universität wien

DIPLOMARBEIT / DIPLOMA THESIS

Titel der Diplomarbeit / Title of the Diploma Thesis

Diskrete Mathematik und Algorithmen im Kontext Schule

verfasst von / submitted by

Dipl.-Ing. Dr. rer. nat. Lukas Riegler

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magister der Naturwissenschaften (Mag. rer. nat.)

Wien, 2020 / Vienna, 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 190 406 884
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Lehramtsstudium UF Mathematik UF Informatik und Informatikmanagement
Betreut von / Supervisor:	Univ.-Prof. Mag. Dr. Ilse Fischer, Privatdoz.

Inhaltsverzeichnis

Zusammenfassung

In Österreich stehen der Funktionsbegriff und die Analysis im Zentrum des Mathematik-Unterrichts der Sekundarstufe II. Im Gegensatz dazu spielen diskrete Mathematik und Algorithmen nur eine untergeordnete Rolle. In der vorliegenden Diplomarbeit wurden ausgewählte Themen der diskreten Mathematik in Form von Arbeitsblättern für den unterstützenden Einsatz im Unterricht aufbereitet.

Im ersten Teil der Diplomarbeit wird das im Jahr 1978 publizierte RSA-Verfahren behandelt, welches einen Meilenstein der modernen Kryptographie darstellt. Diese Methode ermöglicht nämlich die Verschlüsselung, Übertragung und Entschlüsselung von Nachrichten, ohne dass die kommunizierenden Personen zuvor einen geheimen Schlüssel austauschen müssen. Das RSA-Verfahren baut dabei auf Algorithmen und zahlentheoretischen Resultaten auf, die vor mehr als 200 Jahren – teilweise sogar vor mehr als 2000 Jahren – entdeckt wurden. Die ersten 8 Arbeitsblätter sollen es ermöglichen, das RSA-Verfahren und die zahlentheoretischen Hintergründe ohne zugehöriges Vorwissen lückenlos zu erfassen.

Algorithmen und deren Implementierung in einer Programmiersprache sind von den österreichischen Lehrplänen für den Informatik-Unterricht der Sekundarstufe II vorgesehen. Im zweiten Teil der Diplomarbeit liegt der Fokus auf Algorithmen im Bereich der diskreten Mathematik, die sich durch eine Kombination aus geringer Einstiegshürde und hoher Praxisrelevanz auszeichnen. Neben einem einführenden Arbeitsblatt zu Algorithmen wurden weitere Materialien zu Sortieralgorithmen, dem Kruskal-Algorithmus und dem Dijkstra-Algorithmus für den Unterricht aufbereitet.

Abstract

Functions and calculus are main parts of Austria's mathematics curricula in upper secondary schools. On the contrary, discrete mathematics as well as algorithms play a subordinate role. In the present diploma thesis several worksheets were created to support teaching selected topics of discrete mathematics.

The first part of this thesis is dedicated to the RSA algorithm published in 1978, representing a milestone of modern cryptography. This method enabled the encryption, transmission and decryption of messages without the need for communicating parties to exchange a secret key in advance. RSA cryptography is based on algorithms and number-theoretic results older than 200 years – partially even older than 2000 years. The goal of the first 8 worksheets is to enable interested students to fully understand how and why the RSA algorithm works without any prior knowledge in this field.

Algorithms and their implementation in a programming language are part of Austria's computer science curricula. The second part of this diploma thesis focuses on selected algorithms related with discrete mathematics, which excel in being both relevant in practice and requiring no prior knowledge to study. Apart from an introductory worksheet on algorithms additional materials for teaching sorting algorithms, Kruskal's algorithm and Dijkstra's algorithm were created.

Danksagung

Im Laufe der letzten 15 Jahre durfte ich zahlreiche Menschen kennenlernen, die ihre Begeisterung für diskrete Mathematik nicht nur über ihre Forschung, sondern auch über ihre Lehre ausstrahlen. Insbesondere möchte ich mich an dieser Stelle (wieder!) bei Ilse Fischer für ihr Engagement über die vielen Jahre bedanken!

