicherbereich vom Kunden aus zugegriffen werden würde.

Für Fans: unter den Aggregationen gibt es wieder besondere Fälle. Diese werden als Kompositionen bezeichnet. Im Beispiel würde dann eine Komposition vorliegen, wenn die Klasse Konto ohne die Klasse Kunde nicht existieren kann. Besser vorstellbar ist dies vielleicht bei der Beziehung zwischen einer BeLL-Dokumentation und ihrem Inhaltsverzeichnis. Wird die BeLL-Dokumentation gelöscht, dann wird automatisch auch das Inhaltsverzeichnis gelöscht; es kann ohne die Dokumentation nicht sinnvoll weiterexistieren.

2.4. Klassisch oder agil

Unterscheidung

Die **Klassische Softwareentwicklung** bezieht sich auf einen sequenziellen, strukturierten Ansatz zur Softwareerstellung. Die Phasen der Softwareentwicklung werden also nacheinander durchlaufen. Jede Phase beginnt erst, wenn die vorherige Phase abgeschlossen ist, was bedeutet, dass Änderungen später im Prozess oft schwer umzusetzen sind.

Die **agile Softwareentwicklung** bezieht sich auf einen flexibleren Ansatz, der auf iterativen und inkrementellen Fortschritten basiert. Dies ermöglicht schnelle Anpassungen an sich ändernde Anforderungen und die Lieferung funktionsfähiger Software bereits in kurzen Entwicklungszyklen.

Im Vergleich

Gemeinsames und Ähnliches	Unterschiede
<ul style="list-style-type: none"> • Formulierung von Projektzielen • planvolles Vorgehen bei der Festlegung von Zeitrahmen und Ressourceneinsatz • in Teamarbeit werden rollenspezifische Aufgaben und Verantwortlichkeiten übernommen • Projektinformationen werden dokumentiert • es erfolgt eine Präsentation und Bewertung der (Teil-)Produkte 	<ul style="list-style-type: none"> • linear ablaufender vs. iterativer Prozess • zentrale Projektleitung vs. selbstorganisiertes Team • kaum Einfluss der Stakeholder vs. konstanter Austausch mit Stakeholdern • einmalige Zieldefinition zu Projektbeginn vs. kontinuierliche Anpassung der Ziele

Wichtige Begriffe über die Modelle hinweg

- **Stakeholder:** Person oder Gruppe, die ein berechtigtes Interesse am Verlauf oder Ergebnis eines Prozesses oder Projekts hat (übersetzt: Interessengruppe)
- **Artefakt:** Element, das während des Entwicklungsprozesses produziert wurde (Pflichtenheft, Datenmodell, Designdokument, ...)

Wasserfallmodell

- klassisches Modell
- Phasen werden nacheinander abgeschlossen und enden mit einem Artefakt, das verbindliche Grundlage für darauffolgende Phasen ist
- im engeren Sinne existiert kein Rückweg zu vorheriger Phase (es gibt Erweiterungen, die dies erlauben)
- eine unvollständige Auswahl möglicher Phasen
 - Planung → Design → Entwicklung → Testen → Implementierung → Wartung
 - Analyse → Entwurf → Implementierung → Test → Inbetriebnahme
 - System requirements → software requirements → analysis → program design → coding → testing → operations [[Winston W. Royce: Managing the Development of Large Software Systems, August 1970](#)]
- Beispiele für Phasen und mögliches zugehöriges Artefakt

2.4. Klassisch oder agil

- Bedarfsanalyse: Machbarkeitsstudie, Lastenheft
- Anforderungsanalyse: Pflichtenheft
- Design / Entwurf: Systemarchitektur
- Codierung und Modultest: Programm
- Integration und Systemtest: Testergebnisse
- Installation und Wartung: Handbuch

V-Modell

- klassisches Modell
- Ursprung: [Berry W. Boehm: Guidelines for Verifying and Validating Software Requirements and Design Specifications, 1979](#)
- mehr Fokus auf die zur jeweiligen Phase gehörenden Tests als Folge der Erkenntnis, dass früh gefundene Fehler in der Beseitigung weniger Ressourcen benötigen als spät gefundene Fehler

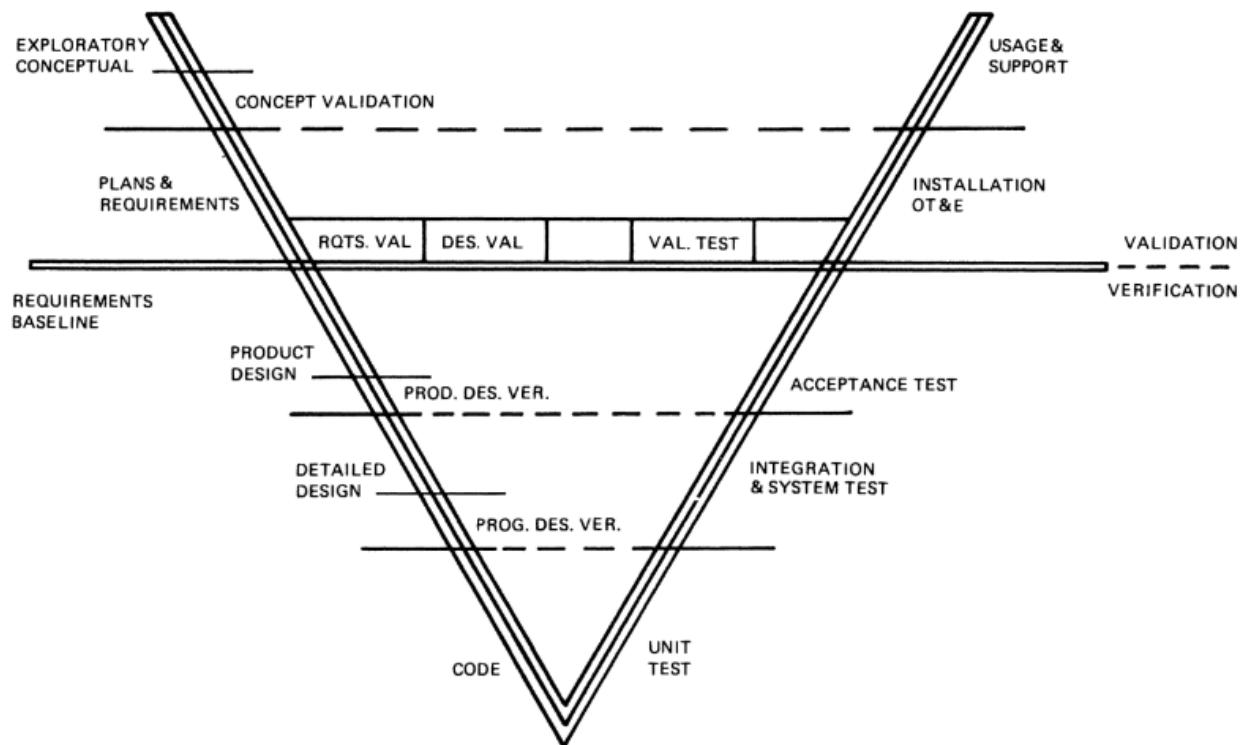
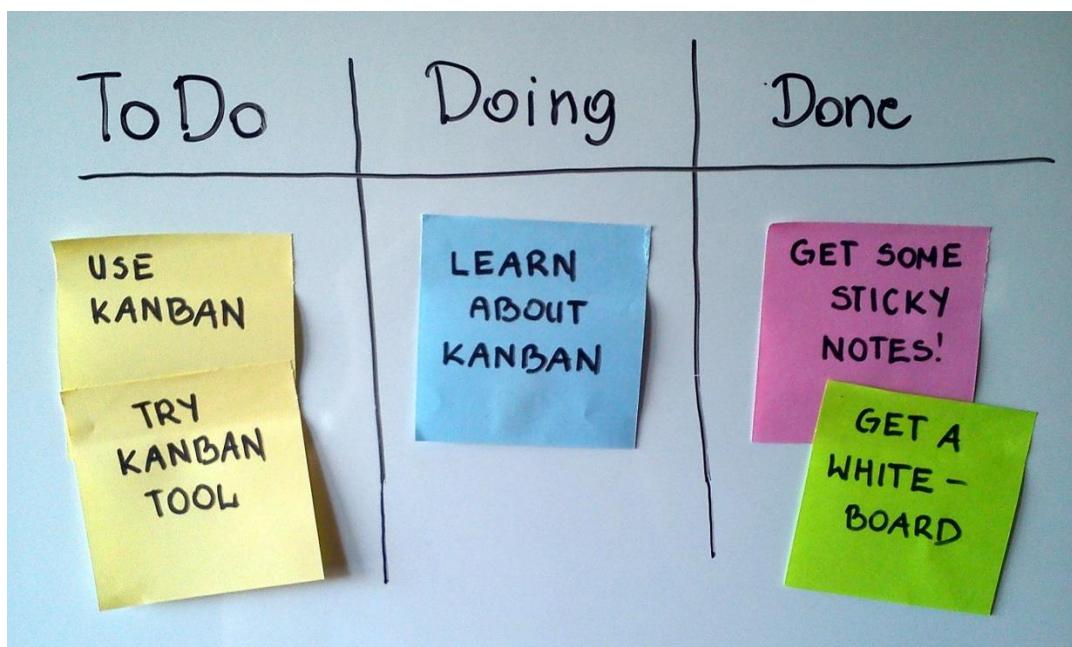


Abbildung aus dem oben angegebenen Dokument

2.4. Klassisch oder agil

Kanban

- agiler Ansatz
- übersetzt: Signalkarte
- Ursprung: 1947 Toyota mit dem Ziel der lean production (schlanke Produktion → Verschwendungen vermeiden, Produktionsfaktoren sparsam und effizient einsetzen)
- Ansatz basiert auf der Visualisierung des Arbeitsablaufs, Begrenzung der gleichzeitig in Bearbeitung befindlichen Aufgaben („Work-In-Progress-Limit“) und kontinuierlichen Verbesserung der Arbeitsprozesse
- Arbeit wird in Form von Aufgaben (Karten, Tickets, ...) auf einem Kanban-Board dargestellt
- Aufgabe wandert von Spalte zu Spalte, während sie durch den Arbeitsprozess fließt
- häufig verwendete Spalten: zu erledigen, in Bearbeitung, fertig



Jeff.lasovski@Wikimedia, letzter Abruf 28.12.2023

Scrum

- agiler Ansatz
- Ursprung: 1993 Jeff Sutherland und Ken Schwaber
- Überblick über die vielen wichtigen Aspekte (Teammitglieder, Sprintablauf, Artefakte, Säulen, Werte): siehe Scrum Guide von 2017 und Scrum Guide Poster im OPAL-Kurs

2.4. Klassisch oder agil

Gemeinsamkeiten von Kanban und Scrum

- agile Methoden
- Visualisierung auf Boards (Kanban: Arbeitsfluss, Scrum: Verwaltung der Aufgaben während eines Sprints)
- Kontinuierliche Verbesserung der Prozesse
- Transparenz und Zusammenarbeit
- Flexibilität
- Kontinuierliche Lieferung

Unterschiede zwischen Kanban und Scrum

Kanban	Scrum
<ul style="list-style-type: none">• flexibel und weniger reglementiert (keine festen Zeiträume)• erlaubt kontinuierliche Lieferung und Änderung im Arbeitsablauf• betont die Optimierung des Arbeitsflusses und Kontrolle der gleichzeitigen Arbeit• Identifizierung von Engpässen und ineffizienten Prozessen• Keine spezifischen Rollen	<ul style="list-style-type: none">• Feste Iterationen (Sprints)• Feste Rollen• Feste Meetings• Fokussierte Planung für jeden Sprint (Ziel, Backlog, ...)

2.5. Softwarearchitektur

3. Technische Informatik

3.1. Systemsoftware

3.2. Zahlen und Codierung

Herausforderung: Darstellen von Zahlen und Zeichen basierend auf Bits und damit verbunden den Ziffern 0 und 1.

Natürliche Zahlen

Gewohnt: Darstellung von Zahlen im Dezimalsystem, also zur Basis 10.

$$z = \overline{z_n z_{n-1} \cdots z_1 z_0} = z_n \cdot 10^n + z_{n-1} \cdot 10^{n-1} + \cdots + z_1 \cdot 10^1 + z_0 \cdot 10^0$$

Beispiel: $2024 = 2 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0$

Um zu kennzeichnen, dass die Zahl im Dezimalsystem gegeben ist, können wir auch $[2024]_{10}$ schreiben (lassen das aber meist aus Gewohnheit weg).

Jetzt: Darstellung im Stellenwertsystem der **Binärzahlen** (=Dualzahlen), also zur Basis 2.

$$z = [z_n z_{n-1} \cdots z_1 z_0]_2 = z_n \cdot 2^n + z_{n-1} \cdot 2^{n-1} + \cdots + z_1 \cdot 2^1 + z_0 \cdot 2^0$$

Beispiel: $[10\ 1001]_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 41$

Bei längeren Binärzahlen fasst man üblicherweise die Ziffern bei der Einerstelle beginnend in Viererblöcke zusammen (im Gegensatz zum Dezimalsystem, dort sind es üblicherweise Dreierblöcke).

Umwandlung von Dezimalzahlen zu Dualzahlen

Option 1: Zerlegen als Summe von Zweierpotenzen

$$100 = 64 + 36 = 64 + 32 + 4 = 2^6 + 2^5 + 2^2 = [110\ 0100]_2$$

Option 2: Fortschreitende Division mit Rest durch 2 bis Quotient 0

$$100 : 2 = 50 \text{ Rest } 0$$

$$50 : 2 = 25 \text{ Rest } 0$$

$$25 : 2 = 12 \text{ Rest } 1$$

$$12 : 2 = 6 \text{ Rest } 0$$

$$6 : 2 = 3 \text{ Rest } 0$$

$$3 : 2 = 1 \text{ Rest } 1$$

$$1 : 2 = 0 \text{ Rest } 1$$

Liest man nun die Reste von unten nach oben, ergibt sich die Binärzahl.

Vorteil: anwendbar auch auf die Umwandlung in andere Stellenwertsysteme, z.B. das **Hexadezimalsystem** mit Basis 16 (Ziffern $A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$).

3.2. Zahlen und Codierung

Vereinbarung: Zum Platzsparen darf die fortschreitende Division auch wie folgt kurznotiert werden.

$$100 \rightarrow^0 50 \rightarrow^0 25 \rightarrow^1 12 \rightarrow^0 6 \rightarrow^0 3 \rightarrow^1 1 \rightarrow^1 0$$

Hier werden die Reste an (oder über) die Pfeile geschrieben und jeweils nur die Quotienten in der Pfeilrichtung notiert.

Umwandlung von Dualzahlen zu Dezimalzahlen

Option 1: Ermitteln als Summe von Zweierpotenzen

$$[10\ 1001]_2 = 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 = 1 + 8 + 32 = 41$$

Option 2: Fortschreitende Multiplikation

Wir starten mit der Ziffer an der höchsten Stelle. Für jede darauffolgende Stelle gilt: Aktuellen Wert verdoppeln und dann die neue Stelle addieren.

Bsp. $[101001]_2$:

- Start mit 1
- Verdoppeln und 0 addieren: 2
- Verdoppeln und 1 addieren: 5
- Verdoppeln und 0 addieren: 10
- Verdoppeln und 0 addieren: 20
- Verdoppeln und 1 addieren: 41

Vorteil: auch das funktioniert gut mit anderen Stellenwertsystemen (dann jeweils mit Basis statt 2 multiplizieren).

Bsp. $[C3A]_{16}$:

- Start mit $C = 12$
- Mit 16 multiplizieren und 3 addieren: $16 \cdot 12 + 3 = 195$
- Mit 16 multiplizieren und $A = 10$ addieren: $16 \cdot 195 + 10 = 3130$

Negative ganze Zahlen

Für die Darstellung negativer ganzer Zahlen gibt es unterschiedliche Ansätze.

Variante 1: Einerkomplement

Der Betrag der Dezimalzahl wird umgewandelt, anschließend wird jedes Bit invertiert (also aus 0 eine 1 und aus 1 eine 0 gemacht).

Bsp. $-41: 41 = [10\ 1001]_2$ wird nun bitweise invertiert: $[01\ 0110]_2$. Tipp: Komplementbildung kennzeichnen (sonst entsteht z.B. oben $-41 = 22$)

Nachteile: für die Zahl 0 gibt es (wegen $-0 = 0$) zwei unterschiedliche Darstellungen, z.B. mit 8 Bit dargestellt: $[0000\ 0000]_2$ und $[1111\ 1111]_2$; addiert man eine Zahl und ihr Komplement, erhält man die Bitfolge aus lauter Einsen.

Variante 2: Zweierkomplement

Der Betrag der Dezimalzahl wird umgewandelt, anschließend wird jedes Bit invertiert (also aus 0 eine 1 und aus 1 eine 0 gemacht) **und abschließend 1 addiert**.

Bsp. $-41: 41 = [10\ 1001]_2$ wird nun bitweise invertiert: $[01\ 0110]_2$ und nun noch 1 addiert: $[01\ 0111]_2$. Hier entsteht ohne Kennzeichnung des Komplements die Gleichheit $-41 = 23$.

Vorteil: die Darstellung der 0 ist eindeutig und beim Addieren von einer Zahl und ihrem Komplement erhält man eine Bitfolge lauter Nullen.

Beim Rückrechnen wird wieder jedes Bit invertiert und am Ende 1 addiert.

Bsp.: $1[01\ 1011]_2$ wird beim Invertieren zu $[10\ 0100]_2$ und nach Addition der 1 zu $[10\ 0101]_2 = 37$. Die Ursprungszahl war also -37 .

Addition und Multiplikation von Binärzahlen

Binärzahlen werden genauso addiert und multipliziert wie Dezimalzahlen.

Beispiele

$$42 + 27 = 69$$

	4	2
+	2	7
	6	9

$$11 \cdot 13 = 143$$

1	1	.	1	3
		1	1	
			3	3
		1	4	3

$$[10\ 1010]_2 + [1\ 1011]_2 = [100\ 0101]_2$$

	1	0	1	1	0	1	0
+			1	1	0	1	1
1		1	1			1	
1	0	0	0	0	1	0	1

$$[1011]_2 \cdot [1101]_2 = [1000\ 1111]_2$$

1	0	1	1	.	1	1	0	1
		1	0	1	1			
			1	0	1	1	0	
					1	0	1	1
	1	1	1	1				
1	0	0	0	0	1	1	1	1

Vorteil beim Multiplizieren: dadurch, dass nur die Ziffern 0 und 1 existieren, ist das Teilergebnis entweder 0 oder die Ausgangszahl

Subtraktion von Binärzahlen

Die Subtraktion wird auf die Addition des Komplements zurückgeführt (wobei die betragsmäßig kleinere Zahl vorher um entsprechend viele Nullen zu erweitern ist, damit das „Vorzeichenbit“ an derselben Stelle landet).

Bsp. 1: $69 - 42 = 69 + (-42)$

$69 = [100\ 0101]_2$ und hat somit ohne Vorzeichenbit 7 Bit.

$42 = [10\ 1010]_2$ und muss somit auf 7 Bit erweitert werden: $[010\ 1010]_2$. Ich bilde nun das Zweierkomplement und erkläre das 8. Bit zum Vorzeichenbit. Nach Invertieren erhalte ich $[101\ 0101]_2$, nach der Addition von 1 daher $[101\ 0110]_2$, wobei nun das Vorzeichenbit hinzutritt: $1[101\ 0110]_2$. Ich muss also $0[100\ 0101]_2 + 1[101\ 0110]_2$ berechnen.

	0	1	0	0	0	1	0	1
+	1	1	0	1	0	1	1	0
1	1				1			
1	0	0	0	1	1	0	1	1

Durch Vernachlässigen der 1 (außerhalb des betrachteten Zahlenbereichs) erhalten wir das korrekte Ergebnis 27 als Binärzahl $[1\ 1011]_2$. Die 0 im Vorzeichenbit zeigt an, dass es sich um eine positive Zahl handelt.

Bsp. 2: $42 - 69 = 42 + (-69)$

$42 = [10\ 1010]_2$ hat ohne Vorzeichenbit 6 Bit.

$69 = [100\ 0101]_2$ hat ohne Vorzeichenbit einen Bedarf von 7 Bit, mit Vorzeichenbit daher 8 Bit. Beim Invertieren erhalte ich $1[011\ 1010]_2$ und damit als Zweierkomplement $1[011\ 1011]_2$.

42 auf 7 Bit plus ein Vorzeichenbit erweitert entspricht $0[010\ 1010]_2$

	0	0	1	0	1	0	1	0
+	1	0	1	1	1	0	1	1
		1	1	1		1		
1	1	1	0	0	1	0	1	

Es ergibt sich $1[1100101]_2$. Invertiert ergibt sich $0[0011010]_2$ und damit nach Addition der 1 die Zahl $[11011]_2$, womit das Ergebnis das Zweierkomplement von 27 war, also -27 .

Zeichencodierung

Es gibt unterschiedliche Ansätze, um Einzelzeichen mit Binärzahlen zu codieren. Zwei bekannte sind ASCII und Unicode.

ASCII-Codierung

- American Standard Code for Information Interchange
- 1963 eingeführt
- ursprünglich mit 7 Bit codiert (heute in der Regel mit 8 Bit)
- jedes Zeichen besitzt dieselbe Codelänge
- wechselseitige Zuordnung zwischen Einzelzeichen und ganzer Zahl kann der ASCII-Tabelle entnommen werden (siehe z.B. Tafelwerk)
- Bsp.: Das Zeichen „*“ erhielt den ASCII-Code 42, das Zeichen „A“ den ASCII-Code 65, das Zeichen „a“ den ASCII-Code 97
- praktisch zu wissen: der (horizontale) Tabulator besitzt den ASCII-Code 9

Unicode

- Version 1.0.0 im Oktober 1991 veröffentlicht
- Codelängen der einzelnen Zeichen variieren zwischen 1 und 4 Byte
- Unicode-Konsortium veröffentlicht auf ihrer Internetseite die jeweils geltenden Codierungen (etwa 150.000 Zeichen enthalten)
- <https://home.unicode.org/> bzw. <http://www.unicode.org/charts/>
- Neben „typischen“ Schriftzeichen sind bspw. auch mathematische Symbole $\propto \partial \wedge \cap$, Braille-Zeichen $\vdash \vdash$ und Emojis enthalten.
- Innerhalb des Unicodes gibt es verschiedene Verfahren, um Zeichen zu speichern, sogenannte Unicode Transformation Formats, kurz UTF. Bsp.: **UTF-8** (derzeit häufig Standard z.B. auch in Textdateien).

Gebrochene Zahlen

Auch für gebrochene Dezimalzahlen gibt es Umrechnungsmöglichkeiten in das Binärsystem. Hier wird üblicherweise der ganzzahlige Anteil und der rationale Anteil separat behandelt.

Umwandlung von Dezimalzahlen zu Dualzahlen

Am Beispiel 42,69

$$42 = [10\ 1010]_2$$

Option 1: Rationalen Anteil als Summe von Zweierbrüchen darstellen

$$0,69 = 0,5 + 0,19 = 0,5 + 0,125 + 0,065 = 0,5 + 0,125 + 0,0625 + 0,0025 = \dots$$

Option 2: fortschreitende Multiplikation

$$0,69 \cdot 2 = 1,38 \rightarrow 1$$

$$0,38 \cdot 2 = 0,76 \rightarrow 0$$

$$0,76 \cdot 2 = 1,52 \rightarrow 1$$

$$0,52 \cdot 2 = 1,04 \rightarrow 1$$

$$0,04 \cdot 2 = 0,08 \rightarrow 0$$

$$0,08 \cdot 2 = 0,16 \rightarrow 0 \dots$$

$$\Rightarrow 42,69 \approx [10\ 1010,1011\ 00 \dots]_2$$

Umwandlung von Dualzahlen zu Dezimalzahlen

Am Beispiel $[101,1101]_2$

Ganzzahliger Anteil: $[101]_2 = 5$

Rationaler Anteil: $[0,1101]_2 = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0,5 + 0,25 + 0,0625 = 0,8125$

Insgesamt: $[101,1101]_2 = 5,8125$

Herausforderung: viele Zahlen können nicht als endliche Dualzahl dargestellt werden.

Lösungsidee 1: Festkommazahlen

Es wird festgelegt, wie viele Vor- und wie viele Nachkommastellen eine Zahl haben darf.

Vorteil: intuitiv verständlich

Nachteil: unpraktikabel, da schnell Rechenfehler entstehen

Beispiel: wir beschränken unsere Darstellung auf 5 Vorkomma- und 3 Nachkommastellen. Die größtmögliche Zahl ist damit theoretisch $[1\ 1111,111]_2 = 31,875$. Die Ungenauigkeiten sind aber recht hoch. Die Zahl $[10101,101]_2$ ist das Ergebnis der Berechnung von z.B. 21,625 aber auch von 21,749 (und allen Zahlen dazwischen). Die Addition zweier Zahlen führt schnell aus dem Zahlenbereich heraus (die Multiplikation noch schneller).

Lösungsidee 2: Gleitkommazahlen

Wir stellen die Zahl in der aus Mathematik bekannten Form $z = m \cdot b^e$ dar, wobei ***m*** als **Mantisse** und ***e*** als **Exponent** bezeichnet wird und fordern $m \in [1; b[$.

Beispiele:

- $426,9 = 4,269 \cdot 10^2$ hat die Mantisse 4,269 und den Exponenten 2
- $602\ 200\ 000\ 000\ 000\ 000\ 000 = 6,022 \cdot 10^{23}$ hat die Mantisse 6,022 und den Exponenten 23 (und entstammt näherungsweise der Avogadro-Konstanten N_A (Anzahl der Teilchen je mol eines Stoffes))
- $0,00000000000000000000000000000006626 = 6,6626 \cdot 10^{-34}$ hat die Mantisse 6,6626 und den Exponenten -34 (und entstammt dem Planck'schen Wirkungsquantum h (Quotient aus Energie und Frequenz eines Photons))
- $[10\ 1110\ 1101\ 0110,101]_2 = [1,0111\ 0110\ 1011\ 0101]_2 \cdot 2^{13}$ hat die Mantisse 1,0111 0110 1011 und den Exponenten 13.

Die Norm IEEE 754 definiert Standarddarstellungen für die Speicherung von Gleitkommazahlen im Computer. Es gibt Darstellungen u.a. mit 32 Bit (binary32, vorher single genannt) und 64 Bit (binary64, vorher double, in Delphi auch real genannt).

Auch die Darstellung als Gleitkommazahl kann zu Rechenfehlern führen. Tipp beim Programmieren: Bedingungen der Form $v = x$ in die Form $|v - x| < \varepsilon$ für selbstgewähltes kleines positives ε umwandeln (also z.B. statt $r==1$ als Bedingung $\text{abs}(r-1)<0.0001$ verwenden).

Für Fans

Typ	Exponent	Mantisse	Wert Exponent	betragkleinste Zahl	betagsgrößte Zahl
binary32	8 Bit	23 Bit	$\{-126; \dots; 127\}$	$\approx 1,2 \cdot 10^{-38}$	$\approx 3,4 \cdot 10^{38}$
binary64	11 Bit	52 Bit	$\{-1022; \dots; 1023\}$	$\approx 4,9 \cdot 10^{-324}$	$\approx 1,8 \cdot 10^{308}$

Da nach obiger Forderung an die Mantisse bei Binärzahlen klar ist, dass diese mit 1 beginnen muss, wird die führende 1 in der Regel nicht mitgespeichert, sondern nur der rationale Anteil der Mantisse. Das jeweils fehlende Bit zur vollständigen Byte-Anzahl ist als Vorzeichenbit in Nutzung.

3.3. Schaltnetze

Grundlage Aussagenlogik

Eine Grundlage für Schaltnetze, die im Computer verwendet werden, bietet die Aussagenlogik.

Dabei ist eine **Aussage** eine Behauptung, der eindeutig ein Wahrheitswert zugewiesen werden kann.

- Boole'sche Logik: zwei mögliche Wahrheitswerte: True und False; benannt nach George BOOLE
- Kleene-Logik – drei Wahrheitswerte: wahr, falsch und weder wahr noch falsch
- Fuzzy-Logik – Wahrheitswert entspricht einer reellen Zahl im Bereich [0; 1]

In der Schule beschränken wir uns auf Boole'sche Logik, verwenden also nur die Wahrheitswerte 1 (True) und 0 (False).

Operationen auf Aussagen

Wir sollten folgende Operationen kennen.

- **Konjunktion** \wedge
 - $A \wedge B$ ist genau dann wahr, wenn sowohl A , als auch B wahr sind
- **Disjunktion** \vee
 - $A \vee B$ ist genau dann wahr, wenn A (und/)oder B wahr ist (ist also nur dann falsch, wenn beide falsch sind)
- **Negation** \neg
 - $\neg A = \overline{A}$ ist wahr, wenn A falsch ist und umgekehrt
- **Implikation** \Rightarrow, \rightarrow
 - $A \Rightarrow B$ ist nur dann falsch, wenn A wahr und B falsch ist (also aus etwas Wahrem etwas Falsches gefolgt wird)
- **Äquivalenz** $\Leftrightarrow, \odot, \equiv, \leftrightarrow$
 - $A \Leftrightarrow B$ ist genau dann wahr, wenn A und B denselben Wahrheitswert besitzen (also beide wahr oder beide falsch sind)
- **Antivalenz** $\oplus, |$
 - $A \oplus B$ ist genau dann wahr, wenn A und B sich im Wahrheitswert unterscheiden (also „entweder A oder B “ gilt).

Wir können die Operationen in einer Wahrheitstabelle angeben.

A	B	$A \wedge B$	$A \vee B$	$\neg A$	$A \Rightarrow B$	$A \equiv B$	$A \oplus B$
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	0	0	1
1	1	1	1	0	1	1	0

Wichtige **Vorrangregeln**: Klammer kommt vor \neg kommt vor \wedge kommt vor \vee

Aussagenlogische Gesetze

Folgende aussagenlogische Gesetze gelten immer.

