Hinweis zur Nutzung dieses Skriptes:

Achtung, drucken Sie das Skript (noch) nicht aus! Es wird parallel zur Vorlesung erstellt und laufend aktualisiert. Wenn Sie jetzt schon ihren Drucker bemühen, ist das entweder eine Verschwendung von Papier (weil Sie es später nochmal drucken) oder sie erhalten ein evtl. inkonsistentes Dokument (wenn Sie verschiedene Teile zusammenheften).
Mit Abschluss des Semesters wird Ihnen das gesamte Skript zur Verfügung stehen!

Dieses Dokument enthält außerdem Hyperlinks zu weiteren Quellen im Internet, die in Word oder mit dem Acrobat Reader durch Anklicken abrufbar sind. Für die Aktualität der Links übernehme ich keinerlei Gewähr..

HS, 22.4.2010

Kapitel 0: Einführung

0.1 Inhalt der Vorlesung, Organisatorisches, Literatur

Die Vorlesung „Grundlagen der Programmierung“ hat laut der Modulbeschreibung des Bachelor-Studienganges Informatik folgende *Lern- und Qualifikationsziele*:

> *Studierende verstehen die Funktionsweise von Computern und die Grundlagen der Programmierung. Sie beherrschen eine objektorientierte Programmiersprache und kennen andere Programmierparadigmen.*

Daraus ergeben sich folgende *Vorlesungsinhalte*:

- Grundlagen: Algorithmus, von-Neumann-Rechner, Programmierparadigmen
- Konzepte imperativer Programmiersprachen: Grundsätzlicher Programmaufbau; Variablen: Datentypen, Wertzuweisungen, Ausdrücke, Sichtbarkeit, Lebensdauer; Anweisungen: Bedinge Ausführung, Zyklen, Iteration; Methoden: Parameterübergabe; Rekursion;
- Konzepte der Objektorientierung: Objekte, Klassen, Abstrakte Datentypen; Objekt -Variablen/-Methoden, Klassen -Variablen/-Methoden; Werte und Referenztypen; Vererbung, Sichtbarkeit, Überladung, Polymorphie; dynamisches Binden; Ausnahmebehandlung; Oberflächenprogrammierung; Nebenläufigkeit (Threads)
- Einführung in eine konkrete objektorientierte Sprache (z.B. JAVA): Grundaufbau eines Programms, Entwicklungsumgebungen, ausgewählte Klassen der Bibliothek, Programmierrichtlinien für eigene Klassen, Techniken zur Fehlersuche (Debugging)
- Einfache Datenstrukturen und Algorithmen: Listen, Stack, Mengen, Bäume, Sortieren und Suchen
- Softwareentwicklung: Softwarelebenszyklus, Software-Qualitätsmerkmale
- Alternative Konzepte: Zeiger, maschinennahe Programmierung, alternative Modularisierungstechniken

Die Vorlesung (4 SWS) ist nur mit begleitender Übung (2 SWS), Praktikum (2 SWS), Selbststudium, Vorlesungsmitschrift, Hausaufgaben (in Zweiergruppen bearbeitet, korrigiert und bewertet, in der Übung besprochen) sinnvoll. Als Prüfung findet eine Abschlussklausur (120 Minuten Dauer) statt; die Zulassung zur Klausur ist an die Erreichung einer bestimmten Punktzahl in Übungen und Praktikum gebunden.

Für die erfolgreiche Teilnahme gibt es 12 Studienpunkte nach dem ECTS-System (European Credit Transfer System). Ein Studienpunkt (SP) entspricht 30 Zeitstunden Arbeitsaufwand; d.h., der Gesamtaufwand liegt bei ca. 360 Arbeitsstunden:

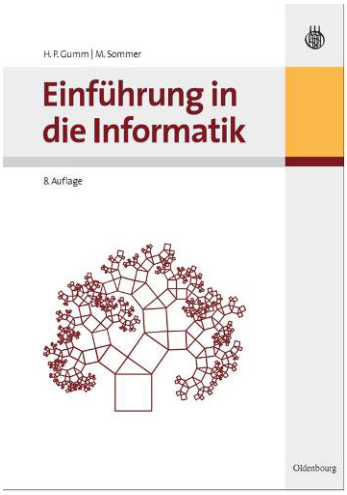
- Vorlesung Grundlagen der Programmierung;
  4 SWS, 6 SP, 60 Anwesenheitsstunden (4h/Woche), 120 h Aufbereitung (8h/Woche)
- Übung Grundlagen der Programmierung;
  2 SWS, 3 SP, 30 Anwesenheitsstunden (2h/Woche), 60 h Aufbereitung (4h/Woche)
- Praktikum Grundlagen der Programmierung;
  2 SWS, 3 SP, 30 Anwesenheitsstunden (2h/Woche), 60 h Aufbereitung (4h/Woche)

Für die Übungen wird 14-tägig ein Übungsblatt verfügbar gemacht, dessen Lösungen zwei Wochen später elektronisch abzugeben sind. Das Aufgabenblatt wird in den Übungen vor- und nachbesprochen. Für die Bearbeitung eines Aufgabenblattes sind also ca. 8h erforderlich.

Während für Vorlesung und Übung die Anwesenheit obligatorisch ist, kann beim Praktikum auf eine Anwesenheit verzichtet werden, wenn der/die Teilnehmer anderweitig über einen Rechner (Laptop) verfügt, auf dem die Aufgaben bearbeitet werden können. Die Gesamtzeit für die Bearbeitung der Praktikumsaufgaben beträgt ca. 6h/Woche.

Literaturhinweise

Leider gibt es kein einzelnes Buch, welches genau den Stoff der Vorlesung enthält.
Als Literatur zur Vorlesung empfehle ich das Buch von Gumm und Sommer.
(Aktuell: 8. Auflage; frühere Auflagen gibt es zum Teil im Sonderangebot)



Als Ergänzung dazu ist nachfolgend eine Liste relevanter Lehrbücher angegeben..

