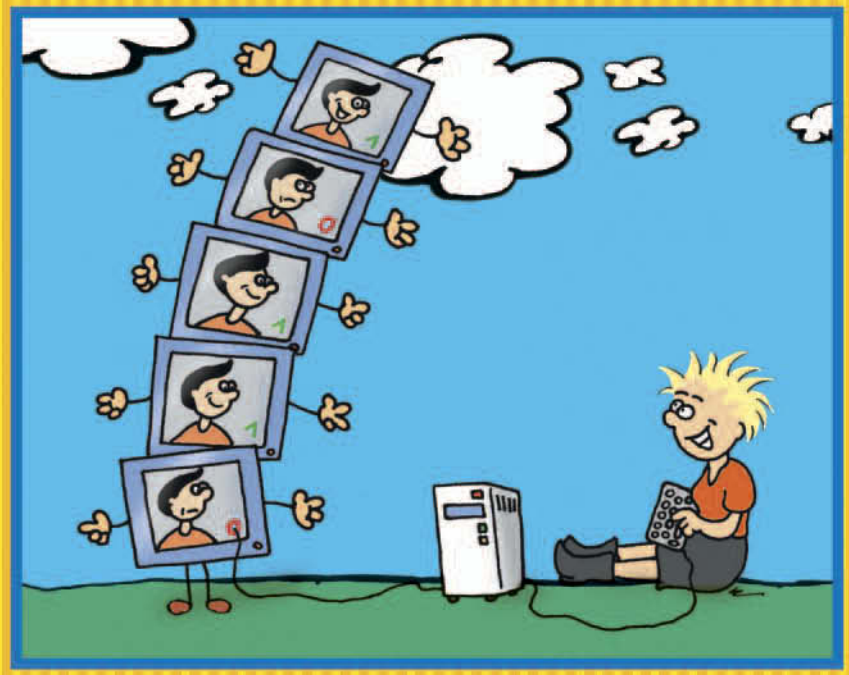
Johannes Magenheim u.a. illustriert von Thomas Müller