Doppelnegation	$\overline{\overline{A}} = A$
Kommutativgesetz	$A \vee B = B \vee A, \quad A \wedge B = B \wedge A,$ $A \equiv B = B \equiv A, \quad A \oplus B = B \oplus A$
Assoziativgesetz	$A \wedge (B \wedge C) = (A \wedge B) \wedge C, \quad A \vee (B \vee C) = (A \vee B) \vee C$
Distributivgesetz	$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C), \quad A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
Idempotenzgesetz	$A \wedge A = A, \quad A \vee A = A$
De Morgansche Regeln	$\overline{A \wedge B} = \overline{A} \vee \overline{B}, \quad \overline{A \vee B} = \overline{A} \wedge \overline{B}$
Absorptionsgesetz	$A \vee (A \wedge B) = A, \quad A \wedge (A \vee B) = A$
Komplementärgesetz	$A \vee \overline{A} = 1, \quad A \wedge \overline{A} = 0$
Neutralitätsgesetz	$A \vee 0 = A, \quad A \wedge 1 = A$
Extremalgesetz	$A \vee 1 = 1, \quad A \wedge 0 = 0$

Bemerkung zur Formatierung: die hervorgehobenen Gesetze sollten namentlich bekannt sein, alle angegebenen Gesetze sollten aber angewendet werden können bei Umformungen.

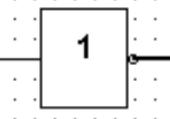
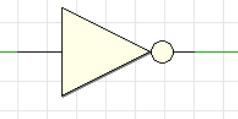
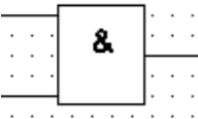
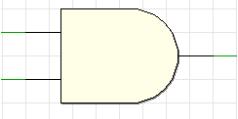
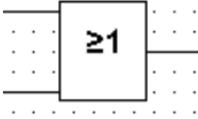
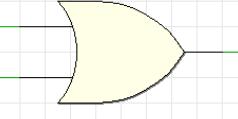
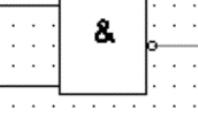
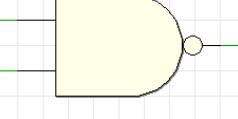
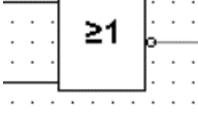
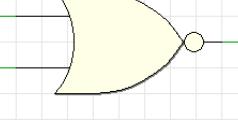
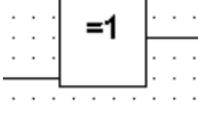
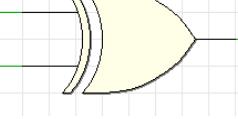
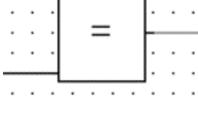
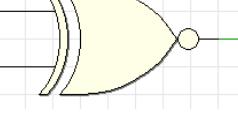
Die Gültigkeit von aussagenlogischen Behauptungen kann durch Umformungen unter Verwendung der oben angegebenen Gesetze oder durch Wahrheitstabellen nachgewiesen werden.

Empfehlung: Wahrheitstabellen stets mit den Spalten für alle beteiligten Variablen beginnen und alle auftretenden Fälle systematisch abarbeiten (z.B. 000, 001, 010, 011, ... bei drei beteiligten Variablen); vgl. Divide-and-Conquer-Heuristik.

Logikgatter

Als **Logikgatter** bezeichnen wir eine abstrakte Darstellung einer logischen Operation, die in einer Schaltung implementiert wird. Diese Gatter können durch Boolesche Funktionen beschrieben werden und bilden die Grundlage für die Konstruktion von Schaltnetzen.

Wir verwenden die folgenden Logikgatter.

Name	Darstellung IEC	Darstellung US	Boolesche Funktion
NOT			Negation
AND			Konjunktion
OR			Disjunktion
NAND ("not-AND")			Negation der Konjunktion
NOR ("not-OR")			Negation der Disjunktion
XOR ("exclusive-OR")			Antivalenz
XNOR ("exclusive-not-OR")			Äquivalenz

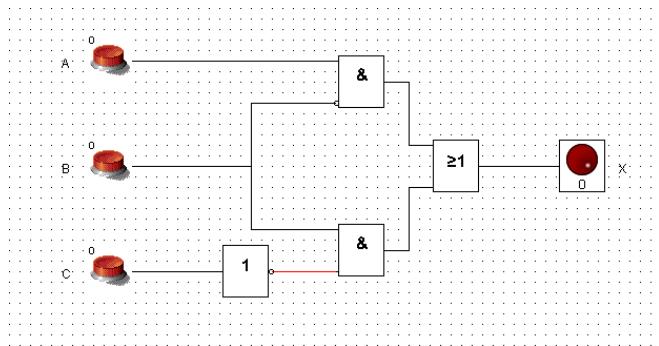
Als Software zur Simulation von Schaltnetzen verwenden wir LogicSim (beide Darstellungen möglich) oder DEEDS (US-Darstellung der Gatter).

Schaltnetzaanalyse

Grundidee: Schaltfunktion zu vorhandenem Schaltnetz ermitteln

Ausgangssituation: Schaltnetz dargestellt oder über Wahrheitstabelle gegeben

Beispiel

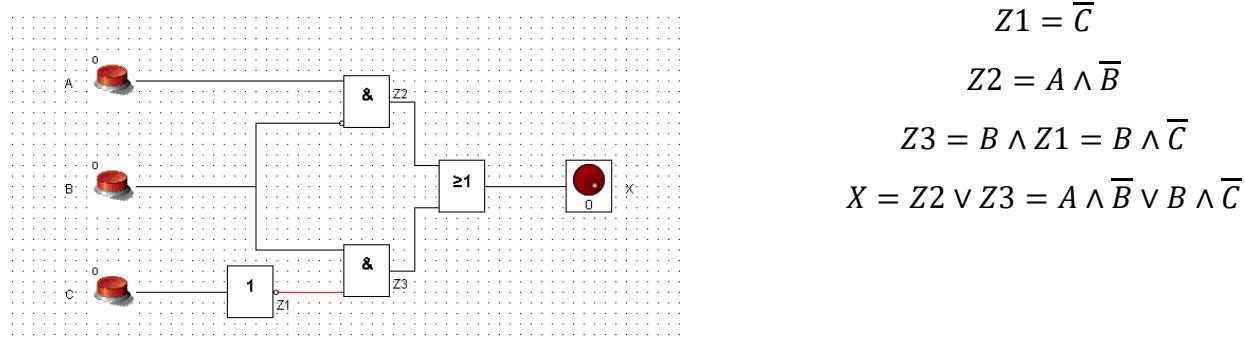


A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Option 1: Zwischengrößen nutzen

Bemerkung: vollständige Darstellung des Schaltnetzes dafür nötig (daher bei BlackBox-Experiment nicht geeignet)

Idee: Zwischengröße für jedes Gatter einfügen, Schaltfunktion schrittweise ermitteln



Option 2: Blöcke im KV-Diagramm nutzen

KV-Diagramm...

- Karnaugh-Veitch-Diagramm
- Darstellung der Wahrheitstabelle in besonderer Form
- Benachbarte Zeilen bzw. Spalten unterscheiden sich stets in genau einem Wahrheitswert der gegebenen Größen
- Randfelder können auch Blöcke bilden
- Suche nach zusammenhängenden Blöcken von „Einsen“

3.3. Schaltnetze

X	BC	$B\bar{C}$	$\bar{B}C$	$\bar{B}\bar{C}$
A	0	1	1	1
\bar{A}	0	1	0	0

$$X = B\bar{C} \vee A\bar{B}$$

Option 3: (K)DNF

Eine Formel befindet sich in **disjunktiver Normalform (DNF)**, wenn sie Disjunktion von Konjunktionstermen ist.

Eine Formel in DNF befindet sich in **kanonisch disjunktiver Normalform (KDNF)**, wenn bei jedem Konjunktionsterm alle gegebenen Größen vorhanden sind.

Taktik: alle Einsen der Wahrheitstabelle mit Disjunktionen verknüpfen

Am Beispiel:

$X = \bar{A}\bar{B}\bar{C} \vee A\bar{B}\bar{C} \vee A\bar{B}C \vee AB\bar{C}$ befindet sich in DNF, sogar in KDNF.

Über Verwendung aussagenlogischer Gesetze sollte die Schaltfunktion nun vereinfacht werden.

$$X = (A \vee \bar{A})B\bar{C} \vee A\bar{B}(C \vee \bar{C}) = B\bar{C} \vee A\bar{B}$$

Option 4: (K)KNF

Eine Formel befindet sich in **konjunktiver Normalform (KNF)**, wenn sie Konjunktion von Disjunktionstermen ist.

Eine Formel in KNF befindet sich in **kanonisch konjunktiver Normalform (KKNF)**, wenn bei jedem Disjunktionsterm alle gegebenen Größen vorhanden sind.

Taktik: alle Nullen der Wahrheitstabelle verwenden (empfiehlt sich daher für Funktionen mit wenigen Nullen), dabei Eingänge invertieren

A	B	C	X	Disjunktionsterm
0	0	0	0	$A \vee B \vee C$
0	0	1	0	$A \vee B \vee \bar{C}$
0	1	0	1	
0	1	1	0	$A \vee \bar{B} \vee \bar{C}$
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	0	$\bar{A} \vee \bar{B} \vee \bar{C}$

$X = (A \vee B \vee C) \wedge (A \vee B \vee \bar{C}) \wedge (A \vee \bar{B} \vee \bar{C}) \wedge (\bar{A} \vee \bar{B} \vee \bar{C})$ befindet sich in KNF, sogar in KKNF.

3.3. Schaltnetze

Über Verwendung aussagenlogischer Gesetze sollte die Schaltfunktion nun vereinfacht werden.

$$\begin{aligned}
 X &= (A \vee B \vee C) \wedge (A \vee B \vee \bar{C}) \wedge (A \vee \bar{B} \vee \bar{C}) \wedge (\bar{A} \vee \bar{B} \vee \bar{C}) \\
 &= ((A \vee B) \vee (C \wedge \bar{C})) \wedge ((A \wedge \bar{A}) \vee (\bar{B} \vee \bar{C})) \\
 &= ((A \vee B) \vee 0) \wedge (0 \vee (\bar{B} \vee \bar{C})) \\
 &= (A \vee B) \wedge (\bar{B} \vee \bar{C})
 \end{aligned}$$

Schaltnetzsynthese

Grundidee: Schaltnetz für gegebene Situation ermitteln

Ausgangssituation: Beschreibung der (Real-)Situation oder Angabe der Schaltfunktion

Ziel: Schaltnetz darstellen (ggf. anpassen auf Vorgaben bezüglich vorhandener Gatterarten und -anzahlen)

Häufig erfolgreicher Ansatz

- aus gegebener Situation Variablen für Eingang und Ausgang ermitteln (inkl. Bedeutung für 1 bzw. 0)
- Wahrheitstabelle für die gewählten Variablen aufstellen
- Schaltnetzaanalyse anhand der Tabelle durchführen
- entstandene Funktion über Grundgatter umsetzen

Beispiel: Im heimischen Labor von Herrn Seifert ist eine Warnleuchte angebracht. Es befinden sich im Haus drei Sensoren: einer in der Küche, einer auf dem Balkon, einer im Flur. Die Warnleuchte soll genau dann leuchten, wenn mindestens zwei der drei Sensoren aktiviert werden.

Eingänge: Sensoren in Küche, Balkon und Flur, z.B. K, B, F ($1 \triangleq$ aktiviert)

Ausgänge: Warnleuchte W ($1 \triangleq$ Leuchte leuchtet)

Wahrheitstabelle

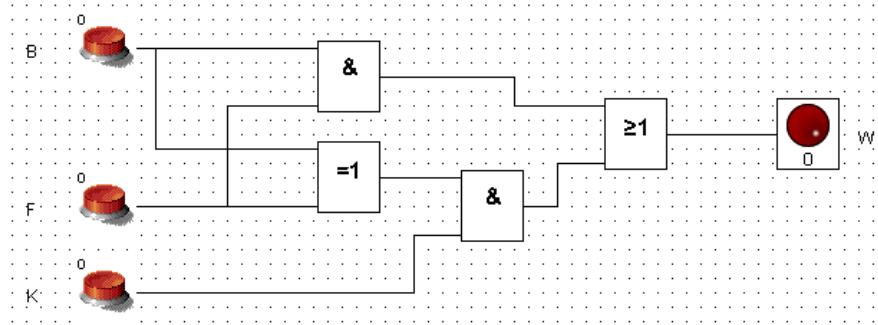
K	B	F	W
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

3.3. Schaltnetze

Entstehende Schaltfunktion

$$W = KBF \vee KB\bar{F} \vee K\bar{B}F \vee \bar{K}BF = BF(K \vee \bar{K}) \vee K(\bar{B}\bar{F} \vee \bar{B}F) = BF \vee K(B \oplus F)$$

Darstellung als Schaltnetz

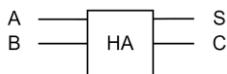


Besondere Schaltungen

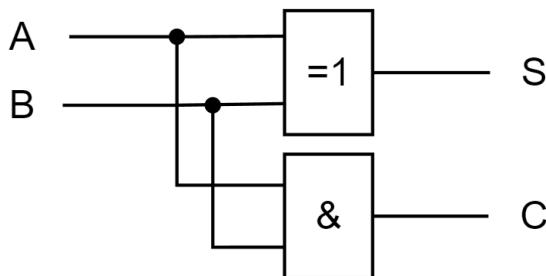
Halbaddierer

Grundidee: Summe und Übertrag zweier 1-Bit-Eingänge wird ermittelt und ausgegeben

Schaltzeichen



Ersatzschaltung



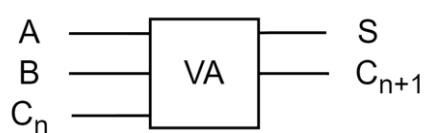
Wahrheitstabelle

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Volladdierer

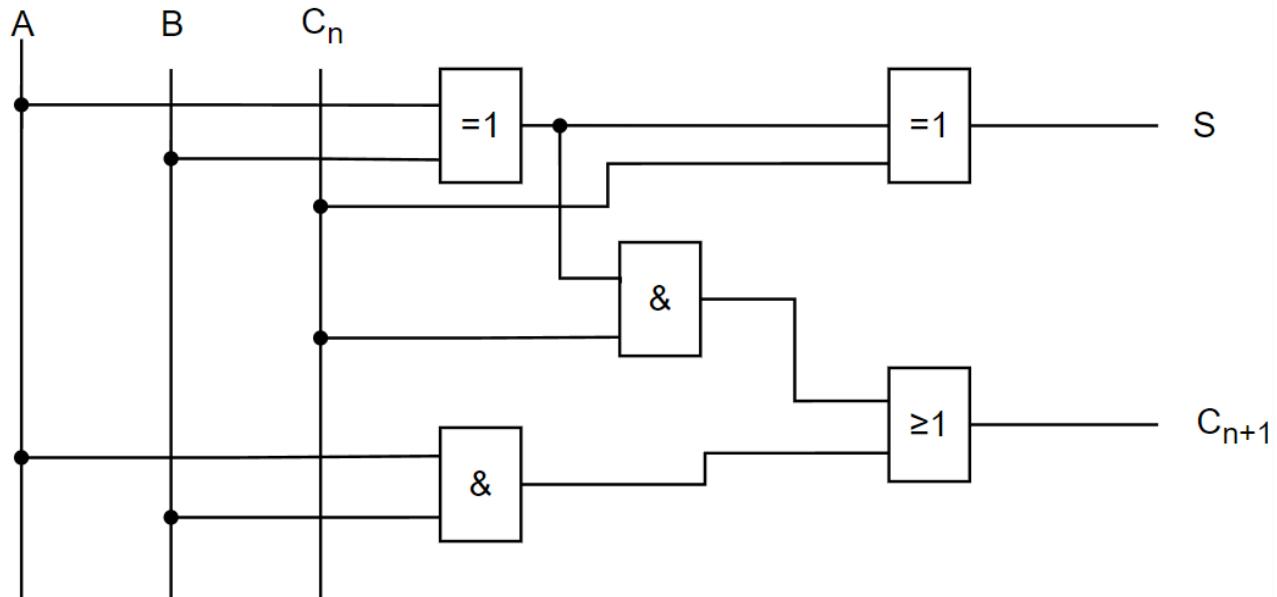
Grundidee: Halbaddierer + ggf. Übertrag aus vorheriger Rechnung

Schaltzeichen



Wahrheitstabelle

A	B	C _n	S	C _{n+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



FlipFlop

... bistabile Kippstufe / bistabiles Kippelement

... der aktuelle Zustand hängt von den Eingangssignalen UND dem vorherigen Zustand ab

... Möglichkeit, Datenmenge eines Bits über unbegrenzte Zeit zu speichern (solange Strom anliegt)

... verschiedene Arten, z.B. RS-FlipFlop, JK-FlipFlop

Multiplexer

... über ein Steuersignal wird entschieden, welcher der Eingänge an den Ausgang übertragen wird

1-MUX: 2 Eingänge E0 und E1, 1 Steuersignal S, 1 Ausgang A ($S=0 \rightarrow A=E0, S=1 \rightarrow A=E1$)

2-MUX: 4 Eingänge E0 bis E3, 2 Steuerbits S0 und S1, 1 Ausgang A

3.4. Rechnerhardware

Von-Neumann-Architektur

Herausforderung: Wie werden Programme und Daten im Speicher des Rechners abgelegt?

Zwei wesentlich unterschiedliche Ansätze

Von-Neumann-Architektur	Harvard-Architektur
<ul style="list-style-type: none"> grundlegende Komponenten: zentrale Verarbeitungseinheit, Speicher, Ein- und Ausgabegeräte Verarbeitung von Befehlen: CPU führt Befehle aus, die aus dem Speicher geladen werden 	
Daten und Befehle teilen sich denselben Adressraum	Getrennte Speicher und Busse für Daten und Befehle
Von-Neumann-Flaschenhals	Höhere Verarbeitungsgeschwindigkeit möglich, da Befehle und Daten gleichzeitig geladen werden können
Anwendung in Desktop-PCs, Notebooks	Anwendung dort, wo hohe Verarbeitungsgeschwindigkeit nötig (Mikrocontroller für spezielle Aufgaben)

Moore'sches Gesetz

- 1965 Gordon Moore: Zahl der Transistoren integrierter Schaltkreise mit minimalen Komponentenkosten verdoppelt sich regelmäßig (etwa alle 2 Jahre)
- kein wissenschaftliches Gesetz, aber Faustformel
- Herausforderungen und Grenzen in moderner Halbleitertechnik (z.B. Quanteneffekte, Wärmeentwicklung, feine Strukturen zu lithographieren)

Herstellung von Prozessoren

- Sand → Silizium → Block → Wafer
- Front end of line: Integration der Transistoren (mehrere dünne Schichten auf Wafer aufgebracht, unbrauchbare Stellen weggeätzt)
- Back end of line: Verbindung der Transistoren, Chips aus Wafer entfernen und in Schutzgehäuse mit Kontakten verbunden

Wichtige Prozessorkenndaten

- Taktfrequenz: wie oft kann Prozessor je Sekunde einen Zyklus von Operationen durchführen
- Kernanzahl: wie viele Kerne vorhanden, die parallel Prozesse abarbeiten können
- Cache-Speicher: Level 1,2,3 – kleiner, schneller Speicher für häufig verwendete Daten und Befehle (Level 1 der schnellste und kleinste)
- Befehlssätze: welche Makrobefehle stehen zur Programmierung zur Verfügung
- Sockel: physisches Interface auf dem Motherboard
- Leistungsaufnahme: wie viel Leistung wird nicht überschritten

Multi-Core

- Vorteil: Parallelverarbeitung, Effizienz
- Herausforderungen: thermische Grenzen, Energieverbrauch, Komplexität des Designs, Management der Kommunikation zwischen den Kernen, spezielle Programmierung nötig

Prozess vs. Thread

- Prozess: Instanz eines ausführbaren Programms
- Thread: kleinste Ausführungseinheit eines Prozesses; Prozess kann mehrere Threads haben, die den Code, die Daten und die Ressourcen teilen

Befehlsabarbeitung (Von-Neumann-Architektur)

- Fetch-Phase
 - der gemäß Befehlszähler nächste Befehl wird aus dem Speicher in den Prozessor geladen
 - eingelegter Befehl wird analysiert
- Execute-Phase
 - Befehl wird gemäß Makrocodetabelle ausgeführt, z.B. Laden von Daten in den Akkumulator der ALU, Speichern von Ergebnissen in Speicherzellen, ...
 - Befehlszähler wird auf nächsten Befehl eingestellt

Arten von Speichermedien

- optisch
 - Bsp.: Bluray, DVD
 - 0en und 1en werden in Form von Pits (kleine Vertiefungen) und Lands (flache Bereiche zwischen Pits) spiralförmig, beginnend im Zentrum gespeichert
 - Übergang von Pit zu Land oder Land zu Pit entspricht 1, fehlender Übergang entspricht 0
 - Laserstrahl wird verwendet zum Speichern und Lesen der Daten
 - Bluray: blauer Laserstrahl wird verwendet (→ kurzwellig → Daten können auf engem Raum gespeichert werden)
- elektromagnetisch
 - Bsp.: HDD
 - Schreibkopf erzeugt ein lokales magnetisches Feld, das die magnetische Ausrichtung der Partikel auf dem Speichermaterial verändert
 - 0en und 1en entsprechen den Orientierungen der Partikel
 - Lesekopf enthält Sensoren, welche die Magnetfelder detektieren
 - Bei Festplatten: Organisation der Daten in Spuren, Sektoren und Clustern
- elektronisch
 - Bsp.: SSD, USB, Speicherkarten, DDR5-RAM
 - Verwendung von elektronischen Schaltungen zum Speichern und Laden von Daten
 - auch als Halbleiterspeicher bezeichnet (Halbleiter siehe Physik Kl. 9 und 11)
 - unterschiedliche Unterkategorien z.B. Flash-Speicher, DRAM, SRAM

Dateisysteme

- Ablageorganisation von Daten auf einem Datenträger
- ausgewählte Beispiele
 - FAT32 – maximale Dateigröße 4 GiB, keine Verschlüsselungs- und Komprimierungsfunktionen, breite Hardwareunterstützung
 - ext4 – maximale Dateigröße 16 TiB, integrierte Verschlüsselung, für Linux und Mac gedacht, für Windows Zusatzsoftware nötig
 - NTFS – maximale Dateigröße 256 TiB, nicht geeignet für Laufwerke und Partitionen unter 400 MB (Verwaltungsaufwand), Verschlüsselung und Komprimierung möglich
 - APFS – maximale Dateigröße 8 EiB, Mac-geeignet, für Windows Zusatzsoftware nötig

Weitere Begriffe zum Speicher

- **Fragmentierung:** die Bestandteile einer Datei können im Speicher an nicht-zusammenhängenden Stellen vorhanden sein, also „Fragmente“ bilden. Hilfsmittel: Defragmentierung
- **NAS:** Network Attached Storage – ein spezieller Dateiserver innerhalb eines Netzwerkes, der es Nutzern ermöglicht, zentral Daten zu speichern und auf sie zuzugreifen (z.B. gemeinsame Erinnerungsfotos, Musik, Videos, ...)

3.5. Netzwerke

3.5.1. Grundlagen

Als **Rechnernetz** bezeichnen wir ein verteiltes System von mehreren unabhängigen, möglicherweise verschiedenen Geräten, die miteinander kommunizieren können.

Zu den Gründen für die Kopplung von Geräten zählen u.a.

- Ressourcenverbund: gemeinsame Nutzung von Hard- und Softwarekomponenten und Daten
- Lastverbund: Verteilung benötigter Rechenleistung
- Kommunikationsverbund: Nachrichtenaustausch

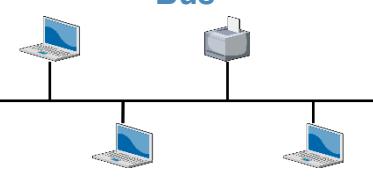
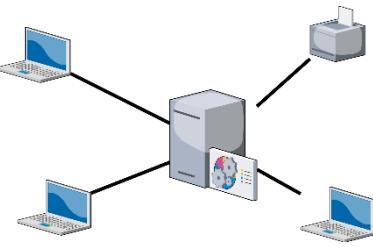
Je nach räumlicher Ausdehnung unterscheiden wir

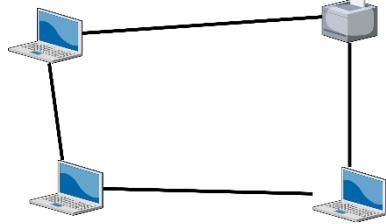
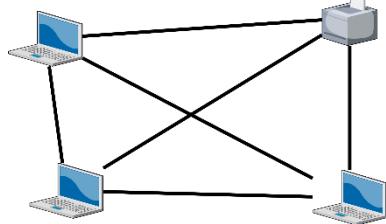
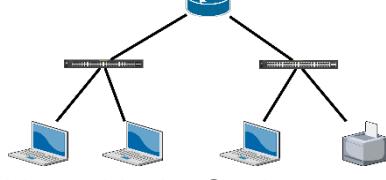
- **LAN**: Local Area Network (einzelne Gebäude(komplexe))
- **MAN**: Metropolitan Area Network (etwa auf Stadtgröße)
- **WAN**: Wide Area Network (überregional)

Topologien

Die **Topologie** bezeichnet die physische oder logische Struktur, in der die Netzwerkgeräte miteinander verbunden sind.

Die **physische Topologie** gibt dabei die real vorliegende Struktur an Geräten und Kabeln an, die **logische Topologie** beschreibt den Weg, den die Daten innerhalb eines Netzwerks nehmen. So kann ein physischer Stern z.B. logisch als Token Ring genutzt werden.

Topologie	Vorteile	Nachteile
<p>Bus</p>  <p>alle Geräte über gemeinsames Übertragungsmedium verbunden</p>	<ul style="list-style-type: none"> • einfacher Aufbau und Installation • Kostengünstig • einfache Erweiterung 	<ul style="list-style-type: none"> • wenn gemeinsames Medium ausfällt, wird gesamtes Netzwerk unterbrochen • Datenkollisionen • begrenzte Kabellänge des zentralen Kabels
<p>Stern</p>  <p>alle Geräte sind über individuelle Kabel mit einem zentralen Knoten verbunden</p>	<ul style="list-style-type: none"> • einfache Fehlersuche und Verwaltung • Erweiterbarkeit • Ausfallsicherheit einzelner Verbindungen 	<ul style="list-style-type: none"> • Abhängigkeit vom zentralen Knoten • Kabelaufwand • Kosten (v.a. für ausfallarmen zentralen Knoten)

<p>(Token) Ring</p>  <p>Geräte sind in ringförmiger Struktur miteinander verbunden, Daten werden in eine Richtung entlang des Rings weitergeleitet</p>	<ul style="list-style-type: none"> • Vermeidung von Datenkollisionen • gleichmäßige Bandbreitennutzung • geordneter Datenfluss 	<ul style="list-style-type: none"> • Ausfall eines Gerätes oder einer Verbindung kann den gesamten Ring unterbrechen • geringere Datenübertragungsgeschwindigkeit • komplexere Verwaltung und Fehlerbehebung
<p>vermascht</p>  <p>Geräte sind teilweise mit mehreren Geräten verbunden (bei vollvermascht wie im Bild: jedes Gerät mit jedem anderen)</p>	<ul style="list-style-type: none"> • Robustheit gegenüber Ausfällen • Erhöhung von Effizienz und Geschwindigkeit durch Existenz mehrerer Wege zwischen Knoten 	<ul style="list-style-type: none"> • Komplexität und Kosten der Installation • hoher Kabelaufwand • Verwaltungskomplexität
<p>Baum</p>  <p>hierarchische Struktur aus verbundenen Stern-Topologien</p>	<ul style="list-style-type: none"> • einfache Erweiterbarkeit • einfache Fehlersuche • Segmentierung des Netzwerks → Effizienz- und Leistungssteigerung • gute Eignung für große Netzwerke 	<ul style="list-style-type: none"> • Abhängigkeit vom Wurzelknoten • Kosten (v.a. für ausfallarmen Wurzelknoten und benötigte Switches usw.) • Verwaltungskomplexität • Leistungsengpässe bei starker Auslastung des Wurzelknotens

Übertragungsmedien

Als **Übertragungsmedien** bezeichnen wir physische Kanäle, über die Daten von einem Netzwerkgerät zu einem anderen Netzwerkgerät übertragen werden.

Wir unterscheiden zwischen kabelgebundenen und kabellosen Übertragungsmedien.

Name	Bemerkungen
Koaxialkabel 	<ul style="list-style-type: none"> einzelner Kupferleiter, umgeben von einer isolierenden Schicht und einem metallischen Geflecht geringe Störanfälligkeit, größere Reichweite als Twisted-Pair-K. höherer Kostenaufwand, schwer zu installieren Verwendung: Antennenkabel, ältere Ethernet-Standards (10BASE2)
Twisted-Pair-Kabel 	<ul style="list-style-type: none"> paarweise verdrillte Kupferkabel Effekt: Reduktion elektromagnetischer Störungen Verwendung: Ethernet-Netzwerke (10BASE-T, 100BASE-TX)
Glasfaserkabel 	<ul style="list-style-type: none"> Übertragung von Daten als Lichtimpulse durch Glasfasern sehr hohe Übertragungsgeschwindigkeit, große Reichweite, unempfindlich gegenüber elektromagnetischen Störungen hohe Kosten, schwierig zu installieren und reparieren Verwendung: Langstreckenkommunikation

Bildquellen (jeweils Abrufdatum 24.08.2024)

Koaxialkabel: Fleshgrinder, Wikimedia: https://commons.wikimedia.org/wiki/File:Coaxial_cable_cutaway_new.svg

Twisted-Pair-Kabel: Baran Ivo: <https://ndla.no/subject:1:81b3892a-78e7-4e43-bc31-fd5f8a5090e7/topic:1:c7717a05-61ae-4d57-a470-8105eac4afad/resource:1:83339>

Glasfaserkabel: Chris Woodford/explainthatstuff.com: <https://www.explainthatstuff.com/fiberoptics.html>

Zu den kabellosen Übertragungsmedien gehören u.a. Funkwellen (WLAN, Mobilfunk), Infrarot und Bluetooth.