Lehrbücher: Einführung in die Informatik

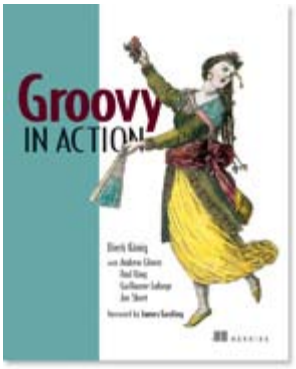
- M. Broy: Informatik, eine grundlegende Einführung. Band 1: Programmierung und Rechnerstrukturen, Band 2: Systemstrukturen und theoretische Informatik. Springer-Lehrbuch (+ Aufgabensammlung) (Monumentalwerk)
- P. Levi, U. Rembold: Einführung in die Informatik für Naturwissenschaftler und Ingenieure, Hanser, (konzise)
- G. Goos: Vorlesungen über Informatik (vier Bände: Bd. 1: Grundlagen und funktionales Programmieren, Bd. 2: Objektorientiertes Programmieren und Algorithmen, Bd. 3: Berechenbarkeit, formale Sprachen, Spezifikationen, Bd. 4: Paralleles Rechnen und nicht-analytische Lösungsverfahren)
- F. L. Bauer, G. Goos: Informatik - Eine einführende Übersicht, 4. Auflage. Bd. 1+2, Springer („der Klassiker“, etwas veraltet)
- L. Goldschlager, A. Lister: Informatik, Eine moderne Einführung. Hanser Studienbücher, (ebenfalls etwas veraltet; in der Bibliothek 40 Ex. vorhanden)
- F. Kröger: Einführung in die Informatik – Algorithmenentwicklung. Springer Lehrbuch(formale Grundlagen, keine OO-Konzepte, als Ergänzung empfohlen)
- H. Balzert: Lehrbuch Grundlagen der Informatik (als Ergänzung der Vorlesungsthemen)
- H.-J. Appelrath, J. Ludewig: Skriptum Informatik - Eine konventionelle Einführung. Teubner/VdF (+ Aufgabensammlung) (Betonung auf „konventionell“!)

- A. Aho, J. Ullman: Informatik - Datenstrukturen und Konzepte der Abstraktion. (deutsche Fassung von: Foundations of Computer Science) Thomson Publishing / Computer Science Press (für Fortgeschrittene)
- R. Sedgewick: Algorithmen in Java (auch für Fortgeschrittene)

Lehrbücher: Programmieren in Groovy

In der Vorlesung „Grundlagen der Programmierung“ und besonders im zugehörigen Praktikum sollen auch Programmierkenntnisse unterrichtet werden. Trotzdem ist diese Veranstaltung auf keinen Fall ein Programmierkurs; es wird erwartet, dass sich die Studierenden selbständig in programmiersprachliche Details an Hand geeigneter Lehrmaterialien (Bücher oder Online-Handbücher) einarbeiten.

Als notationelle Basis dient dabei zunächst die Skriptsprache Groovy, die auf der verbreiteten Programmiersprache Java aufbaut und einige moderne Erweiterungen vorsieht. Später gehen wir dann auf Java zurück. Das Haupt-Referenzbuch zu Groovy ist das Buch von König et. al.



Weitere Bücher zu Groovy sind

- K. Barclay, J. Savage: Groovy Programming: An Introduction for Java Developers Morgan Kaufmann, 2006.
- S. Davis: Groovy Recipes – Greasing the Wheels of Java. Pragmatic Bookshelf, 2008.
- V. Subramaniam: Programming Groovy: Dynamic Productivity for the Java Developer. Pragmatic Bookshelf, 2008.
- C. Judd, J. Nusairat: Beginning Groovy and Grails: From Novice to Professional. Apress 2008.
- B. Abdul-Jawad: Groovy and Grails Recipes – a Problem-Solution Approach. Apress 2008.

Darüber hinaus gibt es zu Groovy eine umfangreiche Online-Dokumentation, siehe http://groovy.codehaus.org/Documentation.

Literatur zu Java

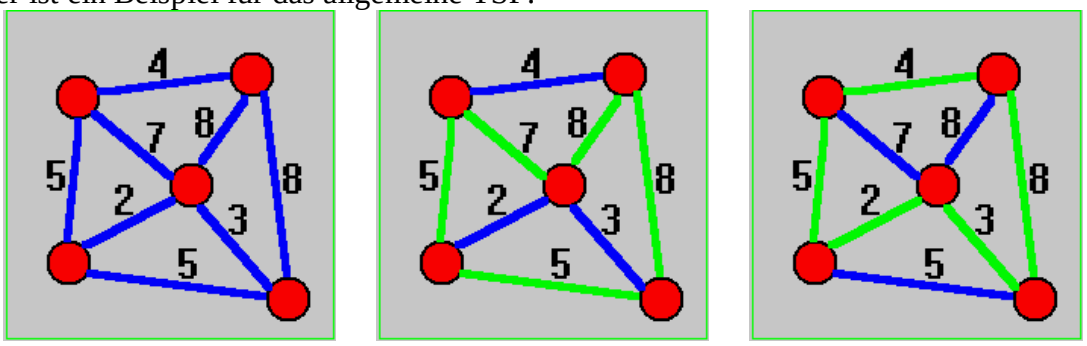
Zum Erlernen der Programmiersprache Java kann entweder das Buch von Bell und Parr oder das von Bishop dienen. Wer sich intensiver mit Java auseinander setzen möchte, dem sei das Buch von Gosling als ultimative Referenz empfohlen.

- J. Bishop: Java lernen (dt. Ausgabe von Java Gently), Addison Wesley / Pearson, 2. Aufl. 2001 (südafrikanisches Flair)
- K. Arnold, J. Gosling, D. Holmes: Die Programmiersprache Java (dt. Ausgabe von The Java Programming Language). Addison-Wesley 1996 („die Referenz“)
- D. Barnes, M. Kölling: Objektorientierte Programmierung mit Java (deutsche Ausgabe von: Objects First with Java - A Practical Introduction using BlueJ). Prentice Hall / Pearson, Sept. 2003 (für Vorlesung empfohlen)
- D. Bell, M. Parr: Java für Studenten – Grundlagen der Programmierung. Prentice Hall / Pearson, 3. Aufl. 2003 (systematischer Java-Lehrgang)
- E.-E. Doberkat, S. Dißmann: Einführung in die objektorientierte Programmierung mit Java. Oldenbourg-Verlag, 2. Auflage 2002 (Wiener Hofzwerge betreiben Informatik)
- H. W. Lang: Algorithmen in Java. Oldenbourg-Verlag 2003
- Küchlin, Weber: Einführung in die Informatik – Objektorientiert mit Java