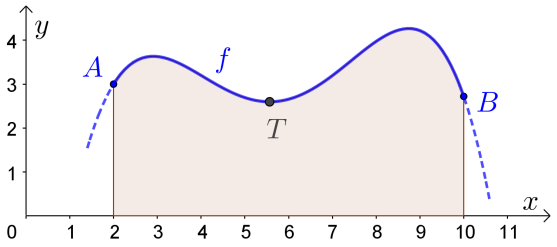
Ohne die gesammelten Erfahrungen der letzten Jahre wäre die Diplomarbeit in dieser Form nicht möglich gewesen. Einerseits danke ich allen SchülerInnen, die mir regelmäßig weiteren Ansporn liefern, meinen Unterricht und die eingesetzten Materialien weiterzuentwickeln. Andererseits hat dabei Michael Eichmair mit dem Projekt „Mathematik macht Freu(n)de“ eine Schlüsselrolle gespielt. Ich danke dir für deinen unermüdlichen Einsatz für das Projekt, von dem eine unglaubliche Anzahl an SchülerInnen, Studierenden und Lehrpersonen profitieren!

Schließlich waren die letzten Jahre von intensiver Arbeit an Schule und Universität geprägt. Ich danke allen KollegInnen, FreundInnen und besonders meiner Familie, die mir immer unterstützend zur Seite gestanden ist!

Wien, 30.12.2019

1 Einleitung

Im Zentrum der „kontinuierlichen“ Mathematik, die in der Schule fest verankert ist, stehen zum Beispiel Funktionen $f\colon \mathbb{R} \to \mathbb{R}$ und ihre Eigenschaften:



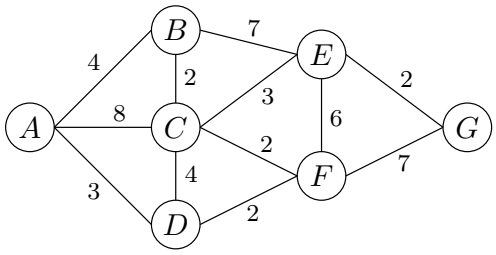
- Wo ist der größte/kleinste Funktionswert von f in $[2;10]$?
- Welche Koordinaten hat der Tiefpunkt T?
- An welchen Stellen ändert sich das Krümmungsverhalten von f?
- Wie groß ist der Inhalt des markierten Flächenstücks?
- Das markierte Flächenstück wird um die x-Achse rotiert.
  Welches Volumen hat der Rotationskörper?
- Welche Länge hat das Kurvenstück vom Punkt A zum Punkt B?
- Was ist der „durchschnittliche“ Funktionswert von f in $[2;10]$?

Im 19. Jahrhundert zeigte Georg Cantor zuerst, dass es gleich viele[1] natürliche Zahlen wie rationale Zahlen gibt, nämlich „abzählbar unendlich“ viele. Danach bewies er, dass es mehr[2] reelle Zahlen als natürliche Zahlen gibt, nämlich „überabzählbar unendlich“ viele ([8]). Bei der Funktion f oben untersuchen wir die überabzählbar unendlich vielen Zahlen im Intervall $[2;10]$ und ihre überabzählbar unendlich vielen Funktionswerte.

Im Zentrum der diskreten Mathematik stehen im Gegensatz dazu solche Strukturen, die sich aus endlich vielen oder abzählbar unendlich vielen Objekten zusammensetzen. Derartige Strukturen treten in zahlreichen Disziplinen der Mathematik und Informatik auf, zum Beispiel Kombinatorik, Graphentheorie, Zahlentheorie, Logik, Mengenlehre, Topologie, Numerik, Spieltheorie, Kodierungstheorie, Kryptographie, Wahrscheinlichkeitsrechnung, Algorithmen und deren Analyse.