INFORMATIK
macchiato

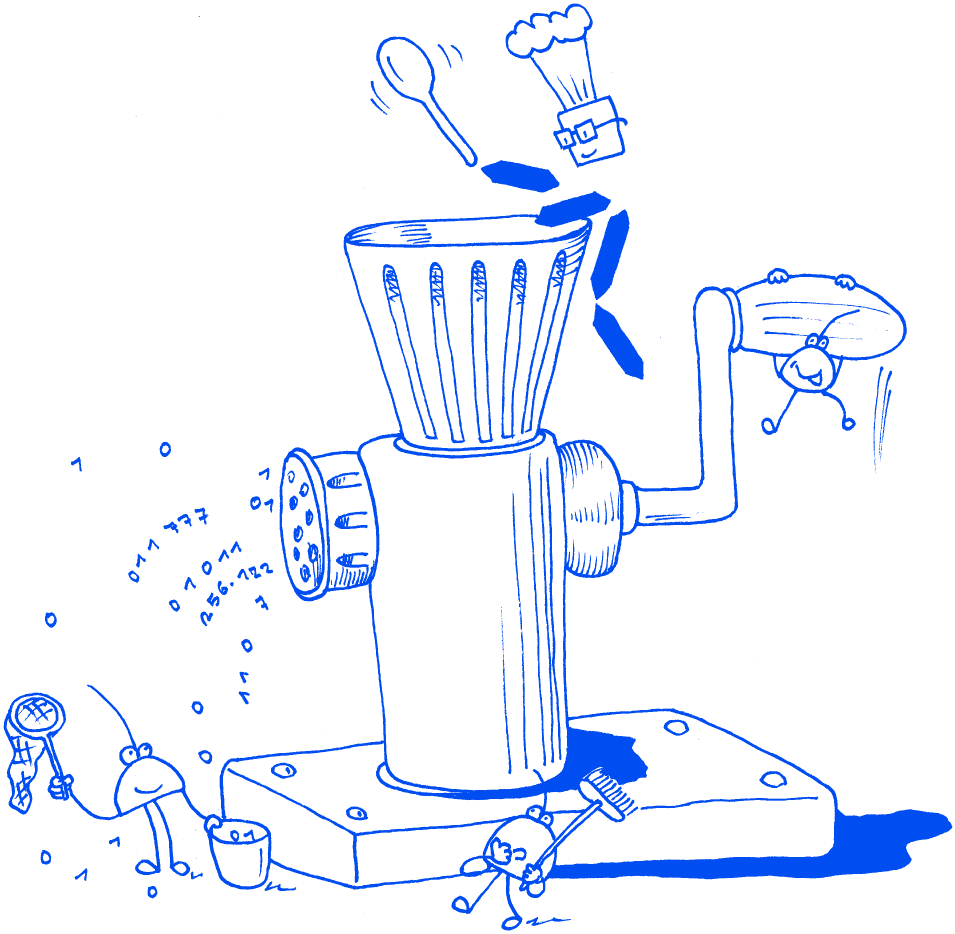


Cartoon-Informatikkurs für Schüler und Studenten

PEARSON
Studium

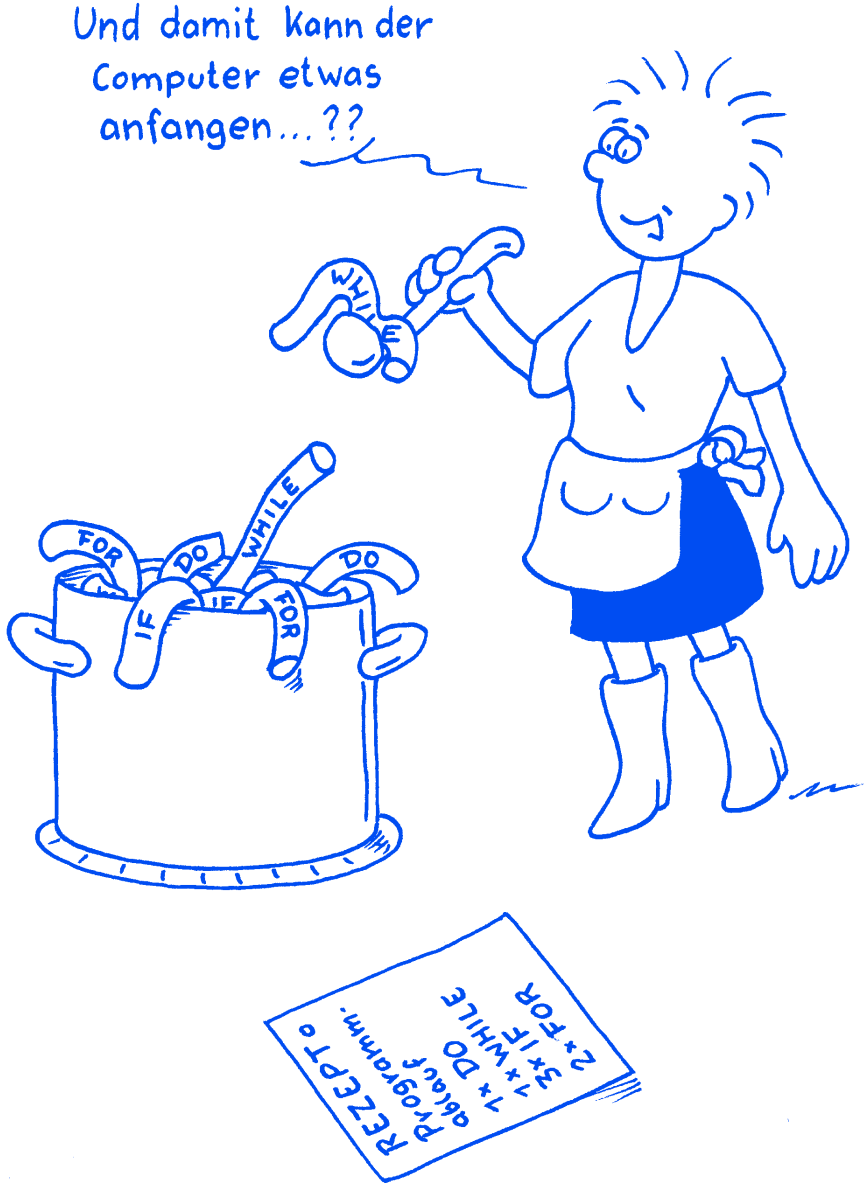
KAPITEL III

DAS INFORMATIK-KOCHSTUDIO



Das Informatik-Kochstudio

Algorithmen



Sortieren

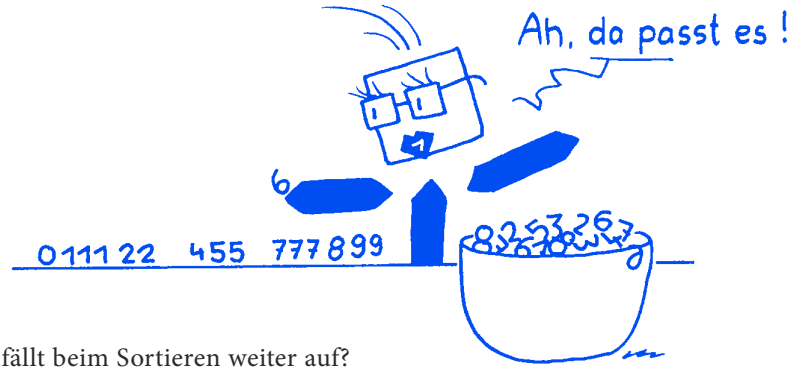
Ordnung ist das halbe Leben. Mit diesem Spruch nerven seit Generationen Eltern ihre Kinder. Aber wer möchte schon sein ganzes Leben damit zubringen, Ordnung zu halten? Deshalb verwenden wir den Computer und übertragen ihm die Aufgabe, **Ordnung** zu halten.

