Zusammenfassung IT

Ich hab hier mal versucht, die Vorlesung IT auf 25 Seiten zusammenzufassen. Ich hoffe es hilft jemandem ;)
Johannes

01_Einleitung_und_Motivation.pdf

Informationstechnik: die technische Umsetzung von Verarbeitung, Übermittlung, Speicherung, Sicherung und Anwendung von Information . IT ist der Oberbegriff für die Informations- und Datenverarbeitung, sowie für die dafür benötigte Hardware und Software

Information: repräsentiert durch Nachrichten in Form von Sprache, Ton, Bild, Text und Daten

Technik: bezeichnet eine Methode, die eingesetzt wird, um ein bestimmtes Ergebnis zu erreichen.
Technologie: bezeichnet das Wissen um diese Technik („Wie funktioniert sie? Welche Erfahrungen mit ihr gibt es, welche Risiken und welche Möglichkeiten?“)

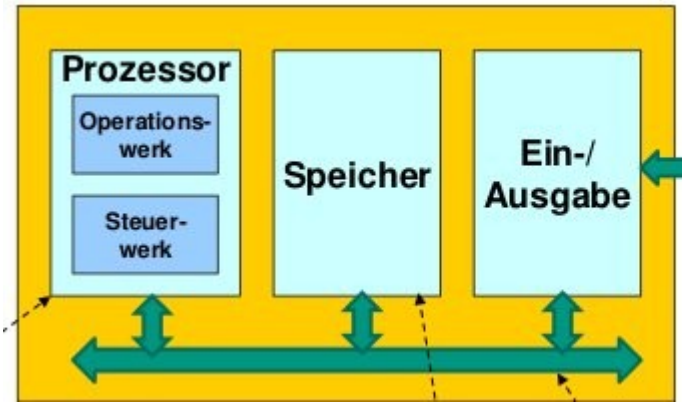
Informatik ist die Wissenschaft, die sich mit der Struktur, der Wirkungsweise, den Konstruktionsprinzipien und den Einsatzmöglichkeiten informationsverarbeitender Systeme sowie ihren Anwendungen beschäftigt .

Informationstechnisches System (abgekürzt IT-System) : jegliche Art elektronischer datenverarbeitender Systeme. Beispiele: Computer, Datenbanksysteme, Informationssysteme, Digitale Messsysteme, Microcontrollersysteme, Kompaktregler, eingebettete Systeme, Mobiltelefone...

Informationssysteme: soziotechnische („Mensch-Maschine“-) Systeme, die menschliche und maschinelle Komponenten (Teilsysteme) umfassen und zum Ziel der optimalen Bereitstellung von Information und Kommunikation nach wirtschaftlichen Kriterien eingesetzt werden . Ein Informationssystem dient der rechnergestützten Erfassung, Speicherung, Verarbeitung, Pflege, Analyse, Benutzung, Verbreitung, Disposition, Übertragung und Anzeige von Information bzw. Daten. Es besteht aus Hardware (Rechner oder Rechnerverbund), Datenbank(en), Software, Daten und all deren Anwendungen .

02_Programmiersprachen.pdf

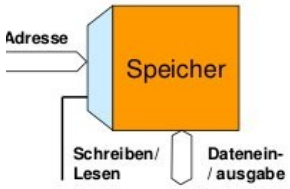
externe Architektur: Maschinenbefehlssatz
interne Architektur: der innere Hardwareaufbau eines Rechners
Universalprozessoren: für alle Anwendungen vom einfachen C-Programm bis zum Compiler, Betriebssystem, Datenbank, Textverarbeitung



Prozessor (Central Prozessing Unit) führt Maschinenprogramm aus.
Arbeitsspeicher: speichert Maschinen-code und zugehörige Daten
Systembus: bidirektional, überträgt Daten, Befehle, Steuersignale, Busbelegung nach Vorrang der Geräte

Speicherwerk:

- Register im Rechnerkern zur Speicherung von Operanden, Bedingungsvektor, Ergebnis
- wahlfrei adressierbarer Schreib/Lese-Speicher (RAM, RWM)
- Die Breite des Adressvektors = Γ log2 (Anzahl Speicherworte)



Speicherhierarchie: Register, Cache, Hauptspeicher Plattenspeicher zunehmende Zugriffszeit, Speicherkapazität →

Speicherorganisation:
Üblich ist die Organisation in Bytes: z.B. 32bit Adresse → 2^{32} Bytes adressierbar . Der Speicher wird in mehrere Teilspeicher unterteilt, ein teil des Adressvektors wählt den Bereich, der andere die Speicherzelle im Bereich. Auch E/A Geräte können wie ein Speicher behandelt werden („memory mapped I/O“) .
Ausrichtung (alignment) : Daten mit einer festen Länge können nur beginnend mit einer festen Adresse im Speicher untergebracht werden. Besteht ein Befehl aus 4 Bytes, so muss die Byte-Adresse um 4 erhöht werden, um den nächsten Befehl zu holen .

Prinzip der Lokalität: benutzte Daten/Befehle werden bald wieder benutzt , auf Daten/ Befehle, die im Adressraum nahe zu gerade benutzten liegen, wird wahrscheinlicher verwiesen als auf weiter weg liegende . Daher werden räumlich nahe beieinander liegende und oft referenzierende Teile (Blöcke), nahe am Prozessor gehalten.

Cacheorganisation sprinzip: nur die tatsächlich gebrauchten Blöcke werden im Cache gehalten .

Fragen der Cacheorganisation:

Wo kann ein Block im Cache platziert werden?

1. Verfahren: voll-assoziativ . Jeder Block kann an allen Adressen gespeichert werden .
2. Verfahren: direkt-abbildend . Ein Block kann nur an einer Adresse mod n (n Cache-Größe) gespeichert werden.
3. Verfahren: n-Wege assoziativ . Ein Block kann an jeder der n Cache-Adressen gespeichert werden, bei denen die Hauptspeicheradresse mod n gleich der Cache-Adresse mod n ist Physikalischer Aufbau von m-unabhängigen Caches

Wie kann festgestellt werden, ob ein Block im Cache ist?
Valid-Bit, Tag etc.

Welcher Block soll ersetzt werden, falls ein Block gebraucht wird?

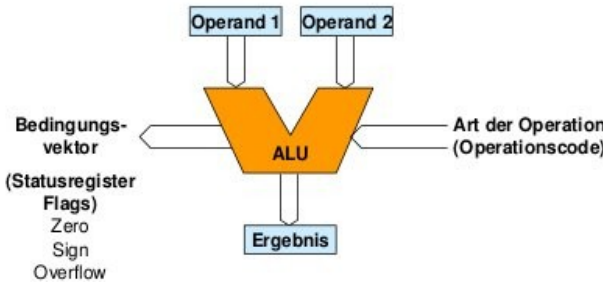
direkt-abbildend: klar, n-Wege assoziativ: LRU (least recently used) , Zufallsgesteuert

Was geschieht beim Schreiben?
Daten in Cache und Hauptspeicher korrespondieren einander nur solange der Prozessor nicht die Daten im Cache überschreibt . 2 Lössungsstrategien: Schreiben in Cache und Speicher (write through) , Schreiben nur in Cache, zurückschreiben in Speicher nur bei Austausch des Blocks (write-back).

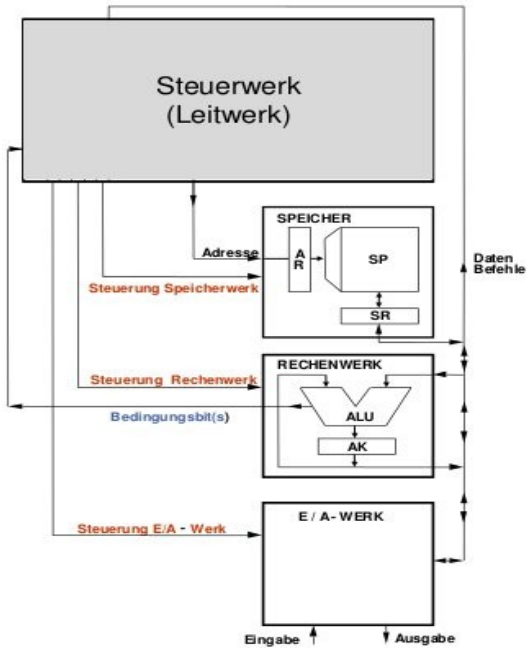
Ein/Ausgabewerk:
Kommunikation mit dem Benutzer via Ein-/Ausgabegeräte (Tastatur, Maus, Bildschirm) und mit den Peripheriegeräten (z.B. Drucker, Plattenspeicher) .
Eingabe: Übernahme der Daten von Gerät, Umsetzung zur Weiterverarbeitung in geeignete Norm
Ausgabe: Datenformatmäßige und elektrische Aufbereitung der Daten , Weiterleitung an ausgewähltes externes Gerät

ALU (Arithmetisch/Logische Einheit)

Verarbeitung der Daten mit arithmetische und logische Basisfunktionen (Addition, 2er-Komplement, NOT, AND, OR), Rückgabe: Flags



Schema eines einfachen Rechners:



Befehle:
Transport-, Verarbeitungs-, Sprung-, Systembefehle (Interrupts, Speicherverwaltung)

Abkürzungen:
SP: Speicher , AR: Adressregister, SR: Speicherregister, ALU: Arithmetik / Logik-Einheit, AK: Akkumulator , E/A: Ein-/Ausgabe-Werk

Verarbeitungsphasen bei der Befehlsausführung
Befehl holen , Befehl dekodieren , Operanden holen, Operation ausführen, Ergebnis abspeichern, Programmzähler erhöhen für nächsten Befehl

Von-Neumann-Bottleneck : Speicherzugriff 10-100 mal langsamer als Befehlsausführung.

Beschleunigung der Befehlsausführung:
Parallellisierung von Befehls- und Datenzugriffen im Hauptspeicher (Harvard Architektur), Überlappende Befehlsausführung (Pipelining), Mehrere Rechenwerke (Superskalar), Beschleunigung Hauptspeicherzugriff (Multi Level Caches.