[1] Es gibt eine Bijektion $f\colon \mathbb{N} \to \mathbb{Q}$.
[2] Es kann keine Bijektion $f\colon \mathbb{N} \to \mathbb{R}$ geben.

Im nachstehenden Bild ist ein Graph dargestellt, der die Knoten A, B, C, D, E, F und G enthält. Manche dieser Knoten sind direkt durch eine Kante verbunden.



In der **Graphentheorie** untersuchen wir zum Beispiel die folgenden Fragen:

- Jeder Knoten soll eingefärbt werden. Wie viele Farben sind mindestens notwendig, wenn benachbarte Knoten verschiedene Farben haben müssen?
- Jede Kante soll eingefärbt werden. Wie viele Farben sind mindestens notwendig, wenn benachbarte Kanten verschiedene Farben haben müssen?
- Die Zahlen auf den Kanten geben die Entfernung der Knoten voneinander an. Welche Gesamtlänge hat der kürzeste Weg von Knoten A zu Knoten G?
- Gibt es einen Weg von Knoten A zu Knoten A, der jeden anderen Knoten *genau* einmal besucht?
- Gibt es einen Weg von Knoten A zu Knoten A, der jede Kante *genau* einmal überquert?

In der abzählenden **Kombinatorik** untersuchen wir zum Beispiel die folgenden Fragen:

- Wie viele Lotto-Tipps muss man bei einer Ziehung von „Lotto 6 aus 45“ abgeben, um *sicher* den Jackpot zu knacken?
- Wie wahrscheinlich ist es, dass bei Roulette 10 Mal hintereinander ein rotes Feld ausgewählt wird?
- Wie wahrscheinlich ist es, bei einem gewöhnlichen 52-Karten-Deck einen „Black Jack“ zu ziehen?

Eine Gemeinsamkeit aller dieser Disziplinen ist, dass sie in der Geschichte der Mathematik vergleichsweise jung sind. Ihre systematische Erforschung begann teilweise erst im 20. Jahrhundert.

In den USA empfahl das „National Council of Teachers of Mathematics“ (NCTM) in den 1980er-Jahren eine weitreichende Aufnahme von diskreter Mathematik in die Schul-Curricula:

> *„The* 1987 *NCTM Curriculum Standards for grades* 9 *to* 12 *include discrete mathematics. The Standards state that students should receive a study of probability, statistics, and discrete mathematics before the formal study of calculus begins. The Standards call for a shift in emphasis from memorization of facts and paper and pencil skills to one of conceptual understanding, mathematical modelling, and mathematical problem solving.“* [10]

Auch in Großbritannien wurde zu dieser Zeit festgestellt, dass Problemlösen im Mathematik-Unterricht zu wenig Raum findet:

> *„The recent U.K. committee of inquiry, chaired by Dr. W. Cockcroft into the teaching of mathematics at secondary school, in paragraph* 243*, claimed that good mathematics teaching should consist of the six components: (i) exposition, (ii) practice of routines, (iii) discussion, (iv) investigations, (v) problem solving, (vi) applications. We all know that there is great emphasis on (i) and (ii), but (iii)-(vi) are rarely seen in the mathematics classroom.“* [6]

Im Jahr 2000 veröffentlichte das NCTM die „Principles and Standards for School Mathematics“. Darin wird Problemlösen als Standard für Kinder ab 3 Jahren definiert [21]:

> *„Solving problems is not only a goal of learning mathematics but also a major means of doing so.* [. . .] *Instructional programs from prekindergarten through grade* 12 *should enable all students to –*
>
> - *build new mathematical knowledge through problem solving;*
> - *solve problems that arise in mathematics and in other contexts;*
> - *apply and adapt a variety of appropriate strategies to solve problems;*
> - *monitor and reflect on the process of mathematical problem solving.“*

Diese NCTM-Standards fanden auch im deutschen Sprachraum Anklang:

> *„Bei den Inhalten fällt auf, dass die Standards stärker als traditionelle Curricula Gebiete wie diskrete Mathematik, Wahrscheinlichkeitsrechnung und Statistik hervorheben.* [. . .] *Die NCTM-Standards sind ein herausragendes Dokument der gegenwärtigen Diskussion in der Mathematikdidaktik. Manche ihrer Vorgaben und Zielformulierungen sind recht optimistisch formuliert, bezogen auf einen Verstehenshorizont der* alle *Schüler einbeziehen will. Die Standards sind ein wichtiger Wegweiser für die schulische Lehre von Mathematik zu Beginn des dritten Jahrtausends.“* [11]

In [25, 26] werden folgende Gründe angeführt, warum sich diskrete Mathematik zur Erfüllung dieser Standards anbietet:

1. *Discrete mathematics provides valuable information an tools.*
2. *Discrete mathematics facilitates a focus on problem-solving and reasoning at all grade levels.*
3. *Discrete mathematics offers students a new start in mathematics.*
4. *Discrete mathematics offers an opportunity to revitalize school mathematics.*

„Discrete mathematics offers a new start for students. For the student who has been unsuccessful with mathematics, it offers the possibility for success. For the talented student who has lost interest in mathematics, it offers the possibility of challenge.“

Tatsächlich zeichnet sich diskrete Mathematik auch aus meiner Sicht durch solche Aufgabenstellungen aus, für die einerseits praktisch kein mathematisches Vorwissen notwendig ist und die andererseits jeden erdenklichen Schwierigkeitsgrad haben können:

Beispiel: Wähle eine beliebige natürliche Zahl $n \geq 1$.
Falls die Zahl gerade ist, dividiere sie durch 2.
Falls die Zahl ungerade ist, multipliziere sie mit 3 und addiere 1.
Wiederhole diesen Schritt für das Ergebnis immer wieder, zum Beispiel:

$$23 \to 70 \to 35 \to 106 \to 53 \to 160 \to 80 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

Startet man mit der Zahl $n = 23$, landet man nach 15 Schritten bei der Zahl 1.
Die Collatz-Vermutung besagt, dass man – unabhängig vom gewählten Startwert $n \geq 1$ – schließlich bei der Zahl 1 landet.
Seit mehr als 50 Jahren konnte diese Vermutung *nicht* bewiesen werden.

Neben innermathematischen Fragen bietet die diskrete Mathematik eine Vielfalt anwendungsbezogener Aufgaben, die sich auch für den Schulunterricht eignen ([33, 35]).

Beispiel: Wie funktionieren Barcodes? Warum kann der Scanner den Strichcode auch um 180° gedreht richtig auslesen?

Beim händischen Eintippen passiert an einer Stelle ein Fehler. Wird dieser Fehler *sicher* erkannt?

Beim händischen Eintippen werden zwei benachbarte Ziffern vertauscht. Wird ein solcher „Zahlendreher“ *sicher* erkannt?

Die NCTM-Standards aus dem Jahr 2000 empfehlen diese Schnittstellen mit diskreter Mathematik verstärkt in den Mathematik-Lehrplänen zu verankern:

> *„As an active branch of contemporary mathematics that is widely used in business and industry, discrete mathematics should be an integral part of the school mathematics curriculum, and these topics naturally occur throughout the other strands of mathematics.“* [21]

Dem Spiralprinzip folgend sehe ich dabei in der Schule auch die Gelegenheit, um Zusammenhänge zwischen diskreter und kontinuierlicher Mathematik herzustellen.

Beispiel: Gegeben ist die quadratische Gleichung $x^2 + p \cdot x + q = 0$.

a) Die Koeffizienten p und q werden jeweils zufällig mit einem fairen Spielwürfel mit Augenzahlen $\{1, 2, 3, 4, 5, 6\}$ festgelegt. Wie groß ist die Wahrscheinlichkeit, dass die zugehörige quadratische Gleichung mindestens eine Lösung in $\mathbb{Z}$ hat?

b) Die Koeffizienten p und q werden jeweils zufällig gleichverteilt im Intervall $[1; 6]$ festgelegt. Wie groß ist die Wahrscheinlichkeit, dass die zugehörige quadratische Gleichung mindestens eine Lösung in $\mathbb{R}$ hat?