3.5.2. Netzwerkprotokolle und Verfahren

Als **Netzwerkprotokoll** bezeichnen wir Standards und Konventionen für die Übermittlung von Daten zwischen mehreren Rechnern, die in einem Netzwerk miteinander verbunden sind.

Das Internet-Protokoll

Eines der bekanntesten Netzwerkprotokolle ist das **Internet Protocol (IP)**. Verwendung finden heute zwei verschiedene Versionen: IPv4 und IPv6.

Beide Verfahren eint, dass jedem Netzgerät eine Adresse zugeteilt wird zur Identifikation.

Aufbau IPv4-Adresse

Jede **IPv4**-Adresse besitzt eine Länge von 32 Bit. Damit sind 2^{32} also etwa 4,3 Milliarden Adressen möglich. Die 32 Bit werden in vier Oktette eingeteilt, deren Werte üblicherweise dezimal, durch Punkte getrennt, angegeben werden. So ist etwa 192.168.0.1 eine gültige IPv4-Adresse.

Aufbau IPv6-Adresse

Jede **IPv6**-Adresse besitzt eine Länge von 128 Bit. Damit sind $2^{128} \approx 3,4 \cdot 10^{38}$ Adressen möglich. Die 128 Bit werden in 8 Hextette (oft auch nur als „Segmente“ bezeichnet) zu je 16 Bit aufgeteilt, deren Werte üblicherweise hexadezimal, durch Doppelpunkte getrennt, angegeben werden. Es gibt besondere Regelungen zur Verkürzung der Adressangabe. So bezeichnen die Angaben 2001:0DB8:0000:0001:0000:0000:0010:01FF und 2001:DB8:0:1::10:1FF dieselben Geräte.

Subnetzmaske

Adressen des Internetprotokolls bestehen aus einem Netzanteil und einem Hostanteil, auch Geräteanteil genannt. Anhand der **Subnetzmaske** kann bei IPv4-Adressen erkannt werden, wie viele Bit der Netzanteil besitzt. Alle zum Netzanteil gehörenden Bits tragen in der Subnetzmaske den Wert 1, alle darauffolgenden Bits gehören zum Hostanteil der IPv4-Adresse.

Beispiel: ein Gerät habe die IPv4-Adresse 192.0.2.155 und gehöre zum Subnetz mit der Subnetzmaske 255.255.192.0. In Bitschreibweise entspricht die Subnetzmaske der Zeichenfolge 11111111.11111111.11000000.00000000, also 18 Einsen mit darauffolgenden 14 Nullen. Alle Geräte, die zu diesem Subnetz gehören, müssen damit die ersten 18 Stellen ihrer IP-Adresse genauso haben, wie dies bei der Adresse 192.0.2.155 der Fall ist (also mit 11000000.00000000.00 beginnen). In diesem Teilnetz sind $2^{14} - 2 = 16.382$ IPv4-Adressen für Geräte zur Verfügung. Die kleinstmögliche Adresse (hier 192.0.0.0) dient der **Identifikation des Netzwerkes**, die größtmögliche Adresse (hier 192.0.63.255) dient als sogenannte **Broadcast**-Adresse und wird für Nachrichten genutzt, bei denen die IP-Adresse des konkreten Empfängers im Subnetz noch nicht bekannt ist. Alle anderen 16.382 der theoretisch möglichen Adressen können nun für Geräte verwendet werden.

CIDR

In der historischen Entwicklung der IPv4-Adressen gab es anfangs ausgewählte Klassen von IPv4-Adressen. Bei Klasse A-Adressen wurde die Subnetzmaske 255.0.0.0 verwendet. Klasse B-Adressen verwendeten die Subnetzmaske 255.255.0.0 und Klasse C-Adressen verwendeten die Subnetzmaske 255.255.255.0.

Durch diese Vorgabe ist man jedoch stark eingeschränkt, was die Anzahl an Netzwerken und zur Verfügung stehenden Gerätedressen angeht.

Classless Inter-Domain Routing, kurz **CIDR**, ist ein 1993 eingeführtes Verfahren, um den IPv4-Adressraum effizienter zu nutzen. Dabei sind auch Subnetzmasken möglich, bei denen die Anzahl der 1-Bits nicht durch acht teilbar ist. Die Anzahl der Einsen wird hinter der IPv4-Adresse und einem Slash notiert.

Das Paar aus IPv4-Adresse 192.168.0.1 und Subnetzmaske 255.255.255.0 (also 24 Einsen) wird so kurz als 192.168.0.1/24 notiert.

Das Paar 192.0.2.255 und Subnetzmaske 255.255.192.0 (18 Einsen) wird so kurz mit der CIDR-Notation als 192.0.2.255/18 angegeben.

DHCP

DHCP ist das Kürzel für **Dynamic Host Configuration Protocol** und ist ein Kommunikationsprotokoll, das insbesondere dafür genutzt wird, um Geräten dynamisch eine IP-Adresse zuzuweisen.

Schritt 1: DHCP Discover

Der Client sendet eine DHCP Discover-Nachricht mit der eigenen MAC-Adresse als Broadcast an alle Geräte im Netzwerk.

Schritt 2: DHCP Offer

Jeder DHCP-Server im Netzwerk, der die Discover-Nachricht empfängt, antwortet mit einer DHCP Offer-Nachricht. Diese enthält die vorgeschlagene IP-Adresse, die Subnetzmaske, das Standard-Gateway, Informationen zu DNS-Servern sowie die Lease-Dauer (also die Zeit, für welche die angebotene IP-Adresse gültig ist). Auch diese Offer-Nachricht wird häufig als Broadcast gesendet, da ja noch keine Kontaktdata des Clients bekannt sind.

Schritt 3: DHCP Request

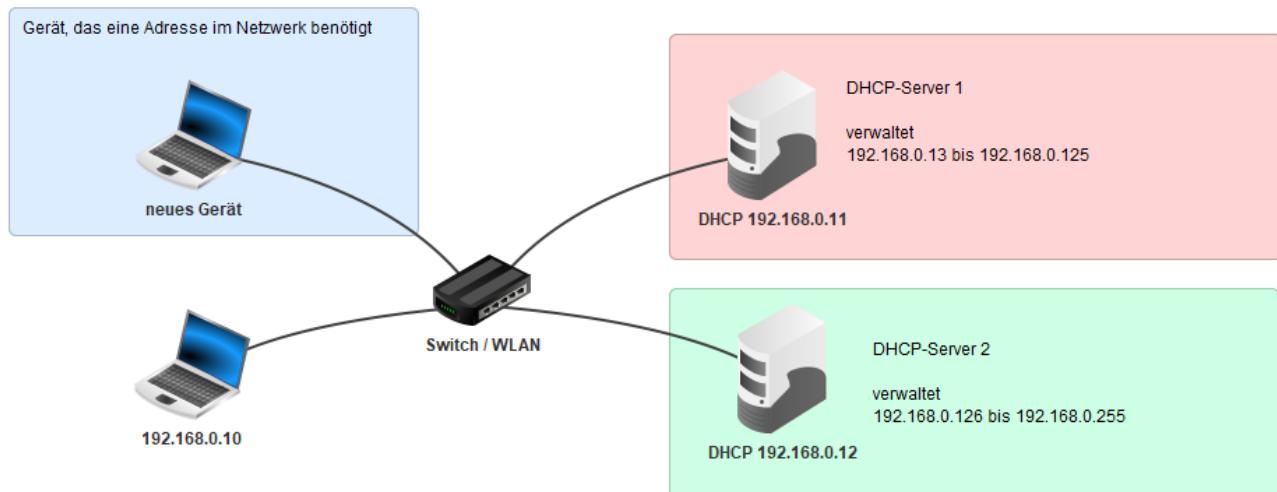
Der Client erhält ggf. mehrere Offer-Nachrichten und wählt sich eine (meist die erste) aus. Er sendet eine DHCP Request-Nachricht, die angibt, welches Angebot er akzeptiert hat. Damit alle anderen Server informiert sind, wird auch diese Nachricht meist im Broadcast gesendet.

Schritt 4: DHCP Acknowledgement

Der Server, dessen Offer-Nachricht akzeptiert wurde, bestätigt die Clientanfrage durch Senden einer DHCP Acknowledgement-Nachricht. Diese beinhaltet die vereinbarten Daten (siehe Offer-

3.5. Netzwerke

Nachricht). Der Client konfiguriert sich nun mit den erhaltenen Parametern und ist bereit, im Netzwerk gezielt zu kommunizieren.



Nr.	Zeit	Quelle	Ziel	Protokoll	Schicht	Bemerkungen / Details
1	13:45:00.003	0.0.0.0:68	255.255.255.255:67	DHCP	Anwendung	DHCPDISCOVER yiaddr=0.0.0 chaddr=F9:BC:07:82:87:49
2	13:45:00.029	192.168.0.12:67	255.255.255.255:68	DHCP	Anwendung	DHCPOFFER yiaddr=192.168.0.12 chaddr=F9:BC:07:82:87:49 router=0.0.0 subnetmask=255.255.255.0 dnsserver=0.0.0 serverident=192.168.0.12
3	13:45:00.029	0.0.0.0	192.168.0.126	ARP	Vermittlung	Suche nach MAC für 192.168.0.126 [op=REQUEST, sender=F9:BC:07:82:87:49 0.0.0.0, target=FF:FF:FF:FF:192.168.0.126]
4	13:45:00.035	192.168.0.11:67	255.255.255.255:68	DHCP	Anwendung	DHCPOFFER yiaddr=192.168.0.13 chaddr=F9:BC:07:82:87:49 router=0.0.0 subnetmask=255.255.255.0 dnsserver=0.0.0 serverident=192.168.0.11
5	13:45:00.279	0.0.0.0:68	255.255.255.255:67	DHCP	Anwendung	DHCPRREQUEST yiaddr=0.0.0 chaddr=F9:BC:07:82:87:49 requested=192.168.0.126 serverident=192.168.0.12
6	13:45:00.306	192.168.0.12:67	255.255.255.255:68	DHCP	Anwendung	DHCPCACK yiaddr=192.168.0.12 chaddr=F9:BC:07:82:87:49 serverident=192.168.0.12

SAT und Routingtabellen

Switches und Router verwenden Tabellen zur gezielten Kommunikation mit Netzwerkgeräten.

Bei Switches wird dabei eine **Source Address Table (SAT)** angelegt. Sobald ein Gerät an einem der Eingänge (**Ports**) des Switches eine Nachricht sendet, wird dessen MAC-Adresse und die zugehörige Portnummer in diese Tabelle eingetragen. Falls die Zieladresse der Nachricht in der Tabelle enthalten ist, wird die Nachricht über den zugehörigen Port weitergeleitet, ansonsten erhalten alle Ports (abgesehen von dem, durch den die Nachricht empfangen wurde) die Nachricht weitergeleitet.

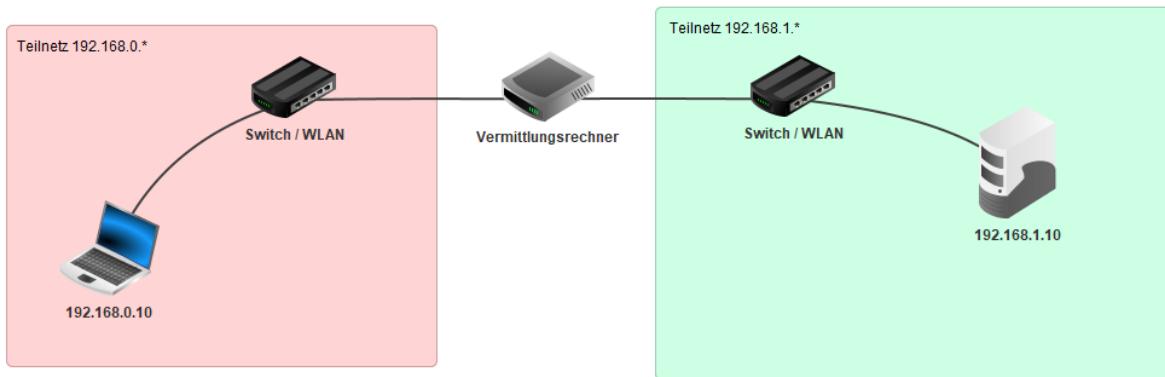
SAT Tabelle Switch / WLAN		
MAC	Port	Letzte Aktualisierung
F4:3E:45:AC:B2:39	Port 4	13:56:08
F9:BC:07:82:87:49	Port 1	13:56:08
BA:26:B0:C3:E5:53	Port 2	13:56:08

Im DHCP-Beispiel hatte das Gerät 192.168.0.10 noch keine Kommunikation im Netzwerk. Deshalb fehlt es noch in der SAT des enthaltenen Switches.

Port 1 entspricht hier dem Rechner „neues Gerät“, Port 2 dem DHCP-Server mit Endung 0.11 und an Port 4 hängt der DHCP-Server mit Endung 0.12.

3.5. Netzwerke

Router verwenden **Routingtabellen** (auch **Weiterleitungstabellen** genannt).



Im Beispiel hier verbindet der Router die beiden angegebenen Teilnetze. Er besitzt zwei Gateways, also Zugangsadressen für die Netzwerkgeräte des jeweiligen Teilnetzes. Im Beispiel wurde für das erste Teilnetz als Gateway 192.168.0.1 gewählt, für das zweite Teilnetz 192.168.1.1. Diese Gateways werden bei den Rechnern auch als Standardgateway in der Konfiguration eingetragen.

Will nun bspw. Gerät 192.168.0.10 das Gerät 192.168.1.10 anpingen, so sendet es die Nachricht über seinen Switch an das Gateway 192.168.0.1 des Routers. Dieser erkennt, anhand der Routingtabelle, dass die Nachricht über das Gateway 192.168.1.1 weitergeleitet werden muss und leitet sie entsprechend in das Teilnetz weiter, wo sie über den Switch an 192.168.1.10 gelangt.

Weiterleitungstabelle			
Ziel	Netzmaske	Nächstes Gateway	Über Schnittstelle
192.168.1.1	255.255.255.255	127.0.0.1	127.0.0.1
192.168.0.1	255.255.255.255	127.0.0.1	127.0.0.1
192.168.1.0	255.255.255.0	192.168.1.1	192.168.1.1
192.168.0.0	255.255.255.0	192.168.0.1	192.168.0.1
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1

Vorteil hier: durch die Verwendung der für das Teilnetz reservierten IP-Adresse (Zeilen 3 und 4 der Tabelle) muss der Router nicht alle Netzwerkgeräte des Teilnetzes kennen.

Dijkstra-Verfahren

Wie finden die Nachrichten schnellstmöglich ihren Weg durch das Internet? Dafür gibt es verschiedene Verfahren. Eines davon ist das Dijkstra-Verfahren, benannt nach dem Niederländer Edsger W. Dijkstra.

Ausgangssituation ist ein gewichteter Graph mit positiven Kantengewichten (z.B. der benötigten Dauer für eine Nachricht zwischen den beiden Netzwerkgeräten, die als Knoten dargestellt werden).

Ziel ist es, den bezüglich dieser Gewichte kürzesten Pfad von einem Startknoten zu einem Zielknoten im Graph zu finden.

Dafür erhält erstmal jeder Knoten den Abstandswert von ∞ zugewiesen mit Ausnahme des Startknotens (dieser erhält logischerweise den Wert 0). Es wird eine Liste geführt mit noch zu besuchenden Knoten, die stets nach dem Abstandswert aufsteigend sortiert wird.

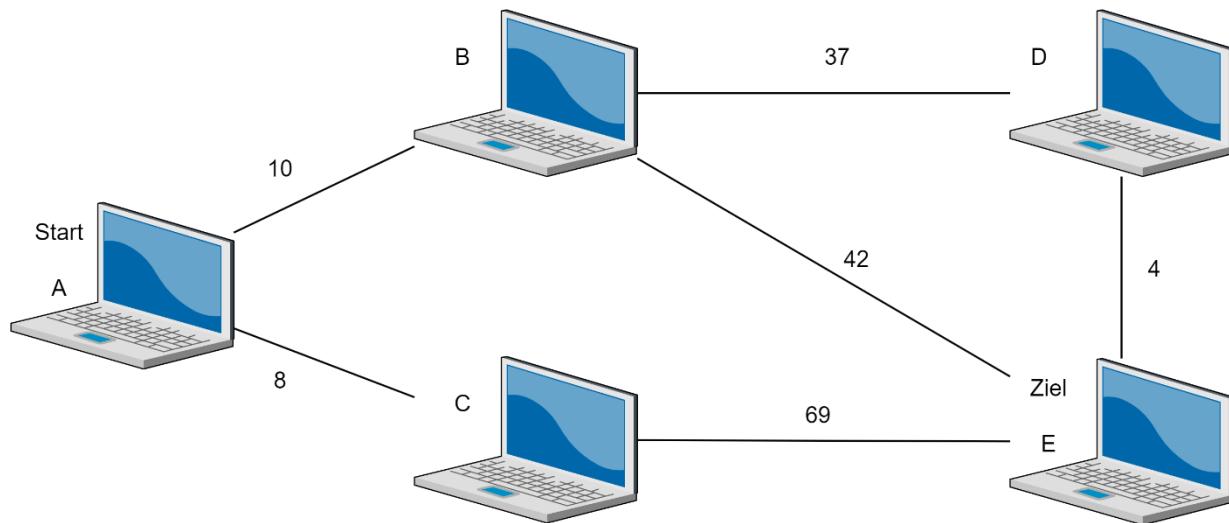
Zu Beginn steht nun in der Liste der Startknoten mit Abstand 0 als erstes Element. Es werden alle direkt mit diesem Knoten verbundenen Knoten „besucht“. Dabei wird deren Abstand zum gerade betrachteten Knoten (also zu Beginn dem Startknoten) zum Abstandswert des gerade betrachteten

3.5. Netzwerke

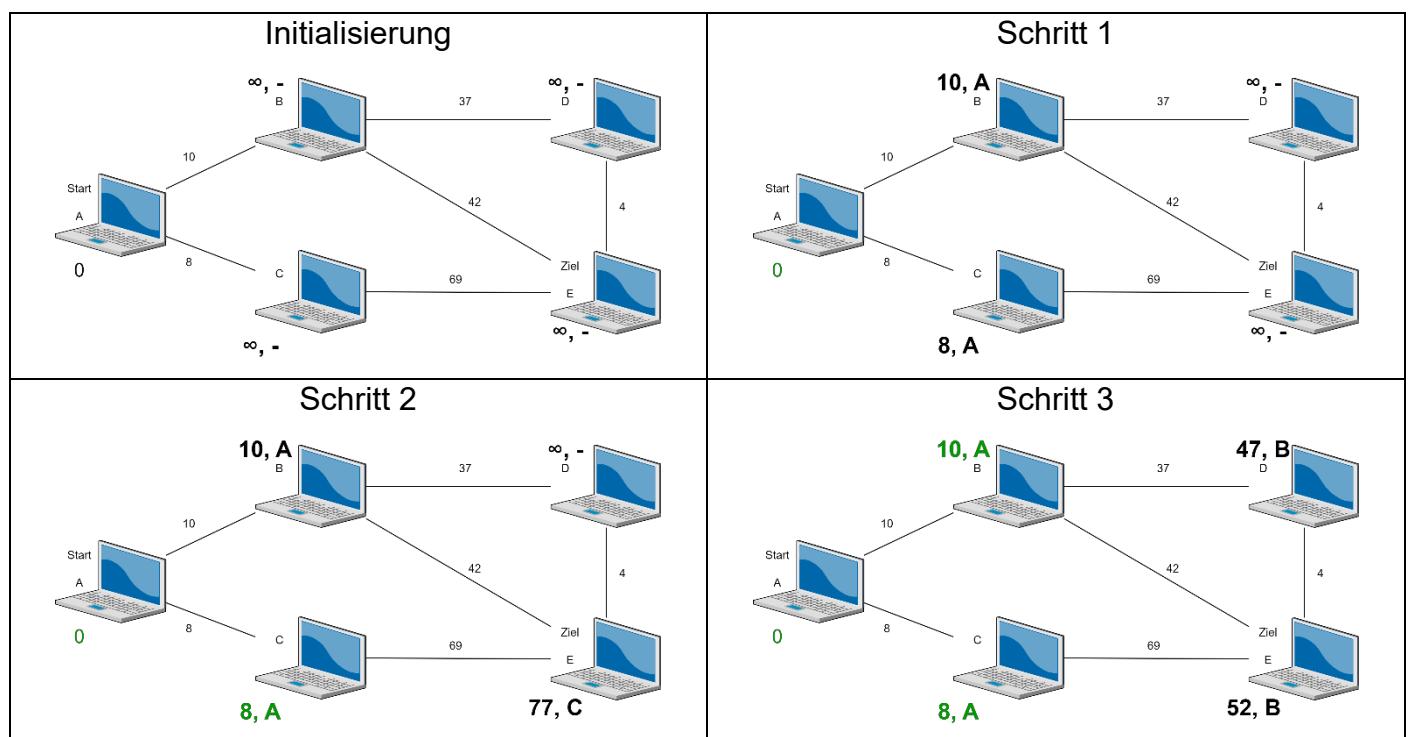
Knotens addiert. Ist der Wert kleiner als der bisher beim Knoten notierte Wert, so wird der Wert dieses Knotens angepasst auf die errechnete Summe (und ggf. mitgespeichert, über welchen Knoten dieses Teilziel erreicht wurde). Wurde dieses Prinzip mit allen Verbindungen des gerade betrachteten Knotens durchgeführt, wird der betrachtete Knoten aus der Liste der noch abzuarbeitenden Knoten entfernt und diese ggf. wieder neu sortiert, sodass als nächstes wieder der Knoten mit dem kleinsten Abstand zu betrachten ist.

Dieses Verfahren endet, wenn kein Knoten mehr übrig ist oder der Zielknoten der nächste zu betrachtende Knoten ist (dann gibt es aufgrund der positiven Kantengewichte keine schnellere Verbindung als die bisher gefundene).

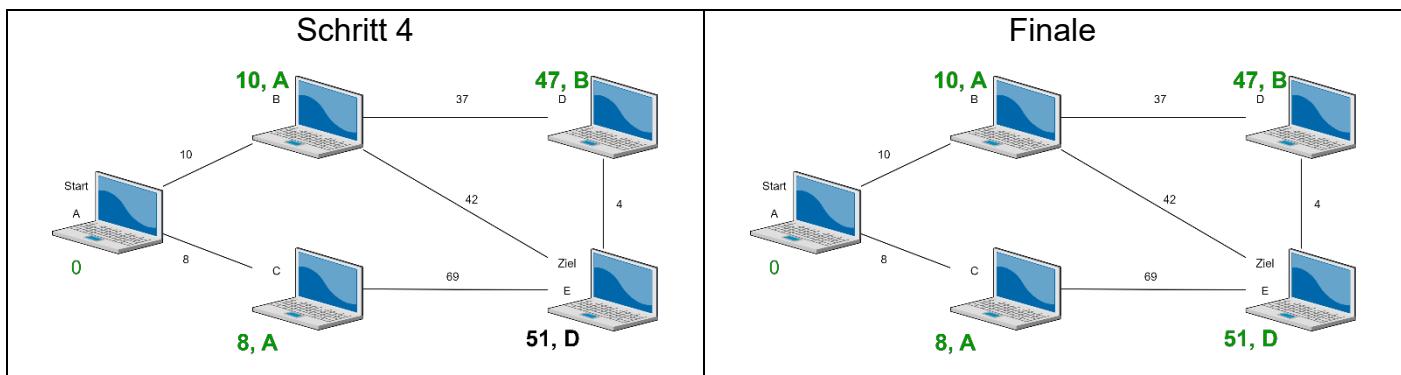
Beispiel



Gesucht wird der schnellste Weg von A zum Knoten E.



3.5. Netzwerke



Anhand der notierten Knoten kann der kürzeste Weg ermittelt werden:

$A \rightarrow B \rightarrow D \rightarrow E$ mit einer Gesamtlänge von 51 Einheiten.

Vermittlungsverfahren

Wir unterscheiden zwei wesentliche Vermittlungsverfahren: Nachrichten- und Paketvermittlung.

Bei der **Nachrichtenvermittlung** wird jede Nachricht vollständig von einem Knoten zum nächsten Knoten übertragen. Dadurch wird die Nachricht an jedem Knoten auf dem Weg zwischen Start- und Zielknoten vollständig zwischengespeichert. Diese Vermittlung sichert ab, dass stets die gesamte Nachricht vorliegt, benötigt aber im Regelfall mehr Speicher und Zeit.

Bei der **Paketvermittlung** wird jede Nachricht in Pakete fester Größe aufgeteilt. Diese Pakete werden dann einzeln durch das Netzwerk gesendet über den jeweils besten zur Zeit verfügbaren Pfad. Häufig wird dadurch weniger Zeit und Speicher benötigt, jedoch muss die Nachricht beim Zielknoten noch zusammengeführt werden. Fehlt ein Paket, muss nur dieses Paket nachgefordert werden beim Startknoten (und nicht die gesamte Nachricht). Das Konzept, wonach jedes Paket, ein sogenanntes Datagramm, als eigenständige Einheit betrachtet wird und unabhängig von anderen Paketen durch das Netzwerk gesendet wird, bezeichnen wir als **Datagrammprinzip**.

TCP / UDP

Es gibt zwei wesentlich unterschiedliche Ansätze der Art und Weise wie Daten zwischen Netzwerkkomponenten ausgetauscht werden sollen.

Das **Transmission Control Protocol (TCP)** ist ein verbindungsorientiertes Protokoll. Dabei wird über den sogenannten TCP-Handshake eine Verbindung mit Rückmeldungen aufgebaut. Der Client sendet ein SYN-Paket (synchronize) an den Server. Ist dieser bereit, über den gewünschten **Socket** (Verbindung aus IP-Adresse und Port) mit dem Client zu kommunizieren, sendet er ein SYN/ACK-Paket an den Client (acknowledgement). Der Client bestätigt den Empfang des SYN/ACK-Pakets durch Senden eines ACK-Pakets. Nun ist die Verbindung zwischen beiden geöffnet und der Datenaustausch kann solange durchgeführt werden, bis beide Seiten durch Senden des FIN-Pakets (finish) die Verbindung trennen (auch hier üblicherweise mit Bestätigungen durch entsprechende ACK-Pakete).

3.5. Netzwerke

Beispiel:

5	11:13:00.133	69.42.111.3:49073	111.69.42.27:80	TCP	Transport	SYN, SEQ: 3,000,000
6	11:13:00.819	111.69.42.27:80	69.42.111.3:49073	TCP	Transport	SYN, SEQ: 2,000,000, ACK: 3,000,001
7	11:13:00.820	69.42.111.3:49073	111.69.42.27:80	TCP	Transport	SEQ: 3,000,001, ACK: 2,000,001
8	11:13:00.878	69.42.111.3:49073	111.69.42.27:80	HTTP	Anwendung	GET / HTTP/1.1 Host: lkurs.de

Der Client sendet vom Socket 69.42.111.3:49073 aus in Zeile 5 die Verbindungsanfrage per SYN-Paket (mit Sequenznummer 3000000) an den Socket 111.69.42.27:80. Dieser bestätigt (ACK: 30000001) und sendet seinerseits ebenso ein SYN-Paket (2000000) an den Client-Socket (Zeile 6). Durch dessen Bestätigung mit ACK 20000001 (Zeile 7) weiß der Server-Client, dass die Verbindung erfolgreich aufgebaut ist, sodass die eigentliche Kommunikation (Zeile 8) beginnen kann.

22	11:13:03.843	111.69.42.27:80	69.42.111.3:49073	HTTP	Anwendung	cK4FHWZZnSvmW50hHirdMT9a/TvN9hy/3v4+eImV9Jtju3W+l4Lm...
23	11:13:03.845	69.42.111.3:49073	111.69.42.27:80	TCP	Transport	SEQ: 3,000,081, ACK: 2,007,880
24	11:13:04.306	111.69.42.27:80	69.42.111.3:49073	HTTP	Anwendung	VOyHM6E6aAaAzJuguRbHmlialnixo3t2GOPdSd8YTi5gQ01/ocvZ...
25	11:13:04.307	69.42.111.3:49073	111.69.42.27:80	TCP	Transport	SEQ: 3,000,081, ACK: 2,009,018
26	11:13:04.363	69.42.111.3:49073	111.69.42.27:80	TCP	Transport	FIN, SEQ: 3,000,081
27	11:13:04.817	111.69.42.27:80	69.42.111.3:49073	TCP	Transport	SEQ: 2,009,018, ACK: 3,000,082
28	11:13:04.875	111.69.42.27:80	69.42.111.3:49073	TCP	Transport	FIN, SEQ: 2,009,018
29	11:13:04.875	69.42.111.3:49073	111.69.42.27:80	TCP	Transport	SEQ: 3,000,082, ACK: 2,009,019

Wie in den Zeilen 23 und 25 sichtbar, sendet der Client stets die Bestätigung für die erhaltenen Daten an den Server. Nachdem die Daten insgesamt erfolgreich übertragen wurden, meldet der Client (Zeile 26) über das FIN-Paket das gewünschte Ende der Verbindung. Der Server bestätigt dies und sendet seinerseits an den Client das ebenfalls gewünschte Ende der Verbindung (Zeilen 27 und 28). Nachdem diese Nachricht beim Client ankam, sendet er die letzte Bestätigung an den Server (Zeile 29).

Diese Art der Verbindung sichert ab, dass die Pakete erfolgreich zwischen Server und Client übertragen wurden. Fehlt eine Bestätigung, kann das zugehörige Paket nochmal gesendet werden. Durch die Bestätigungen wird jedoch auch mehr Zeit für die Übertragung benötigt, was beim Versand großer Datenmengen (etwa bei VoIP oder Gaming) problematisch sein kann.

