Digitalisierung und Programmierung

Vorlesungsskript

Prof. Dr. Nicolas Meseth

15. Februar 2025

Inhaltsverzeichnis

1 Vorwort

2 Problemlösung

2.1 Warum nutzen wir Computer?

Der wichtigste Grund für die Nutzung von Computern ist das Lösen von Problemen. Warum? Weil Computer zwei Eigenschaften besitzen, die für viele Probleme und deren Lösung vorteilhaft sind:

1. Computer machen keine Fehler. Wenn wir einem Computer einen Lösungsweg beibringen, wendet er ihn fehlerfrei auf neue Probleme an.
2. Computer sind unglaublich schnell. Ob einfache Schritte, komplexe Berechnungen oder die Verarbeitung großer Datenmengen – Computer lösen Probleme in einem Bruchteil der Zeit, die wir Menschen benötigen würden.

Diese beiden Eigenschaften ermöglichen es uns, mit Computern besonders solche Probleme effizient zu lösen, die wiederkehrend und in großer Zahl auftreten. Wir sprechen dann von **Automatisierung**.

Um den Begriff des Problems besser zu verstehen und seine Bedeutung im Kontext von Computern einzugrenzen, führen wir zunächst ein einfaches Modell ein.

2.2 Das Eingabe-Verarbeitung-Ausgabe-Modell

Das Eingabe-Verarbeitung-Ausgabe-Modell (EVA-Modell, s. Abbildung 2.1) ist ein grundlegendes Konzept der Informatik, das die Arbeitsweise von Computern vereinfacht erklärt. Es zeigt, wie Computer Probleme lösen: Sie wandeln Eingabedaten durch einen definierten Verarbeitungsprozess in gewünschte Ausgaben um. Ein Taschenrechner veranschaulicht dies gut: Die Eingabe besteht aus Zahlen und der gewünschten Operation, die Verarbeitung erfolgt durch mathematische Berechnung, und die Ausgabe ist das Ergebnis.

Verallgemeinert beschreibt das Modell ein Problem und dessen Lösung als eine Eingabe (*Input*), die dem Computer übergeben wird. Darauf folgt eine Verarbeitung (*Computation*) auf Basis dieser Eingabe, die wiederum eine Ausgabe (*Output*) erzeugt. Beim Taschenrechner lässt sich das sehr eingängig darstellen, und man kann sich bildlich vorstellen, wie ein Mensch die Eingabe tätigt und das Ergebnis abliest. Es ist aber wichtig zu verstehen, dass Eingabe und

Ausgabe sehr unterschiedliche Formen annehmen können und keinesfalls nur über eine Tastatur erfolgen müssen.

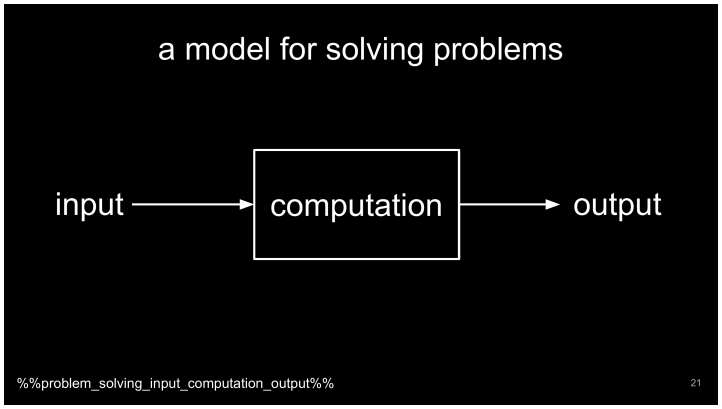


Abbildung 2.1: Das EVA-Modell besteht aus der Eingabe, der Berechnung und der Ausgabe.

2.2.1 Taschenrechner

Das Beispiel des Taschenrechners wird in Abbildung 2.2 anhand einer einfachen Addition zweier Zahlen verdeutlicht. Als Eingabe werden zwei Zahlen benötigt, die Berechnung erfolgt durch eine Addition, dargestellt durch das Plussymbol. Die Ausgabe ist das Ergebnis – die Summe.

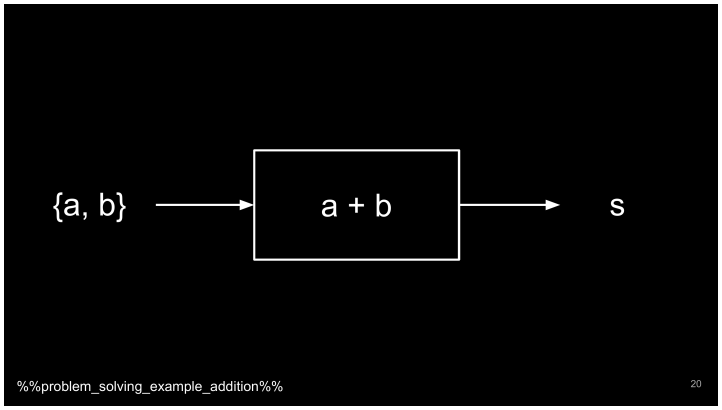


Abbildung 2.2: Das EVA-Modell für die Addition zweier Zahlen.

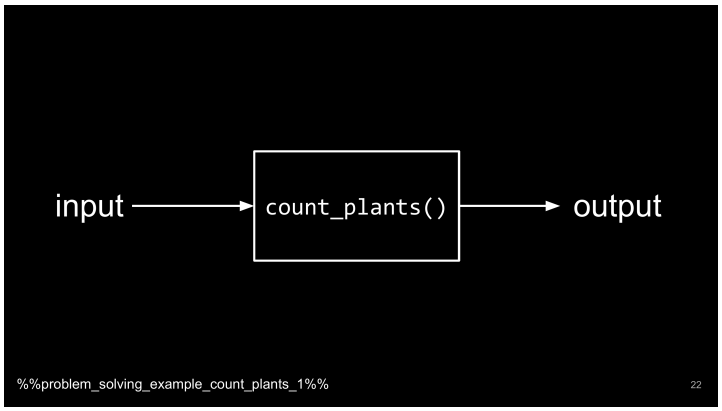
Dieses einfache Beispiel verdeutlicht, dass wir verstehen müssen, wie Computer die drei Bestandteile des EVA-Modells umsetzen. Beim Taschenrechner sind Ein- und Ausgabe jeweils Zahlen. Diese Daten speichert der Computer in seinem Arbeitsspeicher. Dabei ist wichtig zu wissen, dass Computer auf der untersten Ebene ausschließlich Nullen und Einsen speichern.

Wir müssen also verstehen, wie Computer Zahlen mithilfe dieser Binärzahlen darstellen können.

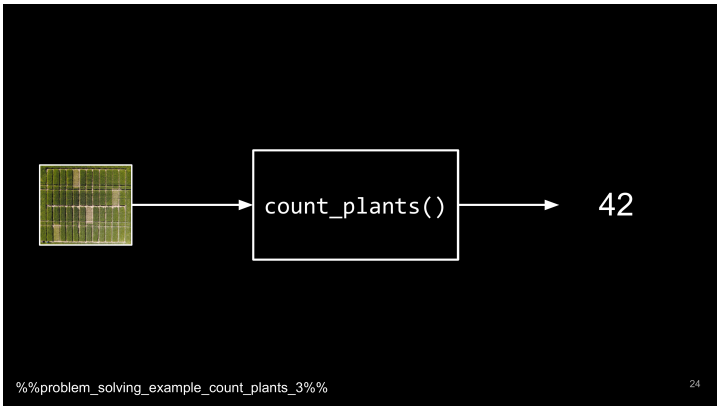
Was passiert bei der Berechnung in der Mitte des Modells? Eine Addition mag uns einfach erscheinen, doch auch hier müssen wir beachten, dass Computer mit Binärzahlen arbeiten. Es stellt sich also die Frage: Wie funktioniert eine Addition, wenn die Zahlen als Folge von Nullen und Einsen dargestellt sind? Auf die beiden Fragen zur **Repräsentation und der Verarbeitung von Informationen** im binären System werden wir im Laufe des Buches Antworten bekommen.