Operationen mit Daten : Datentransfer , Arithmetisch, Floating Point, Logisch,Bedingte und unbedingte Verzweigung , Shift (logisch/arithmetisch), Rotate, Bit Manipulation, Multimedia, (Systemsteuerung, Coprozessorbefehle)

Befehlsformate: 3-Adress (Load/Store Machines), 2-Adress (Register-Memory Machines) , 1-Address (Accumulator Machines), 0-Address (Stack Machines)

Klassifizierung: Stack-, Akkumulator- , Registersatzmaschine (CISC, RISC)

Stack-Maschine: Befehle beziehen sich auf einen Stack zur Speicherung u. a. von Zwischenergebnissen . Der Prozessor hat Zugriff auf einen Stack und verwaltet die Adresse des obersten Elements auf dem Stack (**t**op **o**f **s**tack). Typische Befehle: PUSH m (lege den Wert unter der Adresse m auf den Stack), ADD (addiere die beiden obersten Werte des Stacks, entferne sie und lege die Summe als oberstes Elemente auf den Stack), STORE m (speichere das oberste Element auf dem Stack nach der Adresse m)

Akkumulatormaschine : Ergebnisse werden im Akkumulatorregister AK gespeichert. Der Prozessor beinhaltet AC, der Speicher die Register und einen Adress-MUX. Typische Befehle: LAC m (lade Wert unter Adresse m in den AC), SAC m (speichere den Inhalt von AC nach Adresse m), ADD m: (addiere den Inhalt von AC mit dem Wert unter Adresse m und speichere das Resultat nach AC)

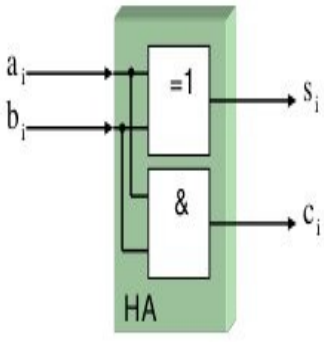
Registersatzmaschine : Der Prozessor enthält z.B. 16 oder 32 Universalregister . Im Prozessor befindet sich weder Akkumulator noch Stack , zur Berechnung befüllt der Prozessor seinen internen Registersatz mit Werten aus dem Speicher

Schaltnetz: ist eine Digitalschaltung, in der es für jede mögliche Kombination von digitalen Signalen an den Eingängen eine – und nur eine – Kombination von digitalen Signalen an den Ausgängen gibt .

Vorgehensweise beim Entwurf von Schaltnetzen:

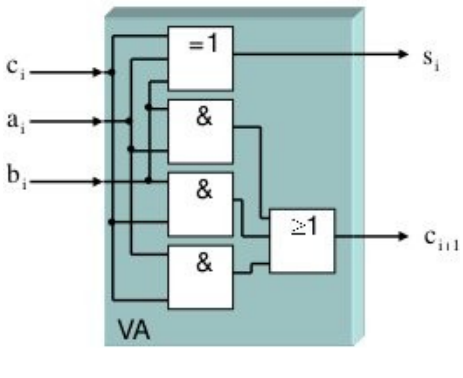
1. Umsetzung der verbalen Aufgabenstellung in eine formale Form, z.B. in eine Funktionstabelle; (dazu gehört die Festlegung der Wertezuordnung für die Binärvariablen)
2. Bildung einer Normalform; Bildung einer vollständigen Blocküberdeckung
3. Bildung einer Minimalform (bspw. nach S-Diagramm oder Nelson/Petrick)
4. Umformung in das gewählte Basissystem
5. Umformen in den Strukturausdruck
6. Umsetzen in das entsprechende Schaltnetz

Halbaddierer



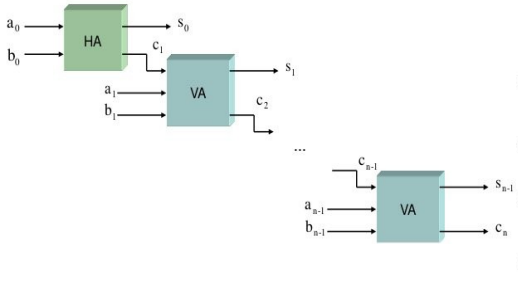
a_i	b_i	s_i	c_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Volladdierer

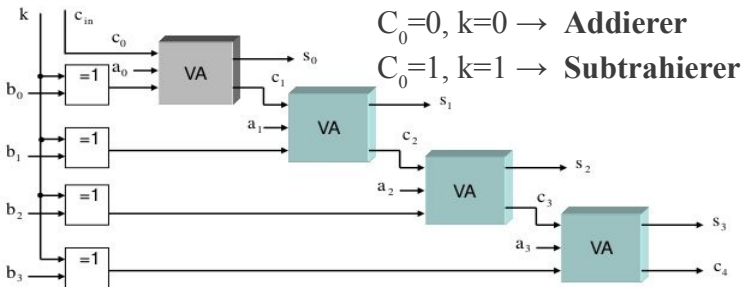


a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Ripple-Carry-Addierer:



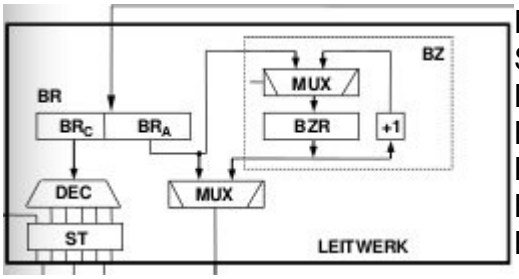
Addierer/Subtrahierer:



C_0=0, k=0 → **Addierer**

C_0=1, k=1 → **Subtrahierer**

Leitwerk:



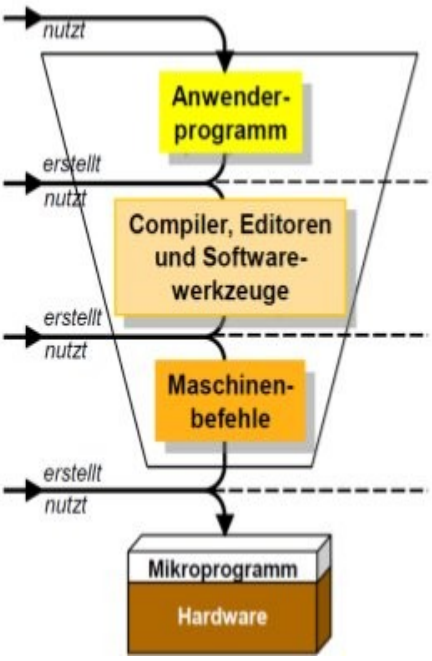
DEC: Befehlsdecoder
ST: Befehlszähler
MUX: Multiplexer
BZR: Befehlszählerregister
BR: Befehlsregister
BR_A: Adressteil von BR
BR_C: Codeteil von BR

Programmiersprachen:
bieten zur effizienteren Entwicklung von Software die Möglichkeit, den Befehlssatz durch mnemo s (Assembler) zu abstrahieren. Hochsprachen sind eine weitere Abstraktion. Auf diese Weise wird der Code menschenverständlicher und dadurch weniger fehleranfällig. Programmiersprachen sind also Notationen für Computerprogramme. Sie müssen für eine maschinelle Analyse geeignet sein und basieren daher auf formalen Sprachen. Sie dienen dazu, Anweisungen an einen Computer zu übermitteln

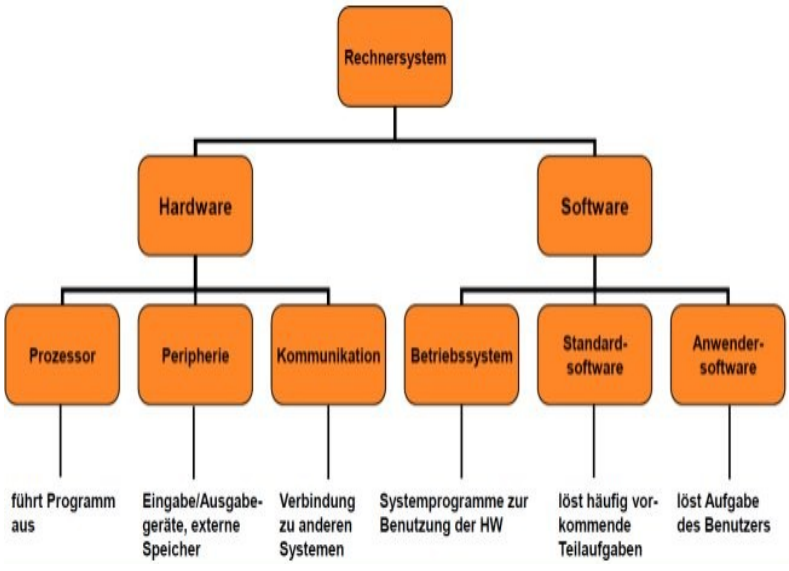
ASSP: Applikationsspezifischen Standardprozessoren → speziellen Befehlssatz ↔
Standard Mikroprozessor → standardisierter Befehlssatz

Programm: ist ein in einer Programmiersprache formulierter Alogirthmus, also eine Abfolge elementarer Schritte. Die Abfolge wird durch Kontrollstrukturen wie Konkatention, Sequenzen, Alternativen (if,else,switch), Iterationen (while,for) oder Unterprogrammaufrufe gesteuert. Komplexere Anweisungen werden aus elementaren zusammengebaut, Daten werden in Variablen gespeichert die einen Datentyp haben. Elementare Datentypen sind Integer, Real, Boolean, Char, String, Array...)

Sichtweise auf Programme:



Rechnersysteme:



Klassifikation von Betriebssystemen:
nach Betriebsart: Stapelverarbeitung, Dialogbetrieb (WIN LINUX), Echtzeit, Verteilte Verarbeitung
nach Anzahl der Benuntzer: single user (MS-DOS) / multi user (UNIX)
nach Aufträgen: single tasking (MS DOS) / multi tasking (Linux, Windows)

Typische Funktionen eines Betriebssystem:
Benutzer-, Auftrag-, Prozess-, Datei-, Betriebsmittel-,Hauptspeicherverwaltung, Scheduling, I/O-Steuerung, Kommunikation mit Umgebung

Systemsoftware: alle Programme, die die effiziente und komfortable Nutzung ermöglichen.
Programmierumgebung: Editor, Compiler, Linker, Lader, Testhilfen
Dienstprogramme: Programme zum suchen, kopieren, sortieren, installieren, konfigurieren, administrieren

Formale Sprache: Menge von Zeichenketten aus einem Alphabeth (Zeichenvorrat) Formale Sprachen sind die Grundlage höherer Programmiersprachen und die Voraussetzung für den von Compilern. Sie folgt einer strikten, eindeutigen Grammatik, aus jeder Anweisung kann eindeutig ein Maschinencode gebildet werden.