Das **User Datagram Protocol (UDP)** ist im Gegensatz zum TCP ein verbindungsloses Protokoll. Es verwendet das Datagrammprinzip. Jedes Datagramm enthält den Quell- und Zielport, eine Angabe über die Länge des Datagramms, ggf. eine Prüfsumme (16 Bit, wenn nicht gewünscht, wird die Prüfsumme auf 0 gesetzt) und die eigentlichen Daten.

Dadurch, dass im Gegensatz zu TCP keine Empfangsbestätigungen übermittelt werden, ist die Übertragung der Daten schneller möglich, jedoch mit Verlust der Sicherheit behaftet, dass die Pakete angekommen sind.

3.5. Netzwerke

Nr.	Zeit	Quelle	Ziel	Protokoll	Schicht	Bemerkungen / Details
1	11:31:20.864	0.0.0.0:68	255.255.255.255:67	DHCP	Anwendung	DHCPDISCOVER yiaddr=0.0.0.0 chaddr=F9:BC:07:82:87:49
2	11:31:21.111	192.168.0.12:67	255.255.255.255:68	DHCP	Anwendung	DHCPOFFER yiaddr=192.168.0.126 chaddr=F9:BC:07:82:87:49
3	11:31:21.112	0.0.0.0	192.168.0.126	ARP	Vermittlung	Suche nach MAC für 192.168.0.126 [op=REQUEST, sender=...]
4	11:31:21.168	192.168.0.11:67	255.255.255.255:68	DHCP	Anwendung	DHCPOFFER yiaddr=192.168.0.13 chaddr=F9:BC:07:82:87:49
5	11:31:23.863	0.0.0.0:68	255.255.255.255:67	DHCP	Anwendung	DHCPREQUEST yiaddr=0.0.0.0 chaddr=F9:BC:07:82:87:49 r...
6	11:31:24.088	192.168.0.12:67	255.255.255.255:68	DHCP	Anwendung	DHCPACK yiaddr=192.168.0.126 chaddr=F9:BC:07:82:87:49...

Nr.: 4 / Zeit: 11:31:21.168
Netzzugang
Quelle: BA:26:B0:C3:E5:53
Ziel: FF:FF:FF:FF:FF:FF
Bemerkungen / Details: 0x800
Vermittlung
Quelle: 192.168.0.11
Ziel: 255.255.255.255
Protokoll: IP
Bemerkungen / Details: Protokoll: 17, TTL: 1
Transport
Quelle: 67
Ziel: 68
Protokoll: UDP
Anwendung
Protokoll: DHCP
Bemerkungen / Details (137 Bytes):

Im Beispiel sehen wir, dass das DHCP-Protokoll zur Übertragung UDP verwendet (siehe Details bei „Transport“ → „Protokoll“).

3.5.3. Netzwerkdienste und Sicherheit

Als **Netzwerkdienst** bezeichnen wir eine in sich geschlossene Funktion, die von einem Computernetzwerk bereitgestellt wird.

Wir unterscheiden dabei in zentrale und dezentrale Netzwerkdienste.

Bei **zentralen Netzwerkdiensten** wird der Dienst von einem oder wenigen zentralen Servern bereitgestellt und verwaltet.

Bsp.: E-Mail, DNS, FTP, HTTP(S)

Bei **dezentralen Netzwerkdiensten** wird der Dienst auf viele Server verteilt, die voneinander unabhängig arbeiten können.

Bsp.: Blockchain

Je nach Implementierung können Dienste auch zentral oder dezentral sein, z.B. Messenger-Dienste (zentral: WhatsApp, Instagram, Snapchat; dezentral: Quicksy, FluffyChat).

E-Mail

Im Zusammenhang mit der elektronischen Kommunikation über E-Mail liest man häufig von drei Protokollen: POP3, IMAP und SMTP

Bei **POP3 (Post Office Protocol Version 3)** und **IMAP (Internet Message Access Protocol)** handelt es sich um Protokolle zum Abrufen von E-Mails bzw. einem Netzwerkdateisystem für E-Mails.

POP3

Nutzt man POP3, so werden die E-Mails vom E-Mail-Server abgerufen und dort im Regelfall gelöscht. Dies führt dazu, dass beim Zugriff über einen anderen Client die bereits abgerufene E-Mail nicht mehr vorliegt und bietet sich daher eher für Nutzer an, die nur einen Client nutzen wollen oder deren Speicherplatz auf dem E-Mail-Server gering ist. POP3 verwendet üblicherweise Port 110 (unverschlüsselt) oder 995 (verschlüsselt). Bei der textbasierten Kommunikation zwischen Server und Client wird bei POP3 das Passwort des Nutzers im Klartext übertragen. Durch Verwendung von SSL/TLS kann dieses Manko behoben werden.

3.5. Netzwerke

3	09:44:43.470	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	SYN, SEQ: 7,000,000
4	09:44:43.937	42.69.111.27:110	69.42.111.3:54541	TCP	Transport	SYN, SEQ: 4,000,000, ACK: 7,000,001
5	09:44:43.937	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	SEQ: 7,000,001, ACK: 4,000,001
6	09:44:44.311	42.69.111.27:110	69.42.111.3:54541	POP3	Anwendung	+OK POP3 server ready
7	09:44:44.311	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	SEQ: 7,000,001, ACK: 4,000,022
8	09:44:44.370	69.42.111.3:54541	42.69.111.27:110	POP3	Anwendung	USER lennox
9	09:44:44.717	42.69.111.27:110	69.42.111.3:54541	TCP	Transport	SEQ: 4,000,022, ACK: 7,000,012
10	09:44:44.774	42.69.111.27:110	69.42.111.3:54541	POP3	Anwendung	+OK enter password
11	09:44:44.775	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	SEQ: 7,000,012, ACK: 4,000,040
12	09:44:44.833	69.42.111.3:54541	42.69.111.27:110	POP3	Anwendung	PASS handball
13	09:44:45.181	42.69.111.27:110	69.42.111.3:54541	TCP	Transport	SEQ: 4,000,040, ACK: 7,000,025
14	09:44:45.239	42.69.111.27:110	69.42.111.3:54541	POP3	Anwendung	+OK Mailbox locked and ready
15	09:44:45.240	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	SEQ: 7,000,025, ACK: 4,000,068
16	09:44:45.298	69.42.111.3:54541	42.69.111.27:110		Anwendung	STAT
17	09:44:45.646	42.69.111.27:110	69.42.111.3:54541	TCP	Transport	SEQ: 4,000,068, ACK: 7,000,029
18	09:44:45.703	42.69.111.27:110	69.42.111.3:54541	POP3	Anwendung	+OK 0 0
19	09:44:45.703	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	SEQ: 7,000,029, ACK: 4,000,075
20	09:44:45.762	69.42.111.3:54541	42.69.111.27:110	POP3	Anwendung	QUIT
21	09:44:46.113	42.69.111.27:110	69.42.111.3:54541	TCP	Transport	SEQ: 4,000,075, ACK: 7,000,033
22	09:44:46.172	42.69.111.27:110	69.42.111.3:54541	POP3	Anwendung	+OK
23	09:44:46.173	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	SEQ: 7,000,033, ACK: 4,000,078
24	09:44:46.232	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	FIN, SEQ: 7,000,033
25	09:44:46.519	42.69.111.27:110	69.42.111.3:54541	TCP	Transport	FIN, SEQ: 4,000,078
26	09:44:46.520	69.42.111.3:54541	42.69.111.27:110	TCP	Transport	SEQ: 7,000,034, ACK: 4,000,079
27	09:44:46.577	42.69.111.27:110	69.42.111.3:54541	TCP	Transport	SEQ: 4,000,079, ACK: 7,000,034

Im Beispiel sieht man das Abrufen eines E-Mail-Kontos über POP3. Es meldet sich der Nutzer lennox (Zeile 8) mit dem Kennwort handball (Zeile 12) an. Anhand von Zeile 18 sehen wir, dass keine E-Mail vorliegt.

15	09:48:27.280	42.69.111.27:110	69.42.111.3:19453	TCP	Transport	SEQ: 10,000,068, ACK: 13,000,029
16	09:48:27.336	42.69.111.27:110	69.42.111.3:19453	POP3	Anwendung	+OK 1 145
17	09:48:27.336	69.42.111.3:19453	42.69.111.27:110	TCP	Transport	SEQ: 13,000,029, ACK: 10,000,077
18	09:48:27.392	69.42.111.3:19453	42.69.111.27:110		Anwendung	LIST
19	09:48:27.734	42.69.111.27:110	69.42.111.3:19453	TCP	Transport	SEQ: 10,000,077, ACK: 13,000,033
20	09:48:27.791	42.69.111.27:110	69.42.111.3:19453	POP3	Anwendung	+OK 1 145 0 145
21	09:48:27.792	69.42.111.3:19453	42.69.111.27:110	TCP	Transport	SEQ: 13,000,033, ACK: 10,000,093
22	09:48:27.848	69.42.111.3:19453	42.69.111.27:110	POP3	Anwendung	RETR 0
23	09:48:28.198	42.69.111.27:110	69.42.111.3:19453	TCP	Transport	SEQ: 10,000,093, ACK: 13,000,039
24	09:48:28.254	42.69.111.27:110	69.42.111.3:19453	POP3	Anwendung	+OK message follows From: Timon <timon@lkurs.de> To: <lennox@lkurs.de> Sub...
25	09:48:28.256	69.42.111.3:19453	42.69.111.27:110	TCP	Transport	SEQ: 13,000,039, ACK: 10,000,279
26	09:48:28.312	69.42.111.3:19453	42.69.111.27:110	POP3	Anwendung	DELE 0
27	09:48:28.653	42.69.111.27:110	69.42.111.3:19453	TCP	Transport	SEQ: 10,000,279, ACK: 13,000,045
28	09:48:28.710	42.69.111.27:110	69.42.111.3:19453	POP3	Anwendung	+OK message marked for delete
29	09:48:28.713	69.42.111.3:19453	42.69.111.27:110	TCP	Transport	SEQ: 13,000,045, ACK: 10,000,308
30	09:48:28.769	69.42.111.3:19453	42.69.111.27:110	POP3	Anwendung	QUIT
31	09:48:29.114	42.69.111.27:110	69.42.111.3:19453	TCP	Transport	SEQ: 10,000,308, ACK: 13,000,049

```
+OK message follows
From: Timon <timon@lkurs.de>
To: <lennox@lkurs.de>
Subject: Testnachricht

Lieber Lennox,
```

viel Erfolg beim nächsten Handballspiel wünsch ich dir.

Liebe Grüße
Timon

Bei diesem Beispiel lag eine Mail vor (Zeile 16), die dann angefordert wird (Zeile 22) und vom POP3-Server an den Client gesendet (Zeile 24) und vom Server gelöscht (Zeile 26) wird.

IMAP

Das IMAP-Protokoll ist ebenfalls ein textbasiertes Protokoll zum Zugriff auf E-Mails. Der Client stellt Anfragen nach den aktuell benötigten Informationen (Gibt es neue Nachrichten? Welche Nachrichten befinden sich im Ordner ...? Bitte lösche die Mail ...! Bitte verschiebe die Mail ... in den Ordner ...; Bitte kennzeichne die Mail ... als (un)gelesen!). Die Nachrichten werden auf dem Server behalten, sodass sie bei Nutzung mehrerer Clients (Heim-PC, Leih-iPad, Smartphone, ...) stets identisch organisiert sind. IMAP verwendet standardmäßig den Port 143 (unverschlüsselt) bzw. 993 (verschlüsselt).

Der Nutzer muss sich authentifizieren, bevor er Zugriff auf die Mails und Ordner erhält. Auch bei IMAP muss ggf. mit SSL-Verschlüsselung gearbeitet werden, wenn das Kennwort nicht im Klartext übertragen werden soll.

3.5. Netzwerke

SMTP

Das **Simple Mail Transfer Protocol (SMTP)** wird zum Versenden von E-Mails genutzt.

Da SMTP ebenfalls ein textbasiertes Protokoll ist, kann z.B. die Absenderadresse manipuliert werden. Prinzipiell antwortet der Server mit dreistelligen Statusnummern und kurzen Texten, auf die der Client reagiert. Standardmäßig ist für SMTP der Port 25 vorgesehen, es können jedoch mit SSL/TLS auch 465 bzw. mit STARTTLS 587 als Ports häufig genutzt werden, sodass die Nachrichten verschlüsselt übertragen werden.

1	10:10:29.300	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SYN, SEQ: 5,000,000
2	10:10:29.337	42.69.111.27:25	69.42.111.3:24525	TCP	Transport	SYN, SEQ: 4,000,000, ACK: 5,000,001
3	10:10:29.337	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SEQ: 5,000,001, ACK: 4,000,001
4	10:10:29.456	42.69.111.27:25	69.42.111.3:24525	SMTP	Anwendung	220 Willkommen bei lkurs.de
5	10:10:29.456	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SEQ: 5,000,001, ACK: 4,000,028
6	10:10:29.462	69.42.111.3:24525	42.69.111.27:25	SMTP	Anwendung	HELO 69.42.111.3
7	10:10:29.497	42.69.111.27:25	69.42.111.3:24525	TCP	Transport	SEQ: 4,000,028, ACK: 5,000,017
8	10:10:29.504	42.69.111.27:25	69.42.111.3:24525	SMTP	Anwendung	250 Hello 69.42.111.3
9	10:10:29.504	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SEQ: 5,000,017, ACK: 4,000,049
10	10:10:29.512	69.42.111.3:24525	42.69.111.27:25	SMTP	Anwendung	MAIL FROM: <lennox@lkurs.de>
11	10:10:29.548	42.69.111.27:25	69.42.111.3:24525	TCP	Transport	SEQ: 4,000,049, ACK: 5,000,045
12	10:10:29.555	42.69.111.27:25	69.42.111.3:24525	SMTP	Anwendung	250 Sender OK
13	10:10:29.555	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SEQ: 5,000,045, ACK: 4,000,062
14	10:10:29.561	69.42.111.3:24525	42.69.111.27:25	SMTP	Anwendung	RCPT TO:<timon@lkurs.de>
15	10:10:29.598	42.69.111.27:25	69.42.111.3:24525	TCP	Transport	SEQ: 4,000,062, ACK: 5,000,069
16	10:10:29.604	42.69.111.27:25	69.42.111.3:24525	SMTP	Anwendung	250 Recipient OK
17	10:10:29.604	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SEQ: 5,000,069, ACK: 4,000,078
18	10:10:29.611	69.42.111.3:24525	42.69.111.27:25	SMTP	Anwendung	DATA
19	10:10:29.647	42.69.111.27:25	69.42.111.3:24525	TCP	Transport	SEQ: 4,000,078, ACK: 5,000,073
20	10:10:29.654	42.69.111.27:25	69.42.111.3:24525	SMTP	Anwendung	354 End data with <CR><LF>.<CR><LF>
21	10:10:29.654	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SEQ: 5,000,073, ACK: 4,000,113
22	10:10:29.661	69.42.111.3:24525	42.69.111.27:25	SMTP	Anwendung	From: Lennox <lennox@lkurs.de> To: Timon <timon@lkurs.d...
23	10:10:29.697	42.69.111.27:25	69.42.111.3:24525	TCP	Transport	SEQ: 4,000,113, ACK: 5,000,390
24	10:10:29.702	42.69.111.27:25	69.42.111.3:24525	SMTP	Anwendung	250 Mail queued for delivery
25	10:10:29.702	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SEQ: 5,000,390, ACK: 4,000,141
26	10:10:29.709	69.42.111.3:24525	42.69.111.27:25	SMTP	Anwendung	QUIT
27	10:10:29.748	42.69.111.27:25	69.42.111.3:24525	TCP	Transport	SEQ: 4,000,141, ACK: 5,000,394
28	10:10:29.755	42.69.111.27:25	69.42.111.3:24525	SMTP	Anwendung	221 Server beendet Verbindung.
29	10:10:29.755	69.42.111.3:24525	42.69.111.27:25	TCP	Transport	SEQ: 5,000,394, ACK: 4,000,171

Beispiel: Vom Client 69.42.111.3, auf dem der Nutzer lennox@lkurs.de angemeldet ist, wird eine Mail an timon@lkurs.de (Zeile 14) gesendet. Dafür kontaktiert der Client den Server über das TCP-Protokoll (Zeile 1), dieser antwortet (Zeile 4). Nachdem die Verbindung aufgebaut wurde (Zeile 8), wird die Mail gesendet (Zeilen 10 bis 24) und die Verbindung wieder getrennt (Zeilen 26 und 28).

3.5. Netzwerke

DNS

Der **Domain Name System**-Server (kurz **DNS**-Server) übersetzt menschenlesbare Domainnamen in maschinenlesbare IP-Adressen, damit Computer z.B. Webserver finden und mit ihnen kommunizieren können.

Nr.	Zeit	Quelle	Ziel	Protokoll	Schicht	Bemerkungen / Details
1	10:35:56.757	69.42.111.3	69.42.111.1	ARP	Vermittlung	Suche nach MAC für 69.42.111.1 [op=REQUEST, sender=7...
2	10:35:56.794	69.42.111.1	69.42.111.3	ARP	Vermittlung	MAC ist EE:49:15:1C:FD:31 [op=REPLY, sender=EE:49:15...
3	10:35:56.795	69.42.111.3:57373	42.69.111.27:53	DNS	Anwendung	ID=32126 QR=0 RCODE=0 QDCOUNT=1 ANCOUNT=0 NSCOUNT=0 ...
4	10:35:56.851	42.69.111.27:53	69.42.111.3:57373	DNS	Anwendung	ID=32126 QR=1 RCODE=0 QDCOUNT=0 ANCOUNT=1 NSCOUNT=0 ...

Nr.: 4 / Zeit: 10:35:56.851

- Netzzugang
 - Quelle: EE:49:15:1C:FD:31
 - Ziel: 71:87:FC:C2:3F:50
 - Bemerkungen / Details: 0x800
- Vermittlung
 - Quelle: 42.69.111.27
 - Ziel: 69.42.111.3
 - Protokoll: IP
 - Bemerkungen / Details: Protokoll: 17, TTL: 63
- Transport
 - Quelle: 53
 - Ziel: 57373
 - Protokoll: UDP
- Anwendung
 - Protokoll: DNS
 - Bemerkungen / Details (93 Bytes):
 - ID=32126 QR=1 RCODE=0 QDCOUNT=0 ANCOUNT=1 NSCOUNT=0 ARCOUNT=0
 - lkurs.de. A 3600 111.69.42.27

Im Beispiel fragt der Client 69.42.111.3 den DNS-Server 42.69.111.27 nach der Adresse von lkurs.de (Zeile 3) und erhält die detaillierter abgedruckte Antwort (Zeile 4), dass dieser die Adresse 111.69.42.27 besitzt (letzte Information bei Bemerkungen im Fenster unten). Im Anschluss kann er mit dem gewünschten Webserver kommunizieren.

Webseiten

Das **Hypertext Transfer Protocol (HTTP)** dient der Übertragung von Webseiten vom Webserver zum Browser des Clients. Fragt ein Client eine Internetseite an, so wird eine Kopie der auf dem Webserver befindlichen Datei an ihn gesendet. Der Browser interpretiert diese dann (fordert ggf. benötigte Dateien an) und stellt sie dar.

Nr.	Zeit	Quelle	Ziel	Protokoll	Schicht	Bemerkungen / Details
5	10:35:56.851	69.42.111.3:45487	111.69.42.27:80	TCP	Transport	SYN, SEQ: 3,000,000
6	10:35:56.931	111.69.42.27:80	69.42.111.3:45487	TCP	Transport	SYN, SEQ: 2,000,000, ACK: 3,000,001
7	10:35:56.931	69.42.111.3:45487	111.69.42.27:80	TCP	Transport	SEQ: 3,000,001, ACK: 2,000,001
8	10:35:56.985	69.42.111.3:45487	111.69.42.27:80	HTTP	Anwendung	GET / HTTP/1.1 Host: lkurs.de
9	10:35:57.039	111.69.42.27:80	69.42.111.3:45487	TCP	Transport	SEQ: 2,000,001, ACK: 3,000,031
10	10:35:57.045	111.69.42.27:80	69.42.111.3:45487	HTTP	Anwendung	HTTP/1.1 200 OK Content-type: text/html <html> <head>
11	10:35:57.045	69.42.111.3:45487	111.69.42.27:80	TCP	Transport	SEQ: 3,000,031, ACK: 2,000,580
12	10:35:57.065	69.42.111.3:45487	111.69.42.27:80	HTTP	Anwendung	GET splashscreen-mini.png HTTP/1.1 Host: lkurs.de
13	10:35:57.115	111.69.42.27:80	69.42.111.3:45487	TCP	Transport	SEQ: 2,000,580, ACK: 3,000,081
14	10:35:57.121	111.69.42.27:80	69.42.111.3:45487	HTTP	Anwendung	HTTP/1.1 200 OK Content-type: image/png iVBORw0KGgo...
15	10:35:57.121	69.42.111.3:45487	111.69.42.27:80	TCP	Transport	SEQ: 3,000,081, ACK: 2,002,040
16	10:35:57.169	111.69.42.27:80	69.42.111.3:45487	HTTP	Anwendung	J285HmHOqGH4aXrASUOR1BnxDMAWb58uasHZSkOTKR8 8kGKILIJ...
17	10:35:57.169	69.42.111.3:45487	111.69.42.27:80	TCP	Transport	SEQ: 3,000,081, ACK: 2,003,500
18	10:35:57.220	111.69.42.27:80	69.42.111.3:45487	HTTP	Anwendung	SsqATIZhcoDx90PpgCh0GF/t4WemOxlDEbnZEcv+EF6yD Z0aXL...
19	10:35:57.220	69.42.111.3:45487	111.69.42.27:80	TCP	Transport	SEQ: 3,000,081, ACK: 2,004,960

↳ Bemerkungen / Details (579 Bytes):

```

HTTP/1.1 200 OK
Content-type: text/html

<html>
  <head>
    <title>Standardseite</title>
  </head>
  <body bgcolor="#ccddff" style="font-family:Verdana; text-align:center;">
    <h2> FILIUS - Webserver </h2>

    <p>Herzlich Willkommen auf dem Webserver der Anwendung FILIUS!</p>

    <p> Diese Seite wurde automatisch mit der Installation des
      Webservers eingerichtet, es lassen sich jedoch auch
      eigene Seiten hier unterbringen. </p>
  
```

In Zeile 8 dieses Beispiels wird die Startseite von lkurs.de vom Client angefordert. Der Server beantwortet diese Aufforderung, indem er in Zeile 10 eine Kopie der Startseite sendet. Wie wir in den mit abgedruckten Details zur Zeile 10 sehen können, wird die Internetseite unverschlüsselt gesendet (wodurch Man-In-The-Middle-Angreifer Änderungen vornehmen könnten).

Nach Interpretation der Seite fordert der Client die enthaltene Grafik splashscreen-mini.png an (Zeile 14) und erhält eine Kopie dieser Grafik (ab Zeile 16).

Zur verschlüsselten Übertragung der Internetseiten kann anstelle von HTTP auch **HTTPS (Hypertext Transfer Protocol Secure)** verwendet werden. Dabei wird SSL/TLS verwendet.

Blockchain

Bei zentralen Netzwerkdiensten kann die Verwaltung der Daten direkt über den Server geschehen. Bei dezentralen Netzwerkdiensten ist das so nicht möglich. Ein Beispiel für einen solchen dezentralen Dienst ist die Verwendung von Blockchains, z.B. bei der Kryptowährung Bitcoin.

Bei Blockchain wird eine Information als Kette von Blöcken dargestellt. Jede Transaktion wird dabei in einem eigenen Block gespeichert und in chronologischer Reihenfolge mit der bestehenden Kette verbunden. So entstehen lange Ketten von Blöcken, welche die gesamte Transaktionshistorie beinhalten. Das sichert ab, dass trotz fehlendem Server keine Transaktion doppelt getätigter werden kann und niemand den Verlauf der Datenbank verändern kann. Die technische Verifizierung der Transaktionen und das Hinzufügen von Blöcken geschieht über den sogenannten Mining-Prozess (dabei werden komplexe mathematische Probleme gelöst).

3.5.4. Netzwerksegmentierung

Netzwerkarchitektur und -segmentierung sind entscheidend für die Organisation und Effizienz von Netzwerken. Als Subnetting bezeichnen wir die Unterteilung eines Netzwerks in kleinere Netze (Subnetze). Dafür gibt es verschiedene Ansätze, von denen wir zwei betrachten: VLANs und VLSM.

Ein **Virtual Local Area Network (VLAN)** dient der logischen Segmentierung eines physischen Netzwerks. Dabei werden Gruppen von Geräten in verschiedenen logischen Netzwerken isoliert, um Sicherheits- und Managementvorteile zu erzielen. Jedes VLAN funktioniert wie ein eigenes, unabhängiges Netzwerk (vgl. getrennte Teilnetzwerke für die verschiedenen Familien innerhalb eines Mehrfamilienhauses).

Variable Length Subnet Mask (VLSM) dient der effizienten Aufteilung eines IP-Adressraums in Subnetze unterschiedlicher Größe, etwa weniger Zugänge für ein Chiemelabor und mehr Zugänge für ein Streamingzimmer innerhalb eines Hauses.

Beispiel:

Ein Unternehmen besitzt das Netzwerk 123.234.251.0/24 und verwendet dieses für Verkauf, Einkauf, Mitarbeiter und Verwaltung. Beim Verkauf sollen 100 Geräte verbunden, beim Einkauf 50, bei den Mitarbeitern 25 und in der Verwaltung 5 Geräte verbunden werden.

Mit dem Netzwerk stehen (ohne Subnetze) theoretisch 254 Geräte zur Verfügung.

Der Verkauf benötigt (inkl. Netz und Broadcast) 102 Adressen. Daher muss ein Subnetz mit 128 Adressen gebildet werden, das durch /25 gekennzeichnet werden kann. Das Subnetz würde als Netzadresse 123.234.251.0/25 erhalten und damit die Adressen von 123.234.251.0 bis 123.234.251.127 belegen. 26 der möglichen Adressen dieses Subnetzes blieben noch unbenutzt.

Der Einkauf benötigt 52 Adressen. Daher muss ein Subnetz mit 64 Adressen gebildet werden, das durch /26 gekennzeichnet werden kann. Das Subnetz würde als Netzadresse 123.234.251.128/26 erhalten und damit die Adressen von 123.234.251.128 bis 123.234.251.191 belegen. 12 der möglichen Adressen dieses Subnetzes blieben noch unbenutzt.

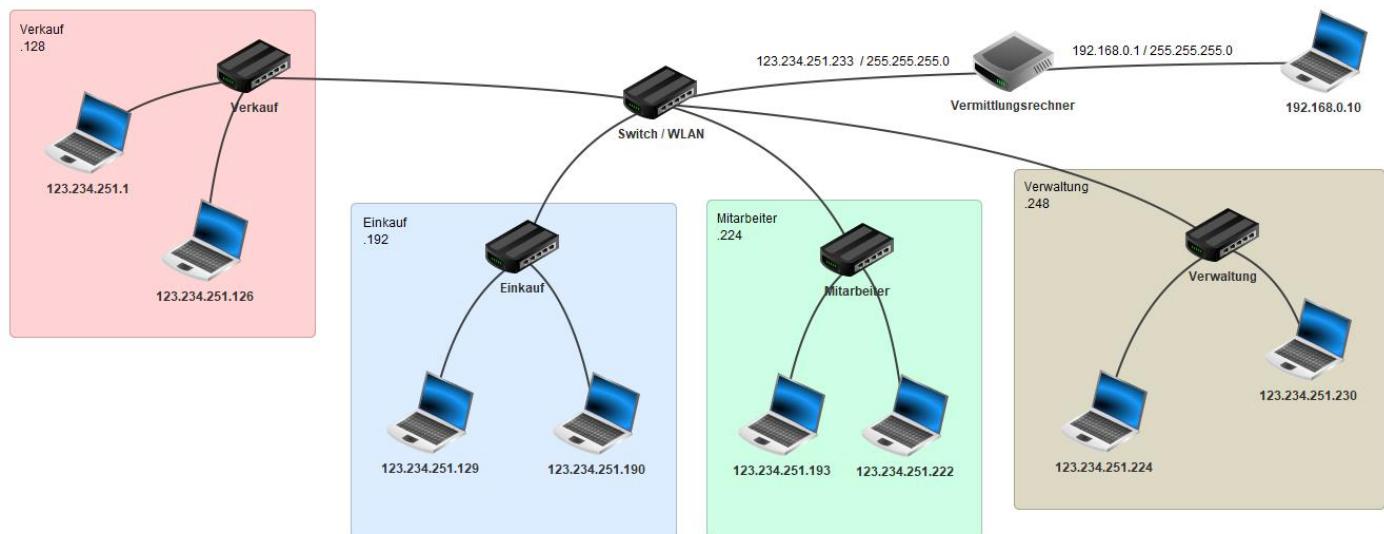
Die Mitarbeiter benötigen 27 Adressen. Daher muss ein Subnetz mit 32 Adressen gebildet werden, das durch /27 gekennzeichnet werden kann. Das Subnetz würde als Netzadresse

3.5. Netzwerke

123.234.251.192/27 erhalten und damit die Adressen von 123.234.251.192 bis 123.234.251.223 belegen. 5 der möglichen Adressen dieses Subnetzes blieben noch unbenutzt.

Die Verwaltung benötigt 7 Adressen. Daher muss ein Subnetz mit 8 Adressen gebildet werden, das durch /29 gekennzeichnet werden kann. Das Subnetz würde als Netzadresse 123.234.251.224/29 erhalten und damit die Adressen von 123.234.251.224 bis 123.234.251.231 belegen. Eine der möglichen Adressen dieses Subnetzes bliebe dabei noch unbenutzt.

Nun stehen noch die 24 Adressen von 123.234.251.232 bis 123.234.251.255 zur Verfügung.



3.6. Kryptologie

Wir unterscheiden fünf Anforderungen an die Informationssicherheit.