2.2.2 Pflanzen zählen

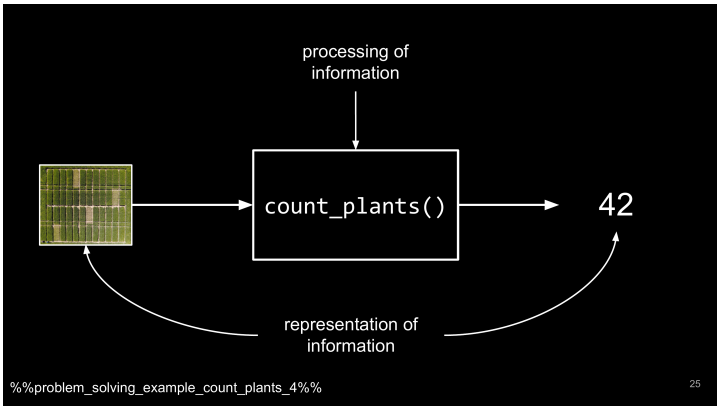
Betrachten wir ein weiteres Beispiel: Stell dir vor, du möchtest einen Computer nutzen, um Maispflanzen auf einer Drohnenaufnahme eines Ackers zu zählen. Diese Aufgabe ist für Menschen zwar einfach zu verstehen, wäre aber sehr zeitaufwändig auszuführen. Moderne Algorithmen ermöglichen es Computern, Objekte auf Bildern präzise zu lokalisieren und zu zählen. Nehmen wir an, wir haben für dieses Problem bereits eine Lösung entwickelt und ein Programm namens `count_plants` erstellt. Nun stellt sich die Frage: Wie sehen die Eingabe und die Ausgabe für dieses Problem aus? Was benötigt das Programm von uns, und was liefert es als Ergebnis?



Die erwartete **Ausgabe** lässt sich einfach beschreiben: Das Ergebnis der Zählung ist eine ganze Zahl. Die **Eingabe** für dieses Problem ist - anders als beim Taschenrechner - kein Tastendruck, sondern ein Bild. Damit der Computer das Bild verarbeiten kann, muss es dem Computer in digitaler Form bereitgestellt werden. Was das genau bedeutet, lernen wir in einem späteren Kapitel. Hier genügt es uns zu verstehen, *dass* wir das Bild digital abbilden müssen.

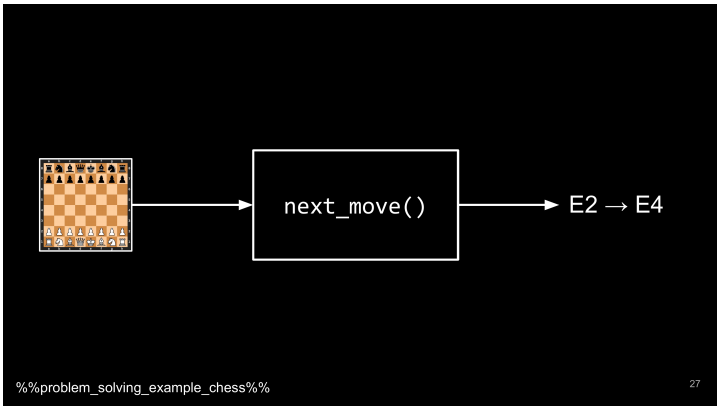


Wie gelangt das Bild in den Computer? Dies ist im Modell nicht näher definiert und für die Problembeschreibung auch nicht wesentlich. Das Bild muss lediglich irgendwie in den Arbeitsspeicher des Programms `count_plants` gelangen. Dies kann auf verschiedene Arten geschehen: Es kann von der Festplatte gelesen werden, über eine drahtlose Verbindung wie Bluetooth direkt übertragen und verarbeitet werden, oder das Programm `count_plants` läuft direkt auf der Drohne und greift unmittelbar auf deren Kamera zu. Die technische Umsetzung ist für unser Modell zunächst irrelevant. In einem späteren Kapitel werden wir uns damit befassen, wie Informationen genau übertragen und gespeichert werden. Genauso werden wir lernen, wie die benötigten Informationen für die Ein- und Ausgabe eines Programms in digitaler Form dargestellt werden können.



2.2.3 Schach spielen

Ein weiteres Beispiel zur Verdeutlichung des EVA-Modells ist das Schachspiel. Das Problem lässt sich einfach beschreiben: Der Computer soll auf Grundlage einer bestehenden Spielsituation den bestmöglichen nächsten Zug vorschlagen. Dieser Zug soll die Gewinnchancen maximieren.



Betrachten wir zunächst die Eingabe für dieses Problem: Wir können dem Computer nicht einfach ein physisches Schachbrett zeigen, sondern müssen überlegen, wie sich ein Schachbrett und die Position der Figuren in digitaler Form darstellen lassen. Dabei kann es durchaus mehrere Möglichkeiten geben, die uns ans Ziel führen.

Ein Schachbrett lässt sich etwa als Liste von 64 Feldern darstellen, die von oben links nach unten rechts durchnummeriert sind. Für jedes Feld speichern wir, ob es leer ist oder welche Figur darauf steht. Die Figuren werden durch Buchstaben dargestellt – zum Beispiel “R” für den Turm (Englisch: Rook) oder “N” für den Springer (Englisch: Knight). Für die Farben Schwarz und Weiß verwenden wir einfach 0 und 1. Diese Darstellungsform reduziert unser Problem auf Listen, Zahlen und Buchstaben in digitaler Form. Für Computer ist das eine leicht zu verarbeitende Struktur, wie wir später noch sehen werden. Ein Beispiel für eine solche Kodierung zeigt Abbildung 2.3.

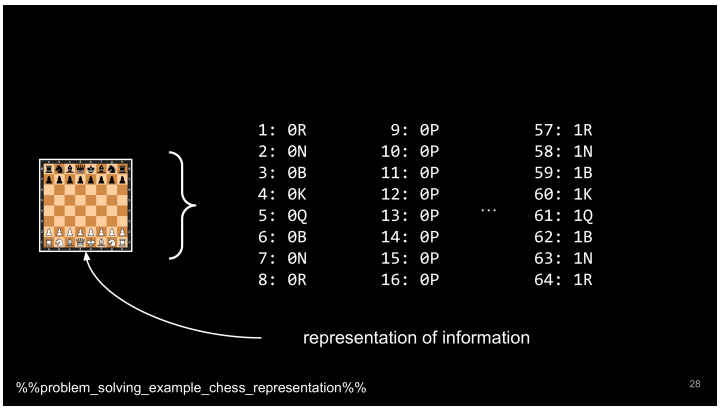


Abbildung 2.3: Beispiel für die Darstellung von Schachfiguren als Zahlen und Buchstaben.

Die Ausgabe, also der nächste Zug, lässt sich ebenfalls durch Zahlen und Buchstaben darstellen. Eine weit verbreitete Notation gibt zunächst die Koordinate des Ausgangsfelds an, von dem eine Figur gezogen werden soll, gefolgt von der Koordinate des Zielfelds. Ein Beispiel wäre

der Zug von “E2 nach E4”. Statt “E2” und “E4” könnten wir ebenso die entsprechende Zahl zwischen 1 und 64 aus unserer Liste verwenden, um mit dem obigen Schema konsistent zu bleiben. Der Zug hieße dann “53 nach 37”.