0.2 Einführungsbeispiel

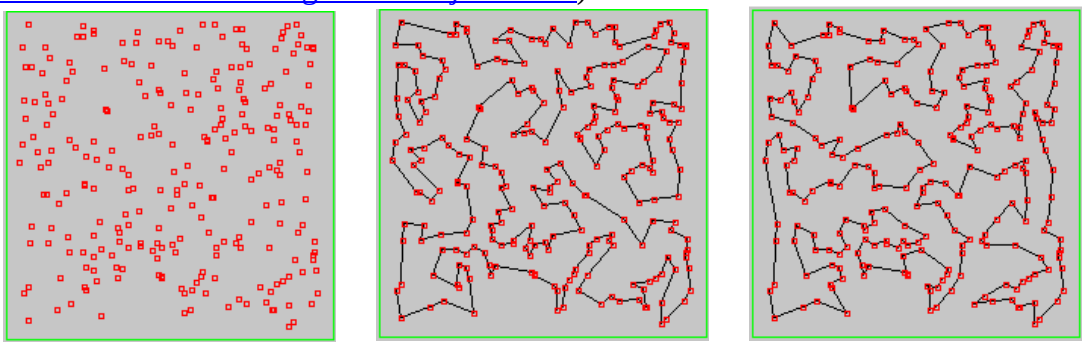
Um sich der Frage zu nähern, was eigentlich praktische Informatik ist, betrachtet man am besten ein Beispiel. Ein bekanntes Problem der Informatik ist das *Problem des Handlungsreisenden (Travelling Salesman Problem, TSP)*. Ein Handlungsreisender muss bei seiner Arbeit Kunden besuchen, die in verschiedenen Orten wohnen. Aufgabe der Sekretärin ist es nun, eine Tour zu planen, die ihn zu jedem Kunden genau einmal führt und am Schluss wieder zum Ausgangspunkt zurück bringt. Natürlich sollte die Tour optimal sein, d.h., möglichst wenig Ressourcen verbrauchen. Abhängig davon, welchen Begriff von Ressource man zu Grunde legt, gibt es verschiedene Varianten der Aufgabenstellung. Beim *allgemeinen* TSP gehen wir davon aus, dass der Handlungsreisende z.B. mit dem Flugzeug unterwegs ist: da es nicht immer von jedem beliebigen Punkt zu jedem anderen eine direkte Flugverbindung gibt, muss die Sekretärin das Streckennetz der Fluggesellschaften und die jeweiligen Flugzeiten (oder Ticketpreise) berücksichtigen. Beim *euklidischen* TSP bewegt sich der Handlungsreisende zu Fuß oder mit dem Auto, d.h. er kommt überall hin und die Kosten sind proportional zu den Entfernungen zwischen den Punkten auf der Landkarte. Das *metrische* TSP ist eine Verallgemeinerung des euklidischen und ein Spezialfall des allgemeinen TSP: zwischen je zwei Punkten besteht eine Verbindung, und die Dreiecksungleichung gilt (die Summe der Kosten von A nach B und der Kosten von B nach C ist größer gleich der Kosten von A nach C).

Hier ist ein Beispiel für das allgemeine TSP:



Die beiden angegebenen Lösungen haben die Länge 33 und 22. Welches ist die optimale Tour?

Hier ist ein Beispiel für das euklidische TSP (nach http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/java/TSP/):



Dieses euklidische TSP hat 250 Punkte. Angegeben sind eine Näherungslösung (Weglänge 12,91) und eine optimale Lösung (Weglänge 12,48).

Was ist nun eine geeignete Methode, um das Problem zu lösen? Eine häufige Herangehensweise der Informatik ist es, ein Problem auf ein einfacheres zurückzuführen. Wir wissen, wie die Lösung eines TSP mit nur 2 Punkten (Firmensitz und ein Kunde) aussieht: Der Handlungsreisende muss hin- zurückfahren. Angenommen, wir wissen, wie wir eine Tour

mit *n* Punkten löst. Dann kommen wir zu einer Lösung für *(n+1)* Punkten, indem wir einen beliebigen Punkt zunächst weglassen, eine optimale Tour für *n* Punkte konstruieren, und den weggelassenen Punkt dann als erstes Ziel in die Tour einfügen. Das ist natürlich noch nicht die optimale Lösung, aber wenn man das für alle Punkte der Reihe nach macht und die Tour mit der minimalen Länge wählt, bekommt man dadurch das Optimum.
Diesen Algorithmus könnte man etwa wie folgt notieren:
Wir nehmen an, der Startpunkt (Firmensitz) sei fest gegeben und notieren eine Tour durch die Folge der zu besuchenden Punkte (ohne den Startpunkt S) und durch die zugehörige Länge.

```
Tour minTour (Punktmenge M) {
  Wenn M einelementig (M={A}) dann
    Rückgabe ((A, 2* |SA|))
  Sonst {
    // Berechne die kürzeste Tour rekursiv
    Sei minT eine neue Tour, wobei zunächst
      Punkte (minT) = undefiniert, Länge(minT)=unendlich;
    Für jedes x aus M {
      Tour rek = minTour (M-{x});
      Sei A der erste Ort von rek;
      Sei Tour try gegeben durch
        Punkte (try) = append(x, Punkte(rek)));
        Länge (try) = Länge(rek) - |SA| + |Sx| + |xA|;
      Wenn Länge(try) < Länge(minT) {minT=try};
    }
    Rückgabe minT;
  }
}
```

Wenn wir den Ablauf dieser rekursiven Funktion z.B. für die Punkte {ABC} betrachten, stellen wir folgende Aufrufe fest:

```
minTour ({ABC}) ruft
  minTour({BC}) ruft
    minTour({C}) und
    minTour({B}).
  minTour({AC}) ruft
    minTour({C}) und
    minTour({A}).
  minTour({AB}) ruft
    minTour({B}) und
    minTour({A}).
```

Das bedeutet, ein Aufruf mit *n* Punkten stützt sich auf *n* Aufrufe mit *(n-1)* Punkten ab, jeder davon stützt sich auf *(n-1)* Aufrufe mit *(n-2)* Punkten ab usw. Insgesamt gibt es

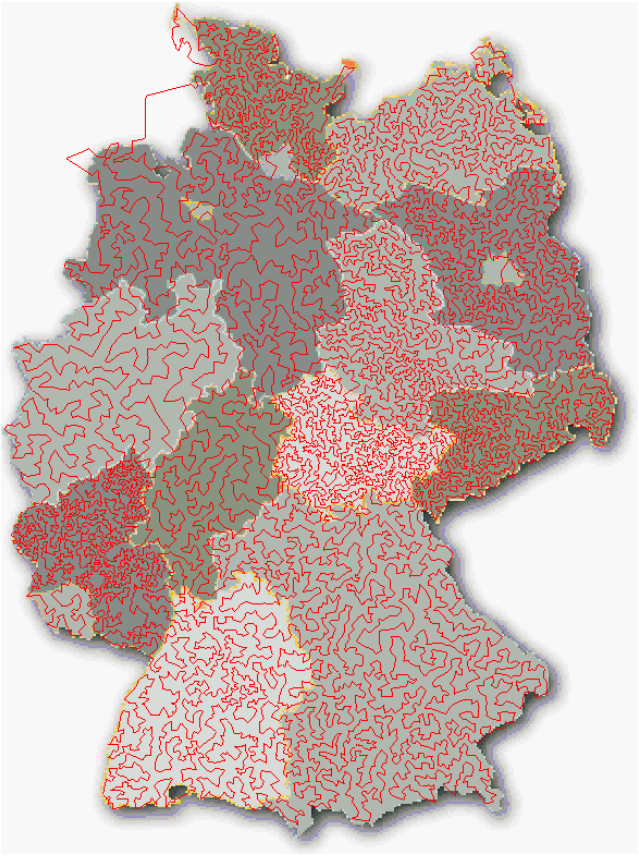
$$n*(n-1)*(n-2)*\ldots*3*2*1=\prod_{i=1}^{n} i = n!$$