„Wer Ordnung hält, ist zu faul zum Suchen." Soll man besser alles unsortiert irgendwo ablegen und muss dann länger suchen oder soll man lieber Ordnung halten und findet dann alles schneller? Normalerweise versucht man, das Chaos erst gar nicht entstehen zu lassen und lieber jedes Objekt an den richtigen Platz zu legen. Aber das ist nicht immer so einfach: Post kommt meist nicht sortiert oder die Bücher im Regal kann man nach Autoren oder Titel sortieren. Und dieses Alltagsproblem tritt im Computer aufgrund der vielen gespeicherten Dokumente noch verstärkt auf. Man denke nur an die unüberschaubare Zahl von Internetseiten, die die Suchmaschine **Google** alle durchsuchen und sortieren muss, damit sie schnell die Ergebnisse ausspucken kann …

Deshalb müssen auch Computer Daten **sortieren** und in den sortierten Daten möglichst schnell etwas finden. Dabei hilft ihnen nicht immer ihre große Geschwindigkeit beim Erledigen solcher Aufgaben, denn auch die Datenmengen werden immer größer.

Der Computer kann nach den gleichen Prinzipien sortieren wie der Mensch, denn wir sortieren in vielen Situationen und können dem Computer das beibringen. Aber wie sortiert denn der Mensch, etwa die Spielkarten bei einem Kartenspiel?

Zunächst einigt man sich auf ein Spiel und damit auf die natürliche Reihenfolge der Karten, die von den Spielregeln des Spiels abhängig ist. Man nimmt die Karten einzeln auf die Hand und sortiert sie in die auf der Hand gehaltenen Karten ein. Der Mensch kann die Karten dabei sehr gut überblicken. Der Computer kann dagegen immer nur einzelne Kartenpaare anschauen und vergleichen. Das muss berücksichtigt werden, wenn man ihm Anweisungen zum Sortieren gibt.



Was fällt beim Sortieren weiter auf?

- Die Dauer des Einsortierens hängt von der Anzahl der Karten ab. Je mehr Karten, desto mehr Zeit erfordert das Einsortieren. Die Kartenzahl beschreibt die sogenannte **Problemgröße**.
- Je mehr Karten man auf der Hand hat, desto aufwändiger wird das Einsortieren. Es ist aber meist nicht so, dass doppelt so viele Karten doppelten **Aufwand** bedeuten würden. Der Aufwand wächst jedoch meistens überproportional mit der Problemgröße.
- Das Sortieren der Karten wird auf ein kleineres leichter handhabbares Problem zurückgeführt, nämlich das Einsortieren einer Karte. Dieses Prinzip heißt „**Teile und herrsche**“. Eine Aufgabe wird in kleinere

Teilaufgaben zerlegt, die getrennt voneinander gelöst werden können. Dieses Prinzip hat sich auch bei der Beschreibung von komplizierten Aufgabenstellungen bewährt.

Aber wie aufwändig ist denn nun das Verfahren? Wovon kann das alles abhängen? Es spielt nicht nur die Zahl der Karten eine Rolle, sondern auch, ob die Karten, die man aufnimmt, gut kommen oder nicht. Was heißt aber gut kommen?

- **worst case**: Jede Karte, die man aufnimmt, muss mit allen vorherigen Karten verglichen werden, bevor man sie einsortieren kann. Das kann passieren, wenn die Karten in besonders ungünstiger Reihenfolge kommen, etwa wenn ich immer von der falsche Seite mit dem Vergleichen beginne.
- **best case**: Man findet die passende Stelle zum Einfügen sofort und muss eigentlich immer nur jede aufgenommene Karte mit einer schon sortierten Karte vergleichen.
- **average case**: Die beiden vorigen Situationen treten eigentlich nie auf. Wenn man lange genug beliebige Karten sortiert, die in zufälliger Reihenfolge liegen, ergibt sich ein guter Mittelwert für den **Aufwand**.

Warum betrachten wir nicht immer nur den „average case“? Das passt doch immer im Mittel! Manchmal möchte man aber sicher sagen können, wie lange ein Sortierverfahren längstens dauert. Etwa wenn die Jahresabrechnung einer Bank mit Millionen Konten läuft, kann nicht gebucht werden, deshalb sollte der Prozess möglichst schnell abgeschlossen sein. Aber häufig hat man schon gut vorsortierte Daten und muss nur noch einige weitere Daten einsortieren. Hier greift der „best case“.

Aber wie formuliert man denn jetzt den Sortieralgorithmus für den Computer?