Syntax: beschreibt die Menge der erlaubten Zeichenketten für Programme
Semantik : definiert die Bedeutung der einzelnen Sprachkonstrukte
Grammatik : legt die Produktionsregeln, also den Zusammenhang zwischen den Zeichen und den Wörtern, fest.
Formale Beschreibungsmittel: Backus-Naur-Form, Syntaxdiagramme

Backus-Naur-Form:
Auflösen des Textes durch schrittweise Analyse, Wahl des Zutreffenden bei Alternativen. Ein Ausdruck ist dabei entweder gültig, wenn er sich auflösen lässt, oder ungültig.
Beispiel: Zahl:= '-' positiveZahl | positive Zahl; positiveZahl := Ziffernfolge | Ziffernfolge '.' Ziffernfolge; Ziffernfolge := Ziffer | Ziffer Ziffernfolge; Ziffer = '0' | '1' |'2' | '3' |'4' | '5' |'6' | '7' |'8' | '9' |
3.14 → Ziffer.14 → Ziffernfolge.14 → Ziffernfolge.Ziffer4 → Ziffernfolge.ZifferZiffer → Ziffernfolge.ZifferZiffernfolge → Ziffernfolge.Ziffernfolge → positiveZahl → Zahl => gültiger Ausdruck!

Syntax Diagramm: Grafische Beschreibung der Syntax, grafische Repräsentation für ein
Datenwörterbuch (Data Directory): ist ein Katalog von Metadaten, Definitionen und Darstellungsregeln für Datenelemente. Es beschreibt Beziehungen zwischen Datenobjekten, mit dem Ziel, diese redundanzfrei und zentral zu definieren.

Höhere Programmiersprachen: sind von der Hardware abstrahiert, sodass einzelne Befehle oft dutzende Maschinenbefehle enthalten und so leistungsfähiger sind. Die Effizienz des Programmierens wird gesteigert. Ein Compiler übersetzt das Programm in Maschinencode. Durch die Abstraktion von der Hardware braucht der Programmierer den konkreten Zustand der Hardware nicht mehr zu kennen, daher kann er sein Programm einfach in Unterprogramme zerlegen und bei bedarf auf einen anderen Rechner portieren indem er neu compiliert. Dies ist bei Java im Gegensatz zu C++ nicht mal nötig: Der Java-Compiler erzeugt sogenannten Bytecode, der vom Interpreter, der Virtual Maschine während der Laufzeit in Maschinencode übersetzt wird. Höhere Programmiersprachen sind durch Bibliotheken erweiterbar.

Paradigma: übergeordnetes Prinzip oder Denkmuster
Programmierparadigma: Übergeordnetes Prinzip nachdem ein Computerprogramm zur Lösung eines Problems erstellt wird. Programmierparadigmen sind: **imperative** Programmierung (strukturiert, prozedural, modular), **deklarative** Programmierung (funktional, logisch), oder **objektorientierte** Programmierung.

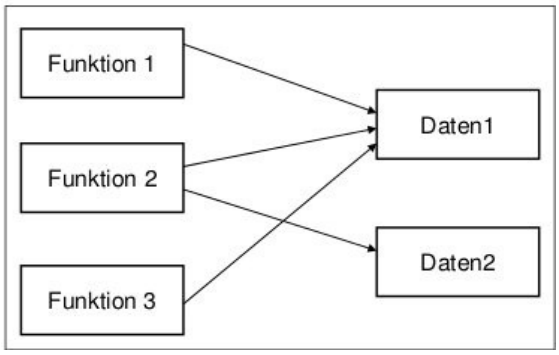
Imperative Programmierung: definiere **WIE** ein Problem gelöst werden soll. Variablen sind im Gegensatz zur Mathematik Speicher für Zwischenwerte. Bei der modularen

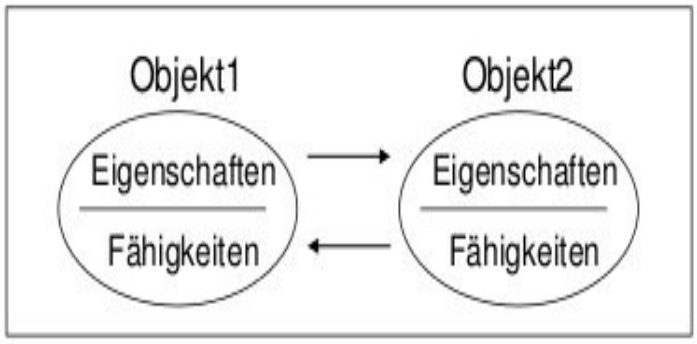
Entwicklung wird das Problem in Module zerteilt. Dadurch verringert sich die Komplexität der einzelnen Module. Die Module werden am Ende logisch zusammengeführt.

Deklarative Programmierung: definiert, **WAS** gelöst werden soll. Funktionale Sprachen (LISP), logische Sprachen (PROLOG) Funktional-logische Sprachen (Babel) und Datenflusssprachen (VAL, Linda) sind deklarativ.

Eigenschaften von Objektorientierung: Abstraktion, Kapselung, Vererbung, Polymorphie

Pozedural vs Objektorientiert





Programmiersprachen für spezielle Einsatzgebiete:
Auszeichnungssprachen (HTML), Skriptsprachen (JavaScript), Datenbanksprachen (SQL), Hardwarebeschreibungssprachen (VHDL, SystemC), Grafische Sprachen (Matlab, Labview)

Funktion in der imperativen Programmierung: Unterprogramm. Ein Mittel zur Modularisierung (Zusammenfassung oft benötigter Codes). Ein Funktionsaufruf erzeugt ein neuen Bereich auf dem Stack (Ablage von Parameter, lokalen Variablen der Funktion, Rückgabewert)

Funktion in der deklarativen Programmierung: mathematische Abbildung

Unterprogramm (subroutine, Prozedur, Funktion, Methode, Operation):
Teil des Programms, das aus dem Programm aufgerufen werden kann und zu diesem zurückkehrt.

03_softwareentwicklung und Qualität.pdf

Software Engineering : Technische Disziplin, die sich mit allen Aspekten der Softwareherstellung beschäftigt: Erfassen des Bedarfs, Systemspezifikation, Analyse, Planung Softwarearchitektur, Design, Implementierung , Test, Dokumentation, Installation beim Kunden, Wartung des Systems, …

Wesentliche Merkmale guter Software:
Zuverlässigkeit: Verlässlichkeit, Zugriffsschutz, Betriebssicherheit (keine körperl, seel. Ök Schäden)
Wartungsfreundlichkeit: SW-Veeränderung, Weiterentwicklung, Diagnostizierbarkeit
Effizienz: Sparsamer Ressourcenverbrauch (Speicher, Prozessorlast), Kosteneffizienz
Benutzerfreundlichkeit: GUI, Dokumentation, ohne übermäßige Anstrengung benutzbar
Nützlichkeit: Erfüllung der Benutzeranforderungen

Software-Entwicklungsprozess:

Anforderungen→	Analyse →	Design →	Umsetzung →	Test/QA

Mit Kunden: Lastenheft (Bedarf), Pflichtenheft, Zielvereinbarung (Vertragsgrundlage) => Einschätzung des Projekts	Warum von uns? Probleme bei Umsetzung? Benötigte Ressourcen? => Beschreibung des Problems	Gliederung des Problems, Definition Schnittstellen, Modularisierung, Einsatz von Bibliotheken, Zukauf von SW. Passt das Design zum Problem? => Architektur	Programmierung nach Design, Verifikation: Design eingehalten? => Code	Bugs fixen Notwendig für Stabilität Erheblicher Zeit-aufwand (oft 50-70% der Entwicklung) => Produkt

Verifikation: Baue ich das Produkt richtig? ↔ **Validierung**: Baue ich das richtige Produkt?

Software-Testprozess:

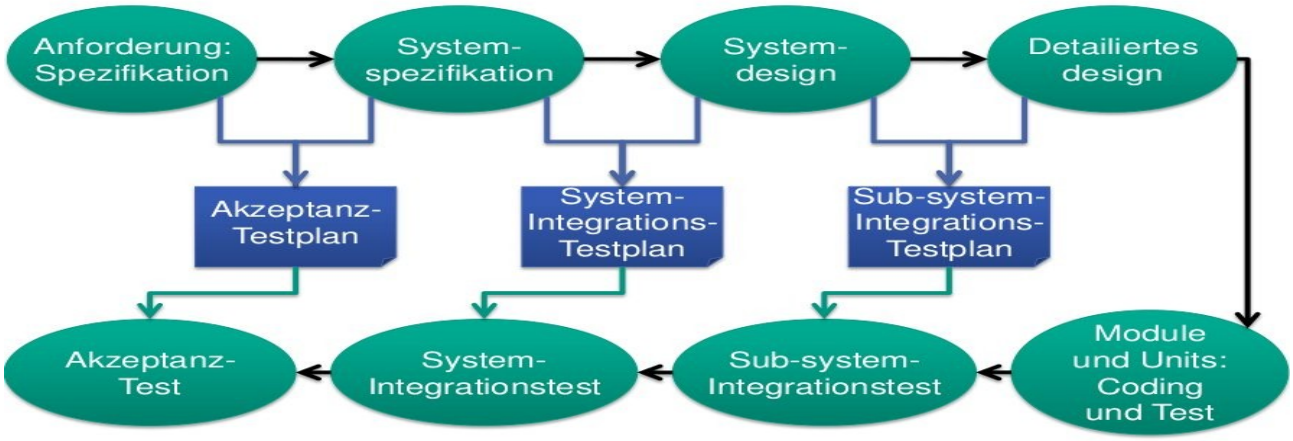
Unit Testing ↔ Module Testing ↔ Subsystem testing ↔ Sytem testing ↔ Akzeptanztest
Component testing Integrationstest user testing

Unit Testing: Test einzelner Komponenten (z.B. Funktionen)
Module Testing: Test des Komponenten-Zusammenspiel
Subsystem Testing: Überprüfen der Stimmigkeit von Schnittstellen
System Testing: Gesamtsystem wird getestet.

Regressionstest: automatisierter Test, sodass der Entwickler mit nur einem Aufruf überprüfen kann ob die neue Programmversion funktiniert.
White Box Test: Mit dem Wissen des Entwicklers, wie das System intern funktioniert, werden alle hypotetischen Eingabe-Möglichkeiten insbes. Grenzfälle getestet.

Akzeptanztests: ALPHA-Test: Test mit Daten aus dem reellen Einsatzgebiet, BETA-Test: Lieferung an potentielle Kunden (friendly user): selbst absurde Eingabe-Kombinationen werden getestet.