Aufrufe. Im Wesentlichen konstruiert der Algorithmus sämtliche Permutationen der Folge der Punkte. Die Anzahl der Permutationen von *n* Elementen ist *n!*. Wir sagen, der Algorithmus hat die Komplexität O(*n!*).
Nachfolgende Tabelle gibt einen Überblick über das Wachstum verschiedener Funktionen.

n	log(n)	256*n	n^2	$17*n^3$	2^n	n!	n^n
1	0,0	256	1	17	2	1	1
2	0,3	512	4	136	4	2	4
3	0,5	768	9	459	8	6	27
10	1,0	2560	100	17000	1024	3628800	10000000000
50	1,7	12800	2500	2125000	1,1259E+15	3,04141E+64	8,88178E+84
256	2,4	65536	65536	285212672	1,15792E+77	#ZAHL!	#ZAHL!
1000	3,0	256000	1000000	1,7E+10	1,0715E+301	#ZAHL!	#ZAHL!
10000	4,0	2560000	100000000	1,7E+13	#ZAHL!	#ZAHL!	#ZAHL!

Wenn wir den Algorithmus auf einem schnellen Pentium DualCore ausführen, der bis zu 4 Milliarden Operationen pro Sekunde ausführt, dann müssen wir für n=10 etwa eine Millisekunde warten. Für n=50 erhöht sich unsere Wartezeit allerdings auf etwa 10^{28}=3.000.000.000.000.000.000.000.000.000 oder 3 Quatrilliarden Jahre. Das heisst, für große Werte von n ist der Algorithmus praktisch nicht verwendbar. Auf der anderen Seite ist es ein offenes Problem, ob es tatsächlich einen substantiell besseren Algorithmus gibt! Wer einen Algorithmus entdeckt, welcher das TSP mit einer polynomialen Anzahl von Schritten löst (abhängig von n), wird mit Sicherheit weltberühmt werden.

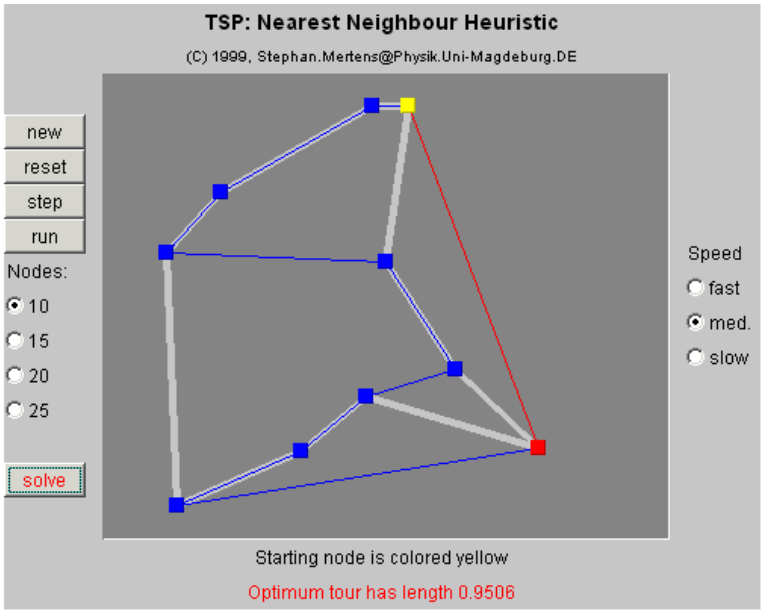
Natürlich gibt es einige Tricks, um die Laufzeit des Algorithmus zu verbessern. Zum Beispiel fällt auf, das während der Rekursion der gleiche Aufruf mehrmals stattfindet. Hier kann man sich mit dem Abspeichern von Zwischenergebnissen behelfen. Dann kann man die Suche stark parallelisieren. Auch hängt es sehr stark von der Implementierung ab, ob man den rekursiven Aufruf naiv oder raffiniert implementiert. Mit solchen Tricks ist es im Jahr 2004 gelungen, ein TSP mit 24.978 schwedischen Städten zu lösen (http://www.tsp.gatech.edu/ http://www.math.princeton.edu/tsp/d15sol/)! Der Weltrekord liegt laut Wikipedia (http://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden) bei der Lösung eines Planungsproblems für das Layout integrierter Schaltkreise mit 85.900 Knoten. Zur Lösung solcher Probleme werden Netze von über hundert Hochleistungscomputern mit einer Gesamtsumme von über 20 Jahren Rechenzeit verwendet. Zum Vergleich: Im Jahr 1977 lag der Rekord noch bei 120 Städten!



Was macht man aber nun, wenn man das Problem in der Praxis (zum Beispiel in einer mittelständischen Reisebüro-Software) lösen will, wo man kein Hochleistungs-Rechnernetz zur Verfügung hat? Die Antwort der praktischen Informatik heißt *Heuristik*. Das Wort „Heuristik“ kommt aus der Seefahrt und bezeichnete früher Verfahren, um seine Position annähernd zu bestimmen. Heute bezeichnen wir damit Näherungsverfahren, die zwar nicht die optimale Lösung, aber eine hinreichend genaue Annäherung liefern. Java-Animationen mit verschiedenen Heuristiken sind zu finden unter (http://web.telia.com/~u85905224/tsp/TSP.htm) und (http://www-e.uni-magdeburg.de/mertens/TSP/index.html).