2.2.4 Mit Computern chatten

Als drittes Beispiel betrachten wir die Verwendung von Chatprogrammen wie ChatGPT. Seit seiner Veröffentlichung im November 2022 hat es die Welt stark verändert und einen regelrechten KI-Hype ausgelöst. Ein großes Sprachmodell wie GPT-4, das hinter dem heutigen ChatGPT steckt, ist eine komplexe Software, die wir in diesem Buch nicht vollständig ergründen können. Das Schöne an Modellen wie dem EVA-Modell ist jedoch, dass sie komplexe Sachverhalte vereinfachen können – so auch bei Sprachmodellen. Das Problem, das Sprachmodelle lösen, lässt sich wie alle Probleme in unserem EVA-Modell einfach darstellen.

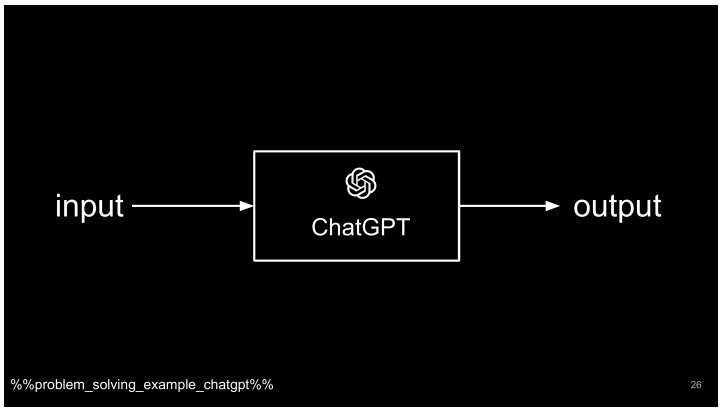


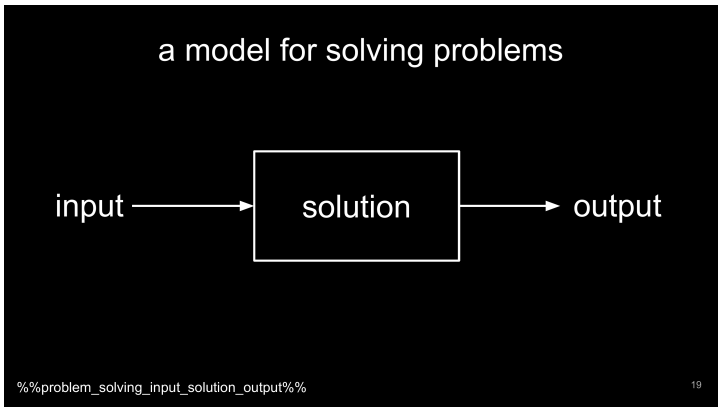
Abbildung 2.4: ChatGPT im EVA-Modell.

Dabei betrachten wir das Sprachmodell – oder ChatGPT – als Blackbox, ohne die internen Prozesse genauer zu definieren. Für unser Modell genügt es zu verstehen, dass wir eine Eingabe in Form einer Nachricht an ChatGPT benötigen und als Ausgabe eine Antwort erhalten. Auch hier stellt sich die Frage, wie wir beides digital repräsentieren können, damit ChatGPT damit arbeiten kann.

Klassischerweise bestehen sowohl Eingabe als auch Ausgabe einfach aus Texten – allerdings beherrschen moderne Sprachmodelle auch andere Eingabeformen wie Bilder oder gesprochene Sprache über ein Mikrofon. Wir sprechen dann von multimodalen KI-Modellen. Bei Bildern stehen wir vor demselben Repräsentationsproblem wie bei unserer Drohnenaufnahme. Bei der Sprache stellt sich neben der Repräsentation von Audioinhalten die Frage, wie wir gesprochene Worte überhaupt in eine digitale Form überführen können. Auch dazu erfahren wir im späteren Verlauf des Buches mehr.

2.3 Die Lösung des Problems

Anhand des EVA-Modells wird deutlich, dass wir dem Computer Informationen in Form von digitalen Daten bereitstellen müssen, mit denen er arbeiten kann. Was aber genau soll er damit machen? Hier kommt der mittlere Kasten des Modells ins Spiel – die eigentliche Lösung des Problems.



In der Informatik nennen wir die Beschreibung zur Lösung eines Problems einen **Algorithmus**. Ein Algorithmus ist eine Schritt-für-Schritt-Anleitung zur Problemlösung und ist zunächst unabhängig von Computern. Das bedeutet, wir können die Lösung eines Problems ohne Bezug zu einem Computer beschreiben und nennen das einen Algorithmus.

Stellt euch dazu zum Beispiel eine IKEA-Aufbauanleitung vor. Sie beschreibt in sequenziellen Schritten, was zu tun ist, um das fertige Möbelstück zu bekommen. Die Eingabe besteht aus den mitgelieferten Teilen, Schrauben und dem benötigten Werkzeug für den Zusammenbau. Die Ausgabe ist das fertige Regal (oder ein anderes Möbelstück) – und das alles ganz ohne Computer.

Oder nehmt das Kochrezept eure Lieblingsessens. Auch ein Kochrezept ist ein Algorithmus: Die Eingabe besteht aus den Zutaten und Küchenutensilien, die Verarbeitung erfolgt durch die Schritt-für-Schritt-Anleitung, und die Ausgabe ist das fertige Gericht. Wie bei der IKEA-Anleitung ist der Algorithmus unabhängig von einem Computer – er beschreibt lediglich die Lösung des Problems “Wie koche ich dieses Gericht?”.

Zu Beginn des Kapitels haben wir festgestellt, dass Computer aufgrund ihrer Geschwindigkeit und Fehlerfreiheit besonders gut zur Problemlösung geeignet sind – insbesondere bei häufig wiederkehrenden Problemen. Die beiden genannten Beispiele, das IKEA-Regal und das Kochrezept, eignen sich allerdings nicht für eine direkte Umsetzung durch Computer. Der Grund liegt in den analogen Eingaben (Baumaterial, Kochzutaten) und Ausgaben (Möbelstück, fertige Mahlzeit). Diese kann ein Computer nicht unmittelbar verarbeiten. Dafür wären Roboter nötig, die mit der physischen Welt interagieren können. Eine solche Automatisierung lohnt sich

heute nur bei Aufgaben, die sehr häufig auftreten und ansonsten mit hohen Kosten verbunden sind – wie etwa in der Automobilindustrie, wo computergesteuerte Roboter in der Produktion zum Einsatz kommen.

Nehmen wir an, dass Haushaltsroboter in Zukunft erschwinglich werden und uns beim Kochen unseres Lieblingsgerichts helfen können. Wie vermitteln wir dann dem Roboter – im Grunde ein Computer mit Armen und Beinen – den Algorithmus für unser Rezept? Die Lösung liegt in der Programmierung: Wir erstellen ein **Programm** in einer computerverständlichen Sprache. Dieses Programm wandelt die Anweisungen aus unserem Kochbuch in Befehle um, die der Computer verstehen und ausführen kann. Genau genommen besteht ein Programm ebenfalls nur aus Informationen, und wir müssen herausfinden, wie wir diese Informationen digital darstellen können. Die Lösung besteht in der Verwendung einer **Programmiersprache** wie Python, die wir später kennenlernen werden.