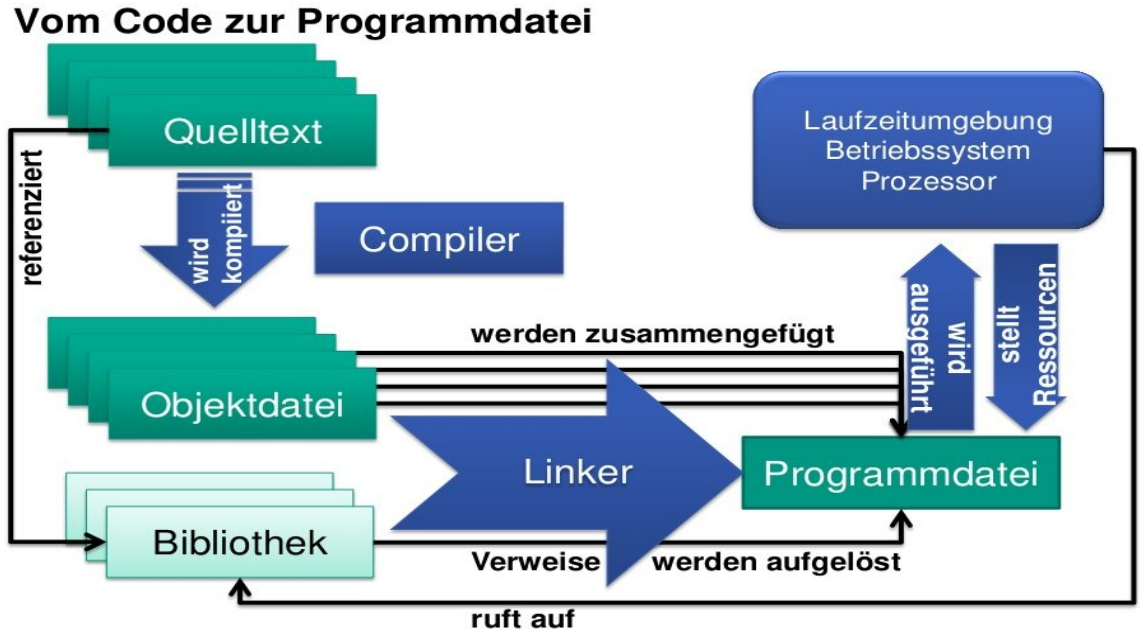
Softwaretest-Planung



Qualitätssicherung von Software:

04_Entwicklungsumgebung.pdf

IDE Integrated Development Environment: Sammlung aller Werkzeuge, die dem Entwickler helfen, seine Programme zu erstellen . Moderne IDES helfen dem Entwickler in allen Phasen der Software-Entwicklung.
Minimale IDE: Programmcodeeditor , Compiler, Bibliotheken, Linker, Laufzeitumgebung
Programmcodeeditor, Compiler: KLAR!
Bibliotheken: Stellen dem Programmierer Funktionalitäten bereit, die er nicht selbst entwickeln muss.
Linker: Verbindet die einzelnen Objektdateien des Programmierers mit den Bibliotheksfunktionen, so dass die Verweise und Sprungbefehle in der resultierenden ausführbaren Datei korrekt sind.
Laufzeitumgebung: Stellt dem Programm benötigte Ressourcen bereit. Zu ihr gehören Betriebssystem ebenso wie weitere Dienste.
Debugger: hilft dem Programmierer beim Entfernen von Fehlern (Bugs). Er führt den mit Debuginformationen angereicherten Maschinencode Schritt für Schritt aus und lässt den Programmierer mitverfolgen, wie sich der Zustand seines Programms Programmzeile für programmzeile ändert. So lassen sich viele Laufzeit-Fehler finden.



Unit Testing-Tools: Erkennen von relevanten Testeingaben und Test der programmierten Einheit zur Verifikation.
Speicherüberwacher: decken Memory-Leaks auf.
code coverage analyzer : ermittelt in Testläufen, welche Codeabschnitte am häufigsten aufgerufen werden und wo somit eine algorithmische Optimierung wahrscheinlich die besten Ergebnisse liefert.

Coding Guidlines Programmierrichtlinien **:** Kein technischer Zwang, führen aber zur verbesserten Lesbarkeit/Wartbarkeit, sinnvolle Namensgebung führt zu Wiederverwendbarkeit, Struktur zu weniger Fehlern.

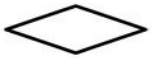
05_Darstellung_von_Programmstrukturen.pdf

Programmablaufplan DIN 66001 :

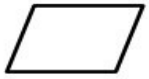
Start/Ende : Markiert Start bzw. Ende des Programmablaufs. Wird üblicherweise mit Start/Ende beschriftet.

Prozess : Ein Kasten kann entweder einen kleinen Schritt oder einen ganzen Unterprozess bezeichnen.

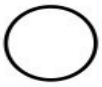
Dokument : Ein gedrucktes Dokument, ein Bericht etc.



Entscheidung : Linien von den Ecken kennzeichnen die verschiedenen möglichen Programmzweige.



Ein-/Ausgabe : Kennzeichnet Material oder Informationen, die in das System kommen oder es verlassen, etwa eine Bestellung (Eingabe) oder ein Produkt (Ausgabe).



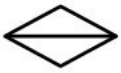
Verbindungspunkt : Weist darauf hin, dass der Ablauf da weitergeht, wo ein anderes, mit dem selben Buchstaben gekennzeichnetes Symbol steht.

Flusslinie : Linien bestimmen den sequentiellen Ablauf der Ablaufschritte.

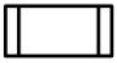
Verzögerung : Weist auf Wartebedingungen oder -zeiten im Ablauf hin.

Vereinigung : Kennzeichnet einen Schritt, wo zwei oder mehrere Unterprozesse oder Teillisten zusammengeführt werden.

Zusammenstellen: Kennzeichnet das Ordnen von Informationen in ein spezifisches Format

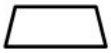


Sortieren : Kennzeichnet das Ordnen einer Liste anhand bestimmer Kriterien (Größe, Umfang etc)

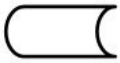


Subroutine/Algorithmus : Kennzeichnet einen komplexeren Ablauf, der in das aktuelle Flussdiagramm eingebettet ist. Dieser Ablauf könnte in einem anderen Diagramm spezifiziert sein.

Manueller Eingriff : Kennzeichnet eine Reihe von Anweisungen, die manuell ausgeführt werden müssen.



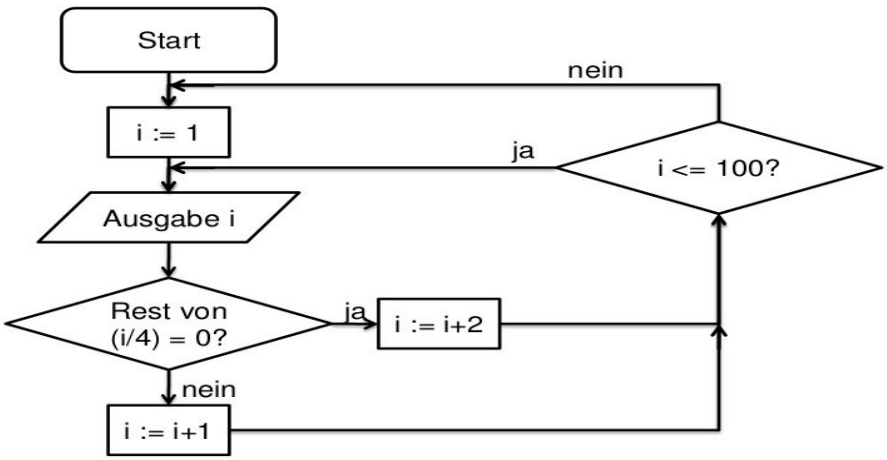
Schleifenende : Kennzeichnet den Punkt, an dem eine Schleife abbrechen soll.



Datenspeicher : Kennzeichnet einen Speicher von Daten.

Datenbank (urspr. Magnetplattenspeicher) : Kennzeichnet eine Sammlung von Informationen, die durchsucht und geordnet ausgegeben werden können.

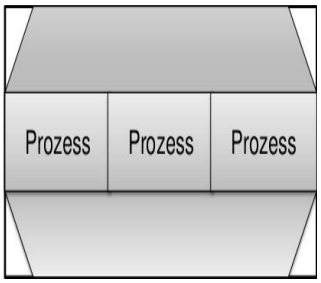
Beispiel für ein PAP:



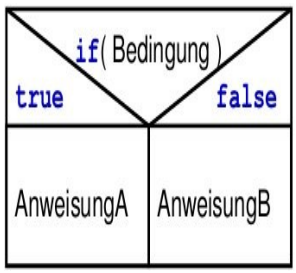
Nassi-Shneiderman Diagramm DIN 66261 :
wurde 1973 von Nassi und Shneiderman definiert. Der Fokus liegt darauf, Blockbildung durch Strukturen hervorzuheben.

Sequenz:

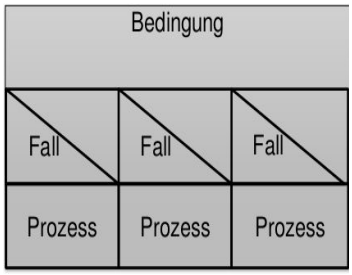
Parallele Prozesse:



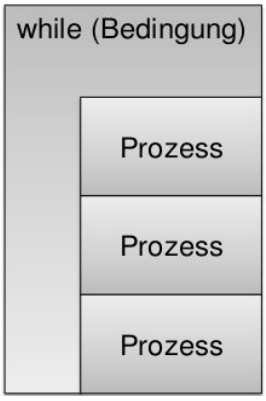
if



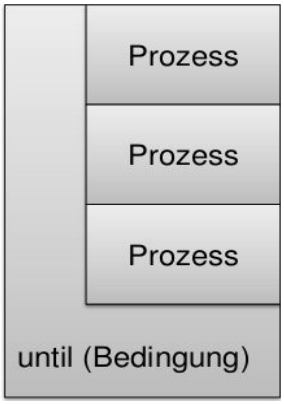
switch:



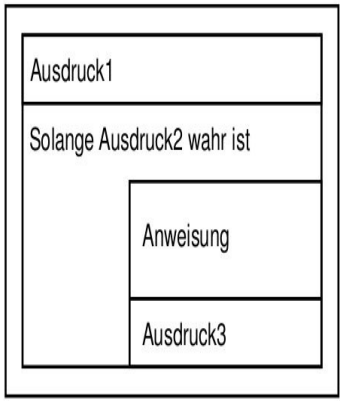
while-Schleife:



until-Schleife:



for-Schleife:



break-Anweisung:

Vorteile von Nassi-Shneiderman:
repräsentiert gut das strukturierte Programmierparadigma, Planung mittels N-S verhindert Sprungbefehle , Blöcke fassen Schleifen eindeutig zusammen
Nachteile von Nassi-Shneiderman:
von Hand schwer zu zeichnen , alle konkreten Aufgaben sind nur Prozesse, Blockform führt oft zum Eindruck der Überfülltheit

Vorteile PAP:
einfach zu zeichnen , Symbole für viele konkrete Einsatzzwecke, sehr leicht nachvollziehbarer Kontrollfluss
Nachteile PAP:
Impliziert Programme, die mit Sprungbefehlen arbeiten , Verschachtelte Schleifen sind nicht erkenntlich.