Zunächst gibt es bestimmte Handlungsfolgen, die nacheinander ablaufen. Dies wird in klaren **Anweisungen** formuliert wie etwa die Anweisungen eines Navigationssystems. Um eine Handlungsfolge zu beschreiben, werden die Anweisungen einfach wie bei einem Rezept hintereinander aufgeschrieben.

Es gibt bestimmte Handlungen, die mehrmals wiederholt werden müssen. Dabei muss man aber auch irgendwann fertig werden. Deshalb braucht man auch eine Bedingung, wann die **Wiederholung** abgeschlossen ist. Beim Sortieren ist das der Fall, wenn der Spieler alle für ihn bestimmten Karten auf der Hand einsortiert hat. Man formuliert hierfür eine Wiederholungsanweisung mit Abbruchbedingung, die zu Beginn oder am Ende der Wiederholung geprüft werden muss.

Beim Einsortieren einer Karte muss immer eine Entscheidung getroffen werden, ob die Karte weiter links oder weiter rechts von der gerade betrachteten Vergleichskarte auf der Hand einsortiert werden muss. Hierbei handelt es sich um eine **bedingte Anweisung**, da der weitere Ablauf von einer Bedingung abhängt.

Ein Algorithmus besteht aus einer Folge von Anweisungen, Wiederholungen und Entscheidungen.

Nun können wir den Algorithmus „**Sortieren durch Einfügen**“ wie folgt formulieren:

Sortieren

- Solange eine Karte für den Spieler auf dem Tisch liegt
  - Nimm die einzusortierende Karte auf
  - Setze die Handkarte auf die linke Karte
  - Solange die Handkarte < einzusortierende Karte
    - Nimm die nächste Karte in die Reihe
  - Schiebe die einzusortierende Karte in die gefundene Lücke

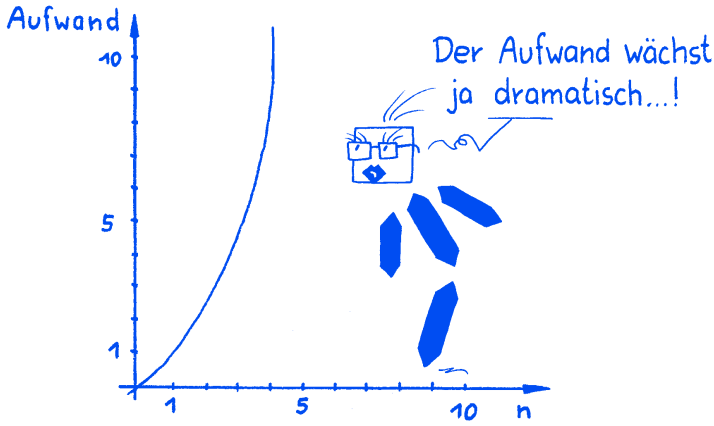
Was ist denn nun anders an der Beschreibung des Algorithmus für den Computer und den Menschen?

Ein **Algorithmus** hat folgende Eigenschaften:

- Durch die Beschreibung ist das Vorgehen stark schematisiert. Jeder Handlungsschritt ist genau vorgeschrieben. Die Vorschrift ist **eindeutig** und **ausführbar**.
- Er funktioniert bei beliebiger Reihenfolge der aufgenommenen Karten und ist deshalb **allgemein**.
- Es gibt nur endlich viele Handlungsanweisungen für die Beschreibung, d.h., er ist **endlich**.
- Die Abarbeitung wird nach endlich vielen Handlungsschritten beendet. Jede Karte benötigt eine bestimmte Anzahl von Schritten zum Einsortieren, d.h., alle Karten sind nach einer bestimmten Zeit einsortiert, die Sortieraufgabe ist damit beendet. Der Algorithmus **terminiert**.

Nun ist das Problem des Sortierens gelöst. Doch wie sieht das mit dem **Aufwand** aus? Bei einer Karte muss man gar nicht vergleichen. Beim Einsortieren der zweiten Karte ist ein Vergleich notwendig, eine weitere Karte braucht im schlimmsten Fall den Vergleich mit zwei weiteren Karten. Im „**worst case**“ muss man beim Einsortieren alle schon sortierten Karten durchgehen und vergleichen. Daraus erhalten wir eine Tabelle.

Anzahl Karten	Aufwand der Vergleiche (worst case)
1	0
2	1
3	3
4	6
5	10
...	
10	45
100	4950
1000	499500
10000	4999500
n	$\frac{1}{2}n^2 - \frac{1}{2}n$