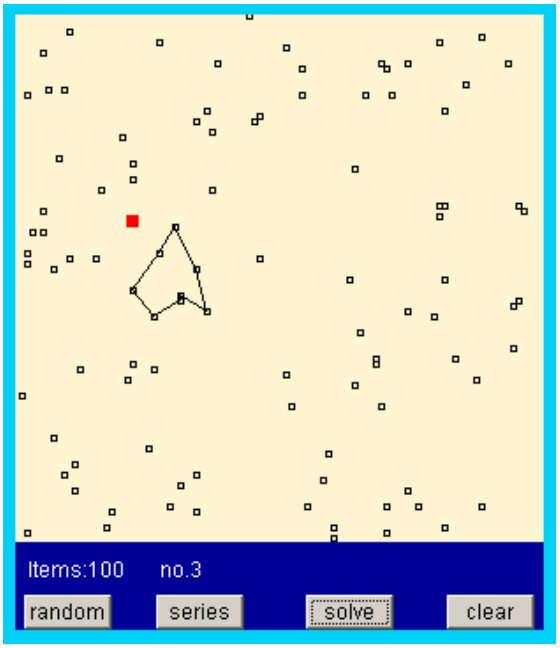
Heuristik 1: nächster Nachbar

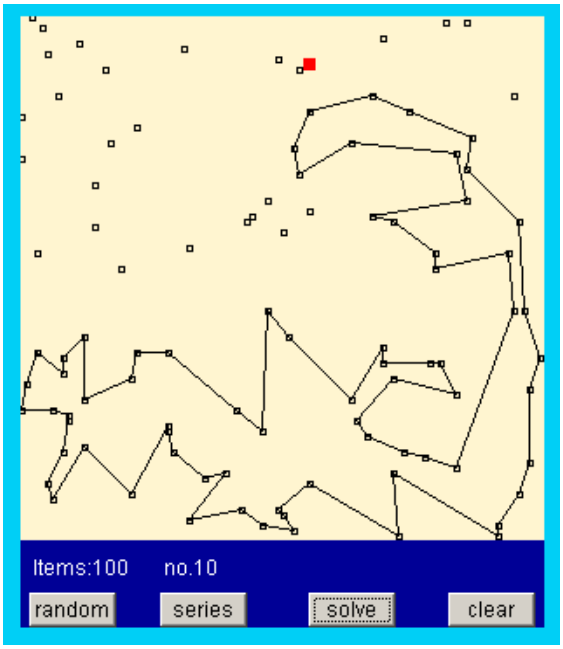
Bei dieser Heuristik sucht der Reisende, ausgehend von einem beliebigen Punkt, zunächst den nächsten Nachbarn des aktuellen Punktes auf der Landkarte. Dann bewegt er sich zu diesem und wendet die Heuristik erneut an. Wenn er alle Kunden besucht hat, fährt er nach Hause zurück. Obwohl diese Heuristik sehr häufig im täglichen Leben angewendet wird, liefert sie schlechte Resultate, da der Heimweg und andere unterwegs „vergessene“ Punkte meist hohe Kosten verursachen.



Heuristik 2: gierige Dreieckstour-Erweiterung

Die Sekretärin wählt zunächst zwei Kunden willkürlich aus und startet mit einer einfachen „Dreieckstour“. Diese wird dann nach und nach erweitert, und zwar wird jede Stadt da in die Tour eingefügt, wo sie am besten „passt“, d.h., wo sie die gegebene Tour am wenigsten erweitert. Solche Strategien nennt man „gierig“, da sie nur auf lokale und Optimierung ausgerichtet sind und das Gesamtergebnis nicht berücksichtigen.



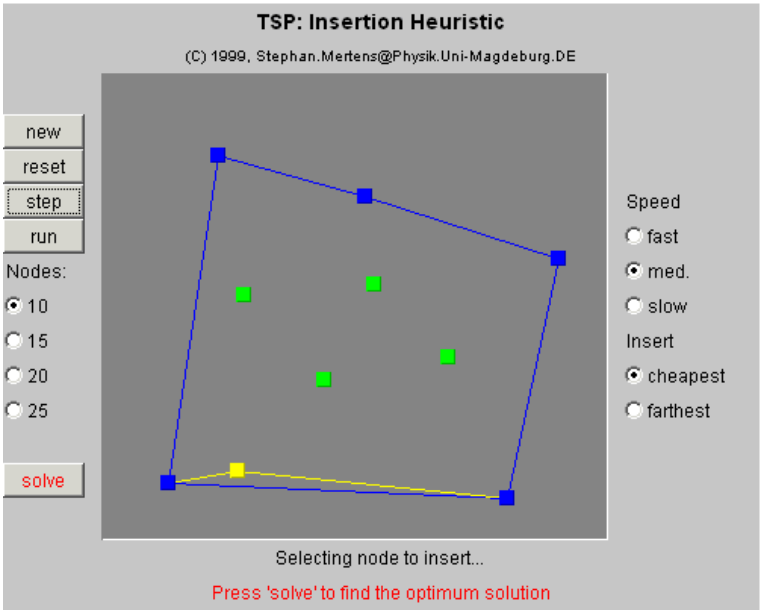


In der geschilderten Form sind noch zwei Zufallselemente enthalten: Die Wahl der ursprünglichen Dreieckstour, und die Reihenfolge, in der die Knoten hinzugefügt werden.

Heuristik 3: Hüllen-Erweiterung

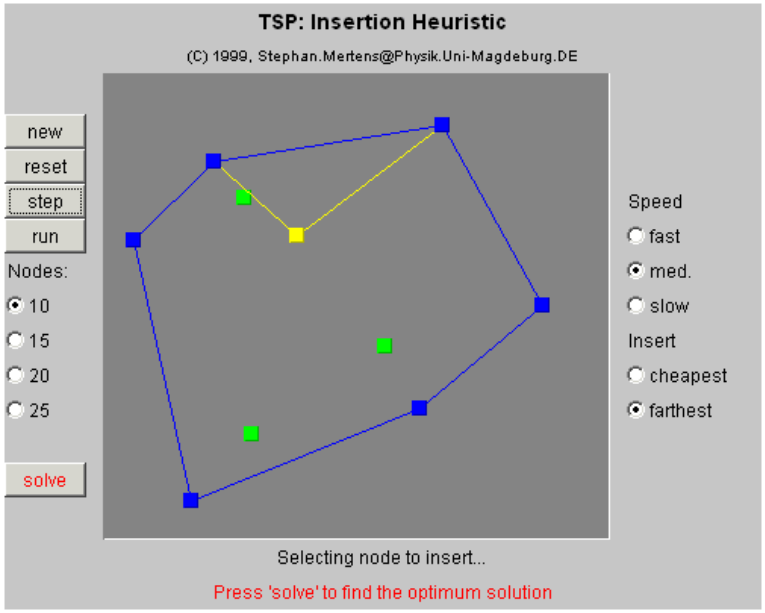
Eine andere Variante dieser Heuristik startet mit der konvexen Hülle aller Punkte (d.h., den Punkten, die am weitesten außen liegen), und fügt der Reihe nach diejenigen Knoten hinzu, die die wenigsten Kosten verursachen. Das heißt, es werden der Reihe nach die Knoten hinzugefügt, die den geringsten Abstand von der bisher konstruierten Tour haben. Das ist

wieder ein „gieriger“ Algorithmus, und zwar in doppelter Hinsicht: bei der Auswahl der Punkte und bei der Bestimmung der Einfügestelle.