Vertraulichkeit	Schutz vor unbefugter Preisgabe von Informationen
Integrität	Schutz vor unbemerkt/unerlaubter Veränderung
Authentizität	Absicherung, dass Information vom angegebenen Autor stammt
Verbindlichkeit	Gewährleistung, dass Versand und Empfang der Informationen nicht in Abrede gestellt werden kann
Verfügbarkeit	Autorisierte Benutzer haben Zugriff zu den Informationen

3.6.1. Datensicherung

Datensicherung bezieht sich auf Maßnahmen und Verfahren, die dazu dienen, Daten vor Verlust zu schützen und deren Wiederherstellung zu ermöglichen.

RAID

Redundant Array of Independent Disks ist eine Technologie, die mehrere Festplatten kombiniert um Datensicherheit, Geschwindigkeit oder beides zu erhöhen

- RAID 0: Verteilt die Daten auf zwei oder mehrere Festplatten → Lese-/Schreibgeschwindigkeit effektiv erhöht
- RAID 1: kopiert identische Daten auf zwei oder mehrere Festplatten → Daten bleiben erhalten, falls eine der Festplatten defekt wird (unter Verlust eines Teils der theoretisch möglichen Speicherkapazität)
- es gibt noch weitere Arten von RAID

Art der Datensicherung

Wir unterscheiden drei unterschiedliche Arten der Datensicherung.

- **Komplettsicherung**: Bei jeder Sicherung werden alle Daten vollständig gesichert.
- **differentielle Datensicherung**: Bei der Sicherung werden alle Daten gesichert, die seit der letzten Komplettsicherung geändert oder hinzugefügt wurden.
- **inkrementelle Datensicherung**: Bei der Sicherung werden alle Daten gesichert, die seit der letzten (ggf. auch differentiellen) Sicherung geändert oder hinzugefügt wurden.

Wie funktioniert die Datensicherung bei git?

Bei jedem Commit vergleicht git die Dateiversionen aller Dateien mit vorherigen Sicherungen. Sind die beiden Versionen identisch (verglichen anhand der SHA-1-Hashprüfung), speichert git einen Verweis zur vorherigen Dateiversion, sonst speichert git die neue Version der Datei.

3.6.2. Datensicherheit

Datensicherheit umfasst Maßnahmen zum Schutz von Daten vor unbefugtem Zugriff, Manipulation und Zerstörung.

Symmetrische Verfahren

Bei **symmetrischen Verschlüsselungsverfahren** besitzen Ver- und Entschlüsselnder denselben Schlüssel. Typische Beispiele sind Cäsar- und Vigenère-Verschlüsselung, AES und DES.

MM-Substitution

Bei **Substitutionsverfahren** werden die Zeichen des Klartexts durch i.d.R. andere Zeichen ersetzt. Unter den Substitutionsverfahren gibt es solche, die **monographisch** arbeiten (wo also jedes Zeichen einzeln ersetzt wird). Ebenso gibt es Verfahren, die **monoalphabetisch** arbeiten (ein Zeichen des Klartexts wird dabei stets durch dasselbe(!) Zeichen des Geheimalphabets ersetzt).

Ein Beispiel für eine monoalphabetisch-monografische Substitution (**MM-Substitution**) ist das Cäsar-Verfahren. Dabei wird das Alphabet um den gegebenen Schlüsselwert zyklisch verschoben.

Ein Schlüssel von 3 bewirkt etwa eine Verschiebung um drei Stellen im Alphabet, womit aus jedem A ein D, aus jedem B ein E, ..., aus jedem Z ein C wird.

Beispiel: der Klartext TILL soll mit D (bzw. der Zahl 3) cäsar-verschlüsselt werden.

Klartext	T	I	L	L
Verschiebung	+3	+3	+3	+3
Geheimtext	W	L	O	O

Als Geheimtext entsteht WL0O.

Die Cäsar-Verschlüsselung kann bei ausreichend langen Texten über die Häufigkeitsanalyse schnell geknackt werden. Da die Häufigkeitsverteilung bestehen bleibt, ist der häufigste Buchstabe im Geheimtext das Ergebnis der Verschiebung des häufigsten Buchstabens im Klartext. Ist der Text lang genug und die Sprache bekannt (in deutschsprachigen Texten ist etwa das E mit 17,40 % mit Abstand der häufigste Buchstabe (vor N mit 9,78 %)), kann so schnell der Schlüssel erraten und damit die Verschlüsselung geknackt werden.

PM-Substitution

Bei der **polyalphabetisch** monografischen Substitution, kurz **PM-Substitution**, wird weiterhin jedes Zeichen des Klartexts einzeln ersetzt. Das ersetzende Zeichen kann sich jedoch unterscheiden (so kann ein E bspw. beim ersten Auftreten durch L, beim nächsten Auftreten durch O ersetzt werden).

Ein Beispiel für PM-Substitution ist der Vigenère-Code. Hier wird anstelle eines einzigen Codezeichens mit einem Codewort verschlüsselt. Der erste Buchstabe des Klartexts wird dann mit dem ersten Buchstaben des Codeworts cäsar-verschlüsselt, der zweite Buchstabe des Klartexts mit dem zweiten Buchstaben des Codeworts cäsar-verschlüsselt usw. Ist der Klartext länger als das Codewort, so wird beim Codewort wieder von vorn begonnen.

3.6. Kryptologie

Beispiel: der Klartext TILL soll mit dem Codewort OLE verschlüsselt werden.

Klartextzeichen	T	I	L	L
ASCII-Code	84	73	76	76
Codezeichen	0	L	E	0
Verschiebung	79-65=14	76-65=11	69-65=4	79-65=14
Summe	98	84	80	90
ggf. Anpassung (65 bis 90)	98-26=72	84	80	90
Zeichen	H	T	P	Z

Als Codewort entsteht hier HTPZ. Wie wir sehen, wurde das L einmal mit P, einmal mit Z verschlüsselt.

Die Sicherheit des Vigenère-Verfahrens ist abhängig von der Länge des verwendeten Schlüsselworts. Ist die Länge des Schlüsselworts bekannt, kann der Text wieder durch Häufigkeitsanalyse entschlüsselt werden. Es gibt unterschiedliche Verfahren zur Abschätzung für die Länge des Schlüsselworts, etwa den Kasiski-Test oder Berechnungen über den sogenannten Koinzidenzindex des Texts.

One-Time-Pad

Verwende ich beim Vigenère-Verfahren ein zufälliges Schlüsselwort, das genauso lang ist, wie der Klartext und nur ein einziges Mal verwendet wird, so sprechen wir von einem **One-Time-Pad**.

Die OTP-Verschlüsselung ist ohne Kenntnis über den Schlüssel nicht entzifferbar. Nachteile sind die einmalige Verwendbarkeit des Schlüssels sowie die Länge des Schlüssels (schließlich benötigt man einen zufälligen(!) Schlüssel, der genauso lang(!) ist wie der Klartext). Die Nachteile machen den OTP zu einem zwar sicheren, in der Praxis aber ineffizienten Verfahren.

Moduloarithmetik

Grundlage vieler Verschlüsselungsverfahren ist das Rechnen mit Resten bei der Division ganzzahliger Werte. Dies wird auch als **Moduloarithmetik** bezeichnet.

Wir verwenden folgende Kurzschrifweisens.

- für die Aussage „25 lässt bei der Division durch 3 den Rest 1“ können wir kurz schreiben:
 $25 \bmod 3 = 1$
- für die Aussage „69 und 42 lassen bei Division durch 3 denselben Rest“ können wir kurz schreiben: $69 \equiv 42 \pmod{3}$
- für die Aussage „3 ist das modulare Inverse von 5 bez. des Moduls 7“ können wir kurz schreiben: $3 = 5^{-1} \pmod{7}$ oder auch $5 \cdot 3 \equiv 1 \pmod{7}$

Erinnerung: in Python liefert der Operator % den Rest und der Operator // den Quotienten bei der ganzzahligen Division.

3.6. Kryptologie

Die Grundrechenarten Addition, Subtraktion und Multiplikation können wie gewohnt durchgeführt werden.

Aus $x + 2 \equiv 7 \pmod{18}$ kann ich etwa durch Subtraktion von 2 schließen: $x \equiv 5 \pmod{18}$. Jede ganze Zahl, die bei der Division durch 18 den Rest 5 lässt, erfüllt damit die Vorgabe $x + 2 \equiv 7 \pmod{18}$.

Da wir uns im Bereich der ganzen Zahlen befinden, stellt die Division ein Problem dar. Die Umkehrung einer Multiplikation ist nicht in jedem Fall möglich.

Wir nennen die Zahl b **modulares Inverses** zur Zahl a bez. des Moduls m , wenn $a \cdot b \equiv 1 \pmod{m}$ gilt. So ist etwa 3 das modulare Inverse von 5 bez. des Moduls 7, da $3 \cdot 5 = 15 \equiv 1 \pmod{7}$.

Das modulare Inverse kann entweder über Brute-Force (sehr ineffizient!) oder den erweiterten euklidischen Algorithmus bestimmt werden.

Beispiel: wir suchen $41^{-1} \pmod{192}$ und berechnen dafür den ggT von 41 und 192 über den (erweiterten) euklidischen Algorithmus.

$$192 : 41 = 4 \text{ Rest } 28 \rightarrow 28 = 1 \cdot 192 - 4 \cdot 41$$

$$41 : 28 = 1 \text{ Rest } 13 \rightarrow 13 = 1 \cdot 41 - 1 \cdot 28$$

$$28 : 13 = 2 \text{ Rest } 2 \rightarrow 2 = 1 \cdot 28 - 2 \cdot 13$$

$$13 : 2 = 6 \text{ Rest } 1 \rightarrow 1 = 1 \cdot 13 - 6 \cdot 2$$

$$2 : 1 = 2 \text{ Rest } 0$$

Anhand der letzten Rechnung sehen wir, dass 1 der ggT von 41 und 192 ist. Nur, wenn die beiden Zahlen wie hier teilerfremd sind, existiert das modulare Inverse.

Durch Rückrechnen der hinter den Pfeilen notierten Gleichungen erhalten wir eine Darstellung der Form $\text{ggT}(a; b) = x \cdot a + y \cdot b$ mit ganzen Zahlen x und y .

$$1 = 1 \cdot 13 - 6 \cdot 2 = 1 \cdot 13 - 6 \cdot (1 \cdot 28 - 2 \cdot 13) = 13 \cdot 13 - 6 \cdot 28$$

$$1 = 13 \cdot (1 \cdot 41 - 1 \cdot 28) - 6 \cdot 28 = 13 \cdot 41 - 19 \cdot 28$$

$$1 = 13 \cdot 41 - 19 \cdot (1 \cdot 192 - 4 \cdot 41) = 89 \cdot 41 - 19 \cdot 192$$

Betrachten wir diese Gleichung nun mod 192, so entfällt der Subtrahend und wir sehen $1 \equiv 89 \cdot 41 \pmod{192}$ und wissen damit, dass $41^{-1} \pmod{192} = 89$ gilt. Wäre der Faktor vor der 41 negativ, so wird die nächstgrößere positive Zahl mit demselben Rest gesucht. Bei der Suche nach $121^{-1} \pmod{169}$ ergibt sich etwa die Gleichung $1 = 58 \cdot 169 + (-81) \cdot 121$.

Damit gilt $121^{-1} \pmod{169} = -81 \equiv -81 + 169 = 88$, also ist 88 das modulare Inverse zur 121 im Modul 169.

Mit dem Wissen über modulare Inverse können wir auch Multiplikationsaufgaben rückgängig machen. Die Gleichung $5x \equiv 4 \pmod{7}$ kann durch Multiplikation mit dem modularen Inversen 3 von 5 zum Modul 7 zu $x \equiv 3 \cdot 4 = 12 \equiv 5 \pmod{7}$. Somit ist jede ganze Zahl, die bei der Division durch 7 den Rest 5 lässt ($5, 12, 19, \dots$) Lösung der Gleichung $5x \equiv 4 \pmod{7}$.

3.6. Kryptologie

Um sich alle Zwischenwerte nicht während der gesamten Berechnung merken zu müssen, gibt es eine Umsetzung des erweiterten euklidischen Algorithmus mit zwei Zahlenfolgen s und t .

Am Beispiel von 41 und 192:

Schritt k	Rest r_k	Wert s_k	Wert t_k	Quotient q_k	Gleichung
0	192	1	0		
1	41	0	1	4	
2	28	1	-4	1	$28 = 1 \cdot 192 + (-4) \cdot 41$
3	13	-1	5	2	$13 = -1 \cdot 192 + 5 \cdot 41$
4	2	3	-14	6	$2 = 3 \cdot 192 + (-14) \cdot 41$
5	1	-19	89		$1 = -19 \cdot 192 + 89 \cdot 41$

Die Zahlenfolge r der Reste startet mit den beiden betrachteten Zahlen. In jeder Zeile berechnet sich der Wert q_k als Quotient aus r_{k-1} durch r_k .

Die Zahlenfolge s startet immer mit den Werten 1 und 0, die Zahlenfolge t mit den Werten 0 und 1. In jedem Schritt danach (also ab $k = 1$) gilt die Rekursionsvorschrift $s_{k+1} = -q_k \cdot s_k + s_{k-1}$ sowie $t_{k+1} = -q_k \cdot t_k + t_{k-1}$. So gilt etwa $s_5 = -q_4 \cdot s_4 + s_3 = -6 \cdot 3 + (-1) = -19$.

Vorteil dieser Variante ist, dass jede Zeile eine Gleichung der Form $r_k = s_k \cdot r_0 + t_k \cdot r_1$ darstellt und wir so in der letzten Zeile direkt die gewünschte Darstellung des ggT besitzen.

Häufig wird neben Addition, Subtraktion, Multiplikation und Division auch die Exponentiation („das Potenzieren“) benötigt. Gerade im Umgang mit großen Zahlen, der für kryptologische Zusammenhänge wichtig ist, ist es nötig Terme der Form $a^b \bmod m$ effizient berechnen zu können.

Dies können wir z.B. über das **Square-and-Multiply-Verfahren** durchführen.

Bei diesem Verfahren wird der Exponent auf Grundlage seiner Binärdarstellung zu einer Zeichenkette mit Anweisungen zum Quadrieren und Multiplizieren umgewandelt. Nach jedem Rechenschritt wird das Ergebnis $\bmod m$ betrachtet.

Am Beispiel: $42^{69} \bmod 99 = ?$

$69 = [1000101]_2$. Statt jeder 1 wird „SM“, statt jeder 0 nur „S“ notiert. So ergibt sich aus der Zeichenkette 1000101 nun die Zeichenkette **SMSSSSMSSM**. Diese wird nun mit 1 beginnend abgearbeitet. Bei jedem S wird der aktuelle Wert quadriert (square) und $\bmod 99$ gerechnet, bei jedem M wird der aktuelle Wert mit der Basis 42 multipliziert (multiply) und $\bmod 99$ gerechnet.

$$1 \rightarrow^S 1 \rightarrow^M 42 \rightarrow^S 1764 \equiv 81 \rightarrow^S 6561 \equiv 27 \rightarrow^S 729 \equiv 36 \rightarrow^S 1296 \equiv 9 \rightarrow^M 378 \equiv 81$$

$$81 \rightarrow^S 6561 \equiv 27 \rightarrow^S 729 \equiv 36 \rightarrow^M 1512 \equiv 27$$

Insgesamt erhalten wir: $42^{69} \bmod 99 = 27$ ohne, dass wir 42^{69} vollständig ausgerechnet haben.

Zur Kontrolle:

$$\begin{aligned}
 42^{69} &= 10\ 097\ 201\ 832\ 880\ 355\ 573\ 875\ 790\ 863\ 214\ 833\ 226\ 896\ 186\ 369\ 872\ 326\ 994\ 250\ 398 \\
 &\quad 570\ 376\ 877\ 433\ 686\ 009\ 543\ 845\ 316\ 266\ 007\ 917\ 815\ 719\ 968\ 899\ 072 \\
 &= 101\ 991\ 937\ 705\ 862\ 177\ 513\ 896\ 877\ 406\ 210\ 436\ 635\ 315\ 013\ 837\ 094\ 212\ 063\ 135\ 339 \\
 &\quad 094\ 715\ 933\ 673\ 596\ 055\ 998\ 437\ 538\ 040\ 484\ 018\ 340\ 605\ 746\ 455 \cdot 99 + 27
 \end{aligned}$$

Auch zur Berechnung der Potenz (also ohne Betrachtung mod 99) kann das Square-and-Multiply-Verfahren effizient genutzt werden, da hier die Anzahl der Rechenoperationen so gering wie möglich ist (Mathe-Fans können das gern nachweisen).

Weiterhin für die kryptologische Anwendung wichtig ist der Begriff des Zahlenkörpers. Für schulische Zwecke reicht die Erkenntnis, dass für **Primzahlen** p die Betrachtung der entstehenden Reste der Division ganzer Zahlen durch p besonders positive Eigenschaften besitzt. Die Algebraiker nennen den entstehenden Zahlenkörper **Galoiskörper** (auch Galois-Feld) mit p Elementen und schreiben kurz $GF(p)$. Die ganze Zahl a heißt dann **Primitivwurzel** dieses Galoiskörpers $GF(p)$, wenn die Potenzen von a jeden von null verschiedenen Wert aus $GF(p)$ erzeugen. Betrachten wir z.B. den Galoiskörper $GF(7)$, also bei der Division durch 7, so gibt es dort die Restklassen 0, 1, 2, 3, 4, 5 und 6. Jede ganze Zahl gehört genau einer dieser Restklassen an; die 9 z.B. der Restklasse zur 2, da sie bei Division durch 7 den Rest 2 lässt.

Die Zahl 3 ist eine Primitivwurzel von $GF(7)$, denn: $3^1 = 3, 3^2 \equiv 2, 3^3 \equiv 6, 3^4 \equiv 4, 3^5 \equiv 5$ und $3^6 \equiv 1$ bei der Division durch 7. Somit wird jede Restklasse außer der 0 genau einmal erreicht durch die Potenzen von 3 bis 3^6 .

Diffie und Hellman

Im November 1976 veröffentlichte IEEE den Artikel „New Directions in Cryptography“ von Whitfield Diffie und Martin E. Hellman. Die beiden beschrieben dabei wesentliche Grundlagen zu Kryptologie, leiteten einen Paradigmenwechsel ein, skizzierten, wie das für anliegende Themen (etwa die Authentifizierung) genutzt werden kann, gaben an, unter welchen Bedingungen das von ihnen beschriebene Diffie-Hellman-Verfahren nicht mehr sicher sein wird und gaben eine Einführung in das P-NP-Problem – die Lektüre dieses Artikels ist also sehr empfehlenswert.

Das Diffie-Hellman-Verfahren selbst beschäftigt sich mit der Frage, wie ein sicherer Schlüssel über einen unsicheren Übertragungskanal erzeugt werden kann. Nehmen wir dafür zwei Kommunikationspartner Alice und Bob an. Sie einigen sich auf den Modul q und eine Primitivwurzel a des zugehörigen Galoiskörpers $GF(q)$ (kann über den unsicheren Kanal geschehen). Beide wählen sich nun eine zufällige, von null verschiedene Zahl aus $GF(q)$, welche beide geheim halten, Alice X_i und Bob X_j . Alice berechnet $a^{X_i} \bmod q$ und überträgt dieses Ergebnis Y_i auf dem unsicheren Kanal an Bob. Bob selbst berechnet dann $Y_i^{X_j} \bmod q$ und erhält als Ergebnis damit den Schlüssel K . Bob seinerseits hat $Y_j := a^{X_j} \bmod q$ berechnet und an Alice übertragen. Berechnet sie nun $Y_j^{X_i} \bmod q$ erhält sie ebenfalls das Ergebnis K , also denselben Schlüssel (beide haben $a^{X_i \cdot X_j} \bmod q$ berechnet). Da die Berechnung der Potenz einfach, die Berechnung des Exponenten aus

gegebener Potenz (das Logarithmieren) aber nicht effizient machbar ist, ist das Verfahren (noch) sicher.

Asymmetrische Verfahren

Im Rahmen ihres Artikels haben Diffie und Hellman den Wechsel von symmetrischen zu asymmetrischen Verschlüsselungsverfahren, auch public-key-Verfahren genannt, vorgeschlagen.

Bei diesen Verfahren existiert ein privater und ein öffentlicher Schlüssel. Ein Sender verschlüsselt die Nachricht mit dem öffentlichen Schlüssel des Empfängers (dieser kann öffentlich eingelesen werden) und sendet die so verschlüsselte Nachricht an den Empfänger. Nur durch dessen privaten Schlüssel kann die Nachricht wieder entschlüsselt werden und somit steht sie nur dem Empfänger zur Verfügung und keinem Dritten.

Diffie und Hellman stellen dabei folgende Anforderungen an die Schlüsselverfahren E_K (für den öffentlichen Schlüssel) und D_K (für den privaten Schlüssel):

- (1) für jeden Schlüssel K sind E_K und D_K zueinander invers
- (2) die Verfahren E_K und D_K sind für jedes Paar aus Schlüssel K und Nachricht M einfach anzuwenden
- (3) für fast jeden Schlüssel K lässt sich jeder einfach zu berechnende Algorithmus, der D_K entspricht, rechnerisch nicht effizient aus E_K ableiten
- (4) für jeden Schlüssel K können die zusammengehörigen Paare E_K und D_K effizient berechnet werden.

Kerckhoffs'sches Prinzip

Kerckhoffs (genauer: Jean Guillaume Auguste Victor François Hubert Kerckhoffs von Nieuwenhof) formulierte in den 1880ern eine Forderung: Die Sicherheit eines Verschlüsselungsverfahrens soll auf der Geheimhaltung des Schlüssels beruhen und nicht auf der Geheimhaltung des Verfahrens.

Falltür-Einwegfunktionen

Bei Falltür-Einwegfunktionen ist die Berechnung des Inversen nur durch Falltürinformationen (z.B. Parameter, die zur Ermittlung des Schlüsselpaares verwendet wurden) effizient möglich. Eine solche Funktion sehen wir beim RSA-Verfahren im Einsatz.

RSA

Die Mathematiker Ronald Rivest, Adi Shamir und Leonard Adleman versuchten am MIT die Annahmen von Diffie und Hellman zu widerlegen. Sie fanden dabei ein Verfahren, bei dem sie keine Angriffspunkte fanden: das nach ihnen benannte **RSA-Verfahren**.

Schritt 0: Erzeugen der beiden Schlüssel

Zum Erzeugen des öffentlichen und des privaten Schlüssels werden zwei große Primzahlen p und q verwendet. Das Produkt der beiden $N := p \cdot q$ wird als RSA-Modul bezeichnet. Da p und q Primzahlen sind, gilt $\varphi(N) = (p - 1) \cdot (q - 1)$, wobei φ die eulersche Phi-Funktion darstellt und die Anzahl der zu N teilerfremden positiven natürlichen Zahlen angibt.

3.6. Kryptologie

Der Erzeuger des Schlüssels wählt nun eine Zahl e , die zu $\varphi(N)$ teilerfremd ist aus und berechnet ihr multiplikativ Inverses d bez. $\varphi(N)$.

Der öffentliche Schlüssel ist nun das Paar $(e|N)$, wohingegen d als privater Schlüssel aufzubewahren ist.

Die Werte für p , q und $\varphi(N)$ sollten aus Sicherheitsgründen vernichtet werden.

Schritt 1: Verschlüsselung

Um eine Nachricht mit RSA zu verschlüsseln, muss sie ggf. in eine ganze Zahl im Bereich $\{0; 1; \dots; N - 1\}$ umgewandelt werden. Bei Texten kann dafür entweder jedes einzelne Zeichen per ASCII-Tabelle umgewandelt werden oder ein Block von maximal so vielen Zeichen, wie N an Byte in der Binärdarstellung benötigt. Die entstandene Zahl m wird nun über $m^e \bmod N$ in eine Codezahl c umgewandelt, die an den Empfänger übertragen werden kann. Der verwendete Exponent e und das RSA-Modul N sind dabei dem öffentlichen Schlüssel des Empfängers(!) zu entnehmen.

Schritt 2: Entschlüsselung

Hat der Empfänger eine Nachricht c erhalten, die mit seinem öffentlichen Schlüssel RSA-verschlüsselt wurde, so kann er mit $c^d \bmod N$ die Ausgangszahl m wieder bestimmen und somit die Nachricht im Klartext rekonstruieren.

Beispiel: $p = 521, q = 503$

- $N = p \cdot q = 262\ 063$
- $\varphi(N) = (p - 1) \cdot (q - 1) = 261\ 040$
- Bei Wahl von $e = 240\ 133$ ergibt sich $d = 248\ 317$ (z.B. über den erweiterten euklidischen Algorithmus).
- öffentlicher Schlüssel: $(240\ 133|262\ 063)$
- privater Schlüssel: $248\ 317$
- Die Ausgangsnachricht „RSA ist toll.“ wird nun zeichenweise über den ASCII-Code umgewandelt. Werte für m : $82, 83, 65, 32, 105, 115, 116, 32, 116, 111, 108, 108, 46$.
- entstehende Zahlen für c : $35\ 514, 232\ 108, 220\ 300, 180\ 086, 166\ 142, 242\ 423, 196\ 293, 180\ 086, 196\ 293, 50\ 728, 121\ 811, 121\ 811, 70\ 043$
- Entschlüsselung mit Exponent d ergibt wieder die für m angegebenen Zahlen, die mittels ASCII-Tabelle wieder zu Zeichen umgewandelt werden können

Bemerkung: die hier verwendeten Zahlen für p und q sind sehr kleine Primzahlen. Das Bundesamt für Sicherheit in der Informationstechnik empfahl bereits 2023 die Verwendung von RSA-Schlüsseln mit mindestens 3000 Bit Schlüssellänge für N (siehe z.B. mit Stand vom 02.02.2024 <https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?blob=publicationFile&v=10>, S. 35, Abrufdatum 30.12.2024). Problem: je größer der Schlüssel, desto größer die benötigte Rechenzeit zur Ver- und Entschlüsselung. Daher wird in der Praxis RSA eher zum Aufbau der Verbindung und zur Vereinbarung eines symmetrischen Schlüssels verwendet, für die eigentliche Verschlüsselung der zu übertragenden Daten aber ein anderes Verfahren genutzt.

SSL/TLS

Das Protokoll **Transport Layer Security (TLS)** wurde bis 1999 als **Secure Sockets Layer (SSL)** bezeichnet. Es ist ein Protokoll zur Authentifizierung und Verschlüsselung von Internetverbindungen.

Der Server erzeugt einen öffentlichen Schlüssel. Für diesen Schlüssel wird durch eine Zertifizierungsstelle (Certificate Authority – CA) ein Zertifikat erstellt. Dieses ist zeitlich begrenzt gültig. Will ein Client nun eine sichere Verbindung zum Server aufbauen, fordert er über einen TLS-Request dessen Zertifikat an. Das so erhaltene Zertifikat wird dann vom Client bei der Zertifizierungsstelle gegengeprüft. Ist dieser Vorgang erfolgreich, kann die verschlüsselte Übertragung von Daten zwischen Client und Server beginnen.

Digitale Signaturen - Zertifikate

Betrachten wir bspw. im Browser die Internetseite <https://www.lernsax.de> so wird die Internetseite verschlüsselt übertragen. Häufig können wir im Browser uns nähere Informationen zur Übertragung anzeigen lassen. Firefox bietet bspw. folgende Information (Stand 19.10.2024).

Technische Details

Verbindung verschlüsselt (TLS_AES_256_GCM_SHA384, 256-Bit-Schlüssel, TLS 1.3)

Die Seite, die Sie ansehen, wurde verschlüsselt, bevor sie über das Internet übermittelt wurde.

Verschlüsselung macht es für unberechtigte Personen schwierig, zwischen Computern übertragene Informationen anzusehen. Daher ist es unwahrscheinlich, dass jemand diese Seite gelesen hat, als sie über das Internet übertragen wurde.

Hilfe

Die Verbindung wurde also über TLS (siehe vorheriger Abschnitt) verschlüsselt. Dabei wurde AES mit 256 Bit und Betriebsmodus GCM verwendet. AES ist eine sogenannte Blockchiffre, ein symmetrisches, 2000 erfundenes Verfahren von Joan Daemen und Vincent Rijmen (auch „Rijndael“ genanntes Verfahren), das Blöcke von 128 Bit verschlüsselt. Die 256 gibt an, dass ein Schlüssel mit 256 Bit verwendet wurde.

Der Betriebsmodus GCM (kurz für Galois/Counter Mode) legt fest, wie Nachrichten mit einer Länge von mehr als 128 Bit so auf die 128-Bit-Blöcke verteilt werden, dass sie mit AES verschlüsselt werden können.

Zur Kontrolle wird ein 384 Bit langer Hash-Wert berechnet über das SHA384-Verfahren.

Hash-Funktion

Hash-Funktionen sind Abbildungen, die Eingaben mit beliebiger Größe in ein Ergebnis fester Größe umwandeln. Sie sind deterministisch (Erinnerung: bei derselben Eingabe in das Verfahren wird stets dieselbe Ausgabe erzeugt) und kleinste Veränderungen der Eingabe führen zu einer völlig anderen Ausgabe („Lawineneffekt“). Darüber hinaus sind sie Einwegfunktionen, d.h. es sollte rechnerisch nicht möglich sein, aus der Ausgabe auf die Eingabe zurückzuschließen.

3.6. Kryptologie

Verwendet werden Hash-Funktionen z.B. beim Speichern von Kennwörtern. Kennwörter werden in der Praxis nicht mehr im Klartext gespeichert, sondern als Ergebnis einer Hash-Funktion. Früher wurden MD5 und SHA-1 als Verfahren verwendet. Inzwischen gelten diese Verfahren jedoch als unsicher, da eine Kollision (d.h. zwei unterschiedliche Eingaben, die zur selben Ausgabe führen) in vertretbarer Zeit gefunden werden könnte.

Häufig werden daher Vertreter von SHA-2 (z.B. SHA256, SHA384 oder SHA512) oder SHA-3 (SHA3-224, SHA3-256, SHA3-384 oder SHA3-512) verwendet. Die Zahl gibt dabei die Länge des entstehenden Hash-Wertes in Bit an.