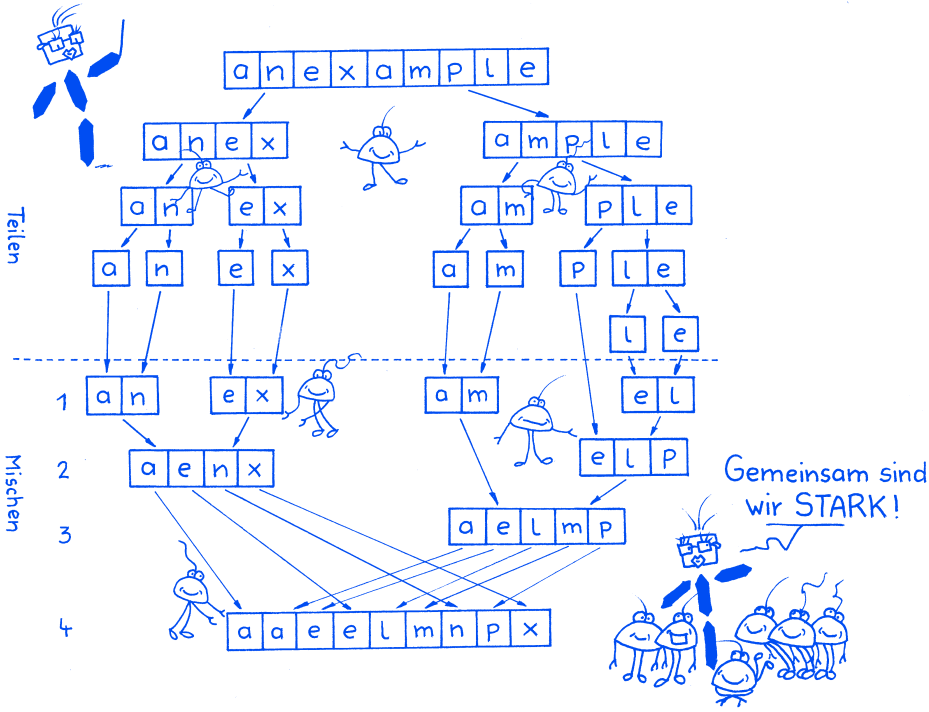
Der Aufwand steigt dramatisch an. Wenn die Zahl der Karten verzehnfacht wird, steigt der Aufwand um den Faktor 100!

Effektives Sortieren

Und geht es vielleicht doch noch etwas schneller? Vielleicht gibt es ja einen anderen Algorithmus, der effizienter ist!

Dazu müssen wir das Prinzip „**Teile und herrsche**“ einsetzen:

Zunächst werden die zu sortierenden Elemente in kleinste Einheiten, etwa einzelne Bücher oder Karten oder Zahlen, zerlegt. Dann beginnt der eigentliche Sortierteil: Die einzelnen Elemente werden zu Zweiergruppen zusammensortiert, danach zu Vierergruppen etc. Beim Zusammensortieren, man sagt auch **Mischen**, benötigen wir immer so viele Vergleiche wie die Anzahl der Elemente. Bei jedem Vergleich der beiden kleinsten Elemente der Gruppen kann man ein Element in die neue Gruppe hineinlegen, also kein großer Aufwand!



Die Zahl der Vergleiche ist zwar immer noch hoch, aber wir benötigen nicht mehr n^2 Vergleiche. Das liegt daran, dass wir nicht so viele Mischvorgänge durchführen. Man braucht für die 9 Buchstaben im Beispiel oben nur 4 Mischvorgänge, also doch eine echte Ersparnis! Aber das wird erst bei vielen Elementen richtig interessant: Man benötigt z.B. beim Telefonbuch von Berlin mit ungefähr 1 Million Einträgen nur 20 Mischvorgänge – wenn das nicht effektiv ist!

Beim **Sortieren durch Mischen** benötigten wir durch das Prinzip „Teile und herrsche“ weniger Zeit zum Zusammenmischen als beim Sortieren durch Einfügen,

denn das Zusammenmischen von kleinen, bereits vorsortierten Teilmengen geht schneller, da wir nur wenige solcher Mischvorgänge benötigen.

Wir haben nun zwei Sortieralgorithmen kennengelernt. Es gibt aber nicht nur Sortieralgorithmen. Man kann Algorithmen auch für viele andere Probleme formulieren.

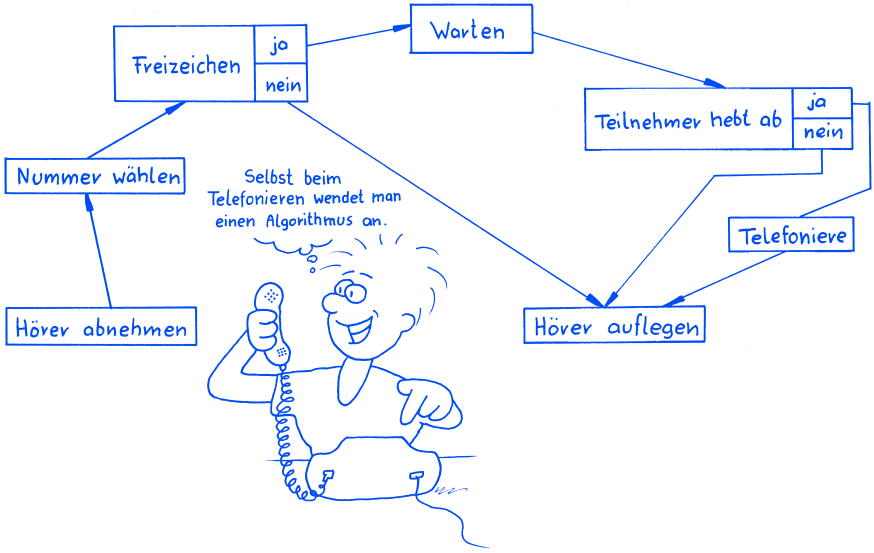
Ein **Algorithmus** ist eine Folge **eindeutiger** und **ausführbarer** Anweisungen zur Herleitung bestimmter Ausgabedaten aus gewissen Eingabedaten mit endlicher Ausführungslänge.