06_Objektorientierung.pdf

Grundidee der Objektorientierung:
Menschliche Auffassung betrachtet Realität meist als Verbund von Objekten (z.B. Menschen, Unternehmen, Fahrzeuge, Formulare, Geschäftsabläufe...)
Entsprechend sollen zumeist reelle Objekte oder Konzepte im Computer abgebildet werden. Diese Objekte haben nicht nur Eigenschaften (Daten) sondern auch Fähigkeiten, die eigenen Eigenschaften zu ändern oder mit anderen Objekten zu interagieren (Methoden).

Prinzipien der Objektorientierung:

- Daten und die zu diesen gehörenden Funktionen möglichst eng in einem sog. „Objekt“ verbinden

- **Kapselung:** Objekt definiert Zugriffsmöglichkeit auf seine Daten und schützt diese so (private), von Außen kann nur auf Schnittstellen (public) zugegriffen werden.
- **Wiederverwendbarkeit**: Nur einmal definieren, wie ein Objekt funktioniert, und dieses immer wieder, auch für neue Typen von Objekten, verwenden

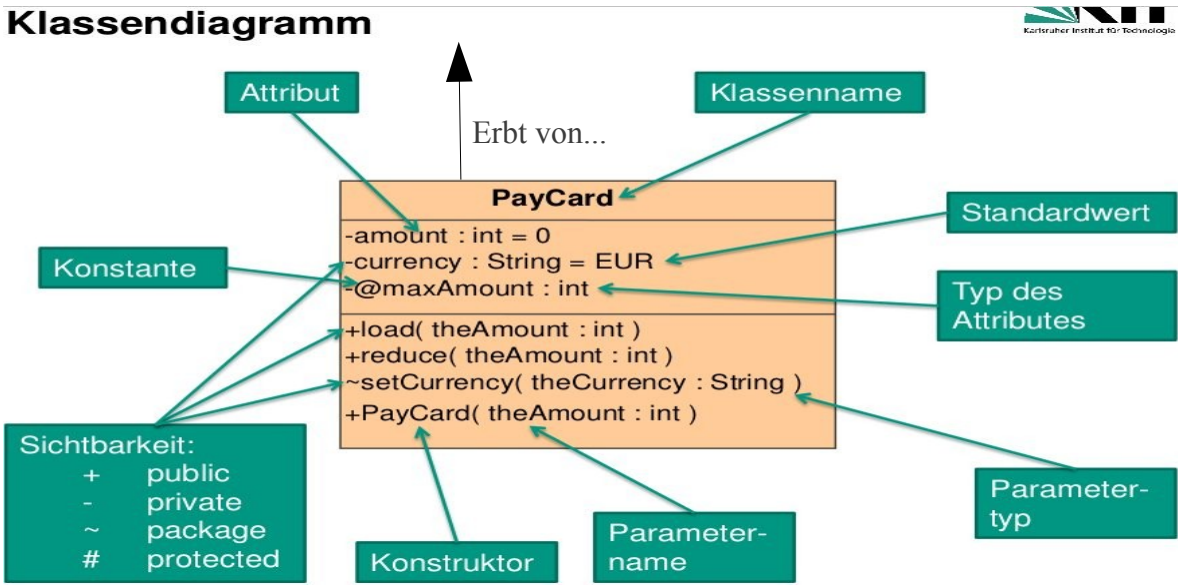
Vorteile der OOP: geringere Fehleranfälligkeit , gute Wiederverwendbarkeit, geringer Wartungsaufwand

Klasse: Bauplan für Objekte des Typs: Daten heißen Eigenschaften bzw. Attribute, Fähigkeiten heißen Mehtoden. Attribute und Methoden haben wiederum Eigenschaften, z.B. Sichtbarkeit, Typ, Wertebereich, Unveränderlichkeit (const) bzw. Sichtbarkeit, Rückgabetyp, Parameter

Instanziierung: Erzeugen eines Objekts nach bzw. aus einer Klasse. Verschiedene Instanzen unterscheiden sich in ihrem Zustand.

Datenabstraktion: Reduktion eines komplexen Zusammenhangs auf das Wesentliche.

Objekt: Eine zur Laufzeit des Programms existierende Datenstruktur, die aus 0 oder mehr Werten besteht, welche Attribute genannt werden. Sie dient als computerinterne Darstellung eines abstrakten Objekts . Objekte sind Instanzen von Klassen. Der Objektzustand wird durch den Wert der Daten bestimmt.



Entitäten:

Polimorphie:

Symbol	Bedeutung
1	Exakt ein mal
0..1	Kein oder ein mal
* (or 0..*)	Kein mal oder häufiger
1..*	Ein mal oder häufiger
3..*	Drei mal oder häufiger
4..6	Vier bis sechs mal
2, 4, 6	Zwei, vier oder sechs mal
1..5, 8, 10..*	Ein bis fünf mal, acht mal oder zehn mal oder häufiger

Methoden, die das gleiche tun haben den selben Namen. Die Methode eines Objekts wird aufgerufen, ohne die genaue Klasse des Objekts zu kennen. Beim sogenannten „**late binding**“ entscheidet sich erst zur Laufzeit, aus welcher Klasse die Methode aufgerufen wird. In C++ ist dazu das Schlüsselwort „**virtual**“ nötig.

07_Datenstrukturen.pdf

Datenstruktur: ist eine Methode, um Daten strukturiert abzuspeichern und zu organisieren sowie den Zugriff auf die Daten und die Modifikation der Daten zu erleichtern. In der Informatik ist eine Datenstruktur ein mathematisches Objekt zur Speicherung von Daten. Nach moderner Auffassung sind Datenstrukturen ein Bestandteil von Algorithmen. In C++ sind Datenstrukturen üblicherweise als Klassen definiert.

Standard-Operationen auf Datenstrukturen: sind zum Beispiel einfügen, löschen, suchen, Anzahl an Elementen, den Vorgänger oder Nachfolger ermitteln. Verschieden Datenstrukturen sind für verschiedene Standard-Operationen verschieden effizient. Dies bietet die Möglichkeit, die Datenstrukturen zu klassifizieren.

Standard-Tamplate-Library (STL): gängige Klassenbibliothek für C++, die die Implementierungen gängiger Datenstrukturen, Klassen zur Ein- und Ausgabe, zum Beispiel für Dateien sowie gängige Algorithmen zum Sortieren oder ähnlichem enthält. Die STL bietet Datentypunabhängige Container (Tamplates) und Iteratoren an.
Iteratoren: Klassen, die wie erweiterte Zeiger Arbeiten.

Wichtige Datenstrukturen: Array, List, Stack, Queue, Hash-Table, Graph, Baum, Heap

Datenstruktur	Beschreibung	Vorteile	Nachteile
Array (Feld)	Einfachste Datenstruktur: speichert mehrere Variablen des selben Typs	Extrem schneller, wahlfreier Zugriff über Index	Keine dynamische Größenanpassung
List (Liste)	Verkettung der Elemente unter einander: Elemente enthalten Verweis auf Vorgänger und Nachfolger (doppelt verkettet). Eine Liste wird mit NIL (not in List) terminiert! NIL ≠ 0 !	Dynamische Speicherung beliebig vieler Elemente	Langsamerer Zugriff als bei Array, kein indizierter Zugriff

Datenstruktur	Beschreibung
Stack (Stapel)	LIFO-Prinzip: Lesen nur in umgekehrter Reihenfolge wie beim Schreiben (Intern oft als Liste realisiert)
Queue (Warteschlange)	FiFo-Prinzip: Lesen nur in Schreibreihenfolge (intern oft als Liste implementiert) Spezialfall: priority Queue: Einordnung nach Priorität
Hash table (Hash-Tabelle)	Implementierung einer dynamischen Menge durch eine Adresstabelle mit direktem Zugriff T **Hash**: mathematische Funktion, die die Position des Datenelements in der Menge berechnet z.B. h(k) = k modulo m Jedes Element der universalmenge hat einen eigenen Arrayeintrag, aber nur die Tatsächlichen Schlüssel verweisen auf Daten, der Rest auf NIL. Verweist die Hashfunktion h(k) mehrere ks auf einen Arrayeintrag, entsteht dort eine verkettete Liste.
Graph	Siehe unten
Baum	Siehe Unten
Heap (Halde)	Vereinigung der Eigenschaften eines Baumes mit den Operationen einer priority queue: Je nach Priorität wird minHeap oder maxHeap verwendet. Heaps werden

	meist über Bäume aufgebaut. **Eigenschaften maxHeap:** Für jeden Knoten (außer Wurzel) gilt : Der Wert eines Knotens i ist kleinergleich dem Wert des Vaterknotens V(i)

Graphen können mithilfe von zwei Mengen und einer Abbildung beschrieben werden:
V = Menge der Knoten , E = Menge der Kanten , Φ(e) (**Inzidenzabbildung**) ordnet jeder Kante e ∈ E die zwei Knoten aus V zu, die die Ecke aufspannen. **G (V, E, Φ)** wird **abstrakter Graph** genannt.
Φ(e) = (g,h) <=> e ist **inzident** zu g bzw. zu h <=> g,h sind **adjazent** zu e. Die **Adiazenzmatrix bzw. Inzidenzmatrix** beschreibt einen Graphen eindeutig.
E= {} <=> G(V, E , Φ) ist **entarted**, besteht nur aus **isolierten Knoten**.
Isomorphie = Strukturgleichheit: bijektive Abbildungen die die selben Inzidenzbeziehungen haben.

Zusammenhängend: von jedem beliebigen Knoten kann jeder andere erreicht werden. Ein gerichteter Graph gilt als zusammenhängend, falls sein ungerichter Graph zusammenhängend ist.

Schlinge/Schleife: Verbindung mit sich selbst. Φ(e) = (g,g)
e, f parallel: Φ(e) = (g,h), Φ(f) = (g,h)

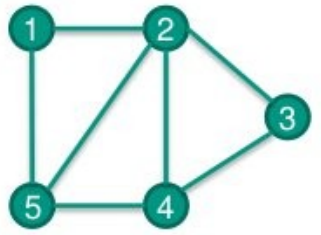
Grad d(g) = Anzahl zu g inzidenter Knoten; **d+(g)** : Ausgangsgrad **d-(g) :** Eingangsgrad
V‘(g) : Menge der unmittelbar benachtbarten (mit kante verbundenen) Knoten
mittelbar benachbart: es ex. Folge von Kanten, sodass man vom einen zum anderen Knoten kommt.