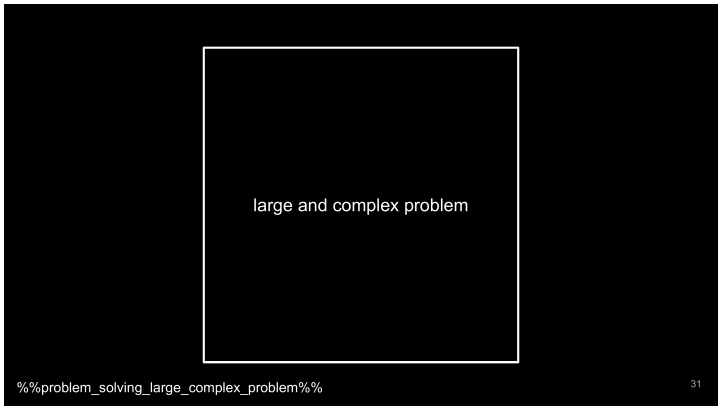
2.4 Problemlösungsstrategien

Die konkrete Lösung und der zugehörige Algorithmus sehen für jedes Problem unterschiedlich aus. Das Erkennen von Pflanzen folgt einer anderen Logik als die Entscheidung, welche Schachfigur als nächstes gezogen werden soll. Dennoch gibt es universelle Lösungsstrategien, die auf viele Probleme anwendbar sind, um sie möglichst effizient zu lösen.

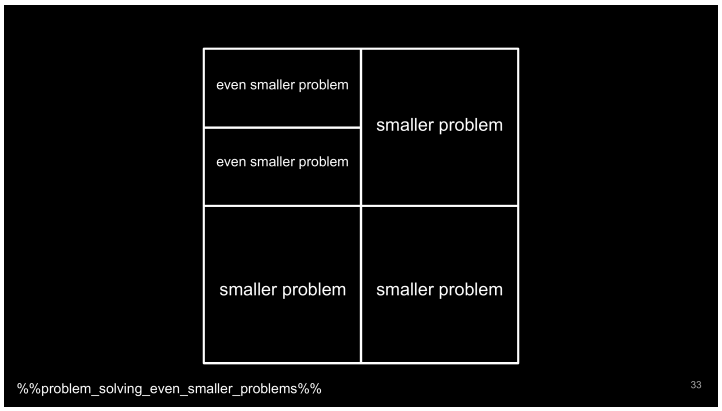
2.4.1 Problemzerlegung (*Problem Decomposition*)

Eine universelle Strategie zur Lösung komplexer Probleme ist die Zerlegung in kleinere Schritte oder Teilprobleme. Jedes Teilproblem hat dabei seine eigene Charakteristik und erfordert einen spezifischen Lösungsansatz. Nehmen wir als Beispiel das Zählen von Pflanzen auf einer Drohnenaufnahme. Dieses komplexe Problem lässt sich, ausgehend von der Eingabe – dem digitalen Bild – in drei Teilprobleme zerlegen:

1. Pflanzen auf dem Bild lokalisieren
2. Lokalisierte Pflanzen klassifizieren: Maispflanze oder nicht?
3. Identifizierte Maispflanzen zählen



Jedes dieser Teilprobleme erfordert einen eigenen Algorithmus. Dabei ist es möglich, die Teilprobleme noch weiter zu zerlegen, um sie besser bearbeiten zu können.



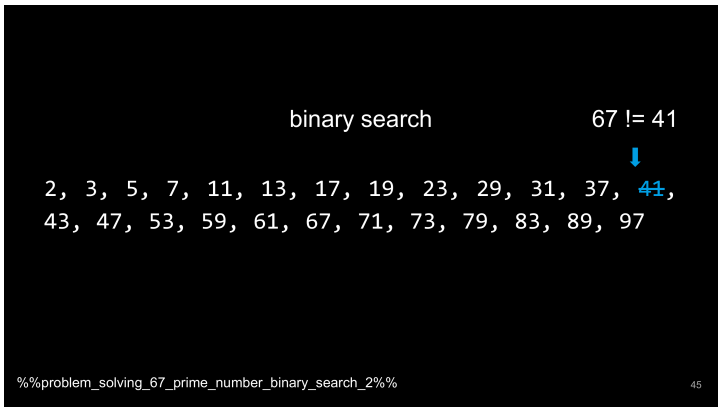
Die Identifizierung sinnvoller Teilprobleme erfordert ein ausgeprägtes analytisches Denkvermögen. Dies ist besonders wichtig im Umgang mit Computern. Wie wir beim Erlernen der Programmierung sehen werden, ist die Zerlegung eines Problems in kleine, lösbare Schritte der Schlüssel zur Beherrschung seiner Komplexität.

2.4.2 Teile und Herrsche (*Divide and Conquer*)

Die "Teile und Herrsche"-Strategie ist ein Ansatz zur Lösung komplexer Probleme, bei dem wir das Hauptproblem schrittweise in immer kleinere Teilprobleme zerlegen, bis diese einfach zu lösen sind. Dabei gehen wir rekursiv vor: Wir teilen das ursprüngliche Problem in kleinere Teile, diese kleineren Probleme wiederum in noch kleinere Teile und so weiter. Die Rekursion endet, wenn die Probleme so klein sind, dass sie sich nicht weiter aufteilen lassen und die Lösung direkt ersichtlich ist.

Anders als bei der Problemzerlegung sind die Teilprobleme beim Divide and Conquer-Ansatz dadurch gleichartig und stellen nur kleinere Instanzen des ursprünglichen Problems dar. Die einzelnen Lösungen für jedes Teilproblem werden dann schrittweise wieder zusammengeführt, um die Gesamtlösung zu erhalten. Ein klassisches Beispiel ist die Sortierung einer langen Liste von Zahlen: Wir teilen die Liste immer wieder in der Mitte, bis nur noch einzelne Zahlen übrig sind, und fügen diese dann in sortierter Reihenfolge wieder zusammen.

Ein anderes Beispiel ist die binäre Suche in einer sortierten Liste. Hier betrachten wir das Element in der Mitte der Liste und vergleichen es mit dem gesuchten Element. Da die Liste sortiert ist, können wir entscheiden, in welchem Teil der Liste wir weitersuchen müssen. Im zweiten Schritt suchen wir nur in diesem Teil weiter und haben damit das Problem halbiert. Die Natur des Problems bleibt dabei gleich, und wir können erneut genauso verfahren – so lange, bis wir nur noch ein Element übrig haben, das entweder das gesuchte Element ist oder nicht.



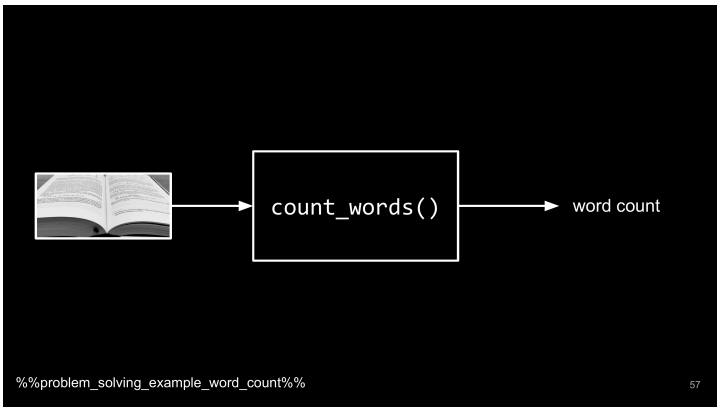
2.4.3 Verteile und Parallelisiere (*Distribute and Parallelize*)

Manche Probleme lassen sich effizienter lösen, wenn mehrere Personen gleichzeitig daran arbeiten. Anstatt eine einzelne Person mit der gesamten Aufgabe zu betrauen, verteilen wir die Arbeit auf mehrere Schultern und arbeiten parallel an der Lösung. Allerdings eignet sich nicht jedes Problem für diesen Ansatz.

Ein gutes Beispiel ist das Aufräumen eines Zimmers: Eine einzelne Person müsste nacheinander verschiedene Bereiche aufräumen, während mehrere Personen gleichzeitig unterschiedliche Ecken des Raums in Angriff nehmen können. Je größer das Zimmer, desto mehr Personen werden benötigt, um es in der gleichen Zeit aufzuräumen. Ein Gegenbeispiel, bei dem diese Strategie nicht funktioniert, ist das Lösen einer mathematischen Gleichung. Hier müssen die einzelnen Rechenschritte aufeinander aufbauen, weshalb das Problem nicht gleichzeitig an mehrere Personen übergeben werden kann, die unabhängig daran arbeiten.