Die Plausibilität der berechneten Wahrscheinlichkeiten kann dann auch mit einem Tabellenkalkulationsprogramm oder durch Implementierung eines Algorithmus überprüft werden.

Die folgenden Sätze beschreiben die Intention der vorliegenden Arbeit bestens:

> *„Discrete mathematics, in contrast to geometry, for instance, does not have a centuries-old tradition that helps (though sometimes also hinders) its teaching. Discrete mathematics as a school subject is being established and shaped literally before our eyes. Any teacher, including the one who is only just embarking on this career, is capable of enriching it.“* [35]

Die im Rahmen dieser Diplomarbeit erstellten Materialien zur Zahlentheorie, dem RSA-Verfahren, Sortieralgorithmen, dem Kruskal-Algorithmus und dem Dijkstra-Algorithmus stellen einen solchen Versuch dar, eine (kleine) Auswahl an Themen im Bereich der diskreten Mathematik zugänglich aufzubereiten.

2 Zahlentheorie & RSA-Verfahren

Als Kinder lernen wir, wie man (kleine) natürliche Zahlen händisch in Primfaktoren zerlegen kann:

$$84 = 2 \cdot 2 \cdot 3 \cdot 7$$

Sortiert man die Primfaktoren aufsteigend nach ihrem Wert, dann ist die Zerlegung für jede natürliche Zahl $n \geq 2$ eindeutig.

Auch wenn wir die Primfaktorzerlegung in der 5. Schulstufe lernen, ist ihre Existenz und Eindeutigkeit keineswegs trivial. Zum Beispiel sind 1987, 2383, 2027 und 2333 jeweils Primzahlen. Es ist aber (auf den ersten Blick) *nicht* offensichtlich, warum $1987 \cdot 2383$ und $2027 \cdot 2333$ verschiedene Zahlen sein müssen. Der Fundamentalsatz der Arithmetik, der die Existenz und Eindeutigkeit der Primfaktorzerlegung aussagt, ist auf den ersten drei Arbeitsblättern aufbereitet.

Das erste Arbeitsblatt zur Teilbarkeit und Primfaktorzerlegung ist für den Mathematikunterricht der 9. Schulstufe konzipiert, in dem die zahlentheoretischen Grundbegriffe aus der 5. Schulstufe wiederaufgegriffen werden. Die beiden folgenden Arbeitsblätter zum Euklidischen Algorithmus und dem Fundamentalsatz der Arithmetik richten sich an interessierte SchülerInnen und können zum Beispiel im Rahmen eines Wahlpflichtfachs behandelt werden.

Hinter dem RSA-Verfahren steckt mathematisch der Satz von Euler, welcher den Kleinen Satz von Fermat verallgemeinert. Die dafür benötigten Begriffe werden auf dem vierten Arbeitsblatt Kongruenz und Restklassen eingeführt. Auch wenn der Kleine Satz von Fermat so wie der Satz von Euler ohne vollständige Induktion und den Binomischen Lehrsatz bewiesen werden kann, bietet sich eine Behandlung dieser beiden Arbeitsblätter zu diesem Zeitpunkt an. Der Kleine Satz von Fermat und der Satz von Euler sowie deren Beweise sind Inhalt des darauffolgenden Arbeitsblatts. Am achten Arbeitsblatt wird schließlich das RSA-Verfahren behandelt.

Die ersten vier Arbeitsblätter bilden die Grundlage, um zu verstehen, *wie* das RSA-Verfahren funktioniert. Durch Kombination aller acht Arbeitsblätter kann schließlich lückenlos verstanden werden, *warum* das RSA-Verfahren funktioniert.