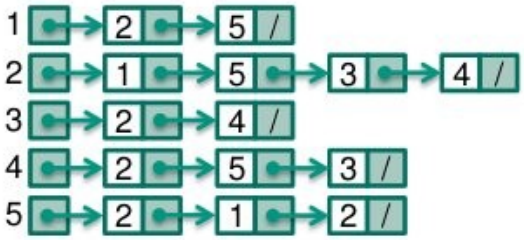
Spezielle Kantenfolgen:

Ungerichteter Graph		Gerichteter Graph	
$g^1 \neq g^{n+1}$	$g^1 = g^{n+1}$	$g^1 \neq g^{n+1}$	$g^1 = g^{n+1}$
offene Kanten-progression	geschlossene Kanten-progression	offene Kanten-progression	geschlossene Kanten-progression
Sind alle Kanten einer Progression voneinander verschieden			
Ketten-progression	geschlossene Kantenzug-progression	Weg-progression	Zyklus-progression
Berücksichtigt man die Kanten einer Progression ohne Ordnung			
Kette	geschlossener Kantenzug	Weg	Zyklus

zyklisch: Ein Graph, der mindestens einen Zyklus (incl. Schleife) enthält.
Baum: zusammenhängender, zyklenfreier Graph.
Wurzel: Knoten ohne Vorgänger d- = Ø . Ungerichteter Graph: frei wählbar.
Blätter : Knoten mit Knotengrad eins , Knoten ohne Nachfolger
Tiefe: Der Abstand von der Wurzel zu den Blättern . Ist die Tiefe für alle Blätter konstant, handelt es sich um einen **symmetrischen** Baum. Unterscheiden sich die Tiefen um max. 1 heißt der Baum **ausgeglichen**.
Binärbaum: Jeder Knoten außer die Blätter hat genau zwei Nachfolger.
Aufspannender Baum: mit dem Kruskal Algorithmus kann man aus jedem zusammenhängenden Baum durch gezieltes entfernen von Kanten einen Baum herstellen.

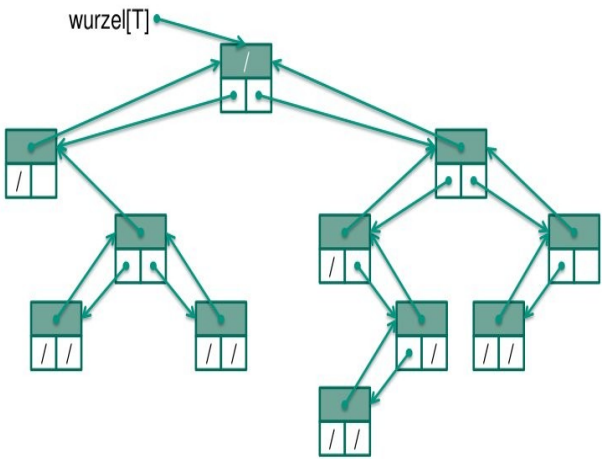
Adiazenzliste und Adiazenzmatrix:



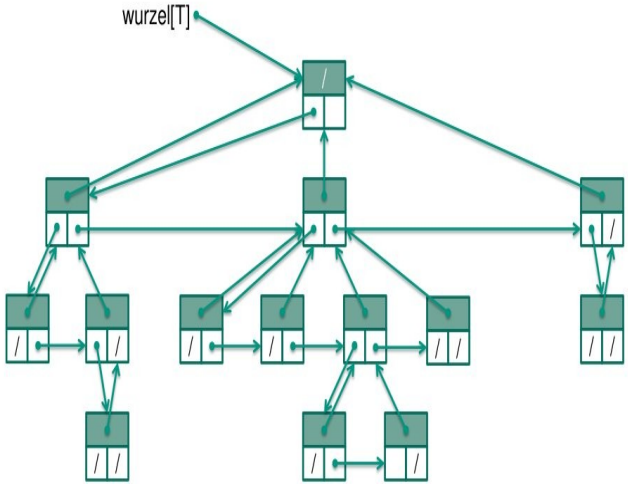


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Binärbaum als Liste



allgemeiner Baum als Liste



08_Algorithmen.pdf

Algorithmus: Genau definierte Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen in endlich vielen Schritten . Ein Algorithmus erzeugt aus einer Menge Eingabe-Größen eine Menge Ausgabe-Größen. Ein Algorithmus muss folgende Eigenschaften erfüllen:
Finitheit: das Verfahren muss in einem endlichen Text eindeutig beschreibbar sein
Ausführbarkeit: jeder Schritt des Verfahrens muss tatsächlich ausführbar sein
Dynamische Finitheit: das Verfahren darf zu jedem Zeitpunkt nur endlich viel Speicherplatz benötigen (Platzkomplexität)
Terminierung: das Verfahren darf nur endlich viele Schritte benötigen (Zeitkomplexität) .

Zusätzlich oft gefordert:
Determiniertheit: der Algorithmus muss bei denselben Voraussetzungen das gleiche Ergebnis liefern
Determinismus: die nächste anzuwendende Regel im Verfahren ist zu jedem Zeitpunkt eindeutig definiert .

Korrekter Algorithmus: stoppt nach jeder Eingabeinstanz mit der korrekten Ausgabe.
Beschreibung von Algorithmen: mit natürlichen Sprache, als Computerprogramm , als Hardwareentwurf , mit Pseudo Code , grafisch mit Nassi-Shneiderman etc.

Klassifikationsmerkmale von Algorithmen: Komplexität, Verfahren, Problemstellung, Anwendung
Platz/Zeitkomplexität: linear, logarithmisch, polynomial, exponentiell, wird asymtotisch in Abhängigkeit von der Eingabelänge betrachtet.
Verfahren: Aproximationsalgorithmus , dynamischer Algorithmus, Greedy-Algorithmus , genetischer Algorithmus , probabilistischer Algortihmus
Problemstellung: Entscheidungsalgorithmus , Optimierungsalgorithmus
Anwendung: Sortieren, Suchen, Graphentheorie, Kryptographie, Bilderkennung etc.
weitere Merkmale: Itterativ, rekursiv, halbiterativ, parallel (kann auf mehreren Prozessoren laufen), inkrementell, divide and conquer , in place , stabil, heuristisch , probabilistisch , abbrechbar .
Iterativ: es kommen keine Rekursionen vor.
Rekursiv: es kommen keine Schleifen vor, der Algorithmus ruft sich aus sich selbst heraraus auf. Jeder Rekursive Algorithmus kann (theoretisch) in einen itterativen umgewandelt werden.
Halb-iterativ/rekursiv: weder rekursiv noch itterativ, jedoch hauptsächlich Merkmale von iterativ/rekursiv.
Inkrementell: fügt nach und nach die Daten hinzu und aktualisiert das Ergebnis.
Devide and conquer: Problem wird solange in Teilprobleme zerlegt, bis sich diese lösen lassen.
In place: Zusätzlich zu den zu verarbeitenden Daten wird nur ein konstante Datenmenge an Speicher gebraucht.
Stabil: ein Sortieralgorithmus sortiert stabil, wenn bei gleichen Schlüsseln, die Reihenfolge unverändert bleibt.
Heuristisch: versuchen mit geringem Rechenaufwand eine zulässige Lösungen zu finden. Dafür, wegen Hilfsannahmen, keine optimale Lösung (z.B.Try and error)
Klassisch: versuchen optimale Rechenzeit und optimale Lösung zu garantieren. (z.B. Brute force)

probabilistisch: Verwendung von Zufallswerten (↔ deterministisch). Muss nicht immer

optimale Lösung liefern, ist dafür aber einfacher. Es gibt 3 Arten:
Macao Algorithmus : indefinit, Alogritmus liefert trotzdem immereine korrekte Antwort. Verwendung: mittlere Laufzeit << maximale Laufzeit
Monte-Carlo Algorithmus: indefinit, korrekte Ausgabe mit P=1-Q, wobei $Q \rightarrow 0$ ($t \rightarrow \infty$)
Las-Vegas Algorithmus : indefinit, Algorithmus liefert nie eine falsche Anwort, ist aber nicht immer determiniert.
Abbrechbar: Algorithmus kann jeder Zeit abgebrochen werden und liefert ein gültiges (aber meist nicht optimales) Ergebnis.

Algorithmenanalyse:

1. Untersuchung der Eingabe-Daten:
   Ziel: Herleitung der Laufzeit für beliebige Verteilungen. Evtl. bestimmen von (kleinsten) oberen Schranken (O-Notation), bzw. durchschnittliche Laufzeit.
2. Bestimmung der Abstrakten Operationen:
   Möglichst unabhängige Bestimmung der Laufzeit
   (unabhängig von: Hardware , OS, Programmiersprache , Verwendete Bibliotheken , Compileroptimierungen)
3. Iterative Verbesserung
   Finde innerste Schleife, überprüfe Befehle, ersetzte rekursiv durch iterativ, verwende Assembler

1:50 Regel: 1% des Programm-Codes bewirkt 50% der Laufzeit
90/10-Regel: 90% der Laufzeit bewirkt durch nur 10% des Programm-Codes

Aufwandsbetrachtung: Struktur- (verstehen und testen) , Speicher-, Laufzeitkomplexität
Effektivität: Wirksamkeit des Algorithmus zur möglichst guten Lösung der Aufgabenstellung unabhängig vom Aufwand
Effizienz: Verhältnis von Mitteleinsatz zum Grad der Zielerreichung, Wirtschaftlichkeit
Effizienz setzt Effektivität voraus !

Grobanalyse:

1) identifiziere Laufzeit-signifikante Teile (innerste Schleife)
2) Analysiere prinzipielles Laufzeitverhalten in Abhängigkeit von der Problemgröße n
3) Untersuche Folgende Szenarien: Günstigster Fall (**best case**), minimale Zahl an Schritten , Ungünstigster Fall (**worst case**), maximale Zahl an Schritten, Durchschnittlicher Fall (**average case**), typische, gemittelte Zahl an Arbeitsschritten

Kostenfunktionen hängen vom Maschinenmodell ab. Im Allgemeinen ist es nicht möglich, für jede Art von Eingabe eine Wachstumsfunktion zu formulieren.

Algorithmen-Wahl: Erarbeiten von Alternativen (z.B. 3), Analyse der Algorithmen, begründete Auswahl, Implementierung, Test.

Lauzeitkomplexitätsklassen:

Komplexität	Bezeichnung	
O(1)	Konstant	Laufzeit von n unabhängig. Perfekt! (Kommt fast nie vor)
O(log(n))	logarithmisch	Verdopplung von n bewirkt Anstieg um Konstante. Sehr günstig!
O(n)	Linear	Verdopplung von n führt zu Verdopplung der Zeit. Gut!
O(n log(n))		Log n << n, etwas mehr als Verdopplung. Immernoch gut!
$O(n^2)$	Quadratisch	Verdopplung von n führt zu vierfacher Zeit. Ungünstig!