Verwendung in Python

Die Berechnung von Hash-Werten mit bekannten Verfahren kann unter Nutzung des Moduls `hashlib` umgesetzt werden. Dabei gibt `hashlib.algorithms_available` an, welche Hash-Verfahren zur Verfügung stehen.

Beispielcode:

```
import hashlib

def main():
    zkliste = ["Hallo Informatikkurs", "Niners", "Erstaunlich, wie ein geändertes
Zeichen sich auswirkt", "Erstaunlich wie ein geändertes Zeichen sich auswirkt"]
    for eingabe in zkliste:
        print()
        print("Zeichenkette: "+eingabe)

        h1 = hashlib.sha1()
        h1.update(bytes(eingabe, 'utf-8'))
        print('SHA1: '+h1.hexdigest())

        h256 = hashlib.sha256()
        h256.update(bytes(eingabe, 'utf-8'))
        print('SHA256: '+h256.hexdigest())

        h3_256 = hashlib.sha3_256()
        h3_256.update(bytes(eingabe, 'utf-8'))
        print('SHA3-256: '+h3_256.hexdigest())
```

Bei diesem Python-Skript werden für die in `zkliste` eingespeicherten Zeichenketten jeweils die Hash-Werte nach SHA-1, SHA256 und SHA3-256 bestimmt und ausgegeben. Dafür muss die Zeichenkette `eingabe` in Byte-Arrays umgewandelt werden, was über die Methode `bytes` unter Angabe der verwendeten Zeichencodierung `utf-8` geschieht.

Seit Python-Version 3.11 kann auch der Hash-Wert einer Datei direkt ermittelt und ausgegeben werden. Eine mögliche Umsetzung folgt im kommenden Codebeispiel.

3.6. Kryptologie

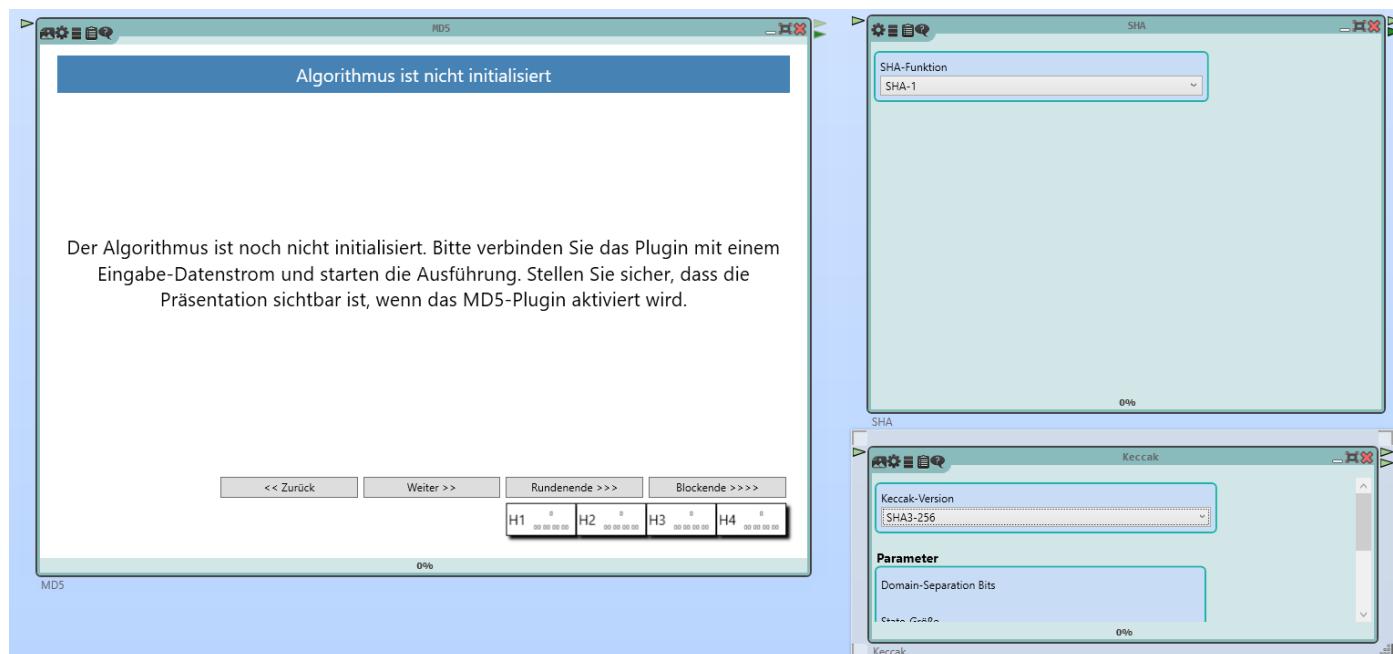
```
datei = open('hashtest.py', 'rb')
code = hashlib.file_digest(datei, 'sha256')
datei.close()
print("Funktionswert: "+code.hexdigest())
print("Anzahl an Bytes: "+str(code.digest_size))
```

Hier wird der SHA256-Wert der Datei hashtest.py ermittelt und ausgegeben. Solche Angaben können z.B. auf Internetseiten als Sicherheit für den Nutzer angeboten werden. Er kann zur heruntergeladenen Datei den Hash-Wert selbst ermitteln und über Vergleich mit der Vorgabe herausfinden, ob die Datei manipuliert wurde.

Version	Operating System	Description	MD5 Sum	File Size	PGP	Sigstore	SPDX
Gzipped source tarball	Source release		304473cf367fa65e450edf4b06b55fcc	25.8 MB	SIG	.sigstore	SPDX
XZ compressed source tarball	Source release		d46e5bf9f2e596a3ba45fc0b3c053dd2	19.5 MB	SIG	.sigstore	SPDX
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	dc762fdc78e9cfecf516db31054de9fd	44.0 MB	SIG	.sigstore	
Windows installer (64-bit)	Windows	Recommended	2f2ab2472a6aa29f8755c72c58f58f4b	25.8 MB	SIG	.sigstore	SPDX
Windows installer (32-bit)	Windows		745f11c8474893da55e5966173375cc8	24.6 MB	SIG	.sigstore	SPDX
Windows installer (ARM64)	Windows	Experimental	ff0d440c2cc4aaddf81c9e247682bfa9	25.1 MB	SIG	.sigstore	SPDX
Windows embeddable package (64-bit)	Windows		1e86b04bc7d27c5c06edf8f617e1184a	10.6 MB	SIG	.sigstore	SPDX
Windows embeddable package (32-bit)	Windows		cd4a16b1d27540b84e7a44327f69ee5a	9.5 MB	SIG	.sigstore	SPDX
Windows embeddable package (ARM64)	Windows		a1631f5cb0b3d5d1a27b5c3edc0f80e3	9.9 MB	SIG	.sigstore	SPDX

Hier bietet z.B. Python für die Version 3.12.8 die MD5-Hashwerte ihrer Datei an. Für solche Zwecke ist MD5 (trotz bekannter Schwächen) nach wie vor gut geeignet.

Die Berechnung von Hash-Werten ist jedoch auch durch entsprechende Komponenten in CrypTool möglich (Hash-Funktionen → MD5, SHA, Keccak), wobei das genau angewendete Verfahren meist in den Einstellungen zu wählen ist. Die Funktionsweise von MD5 oder SHA-3 kann dort z.B. auch in einer Vorlage (bei SHA-3: Keccak-Hash) genauer betrachtet werden.



PGP und S/MIME

PGP und S/MIME sind weitere Verschlüsselungsverfahren. Bei **PGP** (Pretty Good Privacy, erste Version 1991 durch Phil Zimmermann) wird die eigentliche Nachricht symmetrisch verschlüsselt. Der dazu verwendete Schlüssel selbst wird jedoch asymmetrisch (anfangs mit RSA, später mit ElGamal) verschlüsselt. Wir sprechen daher von einem **hybriden Verfahren**, also eines, das sowohl auf symmetrischer, als auch asymmetrischer Verschlüsselung beruht.

S/MIME (Secure/Multipurpose Internet Mail Extensions) ist wiederum ein Standard für die Verschlüsselung und das Signieren von Objekten. Während PGP auf zentrale Zertifizierungsstellen verzichtet, werden bei S/MIME vertrauenswürdige Zertifizierungsstellen (sogenannte CAs) verwendet. Auch S/MIME ist ein hybrides Verfahren, da es die Nachricht symmetrisch, den dafür nötigen Schlüssel (hier: Session Key genannt) aber asymmetrisch verschlüsselt.

Beide Verfahren kombinieren die Stärken der Verschlüsselungsarten: Geschwindigkeit und Sicherheit der symmetrischen Verschlüsselung und sicherer Schlüsselaustausch durch asymmetrische Verschlüsselung.

3.6.3. Datenschutz

Datenschutz bezieht sich auf den Schutz personenbezogener Daten vor Missbrauch und die Wahrung der Privatsphäre von Individuen.

Personenbezogene Daten

Als **personenbezogene Daten** werden (nach Art. 4 DSGVO) alle Informationen bezeichnet, die sich auf eine identifizierte oder identifizierbare natürliche Person beziehen. Dazu gehören z.B. körperliche Eigenschaften, Standortdaten, KFZ-Kennzeichen, Rentenversicherungsnummer usw. Unter ihnen gibt es besonders schützenswerte Daten (Art. 9 DSGVO). Dazu gehören u.a. Daten zur Herkunft, politischer Meinungen, religiöser oder weltanschaulicher Überzeugungen, Gesundheitsdaten sowie Daten zur sexuellen Orientierung.

DSGVO

Die **Datenschutz-Grundverordnung** enthält Regeln zur Verarbeitung personenbezogener Daten durch private oder öffentliche Verantwortliche. Sie wurde EU-weit vereinheitlicht und soll einerseits den Schutz personenbezogener Daten innerhalb der EU sicherstellen und andererseits den freien Datenverkehr innerhalb des europäischen Binnenmarktes gewährleisten.

Grundsätze der DSGVO: Rechtmäßigkeit der Datenverarbeitung, Transparenz über gespeicherte Daten, Zweckbindung der Daten, Datenminimierung, Richtigkeit der Datenverarbeitung, Speicherbegrenzung (Recht auf Vergessenwerden), Integrität und Vertraulichkeit, Rechenschaftspflicht

BDSG

Das **Bundesdatenschutzgesetz** ergänzt und präzisiert deutschlandweit die DSGVO bei der Regelung derjenigen Punkte, bei denen die DSGVO den EU-Mitgliedsstaaten die Umsetzung überlässt (z.B. Verarbeitung von Beschäftigtendaten, Videoüberwachung, Bestellung von Datenschutzbeauftragten).

SächsDSDG

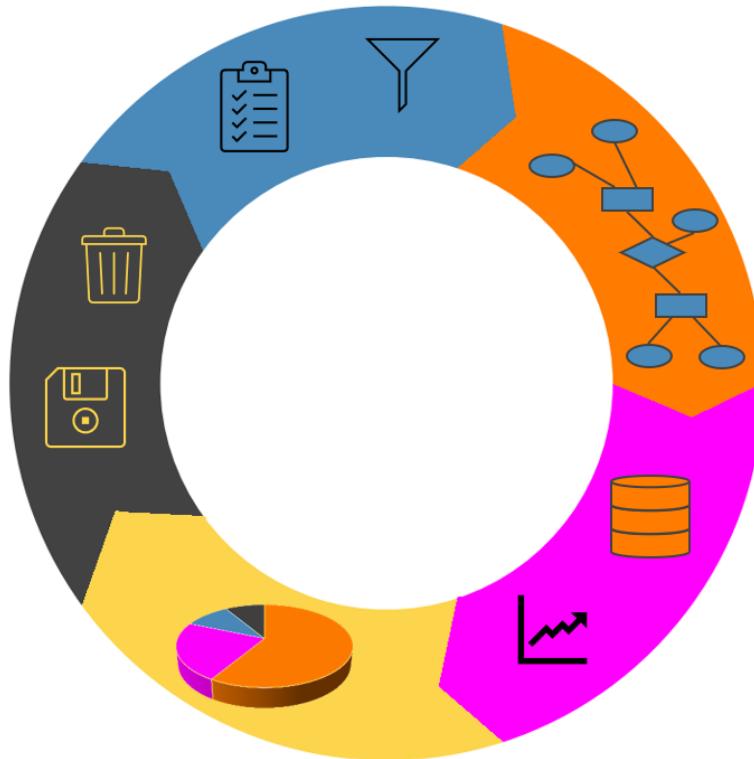
Das **Sächsische Datenschutzdurchführungsgesetz** (seit 25.05.2018) organisiert (zur DSGVO und dem BDSG) ergänzende Regelungen für den Freistaat Sachsen für den öffentlichen Bereich (Behörden, Gemeinden, ...). Es löste das Sächsische Datenschutzgesetz ab (letzte Fassung SächsDSG galt bis 31.12.2020).

Datenschutz am JKG

Am JKG Chemnitz ist (Stand 2024) Herr Weißenfels der Datenschutzbeauftragte.

4. Umgang mit Daten

4.1. Intro



Den Umgang mit Daten können wir im **Datenlebenszyklus (Data-Life-Cycle)** veranschaulichen. Eine mögliche Einteilung der Phasen soll hier am Beispiel einer Umfrage zum Thema „Energy-Drinks in der Schule“ vorgenommen werden.

Erfassen, Beschaffen und Bereinigen von Daten

Die benötigten Daten werden gesammelt. Die Umfrage wird dafür vorher erstellt und verteilt. Dabei werden klare, zielgerichtete Informationen abgefragt (z.B. wie viele Energy-Drinks je Woche konsumiert werden). Bei der Durchführung der Umfrage kann es aber vorkommen, dass bewusst oder unbewusst falsche Informationen eingetragen werden (z.B. 365 Energy-Drinks je Woche angegeben werden) oder nicht alle Fragen beantwortet werden. Die ermittelten Daten müssen daher ggf. bereinigt werden. Je nach Frage bez. Zusammensetzung der Befragtengruppe (z.B. Befragung zur Sonntagsfrage bei Wahlkundgebung einer Partei) kann es auch zu Verzerrungen des Ergebnisses (sogenannter Bias) kommen. Auch diese Verzerrungen sollten bereinigt werden.

Modellierung

Die bereinigten Daten werden strukturiert und modelliert. Die Umfrageergebnisse könnten dafür z.B. in ein relationales Datenbankmodell (→ Tabellen) überführt werden. Ziel ist es hierbei, die Daten so zu organisieren, dass sie effizient gespeichert und verarbeitet werden können.

Implementierung und Optimierung

Die modellierten Daten werden in einem geeigneten System implementiert (z.B. einer Datenbank). Hierfür könnten die Umfrageergebnisse in eine SQL-Datenbank importiert werden. Die Datenbank wird dann optimiert, um schnelle Zugriffe auf relevante Daten zu ermöglichen. So könnten z.B. Filter für Altersgruppen oder Konsumhäufigkeiten definiert werden, um spezifische Analysen zu erleichtern.

Verarbeitung, Analyse und Visualisierung von Daten

In der Analysephase werden die Daten ausgewertet. Für die Umfrage könnten grundlegende Statistiken wie der Durchschnittskonsum oder die beliebtesten Marken ermittelt werden. Dafür können in der Datenbanksoftware Abfragen und Berichte erstellt werden. Die Ergebnisse können z.B. in einer Präsentation vorgestellt werden.

Evaluation, Austausch, Löschung und Archivierung

In der letzten Phase geht es um die Bewertung der Ergebnisse und den Umgang mit den Daten nach der Analyse. Die Ergebnisse der Umfrage könnten in einem Bericht zusammengefasst und der Schulgemeinschaft für die weitere Arbeit zur Verfügung gestellt werden. Gleichzeitig sollten Datenschutzvorgaben berücksichtigt werden: personenbezogene Daten müssen z.B. anonymisiert oder gelöscht werden, wenn sie nicht mehr benötigt werden. Wichtige Erkenntnisse könnten archiviert werden, um später auf sie zurückzugreifen, während andere Daten endgültig gelöscht werden. Die archivierten Daten können als Ausgangsdaten für eine erneute Erhebung einige Jahre später genutzt werden und dienen damit wieder dem Erfassen, Beschaffen und Bereinigen von Daten, sodass ein Kreislauf entstanden ist – der Datenlebenszyklus.

4.2. Erstellen der Datenbasis und KI

Das Erstellen, Erschaffen und Bereinigen von Daten ist der erste Schritt des Datenlebenszyklus und umfasste das Generieren neuer Informationen, etwa durch manuelle Eingabe, automatische Erfassung (z.B. durch Sensoren) oder den Import aus externen Quellen (z.B. vorherigen Umfragen). Dabei ist es essenziell, Daten strukturiert und konsistent zu erfassen, um spätere Nutzungsmöglichkeiten zu gewährleisten. Die Datenbereinigung ist ein kritischer Schritt, um die Qualität und Nutzbarkeit der Daten sicherzustellen. Fehler wie Duplikate, Inkonsistenzen oder fehlende Werte könnten die Analyse beeinträchtigen und zu falschen Schlussfolgerungen führen. Systematische Verzerrungen oder Fehler, die während des Datenlebenszyklus auftreten können und zu fehlerhaften oder unausgewogenen Ergebnissen führen, werden auch als **Bias** (Voreingenommenheit) bezeichnet. Sie können z.B. durch nicht repräsentative Stichproben auftreten. Diese treten auch beim Training künstlicher Intelligenzen auf.

Arten von KI

- **Schwache KI:** spezialisiert auf eng definierte Aufgaben, z.B. Schachcomputer, ChatGPT
- **Starke KI:** menschenähnliche Intelligenz in allen Bereichen
- **Superintelligenz:** Form der KI, die menschliche Intelligenz in allen Aspekten übertrifft

Frühe Formen von KI

- Turing-Test: ein Mensch kommuniziert mit einer Maschine und einem Menschen, weiß aber nicht, wer wer ist. Kann er nach einer Reihe von Fragen, nicht entscheiden, wer der Mensch und wer die Maschine ist, so soll die Maschine als „intelligent“ bezeichnet werden.
- Logic Theorist (1956): Allen Newell, Herbert A. Simon und Cliff Shaw entwickeln ein erstes Computerprogramm, das als eine Form von KI betrachtet wird. Es konnte mathematische Theoreme (38 der ersten 52 Sätze in Bertrand Russells „Principia Mathematica“) beweisen und sogar einige Beweise vereinfachen. Sie stellten ihr Programm auf der Dartmouth Conference vor.
- Perceptron (1957): Frank Rosenblatt entwickelte mit Perceptron eines der ersten künstlichen neuronalen Netze, um einfache Muster zu erkennen und zu klassifizieren.
- ELIZA (1966): Joseph Weizenbaum entwickelt ein Computerprogramm, das eine einfache Form der Verarbeitung natürlicher Sprache nutzte. Es simulierte Gespräche (erkannte dabei Schlüsselwörter in Eingaben und gab vorbereitete Antworten).
- Schachprogramme, z.B. Deep Blue (1996 Sieg in einer Partie gegen Kasparow, 1997 Sieg des Wettkampfs gegen Kasparow)

Maschinelles Lernen

Maschinelles Lernen ist ein Teilbereich der künstlichen Intelligenz der sich damit beschäftigt, Computer zu trainieren, um aus Daten und Erfahrungen zu lernen. Die Maschinen werden dabei auf besondere Fähigkeiten trainiert, etwa Muster und Korrelationen in großen Datensätzen zu finden und Entscheidungen und Vorhersagen zu treffen.

- **supervised learning**... überwachtes Lernen

Modell trainiert anhand bekannter Datensatzpaare (Eingabe und erwartete Ausgabe), Ziel ist Vorhersage oder Klassifikation neuer Daten; Bsp.: Lineare Regression, Entscheidungsbäume, Support Vector Machine, Künstliche Neuronale Netze

- **unsupervised learning**... unüberwachtes Lernen

Modell lernt aus unbeschrifteten Daten ohne vorgegebene Ausgabewerte, Ziel: Erkennen eines Musters oder von Strukturen in den Daten; Bsp.: K-Means, hierarchisches Clustern, Hauptkomponentenanalyse

- **reinforcement learning**... bestärkendes Lernen

Modell lernt durch Interaktionen mit einer Umgebung und der erhaltenen Belohnung oder Bestrafung; Ziel: Entwicklung einer Strategie, um kumulative Belohnung zu maximieren; Bsp.: Q-Learning, Policy-Gradient-Methoden

Es gibt auch Mischformen, etwa semi-supervised learning oder self-supervised learning.

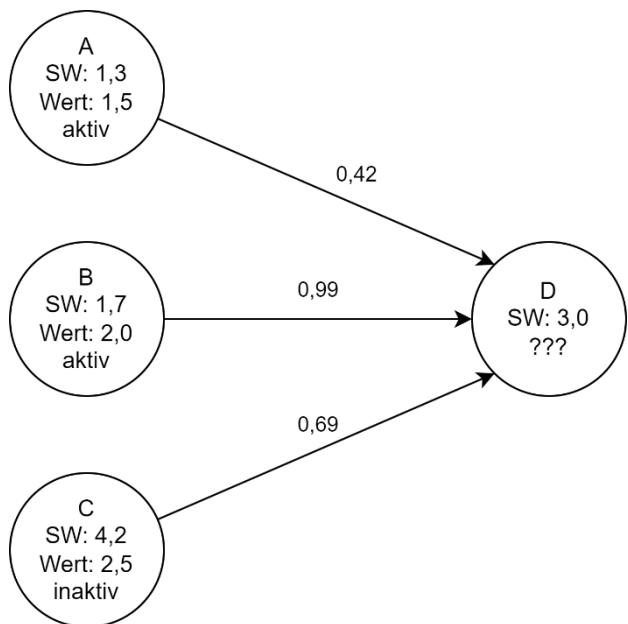
Künstliche Neuronale Netze

Ein **künstliches neuronales Netz** ist ein Modell, das versucht den Lernprozess des menschlichen Gehirns nachzubilden. Dabei werden wesentliche chemische Prozesse (Natrium-Kalium-Pumpen, Myelinscheiden, ...) und biologische Strukturen (Axone, Synapsen, ...) vereinfacht: es gibt Schichten von Neuronen, die miteinander verknüpft werden. Jede Verknüpfung zwischen Neuronen erhält ein Kantengewicht. Jedes Neuron besitzt einen Schwellwert. Wird dieser erreicht oder überschritten, wird das Neuron als „aktiviert“ angesehen. Der erreichte Wert errechnet sich als gewichtete Summe der Werte der zum Neuron hinführenden aktiven Neuronen.

Beispiel

Die Neuronen A und B sind aktiviert, C nicht. Für das Neuron D ergibt sich gemäß der Rechnung ein Wert von $1,5 \cdot 0,42 + 2,0 \cdot 0,99 = 2,61$, sein Schwellwert sei 3,0. Da $2,61 < 3,0$ gilt, zählt dieses Neuron als inaktiv.

Bemerkung: die konkrete Umsetzung des künstlichen neuronalen Netzes hängt von der jeweiligen Anwendung ab. Es kann z.B. sein, dass aktive Neuronen prinzipiell mit dem Zahlenwert 1, inaktive mit 0 als Faktor berechnet werden. Auch die Verwendung spezieller Aktivierungsfunktionen (etwa der Sigmoid-Funktion) hat einen Einfluss auf die Entscheidung, ob ein Neuron aktiv oder inaktiv ist.



Beim Training von Netzen die aus diesen künstlichen Neuronen aufgebaut sind, wird häufig eine zufällige Gewichtung der Kanten zu Beginn gewählt und dann über Backpropagation je nach „erfolgreicher“ oder „erfolgloser“ Ausgabe des Netzes das Kantengewicht angepasst (verringert bzw. erhöht).

Für das Training sind daher viele Daten notwendig.

ChatGPT

ChatGPT, Mitte der 2020er Jahre wohl bekanntester KI-Vertreter, zählt zu den **LLM – Large Language Models**. Dabei wird die KI darauf trainiert nach gegebener Sprachvorgabe („**Prompt**“) die Eingabe zu analysieren und anhand gelernter Muster und Wahrscheinlichkeiten, eine passende Antwort oder Fortsetzung zu erzeugen. Die dabei entstehenden Texte sollten unbedingt fachlich auf ihre Korrektheit geprüft werden. Für das Training von ChatGPT wurde (laut einer Quelle vom 06.03.2024) ein Bestand von 570 GB Textdaten verwendet. Durch die Nutzung von ChatGPT und entsprechende Wertungen der Nutzer (Daumen hoch, Daumen runter) lernt die KI weiter (siehe bestärkendes Lernen). Die Herstellerfirma OpenAI sammelt damit Feedback von Nutzern, um Schwächen zu identifizieren und zukünftige Versionen des Modells zu verbessern um ChatGPT noch präziser, sicherer (schädliche und unangemessene Ausgaben werden vermieden) und nützlicher zu machen.

4.3. Modellierung

Nach Stachowiak (1973) besitzen Modelle drei wesentliche Merkmale:

- Abbildungsmerkmal (jedes Modell ist Abbildung eines Originals)
- Verkürzungsmerkmal (jedes Modell erfasst nicht alle Eigenschaften des Originals)
- Pragmatisches Merkmal (jedes Modell wird aus einem Zweck geschaffen)

Bsp.: Strahlenoptik in der Physik

- Original: Licht
- Modell: Strahlenmodell des Lichts („Licht verhält sich wie Strahlen“)
- Verkürzung: es gibt Phänomene, die mit dem Modell nicht erklärbar sind (z.B. Doppelspaltversuch → Licht hat auch Welleneigenschaften)
- Zweck: grundlegendes Verständnis über Abbildungen (z.B. Lichtbündelung beim Brennpunkt von Sammellinsen)

Beispiele für informative Modelle

- Klasse, Objekte, Attribute und Methoden (KOAM) bei der objektorientierten Programmierung
- Simulationen
- Abläufe bei Automaten (z.B. Pfandautomat)
- Von-Neumann-Modell der Rechnerarchitektur
- Graphen bei der Suche nach kürzesten Wegen innerhalb eines Rechnernetzes
- Symbole (z.B. Disketten-Symbol für das Speichern in einer Anwendungssoftware)
- Modelle im Prozessmanagement oder der Softwareentwicklung (Wasserfallmodell, V-Modell, Kanban, SCRUM, ...)
- Künstliche Neuronale Netze in der KI
- Nachbau von Objekten mit dem 3D-Drucker

Auch für Datenbanken gibt es unterschiedliche Modellansätze. Nach Edgar Codd gibt es dabei drei gemeinsame Eigenschaften: eine generische Datenstruktur (z.B. Relation, Graph, Objekte), generische Operatoren (Umgang mit Daten in der Struktur), sowie eine Menge von Integritätsbedingungen (Einschränkung erlaubter Aktionen).

So gibt es z.B.

- Graphendatenbanken (einzelne Objekte sind Knoten, die miteinander in Verbindung stehen können, was durch Kanten angezeigt wird)
- hierarchische Datenbank (vgl. XML oder HTML-Dokument)
- relationale Datenbank (Grundlage sind Relationen, die meist in Tabellenform dargestellt werden)

Wir fokussieren uns auf relationale Datenbanken. Mathematisch betrachtet sind Relationen Teilmengen des Kreuzprodukts von Mengen. Für unsere Betrachtung genügt es zu vereinfachen: eine Relation entspricht einer Tabelle, die Attribute werden zu Spalten der Tabelle und die einzelnen Datensätze zu Zeilen in der Tabelle.

CAP-Theorem

2000 stellte Prof. Eric Brewer (University of California, Berkeley) eine Vermutung bez. verteilter Systeme auf. Er betrachtete dabei Consistency (Konsistenz), Availability (Verfügbarkeit) und Partition tolerance (Partitionstoleranz) verteilter Systeme.

- **Consistency:** alle Knoten des Systems liefern bei einer Anfrage dieselben Daten (z.B. denselben Kontostand auf dem Geldkonto)
- **Availability:** das System garantiert, dass jede Anfrage eine Antwort erhält (unabhängig davon, ob diese Antwort aktuelle Daten widerspiegelt)
- **Partition tolerance:** das System funktioniert weiterhin (, aber nur in dem Sinne, dass es keinen Datenverlust gibt und Konsistenz gewahrt bleibt, auch wenn es Anfragen blockieren muss).

Seth Gilbert und Nancy Lynch vom MIT bewiesen dann 2002 axiomatisch die Korrektheit der Vermutung von Brewer.

In einem verteilten System ist es unmöglich Konsistenz, Verfügbarkeit und Partitionstoleranz gleichzeitig zu garantieren.

Beispiel 1: DNS-Server

DNS-Server sind ein AP-System, d.h. sie priorisieren Verfügbarkeit und Partitionstoleranz zulasten der Konsistenz. Damit das System hohe Verfügbarkeit (möchte ein Nutzer eine Website aufrufen, sollte er eine Antwort erhalten) und Partitionstoleranz (Netzwerkprobleme in einer Region führen nicht zum Gesamtausfall) gewährleistet, muss es Einschränkungen bei der Konsistenz hinnehmen (Änderungen bei DNS-Einträgen, wie z.B. Aktualisierungen der IP-Adresse einer Website, können Zeit benötigen, um weltweit auf allen DNS-Servern vorgenommen zu werden).

Beispiel 2: Geldautomat

Bei Geldautomaten ist Konsistenz enorm wichtig (der Kunde soll an allen Stellen denselben Kontostand besitzen). Dem folgend muss nun noch zwischen Verfügbarkeit und Partitionstoleranz entschieden werden. Das Gesamtsystem sollte partitionstolerant sein, also auch dann weiterfunktionieren, wenn Teile des Systems z.B. aufgrund von Verbindungsstörungen, nicht erreichbar sind. Dem CAP-Theorem folgend müssen wir dann mit Einbußen bei der Verfügbarkeit leben. So kann es sein, dass einzelne, durch die Störungen belastete, Automaten in der Zeit der Störung für die Kunden nicht verfügbar sind. Entscheidet sich die Bank hingegen dazu, Verfügbarkeit zu priorisieren neben der Konsistenz, so muss sie damit leben, dass das Gesamtsystem nicht partitionstolerant ist, also mit Störungen bei Verbindungen nicht umgehen kann (der Automat könnte dann bei Störungen nur begrenzte Funktionen bieten).