2.1 Teilbarkeit und Primfaktorzerlegung

Die natürlichen Zahlen, Primzahlen und Teilbarkeitsfragen spielen in der Zahlentheorie eine zentrale Rolle. Euklid fand bereits vor rund 2300 Jahren einen Beweis dafür, dass es *unendlich* viele Primzahlen gibt ([37], Book IX, Prop. 20).

In den österreichischen Lehrplänen sind zahlentheoretische Grundlagen in der 5. Schulstufe verankert: SchülerInnen sollen *„anhand von Teilern und Vielfachen Einblicke in Zusammenhänge zwischen natürlichen Zahlen gewinnen“* ([29], [30]). In der 5. Schulstufe lernen SchülerInnen, wie man algorithmisch Primfaktorzerlegungen und daraus den größten gemeinsamen Teiler und das kleinste gemeinsame Vielfache zweier natürlicher Zahlen berechnen kann.

Diese Grundlagen werden in der 9. Schulstufe wieder aufgegriffen: SchülerInnen sollen *„mit Primzahlen und Teilern arbeiten können“* und *„Teilbarkeitsfragen untersuchen können“* (AHS, [29]) bzw. *„den Aufbau von Zahlensystemen wiedergeben und die Erweiterung der Zahlenbereiche argumentieren“* können (BHS, HTL, [31]).
Zu diesem Zeitpunkt können die Algorithmen nicht nur wiederholt, sondern auch kritisch hinterfragt werden. *Warum* liefern diese Algorithmen eigentlich den größten gemeinsamen Teiler bzw. das kleinste gemeinsame Vielfache?

Das Arbeitsblatt setzt stillschweigend die Eindeutigkeit der Primfaktorzerlegung voraus. Dieser Fundamentalsatz der Arithmetik ist auf einem späteren Arbeitsblatt aufbereitet.

Es folgt das Arbeitsblatt – Teilbarkeit und Primfaktorzerlegung und die Ausarbeitung. Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Lernziele:

- ✓ Was sind die **Teiler** bzw. **Vielfache** einer natürlichen Zahl?
- ✓ Was sind **Primzahlen**?
- ✓ Was ist die **Primfaktorzerlegung** einer natürlichen Zahl?
- ✓ Was ist der **größte gemeinsame Teiler** zweier natürlicher Zahlen?
  Warum liefert der Algorithmus, den wir in der 5. Schulstufe lernen, tatsächlich den größten gemeinsamen Teiler?
- ✓ Was ist das **kleinste gemeinsame Vielfache** zweier natürlicher Zahlen?
  Warum liefert der Algorithmus, den wir in der 5. Schulstufe lernen, tatsächlich das kleinste gemeinsame Vielfache?

Teiler

a und b sind natürliche Zahlen mit $a \neq 0$. $a, b \in \mathbb{N},\ a \neq 0$

Bleibt bei der Division $b : a$ *kein* Rest, dann nennen wir a einen **Teiler** von b.

Wir schreiben dafür kurz $\boldsymbol{a \mid b}$. Sprechweisen: „a teilt b“, „b ist durch a teilbar“

Bleibt bei der Division $b : a$ ein positiver Rest, dann ist a kein Teiler von b, und wir schreiben $a \nmid b$.

Teilbarkeit

Entscheide jeweils, ob $\mid$ oder $\nmid$ stimmt: **a)** 3 ☐ 21 **b)** 12 ☐ 4 **c)** 1 ☐ 8 **d)** 4 ☐ 18 **e)** 42 ☐ 42

Primzahlen

Jede natürliche Zahl, die *genau* zwei Teiler hat, heißt **Primzahl**.

Ermittle die 10 kleinsten Primzahlen: $\mathbb{P} = \{ \square, \square, \square, \square, \square, \square, \square, \square, \square, \square, \ldots \}$

Die 3 Punkte (...) deuten an, dass es *unendlich* viele Primzahlen gibt. Es ist *nicht* offensichtlich, dass das stimmt.

Die Zahl 424243 könnte zum Beispiel viele verschiedene Teiler haben. Tatsächlich hat sie aber nur 2 Teiler.