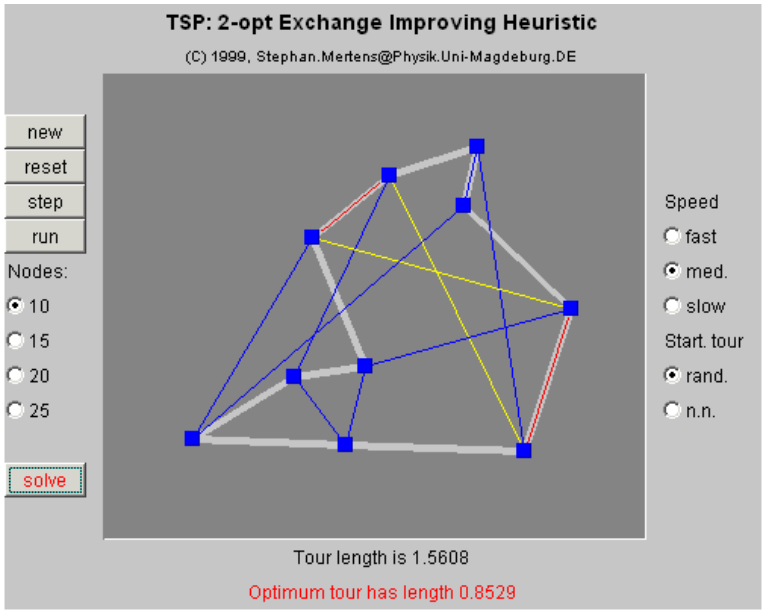
Heuristik 4: inverse Hüllen-Erweiterung

Wieder starten wir mit der konvexen Hülle aller Punkte und fügen der Reihe nach diejenigen Knoten hinzu, die die *meisten* Zusatzkosten verursachen. Das heißt, es werden die Knoten hinzugefügt, die am weitesten von der bisherigen Tour entfernt liegen. Dadurch wird das Gesamtergebnis (die Form der Tour) frühzeitig stabilisiert.



Heuristik 5: Lin-Kernighan

Hier versucht man, eine bestehende Tour zu verbessern, indem man zwei Kantenpaare AB und CD sucht und durch die Kanten AC und BD ersetzt (so genannte *2-opt-Strategie*). Im endgültigen Lin-Kernighan-Verfahren werden nicht nur Kantenpaare, sondern Mengen von Kanten ersetzt.



Wie wir sehen, führen konträre Ideen manchmal beide zu guten Ergebnissen. Es ist sehr schwer, die „Güte“ von Heuristiken abzuschätzen, da die Leistung immer vom verwendeten Beispiel abhängt. Für jede der genannten Heuristiken lassen sich Beispiele konstruieren, so dass das Ergebnis schlecht ist (die doppelte Länge der optimalen Tour oder noch mehr hat). Ist es nicht möglich, eine Heuristik zu finden, die „auf jeden Fall“ ein akzeptables Resultat liefert?

Auch hier hat die praktische Informatik Beiträge zu liefern. S. Arora konstruierte 1996 einen „nahezu linearen“ randomisierten Algorithmus für das euklidische TSP, der für eine beliebige Konstante c eine (1+1/c)-Approximation in $O(n \log(n)^{O(c)})$ Zeit konstruiert (http://www.cs.princeton.edu/~arora/pubs/tsp.ps). Das heißt, z.B. für c=10 bekommt man eine Tour, die höchstens 10% schlechter als die optimale ist, indem wir jeden Knoten durchschnittlich $\log(n)^{10}$ mal betrachten. Wie wir oben gesehen haben, ist log(n) „fast“ eine Konstante (in allen praktischen Fällen kleiner als 5). Daher ist dies „fast“ ein linearer Algorithmus. Der Algorithmus basiert auf raffinierten geometrischen Überlegungen, nämlich einer baumartigen Zerlegung der Ebene in Quadrate und einer Normierung der Schnittkanten der Verbindungslinien zwischen den Punkten.

0.3 Was ist Informatik?

Das Wort „Informatik“ ist ein Kunstwort, welches aus den Bestandteilen „Information“ und „Automatik“ zusammengesetzt ist. Der Begriff „Informatique“ wurde 1962 in Frankreich von P. Dreyfuss geprägt und 1968 vom Forschungsminister Stoltenberg in Berlin bei der Eröffnung einer Tagung übernommen (http://zeitung.informatica-feminale.de/?p=72, http://atrax.uni-muenster.de:8010/Studieren/Scripten/Lippe/geschichte/pdf/Kap1.pdf). Da die Informatik also eine vergleichsweise junge Wissenschaft ist, die eine stürmische Entwicklung durchläuft, gibt es die verschiedensten Definitionen davon, was unter Informatik zu verstehen ist.
Als Beispiel sei hier die Studienordnung der HU von 2003 angeführt:

> (1) Die Informatik erforscht die grundsätzlichen Verfahrensweisen der Informationsverarbeitung und die allgemeinen Methoden der Anwendung solcher Verfahren in den verschiedensten Bereichen. Ihre Aufgabe ist es, durch Abstraktion und Modellbildung von speziellen Gegebenheiten sowohl der technischen Realisierung existierender Datenverarbeitungsanlagen als auch von Besonderheiten spezieller Anwendungen abzusehen und dadurch zu den allgemeinen Gesetzen, die der Informationsverarbeitung zugrunde liegen, vorzustoßen sowie Standardlösungen für Aufgaben der Praxis zu entwickeln. Die Informatik befasst sich deshalb mit
> — der Struktur, der Wirkungsweise, den Fähigkeiten und den Konstruktionsprinzipien von Informations- und Kommunikationssystemen und ihrer technischen Realisierung.
> — Strukturen, Eigenschaften und Beschreibungsmöglichkeiten von Informationen und von Informationsprozessen,
> — Möglichkeiten der Strukturierung, Formalisierung und Mathematisierung von Anwendungsgebieten sowie der Modellbildung und Simulation.
> Dabei spielen Untersuchungen über die Effizienz der Verfahren und über Sinn und Nutzen ihrer Anwendung in der Praxis eine wichtige Rolle.

Andere Fachbereiche haben ähnliche Festlegungen des Studieninhaltes. Als minimaler Konsens für den Begriff „Informatik“ kann dabei die Definition angesehen werden, welche sich aus der Wortbedeutung ergibt:

Informatik ist die Wissenschaft
der automatischen Verarbeitung von Informationen.

(Im Buch von Gumm/Sommer: „Informatik ist die Wissenschaft von der maschinellen Informationsverarbeitung“.) In dieser Definition sind ein paar weitere undefinierte Grundbegriffe enthalten: „Information“, „Verarbeitung“, „automatisch“ oder „maschinell“.