Alltagsalgorithmen

Und welche Bedeutung hat ein Algorithmus für das tägliche Leben?

Es gibt im täglichen Leben viele Vorgänge, die nach einem Schema ablaufen, vielleicht infolge von Tradition, Gewohnheit oder aufgrund von Verhaltens- oder Rechtsregeln. Diese Vorgänge ähneln einem Algorithmus, auch wenn er selten so formuliert ist. Im Alltag werden diese jedoch nicht formal aufgeschrieben, da sie teilweise recht unpräzise sind. Beispiele für **Alltagsalgorithmen** sind etwa

- Herstellen einer Telefonverbindung
- Backen eines Kuchens
- Autofahren
- Putzen der Zähne
- Spielen einer Melodie
- Aufbau eines Regals



Komplexe Algorithmen

Aber nicht alle Algorithmen sind so einfach wie der zum Telefonieren. Wir wollen hier als Beispiel eines komplexen Algorithmus die Suche nach dem kürzesten Weg zwischen zwei Orten kennenlernen. Heute haben fast alle Autofahrer ein **Navigationsgerät** im Auto. Und wer kein „Navi" im Auto hat, sucht sich den kürzesten Weg mit Hilfe eines Routenplaners im Internet oder von CD. Dieses Problem des kürzesten Wegs hatten auch schon unsere Eltern. Und wie haben sie das früher gemacht?

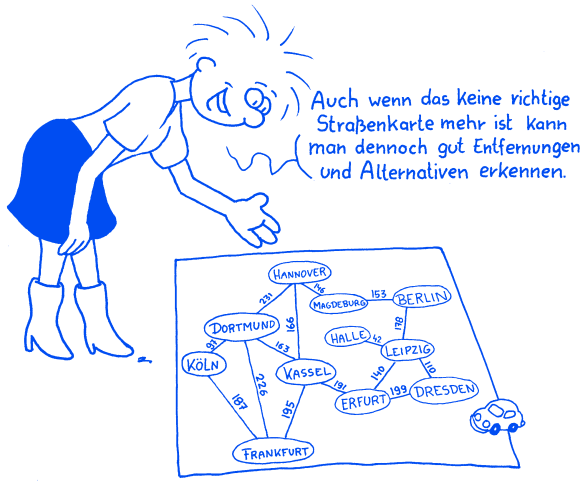
Früher nahm man einen Straßenatlas oder einen Stadtplan und versuchte durch „Draufschauen" die kürzeste Route etwa von Köln nach Leipzig zu finden. Das ist nicht immer ganz einfach. Wie soll einem dabei ein Algorithmus helfen? Der Computer kann sich doch nicht durch einfaches „Draufschauen" einen Überblick verschaffen. Da müssen wir schon etwas genauer beschreiben, wie der Computer den optimalen Weg findet.

Was heißt eigentlich **optimal**? Soll es möglichst schnell gehen oder möchte man wegen der hohen Spritpreise lieber die kürzeste Route haben? Genau wie beim Einsortieren der Karten in Abhängigkeit von der Spielregel müssen wir uns auch hier auf ein Kriterium einigen: Entweder gibt man die Straßenlänge oder, was seltener ist, Fahrzeiten auf der Karte an. Bei den Routenplanern können wir das meistens in der Eingabemaske angeben.