Euklid fand vor mehr als 2000 Jahren einen Beweis dafür, dass es tatsächlich unendlich viele Primzahlen gibt.

Die größte *bekannte* Primzahl (Stand: 2018) ist $2^{82\,589\,933} - 1$. Sie hat 24 862 048 Ziffern und füllt damit rund 25 000 Buchseiten.

Teilbarkeitsregeln

Mit Hilfe der **Teilbarkeitsregeln** können wir für bestimmte Primzahlen schneller entscheiden, ob sie eine gegebene (große) Zahl teilen: Mehr dazu findest du am Arbeitsblatt – Kongruenz und Restklassen.

- **Teilbarkeit durch 2**:
  Eine natürliche Zahl ist genau dann durch 2 teilbar, wenn die Einerziffer 0, 2, 4, 6 oder 8 ist.
  Zum Beispiel: **a)** 2 ☐ 1395663458 **b)** 2 ☐ 475682348**7**
- **Teilbarkeit durch 3**:
  Eine natürliche Zahl ist genau dann durch 3 teilbar, wenn ihre Ziffernsumme durch 3 teilbar ist.
  Zum Beispiel: **a)** 3 ☐ 6435, weil 3 ☐☐. **b)** 3 ☐ 728, weil 3 ☐☐.
- **Teilbarkeit durch 5**:
  Eine natürliche Zahl ist genau dann durch 5 teilbar, wenn die Einerziffer 0 oder 5 ist.
  Zum Beispiel: **a)** 5 ☐ 75463410**5** **b)** 5 ☐ 542167**1**

Primfaktorzerlegung

Jede natürliche Zahl $n > 1$ kann als Produkt von Primzahlen geschrieben werden.
Zum Beispiel: $42 = 2 \cdot 3 \cdot 7$ oder $23\,061\,987 = 3 \cdot 3 \cdot 13 \cdot 439 \cdot 449$
Diese Zerlegung ist für jede Zahl eindeutig bis auf die Reihenfolge der Faktoren.

Mehr dazu findest du am Arbeitsblatt – Fundamentalsatz der Arithmetik.

Primfaktorzerlegung

Mit dem folgenden Verfahren können wir eine Zerlegung in Primfaktoren ermitteln:

1) Dividiere so oft wie möglich durch 2 ohne Rest.

2) Dividiere so oft wie möglich durch 3 ohne Rest.

3) Dividiere so oft wie möglich durch 5 ohne Rest.

4) Setze mit den weiteren Primzahlen fort, bis das Ergebnis 1 ist.

150	2	$2 \mid 150$	$150 = 2 \cdot 75$
75	3	$2 \nmid 75$, aber $3 \mid 75$	$150 = 2 \cdot 3 \cdot 25$
25	5	$3 \nmid 25$, aber $5 \mid 25$	$150 = 2 \cdot 3 \cdot 5 \cdot 5$
5	5	$5 \mid 5$	
1			

Eine Primfaktorzerlegung von 150 ist also $150 = 2 \cdot 3 \cdot 5 \cdot 5$.

Teiler

a und b sind natürliche Zahlen mit $a \neq 0$. $a, b \in \mathbb{N},\ a \neq 0$

Bleibt bei der Division $b : a$ *kein* Rest, dann nennen wir a einen **Teiler** von b.

Wir schreiben dafür kurz $\boldsymbol{a \mid b}$. Sprechweisen: „a teilt b“, „b ist durch a teilbar“

Bleibt bei der Division $b : a$ ein positiver Rest, dann ist a kein Teiler von b, und wir schreiben $a \nmid b$.

Teilbarkeit

Entscheide jeweils, ob $\mid$ oder $\nmid$ stimmt: **a)** $3 \mid 21$ **b)** $12 \nmid 4$ **c)** $1 \mid 8$ **d)** $4 \nmid 18$ **e)** $42 \mid 42$

Primzahlen

Jede natürliche Zahl, die *genau* zwei Teiler hat, heißt **Primzahl**.