ER-Modell

1976 stellte Peter Chen ein Modell zur Beschreibung von Datenbeständen vor – das **Entity-Relationship-Modell**.

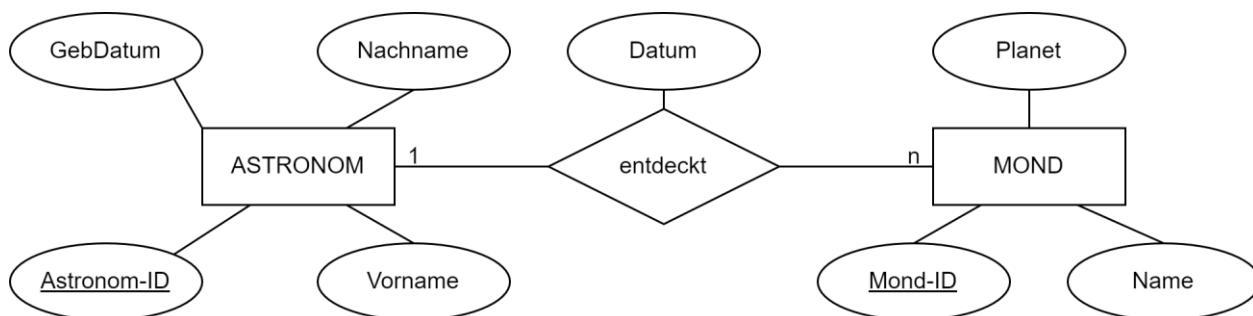
Grundbegriffe dieses Modells

- **Entity**... konkretes Objekt unserer Betrachtung (ein Astronom, ein Mond, ...)
- **Entity Set**... Klasse von Objekten unserer Betrachtung (alle Astronomen, alle Monde, ...)
- **Relationship**... Beziehung zwischen Entity Sets (Astronomen entdecken Monde, Schüler der Sek. II haben Tutoren, ...)
- **Attribut**... sowohl Entity Sets, als auch Relationships können Eigenschaften haben. Alle Entities eines Entity Sets haben dieselben Attribute, aber möglicherweise unterschiedliche Attributwerte. Attribute können auch zusammengesetzt sein (z.B. Adresse aus Straße, Hausnummer, PLZ, Ort).
- **Primärschlüssel**... ein Attribut oder eine Menge von Attributen, die jedes konkrete Objekt eines Entity Sets eindeutig identifiziert

Grundlagen der Darstellung

- Es werden keine einzelnen Entities dargestellt.
- Jedes Entity Set wird von einem Rechteck umrandet.
- Jede Relationship wird von einer Raute umrandet und mit den zugehörigen Entity Sets geradlinig verbunden.
- Jedes Attribut wird von einer Ellipse umrandet und mit dem zugehörigen Entity Set oder der zugehörigen Relationship geradlinig verbunden.
- Primärschlüsselattribute werden einfach unterstrichen.
- Der Beziehungstyp wird für jede Relationship in Nähe der zugehörigen Entity Sets angegeben.

Beispiel



- Entity Sets: ASTRONOM, MOND
- Relationships: entdeckt
- Attribute: GebDatum, Nachname, Vorname, Astronom-ID, Datum, Mond-ID, Planet, Name
- Primärschlüssel: Astronom-ID für ASTRONOM, Mond-ID für MOND
- Kardinalität: 1:n

Kardinalität / Beziehungstyp

Wir unterscheiden bei Relationships drei unterschiedliche Kardinalitäten bezogen auf die jeweiligen Entity Sets. Erklärung folgt anhand des obigen Beispiels.

- **1:1** – jeder Astronom kann maximal einen Mond entdecken, jeder Mond wird von maximal einem Astronomen entdeckt – bei beiden Entity Sets wäre eine 1 zu notieren
- **1:n** – jeder Astronom kann mehrere Monde decken, jeder Mond wird von maximal einem Astronom entdeckt – in dieser Variante: Notation wie im Bild angegeben; andere Variante wäre: jeder Astronom kann maximal einen Mond entdecken, jeder Mond kann von mehreren Astronomen (zuerst) entdeckt werden (dann müssten oben 1 und n vertauscht werden)
- **m:n** – jeder Astronom kann mehrere Monde entdecken, jeder Mond kann von mehreren Astronomen (zuerst) entdeckt werden – bei beiden Entity Sets muss dann jeweils ein Buchstabe stehen (üblich: unterschiedliche Buchstaben verwenden).

Bez. der Arten von Zuordnungen, die in Mathematik kennengelernt haben, entspricht die 1:1-Beziehung der eineindeutigen Zuordnung, die 1:n-Beziehung der eindeutigen Zuordnung und die m:n-Beziehung der mehrdeutigen Zuordnung.

Relationenmodell

Um Datenbanken in eine auswertbare Form zu bringen, verwenden wir das Relationenmodell. Grundlage dieses Modells sind Relationen, die wir in Form von Tabellen darstellen.

Transformationsregeln

- Jedes Entity Set wird zu einer Tabelle gemacht.
 - Jedes Attribut des Entity Sets wird dabei eine Spalte der Tabelle.
 - Primärschlüsselattribute eines Entity Sets werden i.d.R. als erste Spalten gewählt.
- Für Relationships gilt:
 - Besitzt die Relationship selbst Attribute oder ist sie eine m:n-Beziehung, so wird aus ihr eine eigene Tabelle gemacht, welche die Primärschlüssel der beteiligten Entity Sets beinhaltet.
 - Bei 1:1-Beziehungen wird der Primärschlüssel einer der beiden beteiligten Entity Sets der Tabelle des anderen Entity Sets als Fremdschlüssel hinzugefügt.
 - Bei 1:n-Beziehungen wird der Primärschlüssel der 1-Seite als Fremdschlüssel zur n-Seite hinzugefügt

Nach diesen Regeln ergibt sich folgendes Relationenmodell aus dem obigen ER-Diagramm.

ASTRONOM(Astronom-ID, Vorname, Nachname, GebDatum)

MOND(Mond-ID, Name, Planet)

ENTDECKT(Mond-ID, Astronom-ID, Datum)

4.3. Modellierung

Attribute innerhalb einer Relation, die aufgrund einer Relationship in einer anderen Relation Primärschlüssel sind, werden als **Fremdschlüssel** bezeichnet und entweder unterbrochen unterstrichen oder durchgängig überstrichen oder erhalten (FS) als Bemerkung. Primärschlüssel selbst werden bei Relationen durchgängig unterstrichen (oder erhalten (PS) als Bemerkung).

Die Relation ENTDECKT könnte also wie folgt notiert werden.

Variante 1: ENTDECKT(Mond-ID, Astronom-ID, Datum)

Variante 2: ENTDECKT(Mond-ID, Astronom-ID, Datum)

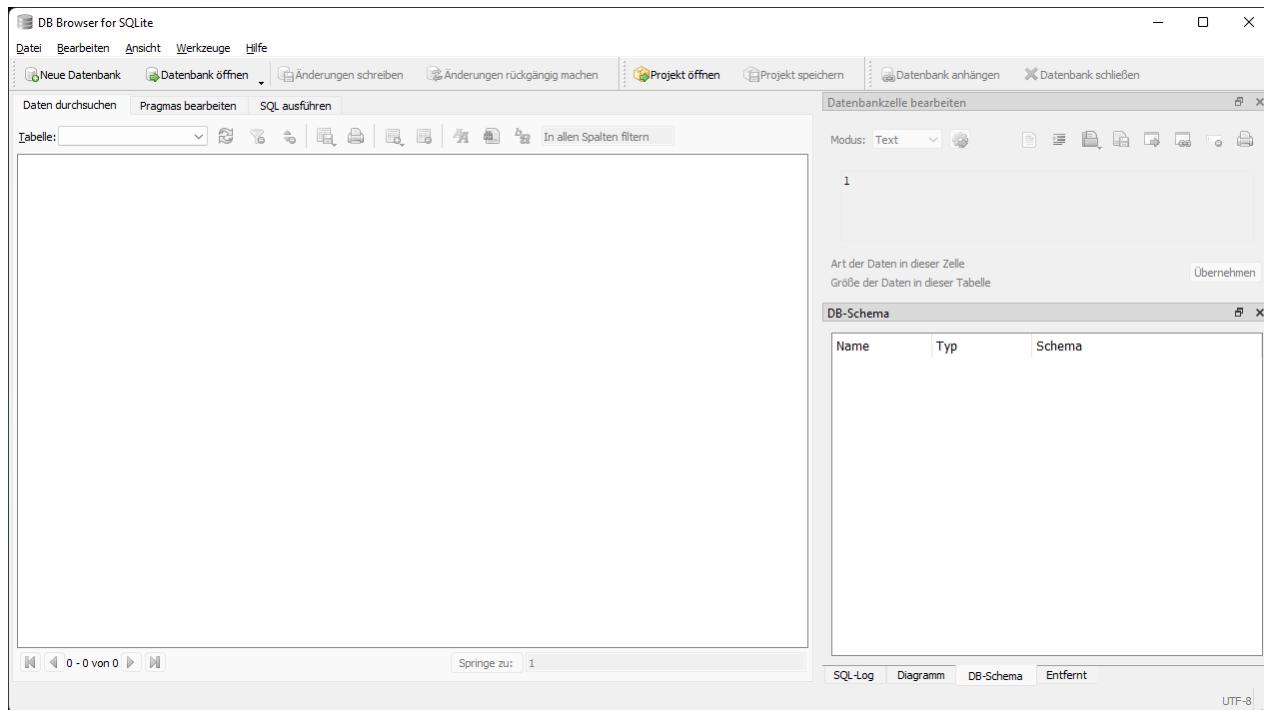
Variante 3: ENTDECKT(Mond-ID (PS, FS), Astronom-ID (FS), Datum)

4.4. Implementierung und Optimierung

4.4. Implementierung und Optimierung

4.4.1. DB Browser for SQLite

Als **Datenbankmanagementsystem (DBMS)** verwenden wir DB Browser for SQLite. Hier können wir eine Datenbank durch Klick auf „Neue Datenbank“ leicht anlegen.

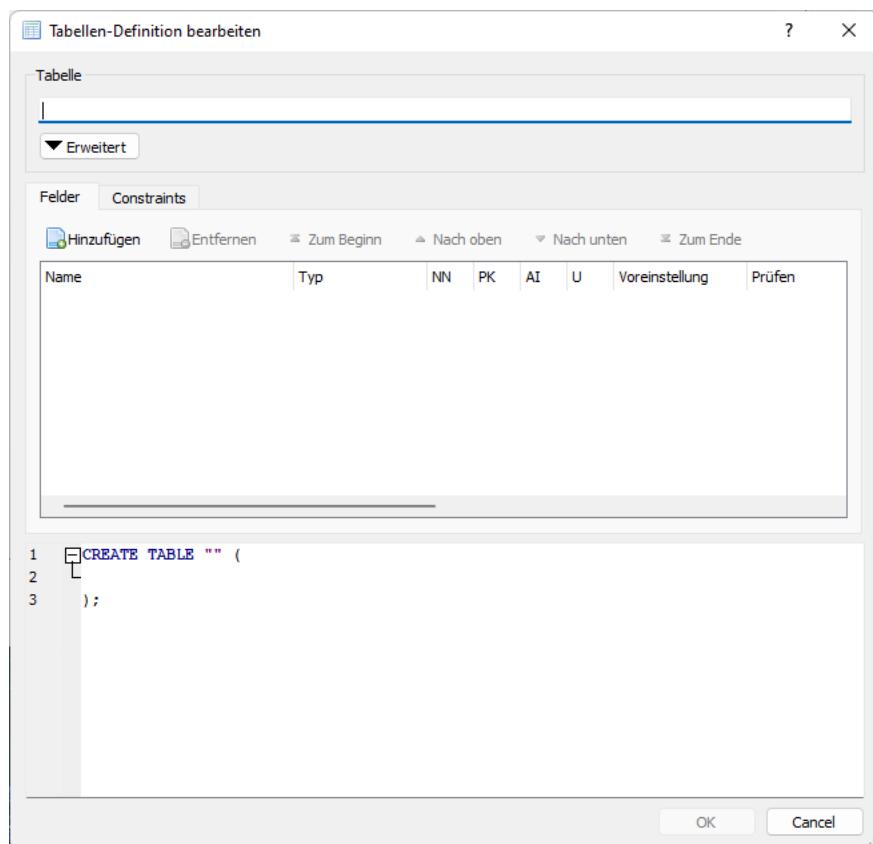


Wir werden dann (nach Wahl des Dateinamens) dazu aufgefordert, eine Tabelle anzulegen.

Dafür geben wir in der Eingabezeile den Namen der Tabelle ein.

Mit Klick auf die „Hinzufügen“-Schaltfläche können wir ein neues Attribut wählen. In der Spalte „PK“ können wir dieses Attribut als Primärschlüsselattribut kennzeichnen. Scrollen wir in diesem Fenster rechts weiter, finden wir auch die Stelle, um ein Attribut als Fremdschlüssel zu kennzeichnen.

Im unteren Fensterabschnitt sehen wir den SQL-Befehl, der sich aus unseren Vorgaben ergibt.



4.4. Implementierung und Optimierung

SQLite-Datenbanken kennen als Datentypen

- TEXT... Zeichenkette (gemäß der Datenbankkodierung z.B. UTF-8)
- INTEGER... ganze Zahl (bis zu 8 Byte)
- BLOB... Binary Large OBject, z.B. Bilder, Tondokumente, ... als Bytestream gespeichert
- REAL... rationale Zahl (als 8-Byte IEEE Fließkommazahl gespeichert)
- NUMERIC... Speicherung numerischer Werte unterschiedlicher Genauigkeit

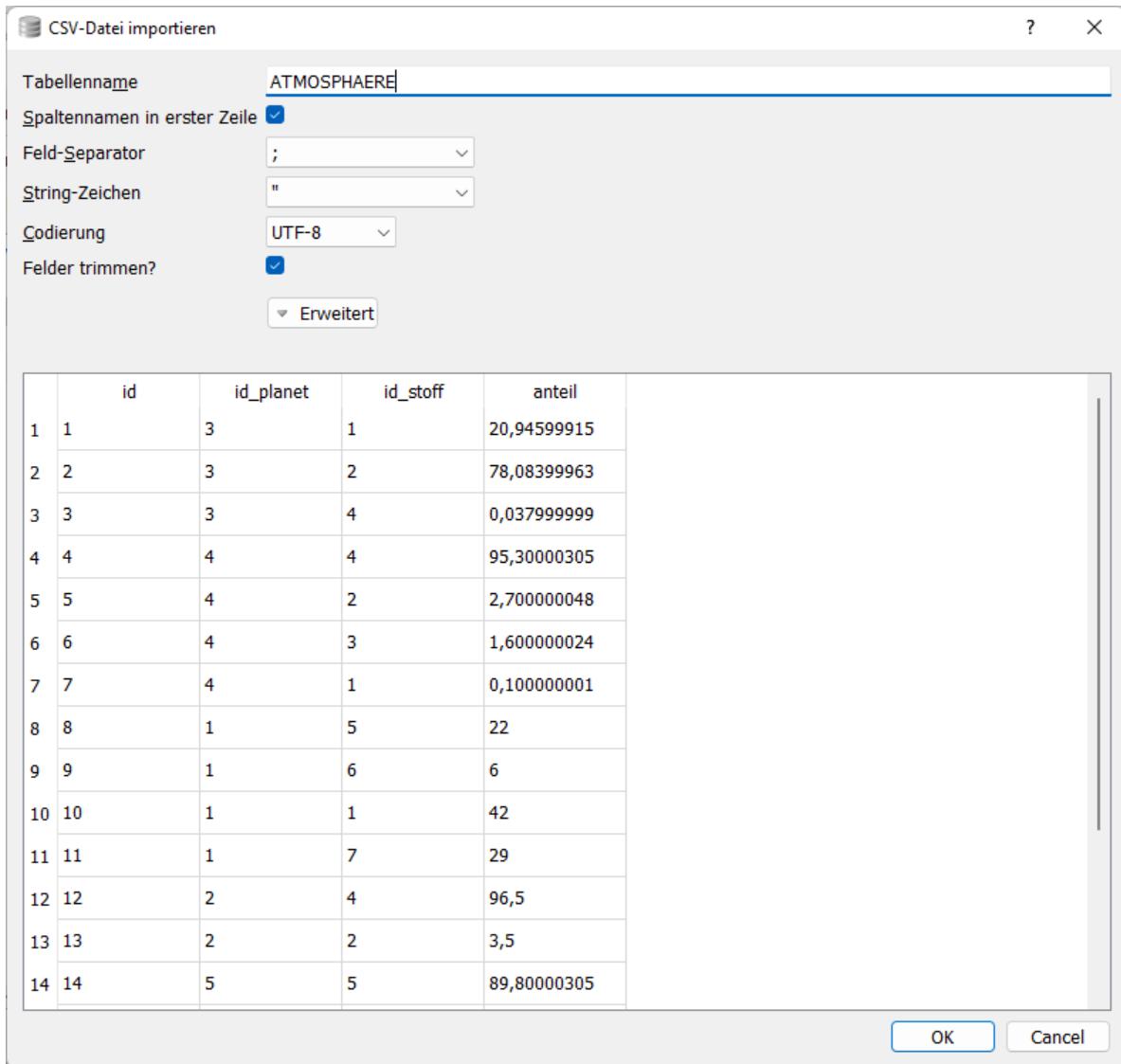
The screenshot shows the 'Tabellen-Definition bearbeiten' (Table Definition Edit) window in SQLite Designer. A new table named 'leih_aus' is being created. The 'Felder' (Fields) tab is selected, showing three columns: 'BuchID' (INTEGER, PK), 'Ausleihernr' (INTEGER), and 'Startdatum' (TEXT). The 'Ausleihernr' column has its 'Fremdschlüssel' (Foreign Key) dropdown open, displaying options: 'Ausleihernr', 'Buch', and 'leih_aus'. The 'Ausleihernr' option is currently selected.

Nach dem Anlegen der Tabellen inkl. Einstellung der Schlüsselattribute können Datensätze hinzugefügt werden. Dafür bei „Daten durchsuchen“ die entsprechende Tabelle wählen. Durch Klick auf die im Bild gekennzeichnete Schaltfläche kann ein neuer Datensatz hinzugefügt werden.

The screenshot shows the DB Browser for SQLite interface. A new row is being inserted into the 'Tabelle2' table. The table has four columns: 'Field1', 'Field2', 'Field3', and 'Field4'. The 'Field4' cell contains the placeholder text 'Fügt eine neue Zeile zur aktuellen Tabelle hinzu'. The status bar at the bottom left also displays this text. On the right side, there are several panes: 'Datenbankzelle bearbeiten' (Database Cell Edit) showing the cell content '1', 'Art der Daten in dieser Zelle: Text / Numerisch 1 Zeichen' (Type of data in this cell: Text / Numerical 1 character), and 'DB-Schema' (DB Schema) pane showing the table structure: CREATE TABLE "Tabelle2" ("Field1" , "Field2" , "Field3" , "Field4").

4.4. Implementierung und Optimierung

Alternativ können auch CSV-Dateien ausgelesen werden. Dafür im Datei-Menü im Menüpunkt „Import“ den Eintrag „Tabelle aus CSV-Datei“ auswählen. Es öffnet sich (nach Auswahl der Datei) der hier dargestellte Dialog, in dem z.B. das Trennzeichen (hier Semikolon) und die Codierung der CSV-Datei passend ausgewählt werden kann.



Im Menüpunkt „Export“ können umgekehrt aus existierenden Tabellen der Datenbank auch z.B. CSV-Dateien erzeugt werden.

4.4.2. Normalformen

Ziel: Vermeidung von **Redundanzen** (Wiederholungen) und **Anomalien** (Probleme beim Einfügen, Löschen oder Aktualisieren).

- **1. Normalform (1NF)**

Jedes Attribut der Relation muss einen atomaren Wertebereich haben.

- **2. Normalform (2NF)**

Relation liegt in 1NF vor und jedes Nicht-Schlüsselattribut ist von jedem Schlüsselkandidaten vollständig funktional abhängig.

Schlüsselkandidat... minimale Menge von Attributen, mit denen die Datensätze eindeutig identifiziert sind.

vollständig funktional abhängig... Nicht-Schlüsselattribut ist nicht nur von einem Teil des Primärschlüssels abhängig

- **3. Normalform (3NF)**

Relation liegt in 2NF vor und jedes Nicht-Schlüsselattribut ist von keinem Schlüsselkandidaten transitiv abhängig.

Transitiv abhängig... Ist Y von X funktional abhängig und Z von Y, so ist Z transitiv von X abhängig.

Neben diesen drei Normalformen existieren weitere Normalformen, etwa die Boyce-Codd-Normalform (BCNF), die 4NF und die 5NF (auch als Project-Join-Normalform PJNF bezeichnet). Für schulische Zwecke sollten die Normalformen bis zur 3NF genügen.

4.5. Verarbeitung und mehr

Zur Abfrage bei Datenbanken verwenden wir **SQL**. Dies ist das Kürzel für **Structured Query Language** und ist eine Abfragesprache.

Abfragen werden bei relationalen Datenbanken in der Regel durch Transaktionen nach dem ACID-Prinzip realisiert. Eine **Transaktion** ist dabei eine Folge von Programmschritten, die als logische Einheit betrachtet werden kann.

ACID-Prinzip

- Atomicity... jede Transaktion wird ganz oder gar nicht umgesetzt
- Consistency... wenn die Datenbasis vor der Transaktion konsistent war, dann ist sie es auch nach der Transaktion
- Isolation... parallel ausgeführte Transaktionen dürfen sich nicht gegenseitig beeinflussen
- Durability... die Auswirkung einer Transaktion ist dauerhaft

Transaktionen werden entweder abgeschlossen (commit) oder abgebrochen (abort).

Index

Ein **Index** ist eine Ansammlung von Zeigern, die eine Ordnungsrelation auf eine oder mehrere Spalten definieren. Damit kann bei Abfragen das Ergebnis schneller ermittelt werden insbesondere beim Vorhandensein großer Datenmengen (siehe Big Data). Primärschlüssel der Relationen werden in der Regel automatisch zu einem Index der relationalen Datenbank gemacht.

Teilsprachen von SQL

Wir unterscheiden zwischen folgenden Teilsprachen von SQL.

- **DQL... Data Query Language**
Abfrage und Aufbereitung der gesuchten Informationen (SELECT ... FROM ... WHERE...)
- **DML... Data Manipulation Language**
Ändern, Einfügen und Löschen von Daten (UPDATE, INSERT INTO, DELETE FROM)
- **DDL... Data Definition Language**
Ändern, Einfügen und Löschen von Relationen ([ALTER|CREATE|DROP] TABLE)
- **DCL... Data Control Language**
Rechteverwaltung für Nutzer (GRANT, REVOKE)
- **TCL... Transaction Control Language**
Kontrolle über die Transaktionen (COMMIT, ROLLBACK)

Im Rahmen des Unterrichts beschränken wir uns auf die ersten drei Teilsprachen.

grundlegende Befehle am Beispiel

CREATE TABLE "MOND" ("MondID" INTEGER, "Name" TEXT, "PlanetID" INTEGER, "Bahnradius" INTEGER, "Masse" INTEGER, FOREIGN KEY("PlanetID") REFERENCES "PLANET"("PlanetID"), PRIMARY KEY("MondID", "Name"));	Erstellen einer Tabelle mit verschiedenen Attributen sowie Angabe von Fremd- und Primärschlüsseln
ALTER TABLE ASTRONOM ADD Jahr INTEGER;	Hinzufügen einer Spalte zur Tabelle
DROP TABLE ASTRONOM;	Entfernen einer Tabelle
INSERT INTO ASTRONOM(FName, VName) VALUES ('Kepler', 'Johannes');	Hinzufügen eines Datensatzes zu einer Tabelle
UPDATE ASTRONOM SET AstronomID=42 WHERE AstronomID=2;	Aktualisieren eines Datensatzes in einer Tabelle
DELETE FROM ASTRONOM WHERE AstronomID=3;	Entfernen eines Datensatzes aus einer Tabelle
DELETE FROM ASTRONOM;	Entfernen aller Datensätze einer Tabelle
SELECT * FROM ASTRONOM;	Anzeige aller Datensätze und Spalten einer Tabelle
SELECT VName AS "Vorname", FName FROM ASTRONOM;	Projektion: Anzeige ausgewählter Spalten einer Tabelle (ggf. mit Umbenennung durch AS)
SELECT * FROM ASTRONOM WHERE AstronomID < 2;	Selektion: Anzeige aller Datensätze, die eine Bedingung erfüllen
SELECT DISTINCT Vname FROM ASTRONOM;	Anzeige unterschiedlicher Attributwerte (sich wiederholende Daten werden nicht mehrfach angezeigt)
SELECT * FROM ASTRONOM WHERE FName LIKE '%a%';	Vergleich bei Zeichenketten innerhalb der Bedingung (Platzhalterzeichen % für „alles oder auch nichts“ kann je nach Software unterschiedlich sein!)
SELECT * FROM GUTHABEN ORDER BY Guthaben DESC;	Sortieren der Datensätze bei der Anzeige (DESC... absteigend, ASC... aufsteigend)
SELECT * FROM GUTHABEN ORDER BY Guthaben ASC LIMIT 1;	Begrenzung der anzuzeigenden Datensätze – hier wird nur der 1. Eintrag der nach Sortierung erhaltenen Tabelle angezeigt
SELECT VName, COUNT(*) AS Anzahl FROM ASTRONOM GROUP BY VName;	Gruppierung und Zählen von Datensätzen COUNT ist eine Aggregationsfunktion . Außer COUNT gibt es noch SUM, MAX, MIN und AVG (arithmetisches Mittel)

Verknüpfen mehrerer Tabellen

Mehrere Tabellen können über das Kreuzprodukt verbunden werden. Dabei wird aber jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle verknüpft. Dies ist in der Regel (bei großen Datenmengen) nicht effizient und sollte durch die Verwendung von Joins vermieden werden.

<code>SELECT * FROM Tab1, Tab2;</code>	Kreuzprodukt der beiden Tabellen
<code>SELECT * FROM Tab1, Tab2 WHERE Tab1.ID = Tab2.ID;</code>	Kreuzprodukt der beiden Tabellen, erst danach wird nach der Bedingung eingeschränkt
<code>SELECT * FROM Tab1 INNER JOIN Tab2 ON Tab1.ID = Tab2.ID;</code>	Inner Join ... selbes Ergebnis wie bei Abfrage vorher, aber i.d.R. wesentlich effizienter Es verbleiben nur diejenigen Datensätze, die in Tab1 und Tab2 vorkommen und dort denselben Wert bei Attribut ID haben
<code>SELECT * FROM Tab1 LEFT JOIN Tab2 ON Tab1.ID = Tab2.ID;</code>	Left Join ... zusätzlich zum Inner Join werden die Datensätze hinzugefügt, die in Tab1 enthalten sind, aber in Tab2 keinen passenden Partner haben (Attribute aus Tab2 werden dann mit NULL belegt)
<code>SELECT * FROM Tab1 RIGHT JOIN Tab2 ON Tab1.ID = Tab2.ID;</code>	Right Join ... zusätzlich zum Inner Join werden die Datensätze hinzugefügt, die in Tab2 enthalten sind, aber in Tab1 keinen passenden Partner haben (Attribute aus Tab1 werden dann mit NULL belegt); wird in DB Browser for SQLite derzeit nicht unterstützt (ggf. durch Left Join ersetzen und Tabellenreihenfolge tauschen)
<code>SELECT * FROM Tab1 FULL JOIN Tab2 ON Tab1.ID = Tab2.ID;</code>	Full Join ... Vereinigung aus Left Join und Right Join; wird in DB Browser for SQLite derzeit nicht unterstützt (ggf. durch UNION von Left Join und umgedrehten Left Join ersetzen)

Verknüpfungen können auch verschachtelt genutzt werden, z.B.

```
SELECT personid, familienname, vorname, ucs, PersoniPadListe.name_intern,
PersoniPadListe.seriennummer, PersoniPadListe.inventarnummer,
PencilListe.objektid, PencilListe.seriennummer FROM (SELECT personid,
familienname, vorname, ucs, name_intern, seriennummer, inventarnummer FROM
(SELECT personid, familienname, vorname, ucs FROM Person WHERE personid IN
(SELECT personid FROM Schueler WHERE klasse LIKE Welche_Klasse)) AS
Personenliste LEFT JOIN (SELECT * FROM Objekt WHERE modell LIKE "*iPad*") AS
iPadListe ON personenliste.personid = iPadListe.verantwortlich) AS
PersonIPadListe LEFT JOIN (SELECT * FROM Objekt WHERE modell LIKE
'*Pencil*') AS PencilListe ON PersonIPadListe.personid =
PencilListe.verantwortlich ORDER BY familienname, vorname;
```

5. Sprachen und Automaten

5.1. Sprachen

Natürliche und Formale Sprachen

Im Gegensatz zu **natürlichen Sprachen**, die sich evolutionär entwickeln und meist mehrdeutig sind (z.B. Deutsch, Englisch, ...) sind **formale Sprachen** künstlich definiert, präzise und eindeutig (z.B. Programmiersprachen, mathematische Logik).

Solche formalen Sprachen können u.a. über Grammatiken angegeben werden.

Grammatiken

Wir können **formale Grammatiken** als Quadrupel $G = (N, T, P, S)$ beschreiben. Dabei stehen die vier Symbole für

- N ... Menge der Nichtterminale (Platzhalter für Strukturen, z.B. Satz, Variable)
- T ... Menge der Terminalen (tatsächliche Zeichen der Sprache)
- P ... Menge der Produktionsregeln (Regeln zur Ersetzung von Nichtterminalen)
- S ... Startsymbol (Ausgangspunkt der Ableitung)

Bsp.: Sei die Grammatik G gegeben über $(\{S; A\}, \{a; b\}, \{S \rightarrow aS|aA; A \rightarrow b\}, S)$. Diese Grammatik besitzt die Nichtterminale S und A , die Terminalen a und b , aus denen jedes Wort dieser Sprache besteht. Jedes Wort der Sprache erhält man, indem man beim Startsymbol S startend alle verfügbaren Produktionsregeln verwendet. In dieser Grammatik gibt es drei Regeln: $S \rightarrow aS$, $S \rightarrow aA$ und $A \rightarrow b$.