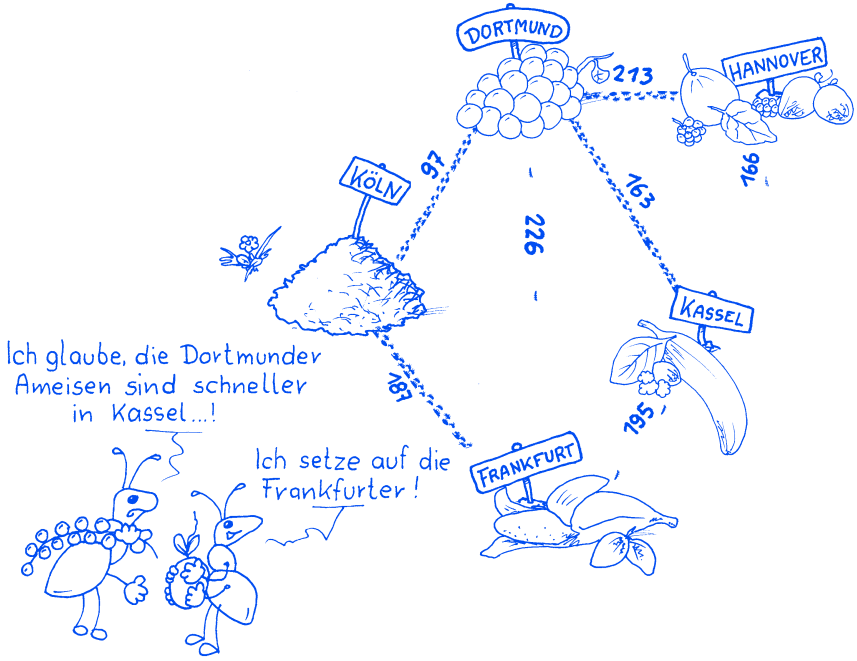
Und wie geht es dann weiter?

Zunächst benötigen wir keine maßstäbliche Karte, da wir die Entfernungen als Zahlen notieren und nicht messen müssen. Für das Suchen der Lösung wird das Problem abstrahiert. Überflüssige Informationen werden weggelassen und wir können uns auf das Wesentliche der Aufgabe konzentrieren. **Abstraktion** ist also eine wichtige Methode zur Problemlösung und begegnet uns auch alltäglich immer wieder.

Ein Modell ist ein abstraktes Abbild der Wirklichkeit, das wesentliche Aspekte enthält, jedoch aufgrund der Reduktion leichter zu untersuchen ist.



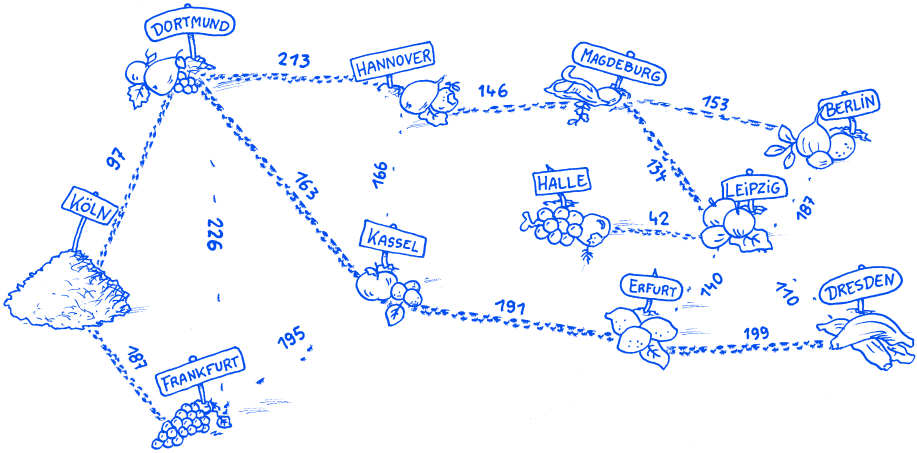
Für die Fahrt von Köln nach Leipzig betrachten wir ein vereinfachtes Straßennetz. Wir starten auf diesem Autobahnnetz von Köln und haben zwei alternative Orte, die wir ansteuern können. Zur Lösung nehmen wir uns nun Anschauungsmaterial aus der Natur zu Hilfe: Ameisen schwärmen auch auf festgelegten Ameisenstraßen, die durch Pheromone markiert sind, aus ihrem Bau aus, um bestimmte Nahrungsquellen zu erreichen. Dabei sind alle Ameisen etwa gleich schnell und sie können gleichzeitig in verschiedene Richtungen auf unterschiedlichen Straßen krabbeln. Sie müssen sich also zunächst nicht entscheiden, sondern probieren alle Straßen gleichzeitig aus!



Der erste Ort, den wir von Köln erreichen, ist Dortmund nach 97 km. Die Ameisen krabbeln weiter von Dortmund Richtung Hannover, Kassel und Frankfurt und parallel von Köln nach Frankfurt. Das zweite Ziel ist nach 187 km Frankfurt. Die Ameisen aus Frankfurt müssen jetzt nicht mehr nach Dortmund, denn dies ist ja schon besucht worden, es geht also der „Run" nach Kassel los. Hier gewinnen die Ameisen aus Dortmund, die Frankfurter können also umkehren, da sie keine neuen Orte besuchen können. Somit transportieren die Ameisen aus Dortmund die Nahrung aus Kassel zum Bau, da dieser Weg ja kürzer ist.

Die Ameisen teilen sich bei jeder neu besuchten Stadt auf und gehen die Wege zu noch nicht besuchten Städten, bis alle Städte besucht sind und sich daraus die jeweils kürzeste Ameisenstraße zu dieser Stadt etabliert hat.