Etwas technischer lässt sich die Strategie des Verteilens und Parallelisierens, die wir im Englischen *Distribute and Parallelize* nennen, wie folgt beschreiben: Wir zerteilen ein großes Problem in unabhängig voneinander lösbare Teile und weisen jeder Ressource (etwa einer Person oder einem Computer) einen Teil zur Bearbeitung zu. Jede Ressource erzeugt ein Ergebnis für ihr Teilproblem, und die Prämisse ist, dass sich am Ende alle Teile zur Gesamtlösung des ursprünglichen Problems zusammenfügen lassen.

Um das Prinzip zu verdeutlichen, betrachten wir ein weiteres Beispiel, das wir zunächst mithilfe des EVA-Modells einführen: das Zählen aller Wörter in einem Buch.



Je nach Dicke des Buches kann das eine sehr mühsame Aufgabe sein, besonders für uns Menschen. Ein Computer wird mit einem einzelnen Buch keine großen Probleme haben – dank der unglaublichen Geschwindigkeit, mit der er arbeitet. Allerdings können wir das Problem beliebig vergrößern, indem wir das Buch symbolisch für alle Texte im Internet oder für Artikel in der Wikipedia-Enzyklopädie stehen lassen. Dann wird es auch für Computer mühsam oder zumindest zeitintensiv.

3 Algorithmen

3.1 Was ist ein Algorithmus?

Der Begriff "**Algorithmus**" stammt vom Namen des persischen Mathematikers Muhammad al-Khwarizmi, der um das Jahr 780 n. Chr. geboren wurde. Al-Khwarizmi war ein bedeutender Gelehrter am Hofe des Kalifen al-Mamun und verfasste dort Schriften, die den Gebrauch der indischen Zahlzeichen erklärten. Diese Schriften wurden im 12. Jahrhundert ins Lateinische übersetzt, wobei der Titel "Algoritmi de numero Indorum" verwendet wurde. Im Laufe der Zeit wurde der Name al-Khwarizmi zur Bezeichnung für die von ihm beschriebenen Rechenverfahren und entwickelte sich schließlich zum modernen Begriff "Algorithmus".

Heute bezeichnet ein **Algorithmus** eine präzise Abfolge von Anweisungen, die ein bestimmtes Problem lösen oder eine Aufgabe erfüllen sollen. Im Alltag begegnen uns Algorithmen ständig, oft, ohne dass wir es merken: beim Kochen, bei der Wegbeschreibung oder beim Aufbau eines IKEA-Regals.

Um besser zu verstehen, was ein Algorithmus ist, grenzen wir die Begriffe Algorithmus, Programm und Prozess voneinander ab:

- **Algorithmus**: Ein allgemeiner Plan oder eine Methode zur Problemlösung, bestehend aus einer endlichen Sequenz von Anweisungen.
- **Programm**: Die konkrete Umsetzung eines oder mehrerer Algorithmen in einer Programmiersprache, die ein Computer ausführen kann.
- **Prozess**: Die Ausführung eines Programms durch einen Computer.

Algorithmen haben also nicht zwangsläufig etwas mit Computern zu tun. Sie beschreiben lediglich, wie man ein Problem lösen kann. Ob das ein Computer macht oder ein Mensch ist dabei offen. Überlege etwa, wie du eine IKEA-Aufbauanleitung befolgst oder wie du ein Rezept in der Küche kochst. Diese alltäglichen Abläufe folgen alle Algorithmen, ohne dass ein Computer beteiligt ist:

- Kochen: Ein Rezept ist ein Algorithmus für die Zubereitung eines Gerichts.

Comment

- Wegbeschreibung: Eine Schritt-für-Schritt-Anleitung, um von Punkt A nach Punkt B zu gelangen.

- Bastelanleitung: Die Anweisungen, um ein Modellflugzeug zusammenzubauen.

Manche Algorithmen können wir einem Computer beibringen und ihn diesen ausführen lassen. Dazu müssen wir eine präzise Übersetzung des Algorithmus in eine Programmiersprache erstellen, die der Computer versteht und ausführen kann. Wir sprechen dann vom Programmieren und das Ergebnis ist ein Programm.

Algorithmen lassen sich basierend auf ihrer Funktion und Anwendung in verschiedene Kategorien einteilen. Hier sind einige wichtige Klassen von Algorithmen:

- Suchalgorithmen
- Sortieralgorithmen
- Optimierungsalgorithmen
- Graphenalgorithmen
- Verschlüsselungsalgorithmen
- Maschinelle Lernalgorithmen

Die oben aufgeführten Klassen sind nicht vollständig und schließen sich nicht gegenseitig aus. Das bedeutet, dass die Klassen Überlappungen aufweisen und sich manche Algorithmen zwei oder mehreren Klassen zuordnen lassen. So ist der **Dijkstra-Algorithmus** für das Finden des kürzesten Weges zwischen zwei Orten ein Graphenalgorithmus, weil er auf einer Datenstruktur arbeitet, die wir als Graphen bezeichnen. Gleichzeitig ist er ein Optimierungsalgorithmus, weil er eine optimale Lösung für ein Problem ermittelt: den kürzesten Weg aus der Menge aller möglichen Wege zu finden.

3.2 Suchen

3.3 Sortieren

3.4 Optimieren

4 Informationen

4.1 Information in der Informatik

Hast du dich jemals gefragt, was eigentlich Information genau ist? Jeder hat eine intuitive Vorstellung davon, was wir mit Information meinen. Aber was ist die genaue Definition? Und wie ist der Begriff *Information* im Zusammenhang mit digtialen Computern gemeint?

Hier schon einmal eine Definition, die wir in diesem Kapitel anhand von einigen Beispielen genauer verstehen wollen.

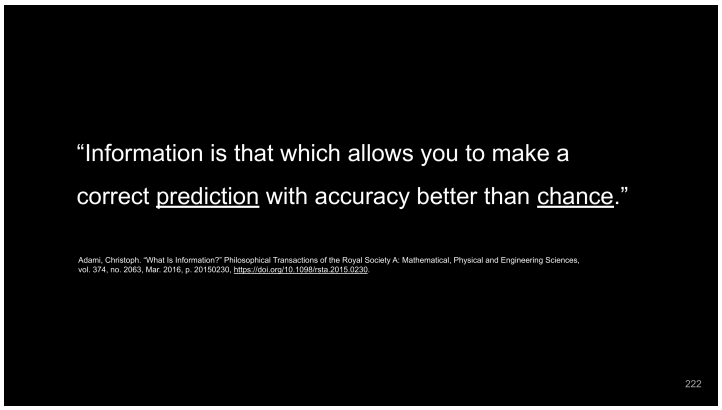


Abbildung 4.1: Lauet dieser Definition erhöht Information unsere Treffsicherheit bei Vorhersagen.

4.1.1 Zahlenraten

Um der Information auf die Spur zu kommen beginnen wir mit einem Gedankenexperiment. Stell dir vor, wir spielen ein Zahlenratespiel. Ich denke an eine Zahl zwischen 1 und 16, und dein Ziel ist es, sie zu erraten. Der Haken ist, dass du nur einen Versuch hast, die richtige Zahl zu erraten, aber du kannst die Möglichkeiten vorher durch Fragen der Form “Ist die Zahl größer als X?” eingrenzen. Für jede Frage werde ich dir mit “ja” oder “nein” antworten und damit die verbleibenden Optionen reduzieren.

Mit jeder Antwort, die du erhältst, wächst dein Wissen über meine Zahl, was bedeutet, dass deine **Unsicherheit** bezüglich der gesuchten Zahl abnimmt. Du kannst einige mögliche Zahlen ausschließen, wenn du eine neue Antwort erhältst.