$O(k^n)$	Exponentiell	Verdopplung von n bedeutet Quadrierung der Zeit! Katastrophal!

Notationen:
$f(n) = O(g(n)) \rightarrow$ f ist höchstens von der Ordnung g(n).
$f(n) = \Omega(g(n)) \rightarrow$ f ist mindestens von der Ordnung g(n)
$f(n) = \Theta(g(n)) \rightarrow$ f ist genau von der Ordnung g(n).

O(g(n)) = {f(n): es existieren positive Konstanten c und n0, so dass 0 < f(n) < cg(n) für alle n > n0 }

09_Sortieralgorithmen.pdf

Sortierungsalgorithmus: findet passende Permutation der Eingangsdaten nach gegebenem Vergleichskriterium. Ein Suchalgorithmus sucht in einer Gesamtmenge von Daten (Suchraum) nach Mustern oder Objekten mit bestimmten Eigenschaften
vier wichtige Suchalgorithmen: Bubble sort, merge sort, insertion sort, quick sort

Instanz eines Problems: Eingabe, die jegliche durch die Formulierung des Problems auferlegten Bedingungen erfüllt und alle notwendigen Daten umfasst, um die Lösung des Problems berechnen zu können.

Algorithmus	Beschreibung
Bubble sort *Sortierung durch Aufsteigen*	Einfaches, stabiles, vergleichsbasiertes in-place-Sortierverfahren. Vergleichsoperator muss Totalordnung der Liste ermöglichen. O(n) bis $O(n^2)$ **Verfahren:** Ein Feld wird vollständig durchlaufen, jeweilige Nachbarn werden verglichen und ggf. vertauscht. Die Eingabe wird von hinten sortiert, die Sortiermenge nimmt pro Schritt um eins ab.
Insertion Sort *Sortieren durch Einfügen*	Einfaches stabiles Sortierverfahren mit minimalem Speicherverbrauch für kleine oder schon vorsortierte Mengen. O(n) bis $O(n^2)$ **Verfahren**: Iteriere durch das Feld, vergleiche den aktuellen Wert mit den Werten, die links davon liegen und sortiere den Wert an der richtigen Stelle ein. Finden der richtigen Stelle im Vorsortierten mit *binärer Suche*.
Merge Sort *Sortieren durch Zusammenfügen*	Rekursiver, stabiler, paralleler Sortieralgorithmus nach divide&conquer-Prinzip, der im allgemeinen nicht in-place arbeitet. Er eignet sich besonders für verkettete Listen. Laufzeit: Immer O(n log n) **Verfahren:** 1. Teile: Teile die zu sortierende Sequenz von n Elementen in zwei Teilsequenzen zu je n/2 Elementen auf. 2. Beherrsche: Sortiere die zwei Teilsequenzen rekursiv mit Hilfe von Sortieren durch Mischen. 3. Verbinde: Mische die zwei sortierten Teilsequenzen, um die sortierte Lösung zu erzeugen.
Quick Sort *nach C. Antony, R. Hoare ca. 1960*	Schneller, rekursiver, nicht-stabiler, paralleler in-place-Sortier-algorithmus nach dem divide&conquer-Prinzip. Der Algorithmus verfügt über eine sehr kurze innerste Schleife, weswegen Quick-Sort im Mittel am schnellsten ist [durchschnittlich O(n log(n)) worst case: $O(n^2)$].

	Verfahren: 1. Teile: Zerlege das Feld A so in zwei evtl. leere Teilfelder A1 und A2, dass gilt: a1 $\in$ A1, a2 $\in$ A2 => a1 $\leq$ S $\leq$ a2 für alle a1, a2. S: Schlüsselelement. Berechne Index q 2. Beherrsche: Sortiere die beiden Teilfelder A1 und A2 durch rekursiven Aufruf von Quicksort. 3. Verbinde: Da die Teilfelder in place sortiert werden, ist keine Arbeit erforderlich, um sie zu verbinden. Das gesamte Feld A ist nun sortiert.

Laufzeit von Quick sort:

schlechteste Zerlegung: ein Teilfeld Länge 0, zweites Teilfeld Länge n-1; Eingabefeld bereits vollständig geordnet => $O(n^2)$.

beste Zerlegung: ein Teilfeld Länge n/2, zweites Teilfeld Länge n/2-1 => $O(n \log_2 n)$.

balancierte Zerlegung: die mittlere Laufzeit von Quicksort ist viel näher an der des besten Falles als an der des schlechtesten Falles: z.B. 1:9 und sogar 1:99 => O(n log2n). Partition liefert in >80% der Fälle besser balanciert als 1:9. Verbesserungspotential durch geeignete Wahl des Pivot-Elementes.

10_Algorithmen_auf_Graphen.pdf

Häufige Problemstellungen auf Graphen: Pfad finden, Knoten finden. Lösung oft mit Hilfe von Suchalgorithmen.

Trade-off Sortieroverhead, Sucheffizienz: Suchen auf geordneten Mengen ist erheblich effizienter, allerdings muss die Menge erst sortiert werden. Anwendung, Datenstruktur, Suchalgorithmus und ggf. Sortieralgorithmus müssen aufeinander abgestimmt sein.

Suchalgorithmus	Beschreibung	Laufzeit
Lineare Suche	In einer unsortierten Liste wird jedes Element verglichen. Durch die lineare Anordnung der Elemente und die Navigierbarkeit zu dem Nachfolger (einfach verkettete Liste) oder Vorgänger und Nachfolger (doppelt verkettet), bleibt der Aufwand bei der linearen Suche asymptotisch gleich.	O(n)
Binäre Suche	Die Binärsuche springt wie beim Intervallschachtel-Verfahren rekursiv immer in die Mitte des noch zu durchsuchenden Intervalls und überprüft, ob der gesuchte Wert größer oder kleiner als das so bestimmte Pivotelement ist (Vorgehenentlang eines virtuellen Entscheidungsbaums) . Der Suchraum wird bei jedem Schritt halbiert. **Voraussetzung**: Sortierte Liste	O(log(n)) worst case
Breitensuche (**B**readth **F**irst **S**earch)	Durchläuft systematisch alle Kanten um alle von s aus erreichbaren Knoten zu finden . Der BFS findet alle von einem vordefinierten Startknoten s erreichbaren Knoten und erzeugt einen Breitensuchbaum mit Wurzel s, der alle erreichbaren Knoten enthält und sie auf dem kürzesten Weg mit s verbindet. Der BFS funktioniert für gerichtete und ungerichtete Graphen	O(V+E) Graph: G = (V, E)
Tiefensuche (**D**epth **F**irst **S**earch)	Findet alle vom Startknoten s erreichbare Knoten, indem der zuletzt gefundene Knoten solange abgearbeitet wird, bis kein Nachfolger mehr gefunden wird. Erst dann kehrt	O(V+E) Graph: G = (V, E)

	der DFS zum Vaterknoten zurück. Es ist also vor allem der Entdeckungszeitpunkt interessant. Je nach Zusammenhang des Graphen ergeben sich so Tiefensuchwälder. Der DFS arbeitet auf gerichteten und ungerichteten Graphen und eignet sich besonders für Labyrinthe.	
Dijkstra-Algorithmus	Löst das Problem des kürzesten Pfades bei einem Startknoten und einem positiv gewichteten, gerichteten Graphen G(V, E) . **Vorgehen:** 1. Initialisierung aller Knoten von G mit einer Distanz „unendlich“ 2. Initialisiere Startknoten mit Distanz gleich 0, markiere ihn als „aktiv“ 3. Berechne temporäre Distanzen der Nachbarknoten als Summe ihrer Distanz mit dem Kantengewicht 4. Aktualisiere die Distanz eines Knotens, wenn die neu berechnete kleiner ist als die aktuelle, setzte den aktuellen Knoten als Vorgänger 5. Wähle den Knoten mit minimaler Distanz, markiere seine Distanz als permanent 6. Wiederhole bis kein Knoten mehr mit temporärer Distanz existiert	$O(V^2+E)$ für Liste $O(V \cdot \log(V) + E)$ für Fibonaggi-Heap Graph: G = (V, E)

Die Breitensuche und die Tiefensuche sind grundlegenden Graphenalgorithmen.

Zyklensuche mit DFS: Führe Tiefensuche durch . Sobald die Tiefensuche auf eine Rückwärtskante stößt, die auf einen Knoten mit Status „in Bearbeitung“ verweist, ist ein Zyklus gefunden .

Critical Path Method : Anwendung der Graphentheorie zum Finden des kritischen Pfades in einem Netzwerk.
11_Optimierungsalgorithmen.pdf

Gründe für exakte Unlösbarkeit: Formulierung als Algorithmus ist nicht möglich , Exakter Algorithmus hat Rechenzeit der Ordnung $O(n^x)$ oder $O(x^n)$ oder sogar O(n!)
Heuristische Verfahren : 1. Anfangslösung 2. Iterativen Verbesserung
I**terative Verfahren**: Greedy, Probabilistisch, Hill climbing
Anwendungsbeispiel: Chiplayout

Partitionierung : Aufteilung der (geschätzten) Fläche eines Chips in einzelne funktionale Blöcke, die mit anderen Blöcken durch möglichst wenige, kurze Leitungen verbunden sind .
Aufgabe der Partitionierung ist es eine Partition P der Modulmenge M = { m1, ..., mi, ...,mn } unter Randbedingungen wie: Zahl der Verbindungen , Flächen der Module zu finden. Zur Vereinfachung wird nach Bipartition P = { B1, B2 } mit minimaler Kostenfunktion gesucht.
Hauptbedingungen Platzierung: geringe Gesamtfläche, geringe Gesamtnetzlänge
Nebenbedingungen Platzierung: Zeit- und Temperaturverhalten
Platzierungsstrategien : Konstruktive Verfahren (Anlagerungsverfahren) , Iterative Verfahren (Random Interchange, Kernighan-Lin, Simulated Annealing, Evolutionäre Algorithmen)
Verdrahtung : Globales Verdrahten, Aufteilung der Verdrahtungsfläche in Kanäle , Zuweisung der Netze auf Kanäle, Detailliertes Verdrahten
Cij: Zahl der Netze, die den Modul mi mit dem Modul mj verbinden,
und mi und mj liegen in verschiedenen Blöcken B1 und B2.