Der Begriff „Information“ ist ein metaphysischer Grundbegriff, mit dem wir uns noch näher beschäftigen werden. An dieser Stelle sei nur bemerkt, dass wir unter „Information“ eine Beschreibung irgendeines Sachverhaltes der uns umgebenden (materiellen oder ideellen) Welt verstehen wollen. Vom Wortstamm her ist eine „Information“ etwas, was *in* eine bestimmte *Form* gebracht worden ist, also auf eine bestimmte Weise *repräsentiert* wird (wenn wir eine Information zur Kenntnis nehmen, bringen wir unseren Verstand in einen bestimmten Zustand). Sehr einfache, wenig strukturierte Informationen bezeichnen wir als *Daten* (daher der altertümliche Begriff „EDV“), eine Menge komplexer Informationen über ein zusammenhängendes Gebiet bezeichnen wir als *Wissen*.
Unter der „Verarbeitung“ von Informationen verstehen wir den Prozess der *Umformung*, d.h. der Veränderung von Informationen aus einer Form in eine andere. Da Informatik sich als Wissenschaft mit der Verarbeitung von Informationen beschäftigt, sind ihr Gegenstand die *Verfahren*, mit denen diese Umformung bewerkstelligt wird: solche Verfahren nennt man

Algorithmen. Häufig erfolgt heutzutage der räumliche Transport von Informationen dadurch, dass sie beim Absender in eine einfachere Form gebracht (*codiert*) werden, durch einen elementaren physikalischen Prozess (elektrische Ströme, Funkwellen etc.) übermittelt und beim Empfänger wieder in die ursprüngliche Form zurückübersetzt (*decodiert*) werden. Daher betrachtet man heute auch die Erforschung von Techniken zur Übertragung von Informationen als Teilgebiet der Informatik.
Der dritte undefinierte Grundbegriff aus der obigen Definition ist „automatisch". Damit soll ausgedrückt werden, dass sich die Informatik nicht mit der Informationsverarbeitung durch Menschen oder andere Lebewesen beschäftigt, sondern durch *Automaten*, d.h. vom Menschen konstruierte Maschinen. Daher gehört zur Informatik auch das Wissen um den Aufbau und die Entwicklung von Technologien zur Konstruktion informationsverarbeitender Geräte. Aus historischen Gründen bezeichnet man diese Geräte oft auch als „*Rechner*" (numerische Berechnungen waren die ersten automatisierten informationsverarbeitenden Prozesse) oder „*Computer*". Daher hat sich im Englischen der Begriff „computer science" für die Informatik durchgesetzt. Dieser Begriff ist allerdings etwas missverständlich, da er suggerieren könnte, dass Informatik die „Wissenschaft von den informationsverarbeitenden Geräten" ist. Einige Leute verwenden daher die Bezeichnung „computing science".

Aus der Bestimmung des Gegenstands der Informatik ergibt sich unmittelbar, dass die Informatik viele Bezüge zu anderen Disziplinen hat: Der Gleichklang zum Wort „Mathematik" ergibt sich nicht von ungefähr. Man kann mit Fug und Recht behaupten, dass die Informatik aus der Mathematik erwachsen ist, so wie die Mathematik ihre Wurzeln in der philosophischen Logik hat. Abgesehen davon, dass die Automatisierung numerischer Berechnungen schon immer ein ureigenstes Interesse der Mathematik war, ist auch die Beschäftigung mit abstrakten Begriffen wie „Berechnungsverfahren" oder „Umformung" ein Gegenstand der Mathematik. Viele Pioniere der Informatik (Pascal, Leibniz, Babbage, Turing, von Neumann, …) waren Mathematiker oder Logiker und haben sich mit den theoretischen Grundlagen automatischer Berechnungsverfahren beschäftigt, bevor es überhaupt Computer gab.
Die zweite Wurzel der Informatik ist die Elektrotechnik. Erst durch den Einsatz elektrischer Schaltungen und Verfahren nach dem zweiten Weltkrieg (durch Zuse, Aiken und andere) wurde gegenüber den davor existierenden mechanischen Rechnern (Hollerith) ein so großer Durchbruch erzielt, dass man über numerische Rechnungen hinausgehen konnte. Da praktisch alle heute existierenden informationsverarbeitenden Prozesse in Automaten auf der Bewegung von elektrischen Ladungen beruhen, ist klar, dass auch heute noch eine enge Verwandschaft zwischen Informatik und Elektrotechnik besteht. Auf der anderen Seite gehören Computer heute mit zu den wichtigsten strombetriebenen Geräten, weshalb sich die Elektrotechnik heute gerne auch als „Informationstechnik" bezeichnet,
Durch die Inhalte der Informatik ergeben sich weiterhin eine ganze Reihe von Querbezügen zu anderen Disziplinen. Wenn Informationsverarbeitung zur Steuerung mechanischer Geräte, etwa von Robotern oder Fertigungsstraßen, genützt wird, müssen Informatiker mit Maschinenbauern und Produktionstechnikern zusammenarbeiten. Durch den Einsatz von Computern zur Übertragung von Informationen per Funkwellen ist eine enge Beziehung zur Nachrichtentechnik gegeben. Wegen der Notwendigkeit der Interaktion von Automaten und Menschen (auf Benutzungs- und Konstruktionsebene) muss die Informatik auf Grundlagen und Ergebnisse der Psychologie, Linguistik, Kommunikationswissenschaften und anderer Fächer zurückgreifen. Da in fast allen Fächern informationsverarbeitende Prozesse vorkommen, die bislang entweder von Menschen durchgeführt wurden oder wegen des hohen Arbeitsaufwandes gar nicht durchgeführt werden konnten, werden Methoden der Informatik in diesen Fächern für die Automatisierung der Verarbeitung von Informationen angewendet.

Dadurch haben sich eine Reihe spezialisierter Studiengänge gebildet, die so genannten Bindestrich-Informatiken, die die Anwendung der Informatik in anderen Fächern betonen. Zu nennen sind hier Wirtschafts-Informatik, Bio-Informatik, Medien-Informatik, Geo-Informatik, Umwelt-, Rechts- oder Medizininformatik, und viele mehr. Wichtig: Mit einer Ausbildung als Informatiker (ohne Bindestrich) kann man sich später für jede dieser Disziplinen weiter qualifizieren!

Aus den geschilderten Wurzeln ergibt sich die heute übliche Struktur der Informatik: Man teilt sie ein in

- theoretische Informatik
- praktische Informatik
- technische Informatik, und
- angewandte Informatik.