Wir halten zunächst fest, dass das Eingrenzen der verbleibenden Möglichkeiten die Unsicherheit reduziert. Wir untersuchen nun, wie diese Idee die Grundlage der Informationstheorie bildet. Durch die Reduzierung der Unsicherheit sammeln wir mit jedem Versuch mehr **Information**. Aber wie viel Information erhältst du mit jeder Antwort? Und wie können wir das messen?

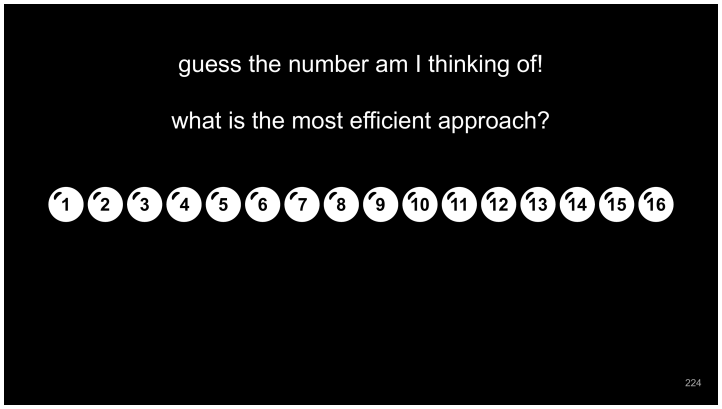


Abbildung 4.2: Ich denke an eine Zahl zwischen 1 und 16. Deine Aufgabe ist es, die Zahl mit so wenig Fragen wie möglich zu erraten.

4.1.2 Bit für Bit

In der Informatik kann Information definiert werden als "das, was es ermöglicht, eine korrekte Vorhersage mit besserer Genauigkeit als der Zufall zu treffen" [CITE]. Einfacher ausgedrückt bedeutet dies, die Unsicherheit durch Eingrenzung der möglichen Optionen zu reduzieren.

Vor diesem Hintergrund wollen wir unser Zahlenratespiel noch einmal genauer betrachten. Wie oben beschrieben, versuchst du meine Zahl zu erraten, und jede Antwort, die du von mir erhältst, grenzt den Bereich der möglichen Zahlen ein. Diese Reduzierung der Unsicherheit kann mit einer Einheit namens **Bit** gemessen werden. Ein Bit, was für **binary digit** (Binärziffer) steht, ist die grundlegende Einheit der Information. Es zeigt eine Halbierung der Unsicherheit an. Einfacher ausgedrückt: Wenn eine neue Antwort nur noch halb so viele Optionen wie zuvor übrig lässt, liefert sie uns genau ein Bit an Information.

Allerdings werden nicht alle Fragen und deren Antworten genau ein Bit Information liefern. Wenn zum Beispiel deine erste Frage im Ratespiel lautet: "Ist deine Zahl größer als 12?" und die Antwort "nein" ist, bleiben die Zahlen 1 bis 12 übrig. Das bedeutet, du hast noch 12 Optionen von ursprünglich 16, was die Unsicherheit nicht halbiert. Es werden nur 4 statt der nötigen 8 Möglichkeiten entfernt. Lautet die Antwort dagegen "ja", so kannst du insgesamt

12 Möglichkeiten streichen, was mehr als der Hälfte entspricht und der Informationsgehalt der Antwort wäre größer als ein Bit.

Um genau ein Bit Information zu erhalten, solltest du darauf abzielen, mit jeder Frage genau die Hälfte der möglichen Zahlen auszuschließen. Wenn du zum Beispiel fragst “Ist deine Zahl größer als 8?”, stellst du sicher, dass du - egal ob die Antwort “ja” oder “nein” ist - in beiden Fällen 8 mögliche Zahlen übrig behältst. Ist die Antwort “ja”, bleiben die Zahlen 9 bis 16 übrig. Ist die Antwort “nein”, bleiben die Zahlen 1 bis 8. In beiden Szenarien wird deine Unsicherheit um die Hälfte reduziert, also um ein Bit.

Indem du deine Fragen sorgfältig so wählst, dass sie die verbleibenden Optionen jedes Mal halbieren, reduzierst du deine Unsicherheit Bit für Bit und machst es dir leichter und schneller, die richtige Zahl zu finden.

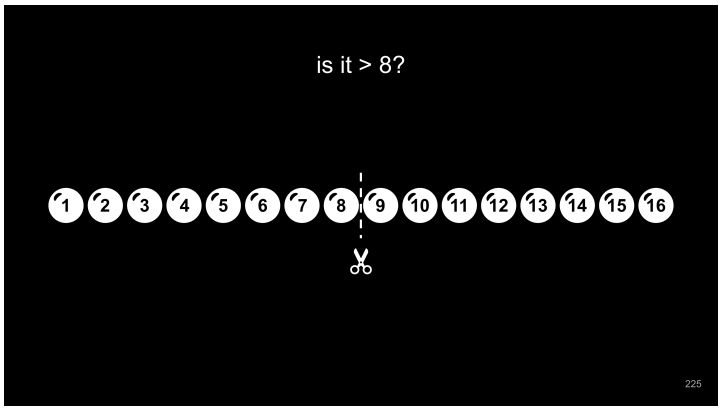


Abbildung 4.3: Deine Fragen sollten die Anzahl der Möglichkeiten halbieren.

Angenommen, die Antwort auf deine erste Frage “Ist deine Zahl größer als 8?” war “nein”. Was sollte deine nächste Frage sein, um die Unsicherheit weiterhin effektiv zu reduzieren? Die beste Strategie ist es zu fragen: “Ist sie größer als 4?”. Diese Vorgehensweise stellt sicher, dass dir immer nur vier mögliche Zahlen bleiben: entweder 1 bis 4, wenn die Antwort “nein” ist, oder 5 bis 8, wenn die Antwort “ja” ist. Erneut halbiert dies die verbleibenden Optionen und liefert genau ein Bit an Information.

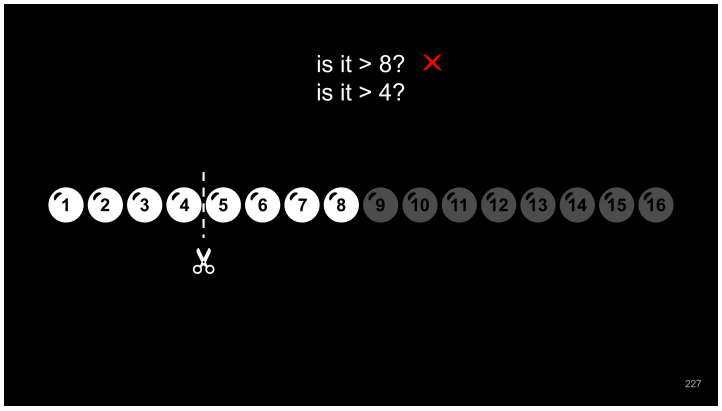


Abbildung 4.4: Nach zwei Fragen bleiben noch 4 Möglichkeitn, wenn die Fragen gut gewählt wurden.

Lass uns mit dieser Methode fortfahren. Angenommen, deine zweite Frage “Ist sie größer als 4?” erhält ein “nein” als Antwort. Deine neue Auswahl an Zahlen ist auf 1, 2, 3 und 4 begrenzt. Um die Unsicherheit weiter zu reduzieren, wäre die nächste logische Frage: “Ist sie größer als 2?” Dies lässt dir entweder die Zahlen 1 und 2 oder 3 und 4, abhängig von der Antwort.

Indem du weiterhin Fragen stellst, die systematisch die verbleibenden Optionen halbieren, kannst du sehen, wie wir schrittweise die Unsicherheit Bit für Bit reduzieren. Schließlich wirst du nach nur vier Fragen die Zahl auf genau eine eingrenzen, da keine anderen Möglichkeiten mehr übrig sind. Das bedeutet, du hast die Unsicherheit auf null reduziert.