Das Finden eines Wortes unter Nutzung der Regeln bezeichnen wir als **Ableiten** des Wortes aus der Grammatik.

So lässt sich z.B. das Wort aaaab aus der Grammatik wie folgt ableiten.

$$S \xrightarrow{r_1} aS \xrightarrow{r_1} aaS \xrightarrow{r_1} aaaS \xrightarrow{r_2} aaaaA \xrightarrow{r_3} aaaab$$

Bemerkung: es gibt auch Autoren, die statt $G = (N, T, P, S)$ das Quadrupel $G = (V, A, R, S)$ verwenden. Dabei gilt: $R = P$, $T = A$ (Alphabet) und $V = N \cup T$ als „Vokabular“ der Grammatik.

Backus-Naur-Form und Syntaxdiagramme

Die **Backus-Naur-Form** (kurz **BNF**) ist eine Notation zur Definition formaler Sprachen.

Dabei werden letztlich die Produktionsregeln in besonderer Weise dargestellt, sodass aus ihnen die Mengen N und T direkt ersichtlich werden. Dafür werden Nichtterminale üblicherweise von `<>` umklammert (darauf kann verzichtet werden, wenn Terminalen durchgängig in Anführungsstriche gesetzt werden). Definitionen werden mit `::=` angegeben.

In der **Erweiterten Backus-Naur-Form** (kurz **EBNF**) werden darüber hinaus Wiederholungen durch geschweifte Klammern `{ }` und optionale Elemente durch eckige Klammern `[]` gekennzeichnet.

5.1. Sprachen

Bsp.: eine ganze Zahl (ggf. mit führenden Nullen) kann in folgender EBNF angegeben werden.

```
<Zahl> ::= [“-“] <Ziffer> {<Ziffer>}  
<Ziffer> ::= “0“ | “1“ | “2“ | “3“ | “4“ | “5“ | “6“ | “7“ | “8“ | “9“
```

Im Beispiel kann optional ein Minus notiert werden, dann muss mindestens eine Ziffer folgen und darauf beliebig viele (auch null-malige Wiederholung ist erlaubt bei { }) weitere Ziffern.

Bemerkung: es gibt auch Autoren, die anstelle von := nur = schreiben.

Reguläre Ausdrücke

Reguläre Ausdrücke sind formale Notationen zur Beschreibung des Aufbaus von Zeichenkettenmustern.

Bsp.: (0b)?[01]+ ist ein regulärer Ausdruck zur Beschreibung einer Binärzahl. Dabei kann optional die Zeichenkette 0b vorangestellt werden zur Kennzeichnung als Binärzahl. Daraufhin folgen beliebig viele, jedoch mindestens ein Zeichen aus der Menge {0;1}. So gehören z.B. 0b1, 0, 1, 0b101010 zu den so beschriebenen Zeichenketten.

Bestandteile:

- . als Platzhalter für ein beliebiges Zeichen
- * als Kennzeichnung für beliebig häufige, mindestens nullmalige Wiederholungen
- + als Kennzeichnung für beliebig häufige, mindestens einmalige Wiederholungen
- ? als Kennzeichnung für optionale Bestandteile
- [abc] als Kennzeichnung dafür, dass aus den gegebenen Zeichen {a;b;c} genau eines zu wählen ist

Bsp. 2: ein möglicher regulärer Ausdruck für eine E-Mail-Adresse lautet:

[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.\[a-zA-Z]{2,}

- beginnt mit dem Benutzernamen, der aus mindestens einem der Zeichen besteht: Buchstaben, Ziffern, Punkt, Unterstrich, Prozentzeichen, Plus- und Minuszeichen
- es folgt ein @
- es folgt der Domain-Name (mindestens ein Zeichen aus: Buchstaben, Ziffern, Punkt, Minus)
- es folgt ein Punkt (muss als \. geschrieben werden, da . Platzhalter für beliebige Zeichen ist)
- es folgt die Top-Level-Domain (bestehend aus mindestens 2 Buchstaben)

Chomsky-Hierarchie

Noam Chomsky ordnete Grammatiken einem jeweiligen Typ zu. Dafür müssen die Bestandteile der Produktionsregelmenge angesehen werden.

Typ 0:

bei jeder Regel steht auf der linken Seite mindestens ein Nichtterminal und es gibt endlich viele Produktionsregeln.

Typ 1 – **kontextsensitive Grammatik**:

Die Grammatik erfüllt Typ 0 und bei jeder Regel ist die linke Seite nicht länger als die rechte Seite. Die Länge der beiden Seiten ergibt sich dabei jeweils durch Auszählen der vorhandenen Nichtterminale und Terminalen. Erlaubte Ausnahme: $S \rightarrow \epsilon$

Typ 2 – **kontextfreie Grammatik**:

Die Grammatik erfüllt Typ 1 und bei jeder Regel steht auf der linken Seite ausschließlich genau ein Nichtterminal (also weder weitere Nichtterminale noch Terminalen).

Typ 3 – **reguläre Grammatik**:

Die Grammatik erfüllt Typ 2 und bei jeder Regel steht auf der rechten Seite genau ein Nichtterminal. Über alle Produktionsregeln hinweg steht es dabei immer auf derselben Seite von etwaigen Terminalen (also immer Nichtterminal rechts oder immer Nichtterminal links).

Steht das Nichtterminal immer rechts, wird die Grammatik als **rechtsregulär** bezeichnet; steht es links, als **linksregulär**.

Bsp.: $(\{S; A\}, \{a; b\}, \{S \rightarrow aS | aA; A \rightarrow b\}, S)$

Diese Grammatik ist mindestens vom Typ 2, da sie endlich viele Regeln (drei) besitzt und jede Regel links nur genau ein Nichtterminal besitzt. Sie ist sogar vom Typ 3, da bei beiden Regeln, bei denen rechts ein Nichtterminal vorhanden ist, nur ein Nichtterminal existiert auf der rechten Seite und dieses auf derselben Seite, nämlich rechts, steht. Diese Grammatik ist damit rechtsregulär.

5.2. Automaten

Eine andere Möglichkeit zur Beschreibung von Sprachen bildet der Automat.

Endliche Automaten

Ein **endlicher Automat** ist ein Quintupel $\mathcal{A} = (Z, \Sigma, \delta, z_0, F)$ mit folgenden Bestandteilen.

- Z ... endliche Menge an Zuständen, die der Automat annehmen kann
- Σ ... Eingabealphabet des Automaten
- δ ... Zustandsübergangsfunktion
- z_0 ... Startzustand des Automaten
- F ... Menge der Finalzustände

Wir unterscheiden zwischen **NEA** (nichtdeterministischer endlicher Automat) und **DEA** (deterministischer endlicher Automat).

DEA: $\delta: Z \times \Sigma \rightarrow Z$

Für jeden Zustand und jedes Eingabezeichen gibt es maximal einen Folgezustand.

NEA: $\delta: Z \times \Sigma \rightarrow \mathfrak{P}(Z)$

Potenzmenge in Word: ALT+120083

Für jeden Zustand und jedes Eingabezeichen gibt es eine (ggf. mehrelementige) Menge möglicher Folgezustände.

Beispiel:

$\mathcal{A} = (\{q_0; q_1\}, \{a; b\}, \delta, q_0, \{q_1\})$ ist ein DEA, der alle auf a endenden Wörter mit den Zeichen a und b akzeptiert, wenn $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_0$, $\delta(q_1, a) = q_1$ und $\delta(q_1, b) = q_0$ gilt.

Der Automat kann wahlweise auch durch eine Tabelle für δ oder ein Zustandsdiagramm angegeben werden. Bei der Tabelle sind Startzustand (Exponent S) und Finalzustand (Exponent F) gekennzeichnet.

Tabelle			Zustandsdiagramm									
<table border="1"> <tr> <td>δ</td><td>q_0^S</td><td>q_1^F</td></tr> <tr> <td>a</td><td>q_1</td><td>q_1</td></tr> <tr> <td>b</td><td>q_0</td><td>q_0</td></tr> </table>			δ	q_0^S	q_1^F	a	q_1	q_1	b	q_0	q_0	
δ	q_0^S	q_1^F										
a	q_1	q_1										
b	q_0	q_0										

Finalzustände werden im Zustandsdiagramm durch zwei konzentrische Kreise gekennzeichnet. Die für den Übergang benötigten Eingabezeichen werden auf die Pfeile zwischen den Zuständen notiert.

Überführen rechtsregulärer Sprachen in Automaten

Bei rechtsregulären Sprachen sind alle Produktionsregeln entweder in der Form $A \rightarrow xB$ oder der Form $A \rightarrow x$ vorhanden.

Der Automat kann wie folgt gebildet werden.

Die Zustandsmenge entspricht der Menge der Nichtterminale ggf. ergänzt um Finalzustände.

Die Menge der Eingabezeichen entspricht der Menge der Terminale.

Der Startzustand ist der Zustand, der aus dem Startsymbol der Grammatik entsteht.

Jede Regel der Form $A \rightarrow xB$ kann in $\delta(A, x) = B$ umgewandelt werden.

Jede Regel der Form $A \rightarrow x$ kann in $\delta(A, x) = F$ umgewandelt werden, wobei F ein ggf. neu gebildeter Finalzustand ist.

So kann die Grammatik mit Produktionsregeln $S \rightarrow aA \mid bS \mid a; A \rightarrow bA \mid a$ in den Automaten $\mathcal{A} = (\{S; A; F\}, \{a; b\}, \delta, S, \{F\})$ mit $\delta(S, a) = \{A; F\}, \delta(S, b) = \{S\}, \delta(A, a) = \{F\}, \delta(A, b) = \{A\}$ umgewandelt werden. Dieser ist offensichtlich ein NEA, kann aber in einen DEA umgewandelt werden.

Umgekehrt kann ein endlicher Automat (egal ob NEA oder DEA) stets in eine rechtsreguläre Sprache umgewandelt werden. Die Grundidee ist dabei identisch zu der oben beschriebenen für die andere Richtung.

Übergang NEA – DEA

Michael Oser Rabin und Dana Stewart Scott wiesen 1959 nach, dass es zu jedem NEA einen äquivalenten DEA gibt (d.h., dass NEA und DEA dieselbe Sprache akzeptieren).

Am Beispiel: $\delta(S, a) = \{A; F\}$, $\delta(S, b) = \{S\}$, $\delta(A, a) = \{F\}$, $\delta(A, b) = \{A\}$

Beginne beim Startzustand S , notiere den Startzustand $q_0 := \{S\}$ des DEA und betrachte alle Folgezustandsmengen die durch ein Eingabezeichen aus S entstehen, hier $\{A; F\}$ und $\{S\}$. Jede noch nicht vorhandene Menge definierst du als neuen Zustand. Hier $q_1 := \{A; F\}$ für den Übergang von q_0 bei Eingabe von a . Bei Eingabe von b ergibt sich $\{S\} = q_0$.

δ_D	$q_0 = \{S\}$
a	$q_1 := \{A; F\}$
b	$q_0 = \{S\}$

Betrachte nun für jeden noch nicht vollständig betrachteten Zustand, hier q_1 alle möglichen Folgezustände. Der NEA führte ausgehend von A bei Eingabe von a zu F , ausgehend von F gab es bei Eingabe von a keinen Folgezustand. Damit muss der DEA bei Eingabe von a aus dem Zustand q_1 in den Zustand $\{F\}$ übergehen, den es bisher nicht gab. Hätte es bei F Folgezustände gegeben, hätten die Zustandsmengen über Vereinigungsmenge verbunden werden müssen.

δ_D	$q_0 = \{S\}$	$q_1 = \{A; F\}$
a	$q_1 = \{A; F\}$	$q_2 := \{F\}$
b	$q_0 = \{S\}$	$q_3 := \{A\}$

Dieses Prinzip wird nun für jeden entstandenen Zustand fortgesetzt, bis sich kein neuer Zustand mehr ergibt. Finalzustände sind alle neuen Zustände, die Finalzustände des NEA beinhalten (hier also den Zustand F in der Menge enthalten).

δ_D	q_0^S	q_1^F	q_2^F	q_3
a	q_1	q_2	-	q_2
b	q_0	q_3	-	q_3

mit $q_0 := \{S\}$, $q_1 := \{A; F\}$, $q_2 := \{F\}$, $q_3 := \{A\}$

Klassifizierung von Automaten

Die von uns betrachteten Automaten - DEA und NEA – benötigen zwingend eine reguläre Sprache als Grundlage. Für Sprachen bis zum Chomsky-Typ 2 existieren besondere Automaten.

- Typ 0: Turingmaschine
- Typ 1: Linear beschränkter Automat
- Typ 2: Kellerautomat

Turingmaschine

Die Turingmaschine besitzt ein unendliches Band, auf dem die Eingabe in einzelnen Feldern notiert ist. Über diesem Band ist ein Schreib-Lese-Kopf positioniert, der beweglich ist und von einem Feld den aktuellen Inhalt lesen bzw. einen neuen Inhalt schreiben kann. Leere Feldelemente werden mit dem Bandsymbol ■ belegt.

$$\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \delta, F)$$

$Q, \Sigma, q_0, F \dots$ siehe NEA/DEA

$\Gamma \dots$ Arbeitsalphabet mit $\Sigma \subseteq \Gamma$,

■ $\in \Gamma \setminus \Sigma \dots$ Bandsymbol für leeres Feld

$\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{l; r; n\} \dots$ Übergangsfunktion

$(q, a, q', a', l) \rightarrow$ Turingmaschine befindet sich im Zustand q und liest ein a im Arbeitsfeld. Sie notiert a' in das Arbeitsfeld, geht in Zustand q' über und bewegt den Schreib-Lese-Kopf um ein Feld nach links (bei r rechts, bei n gar nicht).

Linear beschränkter Automat

Ein linear beschränkter Automat ist eine Turingmaschine, deren Arbeitsalphabet zwei besondere Symbole ■ und □ besitzt, die links bzw. rechts nicht überschritten werden dürfen und selbst nicht geschrieben werden dürfen.

Formaler notiert:

- Übergänge (q, \blacksquare, \dots) dürfen nur in der Form $(q, \blacksquare, q', \blacksquare, r)$ existieren, d.h. ■ darf nicht links überschritten oder beschrieben werden
- Übergänge (q, \square, \dots) dürfen nur in der Form $(q, \square, q', \square, l)$ existieren, d.h. □ darf nicht rechts überschritten oder beschrieben werden.

Kellerautomat / Pushdown automaton (PDA)

Grundidee: Erweitern eines NEA mit einem Kellerstapel

$$\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$$

Q, Σ, q_0, F ... siehe NEA/DEA

Γ ... Kelleralphabet mit Kellerstartsymbol Z_0

$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$... Übergangsfunktion

Es gibt zwei unterschiedliche Arten von Zustandsübergängen.

- (q, a, Z, q', γ) ... Der Kellerautomat befindet sich im Zustand q mit oberstem Kellersymbol Z und Eingabezeichen a . Er geht in den Zustand q' über und schreibt das/die Zeichen γ in den Keller.
- $(q, \varepsilon, Z, q', \gamma)$... Der Kellerautomat befindet sich im Zustand q mit oberstem Kellersymbol Z und kann spontan (also unabhängig davon was das nächste Eingabezeichen ist) in den Zustand q' übergehen und dabei das/die Zeichen γ in den Keller schreiben.

Achtung:

- Bei jedem Übergang wird das oberste Kellersymbol Z vom Kellerstapel entfernt. Soll es stehen bleiben, muss es wieder neu eingekellert werden.
- Befinden sich bei γ mehrere Zeichen, so werden sie von hinten nach vorn eingekellert (Bsp.: $ab1 \rightarrow$ erst wird 1 eingekellert, dann b , dann a , oberstes Symbol des Kellers ist nun a).
- Üblicherweise verwendet man $\#$ als Startsymbol für den Kellerspeicher. Dies ermöglicht das formal korrekte Herausfinden, ob der Kellerstapel vollständig abgearbeitet wurde.

5.3. Übersetzung formaler Sprachen

Interpreter	Compiler
führt den Quelltext direkt zur Laufzeit Zeile für Zeile aus	übersetzt den gesamten Quelltext vor der Ausführung in Maschinencode
Bsp.: Python, VBScript, PHP, JavaScript	Bsp.: C, C++, Java, Rust, ObjectPascal

Der Übersetzungsprozess verläuft in mehreren Schritten.

Schritt 1: Lexikalische Analyse / Tokenisierung

Ein Scanner zerlegt den Quelltext in sogenannte **Tokens**. Diese stellen die Grundbausteine des Programms dar.

Das Ergebnis der lexikalischen Analyse kann in Python bspw. durch folgendes Skript angezeigt werden.

```
import tokenize

dateiname = "Bsp1.py"

with tokenize.open(dateiname) as f:
    tokens = tokenize.generate_tokens(f.readline)
    for token in tokens:
        print(token)
```

Dies ist auch als Kommandozeilenaufruf möglich z.B. mit `python -m tokenize Bsp1.py` in der Windows-Eingabeaufforderung.

Hier werden die erkannten Token angezeigt mit Tokentyp und weiteren Informationen.

Das folgende Skript zeigt alle Tokentypen an, die Python derzeit kennt.

```
import token

# Alle Token-Typen mit Namen ausgeben
for tok_num, tok_name in token.tok_name.items():
    print(f"{tok_num}: {tok_name}")
```

Schritt 2: Syntaktische Analyse / Parsing

Die Programmstruktur wird analysiert und in eine Baumstruktur überführt – einen **Syntaxbaum**, auch als **Parse Tree** oder **AST (Abstract Syntax Tree)** bezeichnet.

5.3. Übersetzung formaler Sprachen

```
import ast

# Datei einlesen
with open("Bsp1.py", "r", encoding="utf-8") as file:
    source_code = file.read()

# AST parsen
tree = ast.parse(source_code)

# AST ausgeben
print(ast.dump(tree, indent=4)) # Schöner formatierte Ausgabe
```

Schritt 3: Semantische Analyse

Beim nächsten Schritt wird z.B. geprüft, ob alle Variablen deklariert sind, die Datentypen zusammenpassen oder es unerlaubte Operationen gibt.

Weitere Schritte

Nach den drei bisher durchgeführten Analysen – lexikalische, syntaktische und semantische Analyse – wird der Quelltext in eine **Zwischenrepräsentation** umgewandelt. Dies wird auch als **Intermediate Representation** kurz mit **IR** bezeichnet.

```
import dis

dateiname = "Bsp1.py"

with open(dateiname, "r", encoding="utf-8") as file:
    source_code = file.read()

# Code kompilieren
compiled_code = compile(source_code, dateiname, "exec")

# Bytecode ausgeben
dis.dis(compiled_code)
```

Die Zeilen folgen einem einheitlichen Muster:

[Zeilennummer im Originalquelltext] [Bytecode-Offset] [Bytecode-Operation]
[Operand] ([Kommentar])

Weiterhin wird der Maschinencode optimiert z.B. durch Konstantenfaltung (aus 10+5 wird direkt 15 gemacht), Tote Code-Elimination (Entfernen unnötigen Codes) und Schleifenoptimierung.

Aus dem nun vorhandenen Code wird nun Maschinencode gemacht, der zum jeweiligen Prozessor passt. Bei kompilierten Sprachen wird zuletzt noch das Linken betrieben (statisches Linken: Platzhalter für externe Module werden durch Speicheradresse ersetzt; dynamisches Linken: es wird sichergestellt, dass die Bibliothek zur Laufzeit geladen wird).

Nochmal Interpreter vs. Compiler

Klassische Interpreter gehen nur bis zur semantischen Analyse, klassische Compiler führen alle beschriebenen Schritte aus. Es gibt Hybridlösungen. So wird bei Python üblicherweise neben den drei Analysen auch noch die Erzeugung von IR und teilweise Optimierung durchgeführt.

Stichwortverzeichnis

<u>__init__</u>	42	Automat	123
—			
1		B	
1NF.....	116	Backtracking.....	30
2		Backus-Naur-Form.....	120
2NF.....	116	Basisoperationen	22
3		Baum	17
3NF.....	116	BDSG	102
A			
Ableiten	120	Befehlssatz.....	68
Absorptionsgesetz	61	Berechenbarkeit.....	36
Abstract Syntax Tree.....	128	Bias.....	105
ACID-Prinzip.....	117	Binäre Suche	33
Adjazenzliste	13	binärer Baum	18
Adjazenzmatrix	13	binary32	59
Aggregation	46	binary64	59
Aggregationsfunktion	118	Binärzahl	53
aktuelle Parameter.....	20	Blatt	17
Algorithmus.....	5	Blockchain	88
Allgemeinheit	5	Bluetooth	73
Analyse		BNF.....	120
lexikalisch	128	Boole'sche Logik	60
semantisch.....	129	branch.....	40
syntaktisch	128	Breitensuche	31
AND.....	62	Broadcast.....	74
Anomalie	116	Bubble-Sort	27
Antivalenz	60	C	
APFS.....	70	Cache-Speicher	68
Äquivalenz	60	Call-by-Reference	19
Artefakt.....	47	Call-by-Value	19
ASCII-Codierung.....	57	Certificate Authority	98
Assemblercode	38	ChatGPT	107
Assoziation.....	46	Chomsky-Hierarchie	122
Assoziativgesetz	61	CIDR	75
Ast.....	17	Classless Inter-Domain Routing	75
AST	128	commit.....	40
Asymmetrische Verfahren	96	Compiler	128
Attribut	42	compilierte Sprachen	38
Attributwert	42	csv-Datei	11
Aussage	60	D	
Aussagenlogische Gesetze.....	61	Datagrammprinzip	79
Authentizität.....	90	Data-Life-Cycle.....	103
		Datenlebenszyklus	103
		Datenschutz	101
		Datensicherheit	91
		Datensicherung	90
		differentiell	90

inkrementell	90
Datentypen	
elementar.....	6
strukturiert	6
DCL	117
DDL	117
De Morgansche Regeln.....	61
DEA	123
DEEDS	62
Determiniertheit	5
Determinismus.....	5
DHCP	75
Dijkstra-Verfahren.....	77
Disjunktion.....	60
Distributivgesetz.....	61
Divide and Conquer	26
DML.....	117
DNF	64
DNS.....	86
Domain Name System	86
Doppelnegation.....	61
double	59
DQL.....	117
DSGVO	101
Dualzahl	53
Dynamic Host Configuration Protocol	75

E

EBNF	120
Effizienz	22
Einerkomplement	54
Einwegfunktion.....	96, 98
elektromagnetisch.....	69
elektronisch	69
endlicher Automat	123
Endlichkeit	5
Entscheidungsproblem	36
erweiterter euklidischer Algorithmus.....	93
Evaluieren	11
ex situ	27
Execute.....	69
Exponent.....	59
ext4	70
Extremalgesetz.....	61

F

Falltürfunktion.....	96
FAT32.....	70
Feld	10
Festkommazahlen	59
Fetch	69
FIFO.....	10
Finitheit.....	5
FlipFlop.....	67

formale Parameter	20
formale Sprache	120
Fragmentierung.....	70
Full Join	119
Funktion	19
Fuzzy-Logik.....	60

G

Galoiskörper	95
Garbage Collector.....	7
gerichtet	12
get	43
gewichteter Graph.....	13
git	90
git	40
Glasfaserkabel.....	73
Gleitkommazahlen	59
Grammatik	120
Graph	12

H

Halbaddierer.....	66
Halteproblem	36
Harvard-Architektur	68
Hash-Funktion	34, 98
hashlib	99
Hexadezimalsystem	53
HTTP	87
HTTPS	87
Hypertext Transfer Protocol.....	87
Hypertext Transfer Protocol Secure	87

I

Idempotenzgesetz	61
IEEE 754	59
IMAP	82
immutable	9
Implikation	60
in situ	27
Index	117
Infrarot	73
Inner Join	119
Insertion-Sort.....	28
instabil	26
Integrität	90
Intermediate Representation	129
Internet Message Access Protocol.....	82
Internet Protocol	74
Interpreter	128
Inzidenzmatrix	14
IPv4.....	74
IPv6.....	74
IR	129

K

Kanban	49
Kapselung	43
Kasiski-Test	92
KDNF	64
Keccak	100
Kellerautomat	127
Kerckhoffs'sches Prinzip	96
KKNF	64
Klassen	42
Kleene-Logik	60
KNF	64
Koaxialkabel	73
Koinzidenzindex	92
Kommutativgesetz	61
Komplementärgesetz	61
Komplettsicherung	90
Komplexität	
exponentiell	22
konstant	22
kubisch	22
linear	22
logarithmisch	22
polynomial	22
quadratisch	22
superlinear	22
Komplexitätsklassen	22
Komposition	46
Konjunktion	60
Konstruktor	42
kontextfrei	122
kontextsensitiv	122
Kreis	13
kreisfrei	13
künstliches neuronales Netz	106
KV-Diagramm	63

L

LAN	71
Landau-Notation	22
Laufzeitkomplexität	22
empirisch	23
Lawineneffekt	98
Left Join	119
LIFO	10
Linear beschränkter Automat	126
Lineare Suche	32
linksregulär	122
Liste	10
LLM	107
LogicSim	62
Logikgatter	62

M

MAN	71
Mantisse	59
Maschinelles Lernen	106
Maschinencode	38
Mastertheorem	23
MD5	99
merge	40
mergen	29
Merge-Sort	29
Methode	19, 42
MM-Substitution	91
Mobilfunk	73
modulares Inverses	93
Moduloarithmetik	92
monoalphabetisch	91
monographisch	91
Moore'sches Gesetz	68
Multi-Core	69
Multiplexer	67
mutable	9

N

Nachrichtenvermittlung	79
NAND	62
NAS	70
natürliche Sprache	120
NEA	123
Negation	60
Netzwerkdienst	82
Netzwerkprotokoll	74
Neutralitätsgesetz	61
NOR	62
Normalform	116
NOT	62
NTFS	70

O

Objekt	42
Objektreferenz	7
One-Time-Pad	92
OOP	43
optisch	69
OR	62

P

Paketvermittlung	79
Paradigma	38
Parse Tree	128
Parsing	128
PDA	127
personenbezogene Daten	101

PGP	101
Pivotelement	29
planar	14
PM-Substitution	91
P-NP-Problem	37
polyalphabetisch	91
Polymorphie	45
POP3	82
Port	76
Post Office Protocol Version 3	82
Primitivwurzel	95
private	43
Produktregel	23
Programmierparadigma	
deklarativ	39
funktional	39
imperativ	38
logisch	39
Projektion	118
Prompt	107
protected	43
Prozedur	19
Prozess	69
public	43
Pushdown automaton	127

Q

Queue	10
Quick-Sort	29

R

RAID	90
Rechnernetz	71
rechtsregulär	122
Redundant Array of Independent Disks	90
Redundanz	116
Referenzparameter	19
Referenzzähler	7
regulär	122
Regulärer Ausdruck	121
reinforcement learning	106
Rekurrenzgleichung	23
Rekursionstiefe	24
rekursiv	24
repository	40
Repräsentation	11
return	19
Right Join	119
Routingtabelle	77
RSA-Verfahren	96

S

S/MIME	101
--------------	-----

SächsDSDG	102
SAT	76
Schaltnetzanalyse	63
Schaltnetzsynthese	65
Schlange	10
Schlüsselaustausch	95
schwache KI	105
Scrum	49
Secure Sockets Layer	98
Selection-Sort	28
Selektion	118
set	43
SHA	99
Simple Mail Transfer Protocol	85
single	59
Skriptsprachen	38
SMTP	85
Sockel	68
Socket	79
Softwareentwicklung	
agil	47
klassisch	47
Sortierschlüssel	26
Source Address Table	76
Speichermedien	69
SQL	117
Square-and-Multiply-Verfahren	94
SSL	98
stabil	26
Stack	10
staged area	40
Stakeholder	47
Stapel	10
starke KI	105
Subnetzmaske	74
Substitutionsverfahren	91
Suchverfahren	32
Summenregel	23
Superintelligenz	105
supervised learning	106

T

Taktfrequenz	68
TCL	117
TCP	79
Terminiertheit	5
Textdatei	11
Thread	69
Tiefensuche	31
TLS	98
Token	128
Tokenisierung	128
Topologie	71
Baum	72
Bus	71

logisch	71
physisch.....	71
Ring.....	72
Stern	71
vermascht	72
Transaktion	117
Transmission Control Protocol.....	79
Transport Layer Security.....	98
Traversierung.....	25
Inorder	25
Postorder.....	25
Preorder	25
Turingmaschine	126
Twisted-Pair-Kabel	73
Typinterpretation	8

U

Übertragungsmedien	73
UDP	80
UML.....	42
Unicode	57
unsupervised learning.....	106
Unterprogramm	19
User Datagram Protocol	80
UTF-8.....	57

V

Variable	
global.....	20
lokal.....	20
Variable Length Subnet Mask.....	88
Variablen	
veränderbar	9
Variablenkonzept	7
Variablentyp	6
Verbindlichkeit	90
Vereinbarung	44

Vererbung.....	45
Verfügbarkeit	90
Verschlüsselungsverfahren	
asymmetrisch	96
symmetrisch	91
Versionsverwaltung.....	40
Vertraulichkeit	90
Vigenère-Code.....	91
Virtual Local Area Network.....	88
VLAN.....	88
VLSM.....	88
V-Modell.....	48
Volladdierer	66
Von-Neumann-Architektur.....	68
Von-Neumann-Flaschenhals	68
Vorgabe	21, 43

W

WAN.....	71
Wasserfallmodell	47
Weiterleitungstabelle	77
Werteparameter.....	19
WLAN	73
Wurzel	17

X

XNOR	62
XOR.....	62

Z

Zeiger	7
Zeitmessung	23
Zertifikat	98
zusammenhängend	12
Zweierkomplement	55
Zwischenrepräsentation	129