Der Weg nach Leipzig lässt sich nun aus den besuchten Ameisenstraßen ablesen: Es gibt immer genau einen Weg von Köln nach Leipzig, der auch der kürzeste ist, denn die Alternativwege etwa über Erfurt waren länger, da die Ameisen mehr Zeit benötigt haben. Durch Addition erhält man die Länge des Wegs von Köln nach Leipzig: Es sind 591 km!

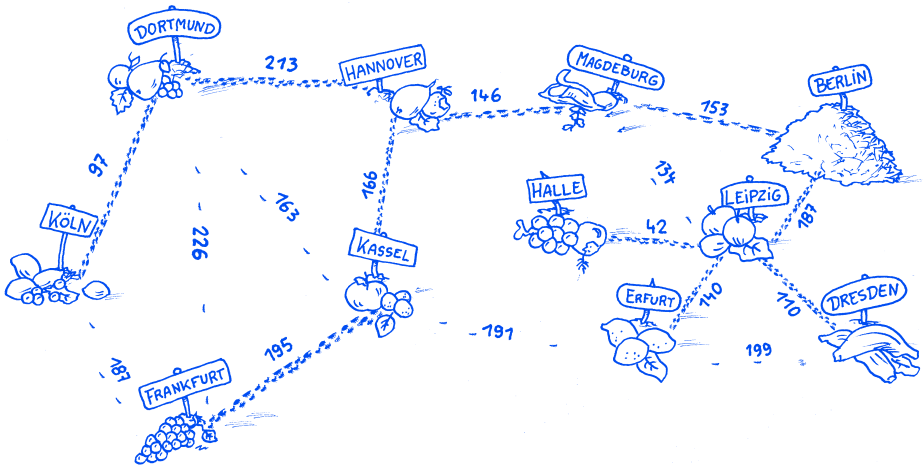


Das Problem ist also gelöst und wir können eine Entfernungstabelle aufstellen.

Für eine vollständige **Entfernungstabelle** müssen wir jedoch den Algorithmus in jeder Stadt erneut starten, um so ein neues Wegenetz für die kürzesten Wege zu dieser Stadt zu erhalten. Erst dadurch entsteht eine vollständige Entfernungstabelle. So ist es beispielsweise sicher nicht sinnvoll, auf dem bisherigen Wegenetz von Berlin nach Dresden über Dortmund und Kassel zu fahren!

	Köln
Berlin	609
Dortmund	97
Dresden	650
Erfurt	451
Frankfurt	187
Halle	632
Hannover	310
Köln	0
Leipzig	591
Magdeburg	456
Paderborn	202

Na, jetzt sollte man aber spätestens doch mal den Computer bemühen. Aber dazu müssen wir aus dem anschaulichen Verfahren einen **Algorithmus** für den kürzesten Weg formulieren. Dann geht das Ganze sicher auch für mehr Städte und der Algorithmus für ein Navigationsgerät ist fertig!



Und wie aufwändig ist dieser nach seinem Entdecker genannte **Dijkstra-Algorithmus**?

Bei jedem Schritt wird eine neue Stadt durch die Ameisen erreicht. Dabei muss man alle Städte untersuchen, die besucht werden können. Dies sind jedoch normalerweise ein paar wenige, im schlimmsten Fall alle noch nicht besuchten Städte. Im Durchschnitt wird man also in einer festen Zeit eine neue Stadt hinzunehmen können, deshalb wächst der Aufwand im Durchschnitt mit der Anzahl der Städte. Und man erhält eine vollständige Entfernungstabelle von seiner Heimatstadt. Also ist das Ganze sehr effektiv!

Aber wie schreibt man den Dijkstra-Algorithmus für den Computer auf? Und wie sieht nun die komplette Entfernungstabelle für alle Städteverbindungen aus?

Dazu gibt es weitere Infos auf der Webseite.

Erkenntnisse dieses Kapitels

- **Algorithmen** werden von Informatikern entwickelt, um Computern bestimmte Aufgaben beizubringen.
- Um dem Computer etwas Neues beizubringen, muss man die Wirklichkeit manchmal **abstrahieren** und **formalisieren**.
- Algorithmen setzen sich aus **eindeutigen**, **endlichen** und **ausführbaren Anweisungen** zusammen. Sie bestehen zum größten Teil aus **Sequenzen**, **Wiederholungen** und **Bedingungen**.
- Der **Aufwand** ist ein Qualitätsmaßstab und gibt an, wie lang ein Algorithmus zur Ermittlung des Ergebnisses bei einer bestimmen Problemgröße braucht. Er kann nicht nur von der Problemgröße, sondern auch von den zu bearbeitenden Daten abhängig sein.
- Viele Probleme lassen sich in kleinere Probleme zerlegen. Dieses Prinzip **„Teile und herrsche“** führt bei der Formalisierung schnell zu brauchbaren Algorithmen.