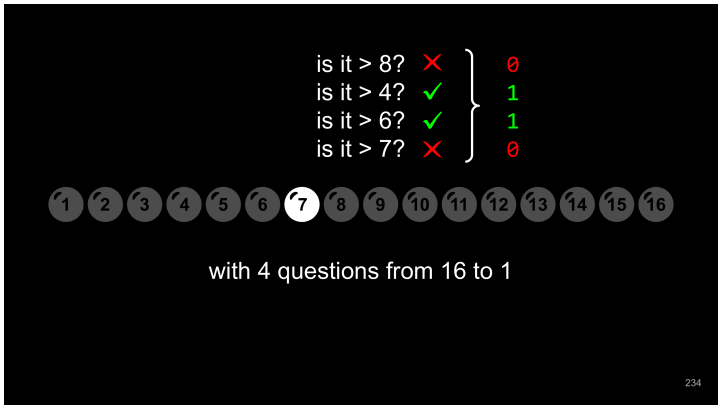


Abbildung 4.5: Wir benötigen vier Ja/Nein-Fragen, um die Optionen von 16 auf eine einzige zu reduzieren.

4.1.3 Unsicherheit

Wir haben gerade gelernt, dass wir mit jeder Ja/Nein-Frage, die die Hälfte der Möglichkeiten eliminiert, ein Bit an Information gewinnen. Aber wie können wir dieses Konzept der Reduktion von Unsicherheit mathematisch quantifizieren? Wir gehen das Schritt für Schritt durch.

Stell dir vor, du bist am Anfang unseres Zahlenratespiels. Es gibt 16 mögliche Zahlen, an die ich denken könnte, daher ist deine Wahrscheinlichkeit, beim ersten Versuch die richtige Zahl zu erraten, ziemlich gering:

$$P_{correct} = \frac{1}{16}$$

Das entspricht einer Wahrscheinlichkeit von nur 0,0625 - definitiv nicht zu deinen Gunsten. Angenommen, du stellst eine Frage, die die Möglichkeiten auf die Hälfte reduziert. Jetzt verbessert sich deine Chance, richtig zu raten:

$$P_{correct} = \frac{1}{8}$$

Mit jeder weiteren Frage verdoppelt sich diese Wahrscheinlichkeit, während sich deine Unsicherheit halbiert. Ist Wahrscheinlichkeit die beste Methode, um Unsicherheit zu messen? Wenn wir möchten, dass unser Maß für Unsicherheit abnimmt, während wir mehr Informationen sammeln, ist es es sinnvoll, den Kehrwert der Wahrscheinlichkeit zu verwenden. Das entspräche der Anzahl an Möglichkeiten, die noch im Rennen sind.

Zu Beginn gibt es 16 Möglichkeiten, also begänne die Unsicherheit bei 16. Nach der ersten Frage fiele sie auf 8, dann auf 4, 2 und schließlich 1. Allerdings würde dieses Maß uns selbst dann, wenn nur noch eine Option übrig ist, eine gewisse Unsicherheit, nämlich 1, anzeigen, was nicht ganz unserem intuitiven Verständnis entspricht. Wenn wir sicher sind, dann sollte die Unsicherheit 0 sein.

Claude Shannon, der Vater der Informationstheorie, schlug einen anderen Ansatz vor. Er empfahl, die Unsicherheit mithilfe des Logarithmus zur Basis 2 der Anzahl der Möglichkeiten zu messen:

$$H = log_2(N)$$

Diese Methode hat mehrere Vorteile. Erstens ist die Unsicherheit gleich null, wenn es nur eine mögliche Antwort gibt (N=1), was mit unserer Intuition übereinstimmt:

$$log_2(1) = 0$$

Ein weiterer Vorteil ist die Einfachheit der Berechnungen, besonders wenn es um mehrere unabhängige Unsicherheiten geht. Nehmen wir an, das Spiel ändert sich: Jetzt denke ich an zwei unabhängige Zahlen zwischen 1 und 16. Bevor du irgendwelche Fragen stellst, beträgt deine Unsicherheit für jede Zahl:

$$log_2(16) = 4\ bits$$

Die Gesamtunsicherheit für beide Zahlen ist einfach die Summe ihrer individuellen Unsicherheiten:

$$log_2(16) + log_2(16) = 8\ bits$$

Wenn wir stattdessen Wahrscheinlichkeiten verwenden, müssten wir die Chancen multiplizieren, um die Wahrscheinlichkeit zu finden, beide Zahlen korrekt zu erraten:

$$P_{correct,correct} = \frac{1}{16} \times \frac{1}{16} = \frac{1}{256}$$

Mit beiden Zahlen gibt es 256 Möglichkeiten. Mit Shannons Methode erhalten wir das gleiche Ergebnis:

$$log_2(256) = 8\ bits$$

As you can see, using logarithms simplifies our calculations, especially when dealing with multiple sources of uncertainty. This clarity and simplicity are why Shannon's approach has become fundamental in the field of information theory.

Wie du siehst, vereinfacht die Verwendung von Logarithmen unsere Berechnungen, besonders wenn es um mehrere Quellen der Unsicherheit geht. Diese Klarheit und Einfachheit sind der Grund, warum Shannons Ansatz fundamental für die Informationstheorie geworden ist.

4.1.4 Information

Um den Begriff *Information* aus dem Begriff der *Unsicherheit* herzuleiten, betrachten wir zwei Zustände: Die ursprüngliche Unsicherheit vor einer Frage, die wir als H_0 bezeichnen, und die reduzierte Unsicherheit nach der Antwort, die wir H_1 nennen. Die Menge an Information I, die wir durch die Antwort gewinnen, entspricht der Differenz dieser beiden Unsicherheiten:

$$I = H_0 - H_1$$

Information ist also die Reduzierung von Unsicherheit. Mit der obigen Formel können wir die Information I präzise quantifizieren. Dies ermöglicht uns zu berechnen, wie viel Information wir durch jede Frage gewinnen. Betrachten wir ein konkretes Beispiel aus unserem Zahlenratespiel: Angenommen, deine erste Frage ist "Ist deine Zahl größer als 12?" und die Antwort lautet "nein". Dann bleiben die Zahlen 1 bis 12 als Möglichkeiten übrig. Die gewonnene Information berechnet sich wie folgt:

$$I = log_2(16) - log_2(12) \approx 4 - 3.585 \approx 0.415\ bits$$

In diesem Fall lieferte die Antwort etwa 0,415 Bits an Information – also weniger als ein Bit. Das ist folgerichtig, da die Antwort weniger als die Hälfte der Möglichkeiten eliminiert hat.

Wie sähe es aus, wenn die Antwort "ja" gewesen wäre? In diesem Fall blieben nur die Zahlen 13, 14, 15 und 16 übrig, also vier mögliche Optionen:

$$I = log_2(16) - log_2(4) = 4 - 2 = 2\ bits$$

Diese Antwort lieferte mehr als ein Bit, nämlich 2 Bits. Wie lässt sich dieser Unterschied erklären?

4.1.5 Unwahrscheinliche Antworten

Die Menge an Information hängt von der Wahrscheinlichkeit der jeweiligen Antwort ab. Bei einer Frage, die die Möglichkeiten halbiert, hat jede Antwort "Ja" oder "Nein" auf die Frage "Ist die Zahl größer als X?" eine Wahrscheinlichkeit von genau 50%. Diese gleichmäßige Verteilung vereinfacht die Berechnung und liefert genau ein Bit an Information.