Die theoretische Informatik beschäftigt sich mit den *formalen Grundlagen*, die praktische Informatik mit den *Verfahren*, die technische Informatik mit den *Maschinen* zur Verarbeitung von Informationen. In der angewandten Informatik werden die *Anwendungen* der Informationsverarbeitung für andere Fächer (z.B. Robotik, Bioinformatik, medizinische Bildverarbeitung) untersucht. Oftmals wird die angewandte Informatik als Teil der praktischen Informatik betrachtet; an einigen Universitäten studiert man im Grundstudium zunächst ein beliebiges Nebenfach und spezialisiert sich dann im Hauptstudium auf die angewandte Informatik in diesem Nebenfach.

Geschichte der Informatik

Informatik im eigentlichen Sinne gibt es erst seit dem Endes des zweiten Weltkrieges.
Die Wurzeln der Informatik reichen dagegen bis ins Mittelalter bzw. ins Altertum zurück:

- ***300 v. Chr:*** Euklid entwickelt sein Verfahren zur Bestimmung des größten gemeinsamen Teilers (ggT)
- ***um 820:*** Al-Chwarizmi fasst in einem Buch Lösungen zu bekannten mathematischen Problemen zusammen.
- ***1524:*** A. Riese veröffentlicht ein Buch über die Grundrechenarten
- ***17-18.Jh.:*** G. W. Leibniz (1646-1714) entwickelt das Dualsystem (1679) und baut eine Rechenmaschine (1673/1694), Pascal, Schickard u.a. entwickeln ebenfalls mechanische Rechenmaschinen, Babbage konzipiert „difference engine“, Ada Lovelace die erste Programmiersprache dafür
- ***Ende 19./ Anf. 20. Jh.:*** Formalisierung der logischen und mathematischen Grundlagen durch Frege (“Begriffschrift”, 1879), Russell u. Whitehead (“Prinicipia mathematica", 1910-13), Peano u.a.
- ***Ende 19./ Mitte 20. Jh.:*** Perfektionierung mechanischer Rechenmaschinen;
- ***1930-40:*** Theorie der Berechenbarkeit, Vollständigkeits- und Entscheid-barkeitssätze (Gödel, Turing, Tarski, Church, Kleene, Post, Markov u.a.)
- ***1930-40:*** erste elektromechanische Computer: Zuses Z1 (1936), Z3 (1943), Aikens Mark1 (1944), Eckert+Mauchlys ENIAC (1946)
- ***1948-49:*** Konrad Zuse entwickelt seinen “Plankalkül”, C. Shannon seine “Informationstheorie”, J. v. Neumann entwickelt den nach ihm benannten Rechnertyp: Daten und Befehle werden gemeinsam im Rechner gespeichert und ähnlich behandelt.
- ***1955:*** Erfindung des Transistors
- ***1959-60:*** erste “höhere Programmiersprachen”: J. McCarthy entwickelt die funktionale Programmiersprache LISP und begründet die “Artificial Intelligence”, Fortran (Masch.bau), Cobol (BWL) und Algol (Math) werden definiert.
- ***1969-70:*** Entwicklung universaler Programmiersprachen wie Algol68 und PL/I
- ***ab 1970:*** Informatikstudium in Deutschland
- ***ab 1980:*** Objektorientierte Sprachen und Systeme
- ***ab 1990:*** Internet (Gopher, Mosaic), Mobilfunk (1991 D-Netz, 1994 E-Netz), WLAN (ca. 1995), PDAs (1997), Java (1995), Java2 (2002)

Die Informatik ist heute in fast alle Aspekte unseres Lebens vorgedrungen:

- Einen Großteil ihres Studiums werden Sie mit „e-Learning“ verbringen, die notwendigen Informationen beschaffen Sie sich im Internet. Vielleicht bestellen Sie hier auch Waren oder vergleichen zumindest bei eBay die Preise.
- Dokumente (Briefe, Steuererklärungen usw.) verfassen Sie natürlich am Computer; mit Ihren Kommilitonen nebenan und dem Onkel in Amerika tauschen Sie e-Mails aus, die den Empfänger in wenigen Sekunden erreichen.
- Wahrscheinlich haben Sie auch ein Mobiltelefon, vielleicht sogar ein Notebook mit WLAN, oder einen elektronischen Organizer.
- Ihr Bankkonto wird von einem Computer geführt, Bargeld holen Sie am Geldautomaten, vielleicht habe Sie auch eine elektronische Brieftasche.
- Wenn Sie mit dem Auto nach Hause fahren, begleiten Sie bis zu 80 eingebaute Steuergeräte, das Auto ist weitgehend von Robotern gebaut worden. Sogar die Schnittmuster Ihrer Kleidung wurden vom Computer optimiert.
- Ihre Armbanduhr, Foto- oder Filmapparat, Ton- und Bildwiedergabegeräte sind schon längst nicht mehr mechanisch, ganz zu schweigen von der Türschließanlage, Fahrstuhlsteuerung, Kühlschrank, Mikrowelle, Waschmaschine, und anderen Geräten zu Hause.

Das ist aber keinesfalls das Ende der Entwicklung. In ein paar Jahren wird Sie wahrscheinlich die Türschließanlage an Ihrem Aussehen und Fingerabdruck erkennen, Sie werden sich vielleicht wie im Roman „per Anhalter durch die Galaxis“ mit dem Fahrstuhl unterhalten, der Kühlschrank könnte Vorräte selbsttätig nachbestellen, der Herd sich Rezepte aus dem Internet holen und die Waschmaschine wissen, wie heiß die Wäsche gewaschen werden muss.

Alle diese „Wunder der Technik“ werden möglich durch systematische Vorschriften für die Verarbeitung von Informationen (Algorithmen, Programme) und Maschinen, die diese Vorschriften ausführen können (Computer, Prozessoren). Natürlich können wir uns in einer Vorlesung nicht mit allen oben genannten Anwendungen beschäftigen, aber die zentralen Gesichtspunkte die in allen gleichermaßen vorhanden sind, bilden den Gegenstand der Vorlesung: Algorithmen und ihre Ausführung auf Rechenanlagen.

Das zentrale Ziel der Vorlesung ist es, eine „algorithmische Denkweise“ zu vermitteln: Ein Verständnis dafür, wann und wie ein (informationsbezogenes) Problem mit welchem Aufwand durch eine Maschine gelöst werden kann. Inhaltlich gibt die Vorlesung einen Überblick über das Gebiet der praktischen Informatik. Dazu gehören unter anderem folgende Themen:

- Repräsentation von Informationen in Rechenanlagen
- programmiersprachliche Konzepte
- Methoden der Softwareentwicklung
- Algorithmen und Datenstrukturen
- Korrektheit und Komplexität von Programmen

In den weiteren Vorlesungen des Bachelor-Studiums werden diese Themen ergänzt und vertieft.