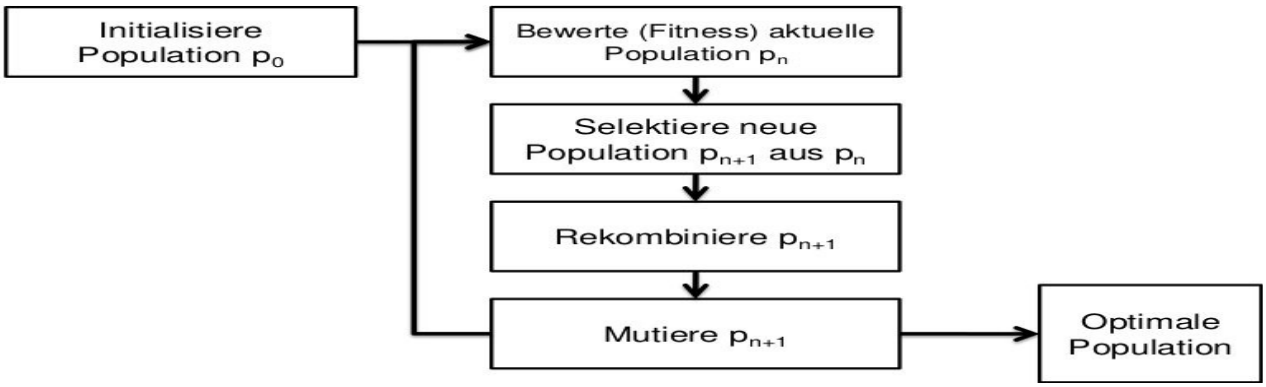
Analgerungsverfahren:
1. Schritt: Auswahl des Saatknotens für Block 1, z.B. der Knoten mit der größten Anzahl an Verbindungen mit anderen Baugruppenknoten
N Schritte: Zuordnung derjenigen Baugruppenknoten zu Block 1, deren Differenz aus den Verbindungen zu Block 2 und den Verbindungen zu Block 1 am kleinsten ist.
Abbruch: Wenn die Anzahl der Knoten, die Fläche oder die Verlustleistung von Block 1 einen vorher festgelegten Wert überschreitet.
Problem: Kann in lokalem Minimum der Kostenfunktion enden .

Random Interchange
1. – N. Schritt : Zufällige Auswahl zweier Knoten , Berechnung der Kosten vor und nach dem Austausch , Beibehalten des Austausches bei Kostenminderung
Abbruch: nach N Austauschversuchen.
Bewertungsfunktion: D(mi, mj) = E(mi) + E(mj) - I(mi) - I(mj) - 2 cij

Kernighan-Lin-Algorithmus
1. – N. Schritt : Auswahl des Knotenpaars mit der besten Kostenfunktion, welches noch nicht getauscht wurde , vertauschen der beiden Knoten, trotz eventueller Verschlechterung der Gesamtkosten , Abspeicherung der Kosten vor und nach dem Tausch.
Abbruch: wenn alle Knoten getauscht wurden, die Knoten also alle den Block gewechselt haben . Auswahl der kostengünstigsten Konfiguration anhand der abgespeicherten Werte.
Bewertungsfunktion: D(mi, mj) = E(mi) + E(mj) - I(mi) - I(mj) - 2 cij

Simmulated Annealing:
1. – N. Schritt : Zufälliges Vertauschen zweier Knoten aus zwei unterschiedlichen Blöcken, berechnung der Kosten vorher,nachher, bei Verbesserung der Kostenfunktion wird getauscht , bei Verschlechterung keine zwingende Abweisung, sondern Akzeptanz mit einer temperaturproportionalen Wahrscheinlichkeit [y=exp(-diff_cost ÷ T)] .
Abbruch: nach N Schritten .

Evolutionäre Algorithmen:



Merkmale: Außer der Zielfunktion sind keinerlei weitergehende Informationen notwendig, denn es wird mit einer Codierungen von der Konfigurationen und nicht mit Konfigurationen selbst gearbeitet. Verbesserungen finden nicht von Punkt zu Punkt, sondern von Population zu Population statt. Es wird stets aus einer Menge (Population) von Konfigurationen (Individuen) eine neue (Generation) erzeugt. Die Übergangsregeln sind probabilistisch, nicht deterministisch. Die Erzeugung einer neuen Generation erfolgt mit Hilfe von Operatoren, die Veränderungen finden anhand der Codierungen statt.

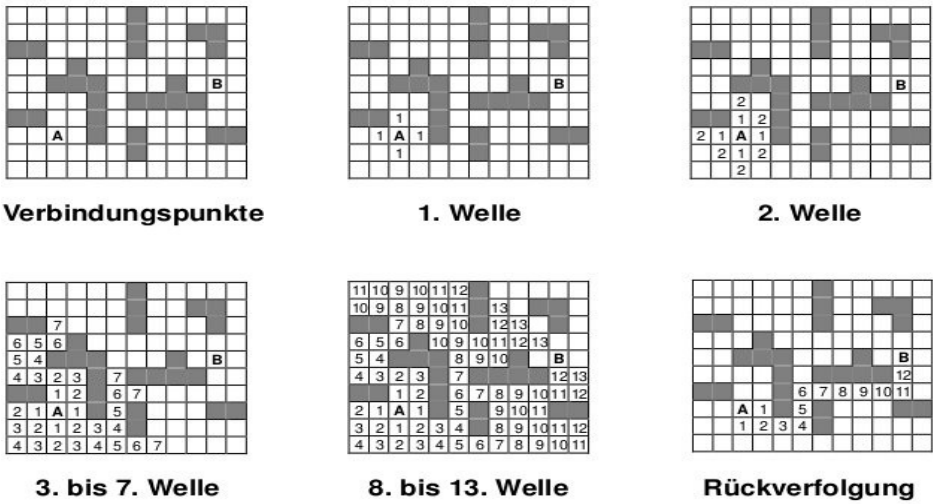
Operatoren: Selektion (Reproduktion) , Rekombination (Kreuzung) , Mutation

Selektion: Je besser die Zielfunktion (engl. fitness) eines Individuums desto häufiger kann es sich clonen (survival of the fittest)
Rekombination: Die Zielfunktionen zweier Individuen wird rekombiniert, die Eltern verschwinden.
Mutation: zufällige Änderung der zielfunktion zur Überwindung lokaler Optima.

Parameter: Codierung Code stellt Individuum J dar , Population P: Menge von (nicht notwendigerweise unterschiedlichen) Codierungen , g: die zu optimierende Fitnessfunktion ,
Kreuzungswahrscheinlichkeit p_c , Mutationswahrscheinlichkeit p_m, Anfangspopulation P_0 .

Verdrahtung : Bestimmung der geometrischen Strukturen für die Verbindungen zwischen den Zellen gemäß der Netzliste unter Berücksichtigung technologie- und schaltungsspezifischer Designregeln .
Technologieabhängigkeit: Standard- und Makrozellen (Verdrahtungsfläche in einzelne Verdrahtungskanäle aufteilen , globale Wegsuche, festlegen der Verdrahtungsreihenfolge der Kanäle), Erzeugen der Endverdrahtung für jeden Kanal , Moderne Gate Arrays (Sea of Gates) (Aufteilung in Blöcke, Block für Block behandeln)

LEE-Algorithmus (maze routing)
1. Schritt: Auswahl des Startpunktes
N. Schritt: Fortlaufende Nummerierung aller Nachbarpunkte in Achsenrichtung
Abbruch: Rückwärtssuche des kürzesten Weges entlang der Nummerierung



Modifikationen des LEE-Algorithmus : Günstige Wahl des Startpunktes , Gleichzeitige Wegsuche von Start- und Zielpunkt , Einführung eines Suchrahmens , Übergang von Zweipunkt- zu Mehrpunktverbindungen , Dreidimensionale Wegesuche möglich (Mehrlagenverbindungen)

12_Rechnerarchitektur_1.pdf

Steuerwerk (Leitwerk) : Steuerung und Koordination des zeitlichen Ablaufs der Komponenten des Prozessors (Versorgung mit Daten und Adressen , Selektion von Geräten , Auswahl von Operanden und Operationen)

Befehl: (maschinenlesbare) Anweisung, aus der hervorgeht welche Operation mit welchen Operanden durchgeführt werden soll . Ein Maschinenbefehl startet eine Mikrobefehlsfolge.
Befehlsformat: Festlegung, wie die für einen Befehl notwendigen Angaben dargestellt werden .

Unterprogrammaufruf mit Stack-Speicher : Die Adresse des nächsten Befehls wird auf den Stack gespeichert, dann wird in die Subroutine gesprungen. Wird aus der Subroutine weiter gesprungen, so wird die Adresse des nächsten Befehls wiederum auf den Stack geschrieben. Damit steht nach verlassen einer Subroutine die Adresse des nächsten Befehls auf dem Stack bereit.

Mikroarchitektur (interne Architektur): interne Struktur des Prozessors . Verschiedene interne Architekturen können dieselbe externe Architektur aufweisen , sie bilden eine Computerfamilie .

***I*nstruction *S*et *A*rchitecture:** externe Architektur oder Befehlssatzarchitektur. Sie definiert, wie Daten repräsentiert werden, wo diese gespeichert und wie auf sie zugegriffen werden kann. Außerdem wird durch die ISA definiert, welche Operationen auf den Daten ausgeführt werden kann und wie die Befehle codiert sind.

Datenformate: Integer (z.B. Byte, 16-bit Word, 32-bit Longword, and 64-bit Quadword), gepackte und ungepackte BCD Zahlen , ASCII Character , Floating-point data formats (ANSI/IEEE 754-1985 standard, basic oder extended, je zwei Längen: single or double) , Multimedia
big-endian: most significant byte first
little-endian: least significant byte first

Adressraum : Auf Assemblerebene gibt es register space, stack space, I/O space, control space . Außer Register werden alle anderen Adressräume auf einen einzigen zusammenhängenden Speicheradressraum abgebildet.

Adressierungsmodi:

Modus	**Beschreibung**	**Befehl**:
Register	Operand wird in einem Register gespeichert	load Reg1,Reg2
Immediate	Operand ist Teil der Anweisung	load Reg1,#const
Direct	Die Adresse des Operanden im Speicher wird in die Anweisung gespeichert	load Reg1,(const)
Register indirect	Die Adresse des Operanden im Speicher wird in ein Register gespeichert	load Reg1,(Reg2)
Post-Increment	Wie Register indirect, nur dass der Inhalt des Registers nach dem Gebrauch der Adresse inkrementiert wird. Das ist z.B. nützlich bei Schleifen.	load Reg1,(Reg2)+
Pre-Decrement	Der Inhalt des Registers wird dekrementiert und anschließend als Register indirect Adresse verwendet. z.B. zum rückwärts durchlaufen eines Feldes	load Reg1,-(Reg2)
Displacement	Die effektive Adresse ist die Adresse aus einem Register plus eine in der Anweisung spezifizierte Konstante (Displacement).	load Reg1,displ(Reg2)
Indexed and scaled indexed	Wie Register indirect, nur dass die Adresse aus dem Register um einen Vorfaktor skaliert werden kann.	load Reg1,(Reg2*scale)