Bei der Frage "Ist deine Zahl größer als 12?" sind die beiden Antworten "Ja" und "Nein" allerdings nicht gleich wahrscheinlich. Es gibt zwölf mögliche Zahlen, die ein "Nein" ergeben und nur vier, die ein "Ja" ergeben, was Wahrscheinlichkeiten von 75% für "Nein" und 25% für "Ja" bedeutet. Da die "Ja"-Antwort weniger wahrscheinlich ist, ist sie überraschender – und liefert deshalb mehr Information. Wie wir berechnet haben, gibt uns ein "Ja" 2 Bits an Information, während ein "Nein" nur 0,415 Bits liefert.

Dieser Zusammenhang ist ein Kernprinzip der Informationstheorie: Je unwahrscheinlicher ein Ereignis ist, desto mehr Information enthält sein Auftreten. Umgekehrt liefert eine sehr wahrscheinliche Antwort, wie das "Nein" im obigen Beispiel, weniger Information, da sie weniger überraschend ist.

Warum zielen wir darauf ab, Fragen zu stellen, die die Möglichkeiten gleichmäßig halbieren? Man könnte auch wild raten in der Hoffnung, durch eine unwahrscheinlichere Antwort mehr Informationen zu bekommen – auch wenn man seltener richtig liegt. Die Antwort ist einfach:

Eine Frage, die den Raum der Möglichkeiten halbiert, maximiert unseren *erwarteten* Informationsgewinn. Zwar können wir mit einer riskanten Frage wie "Ist die Zahl größer 15?" unser Glück herausfordern und im Erfolgsfall die Unsicherheit stark verringern. Dies ist jedoch keine nachhaltige Strategie. Wenn wir das Spiel nicht nur einmal, sondern 10, 100 oder 1000 Mal spielen, nähern wir uns im Mittel dem Erwartungswert für die gewonnene Information $E[I]$. Und dieser Erwartungswert favorisiert eindeutig Fragen, die die Hälfte der Möglichkeiten eliminieren.

Um dies zu verdeutlichen, berechnen wir zunächst den Erwartungswert für die extreme Frage "Ist die Zahl größer 15?":

$$E[I] = \left(\frac{1}{16} \times 4\right) + \left(\frac{15}{16} \times 0.09311\right) = 0.3373$$

Der Erwartungswert berechnet sich durch den mit der Wahrscheinlichkeit gewichteten Informationsgewinn für jede mögliche Antwort. Bei der Frage "Ist die Zahl größer als 15?" gibt es ein "Ja" nur dann, wenn die Zahl 16 ist. Von den sechzehn möglichen Zahlen erzeugt also nur eine diese Antwort. Die Wahrscheinlichkeit beträgt somit $\frac{1}{16}$. In diesem Fall reduziert sich unsere Unsicherheit sofort auf Null, da nur noch eine einzige Möglichkeit übrig bleibt. Die ursprüngliche Unsicherheit betrug 4 Bits:

$$H_0 = log_2(16) = 4\ bits$$

Und wenn die Unsicherheit nach der Antwort "Ja" auf Null reduziert wird:

$$H_1 = log_2(1) = 0\ bits$$

Daraus ergibt sich ein Informationsgehalt von 4 Bits:

$$I = H_0 - H_1 = 4 - 0 = 4\ bits$$

Dies erklärt den ersten Teil der Erwartungswertgleichung. Der zweite Teil folgt demselben Prinzip, allerdings mit einem wichtigen Unterschied: Die Wahrscheinlichkeit für die Antwort "Nein" ist mit $\frac{15}{16}$ deutlich höher. Gleichzeitig verbleibt eine große Restunsicherheit H_1, da wir lediglich eine einzige Zahl ausschließen konnten:

$$H_1 = log_2(15) \approx 3.9069\ bits$$

Die gewonnene Information ist somit sehr klein:

$$I = 4 - 3.9069 \approx 0.0931\ bits$$

Anhand der Erwartungswertformel konnten wir zeigen, dass eine extreme Frage, die mit viel Glück direkt zum Ziel führt, in unserem Ratespiel einen erwarteten Informationsgehalt von etwa einem Drittel Bit (0,3373) hat. Prüfen wir nun, ob der Erwartungswert für Fragen, die die Hälfte der Möglichkeiten eliminieren, tatsächlich höher ist:

$$E[I] = (0.5 \times 1) + (0.5 \times 1) = 0.5 + 0.5 = 1 \; bit$$

Wie erwartet beträgt der Erwartungswert genau 1 Bit, da bei jeder Antwort – ob "Ja" oder "Nein" – jeweils die Hälfte der Optionen eliminiert wird.

4.1.6 Mehr als zwei Antworten

Bisher haben wir in unserem Zahlenratespiel nur Fragen mit zwei Antwortmöglichkeiten betrachtet. Informationen, also Antworten auf Fragen, können jedoch vielfältiger sein. Wie berechnen wir die Information, wenn es mehr als zwei Antwortmöglichkeiten gibt?

Als Beispiel betrachten wir eine Urne mit farbigen Kugeln in drei Farben: Rot, Grün und Blau. Aus dieser Urne ziehen wir zwei Kugeln, wobei wir nach jedem Zug die gezogene Kugel zurück in die Urne legen. Dieses statistische Experiment bezeichnet man als "Ziehen mit Zurücklegen".

4.2 Daten repräsentieren Information

5 Bits

5.1 Die kleinste mögliche Information

5.2 Ein Byte

6 Codesysteme

6.1 ASCII-Code

6.2 Verallgemeinerung von Codesystemen

6.3 Unicode

6.4 RGB-Code

6.5 Code vs Codec

Der Begriff *Codec* leitet sich von den zwei Wörtern en**co**der und **dec**oder ab. Die beiden Begriffe werden zwar ähnlich geschrieben, unterscheiden sich aber in ihrer Bedeutung. Ein Codec beschreibt ein Vorgehen oder einen Algorithmus, um Daten vor dem Senden nach einem definierten Verfahren zu kodieren und nach dem Empfang wieder zu dekodieren. Dies ist besonders sinnvoll, um Informationen effizienter zu übertragen und weniger Bandbreite zu beanspruchen. Zudem können Daten verschlüsselt und die Kommunikation dadurch sicherer gemacht werden. Wir lernen einige Methoden für beide Anwendungsfälle in einem späteren Kapitel kennen. Im Gegensatz dazu beschreibt ein Code ein System, das einer Reihe von Symbolen eine bestimmte Bedeutung zuordnet, wie wir in diesem Abschnitt gesehen haben.

7 Datenstrukturen

7.1 Listen

7.2 Mengen

7.3 Graphen

7.4 Bäume

7.5 Warteschlangen

8 Analog vs. Digital

8.1 Diskrete und kontinuierliche Werte

9 Speicher

9.1 Substratunabhängigkeit

10 Logik und Arithmetik

10.1 Schalter

10.1.1 Relais, Vakuumröhren und Transistoren

10.2 Logikgatter

10.3 Binäre Addition

10.4 Binäre Subtraktion

10.5 Substratunabhängigkeit

11 Computer

11.1 Komponenten eines Computers

- von Neumann-Architektur

12 Signale

- How can we create and receive signals?
  - Analog vs. digital
  - LED / Color Sensor
- How can we give signals meaning? How can we transfer text, images, sound and other file formats?

13 Protokolle

13.1 Peer-To-Peer

13.2 Handshake

13.3 TCP/IP & HTTPs

- How can we communicate between two computers?
  - Protocols / Handshake
  - Netzwerk / Internet / P2P-Communication

14 Verschlüsselung

14.1 Verschlüsselung

14.1.1 Symmetrisch

- Symmetric encryption with Caesar Cipher

14.1.2 Asymmetrisch

- Assymetric with RSA Toy

14.2 Digitale Signaturen

- Digital signatures

15 Kompression

15.1 Verlustfreie Kompression

- Huffman

15.2 Verlustbehaftete Kompression

- JPEG

16

Index