Ermittle die 10 kleinsten Primzahlen: $\mathbb{P} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots\}$

Die 3 Punkte (…) deuten an, dass es *unendlich* viele Primzahlen gibt. Es ist *nicht* offensichtlich, dass das stimmt.

Die Zahl 424243 könnte zum Beispiel viele verschiedene Teiler haben. Tatsächlich hat sie aber nur 2 Teiler.

Euklid fand vor mehr als 2000 Jahren einen Beweis dafür, dass es tatsächlich unendlich viele Primzahlen gibt.

Die größte *bekannte* Primzahl (Stand: 2018) ist $2^{82\,589\,933} - 1$. Sie hat 24 862 048 Ziffern und füllt damit rund 25 000 Buchseiten.

Teilbarkeitsregeln

Mit Hilfe der **Teilbarkeitsregeln** können wir für bestimmte Primzahlen schneller entscheiden, ob sie eine gegebene (große) Zahl teilen: Mehr dazu findest du am Arbeitsblatt – Kongruenz und Restklassen.

- **Teilbarkeit durch 2**:
  Eine natürliche Zahl ist genau dann durch 2 teilbar, wenn die Einerziffer 0, 2, 4, 6 oder 8 ist.
  Zum Beispiel: **a)** $2 \mid 1395663458$ **b)** $2 \nmid 4756823487$
- **Teilbarkeit durch 3**:
  Eine natürliche Zahl ist genau dann durch 3 teilbar, wenn ihre Ziffernsumme durch 3 teilbar ist.
  Zum Beispiel: **a)** $3 \mid 6435$, weil $3 \mid 18$. **b)** $3 \nmid 728$, weil $3 \nmid 17$.
- **Teilbarkeit durch 5**:
  Eine natürliche Zahl ist genau dann durch 5 teilbar, wenn die Einerziffer 0 oder 5 ist.
  Zum Beispiel: **a)** $5 \mid 7546341050$ **b)** $5 \nmid 5421671$

Primfaktorzerlegung

Jede natürliche Zahl $n > 1$ kann als Produkt von Primzahlen geschrieben werden.
Zum Beispiel: $42 = 2 \cdot 3 \cdot 7$ oder $23\,061\,987 = 3 \cdot 3 \cdot 13 \cdot 439 \cdot 449$
Diese Zerlegung ist für jede Zahl eindeutig bis auf die Reihenfolge der Faktoren.

Mehr dazu findest du am Arbeitsblatt – Fundamentalsatz der Arithmetik.

Primfaktorzerlegung

Mit dem folgenden Verfahren können wir eine Zerlegung in Primfaktoren ermitteln:

1) Dividiere so oft wie möglich durch 2 ohne Rest.

2) Dividiere so oft wie möglich durch 3 ohne Rest.

3) Dividiere so oft wie möglich durch 5 ohne Rest.

4) Setze mit den weiteren Primzahlen fort, bis das Ergebnis 1 ist.

150	2	$2 \mid 150$	$150 = 2 \cdot 75$
75	3	$2 \nmid 75$, aber $3 \mid 75$	$150 = 2 \cdot 3 \cdot 25$
25	5	$3 \nmid 25$, aber $5 \mid 25$	$150 = 2 \cdot 3 \cdot 5 \cdot 5$
5	5	$5 \mid 5$	
1			

Eine Primfaktorzerlegung von 150 ist also $150 = 2 \cdot 3 \cdot 5 \cdot 5$.

Primfaktorzerlegung

Schreibe die Zahlen 180, 350 und 495 jeweils als Produkt von Primfaktoren.

Alle Teiler

Wie viele Teiler hat die Zahl 150? Du könntest alle Zahlen von 1 bis 150 durchprobieren. Das dauert aber lange.
Die Zahlen 1 und 150 sind Teiler von 150.
Um die anderen Teiler zu ermitteln, zerlegen wir 150 in Primfaktoren: $150 = 2 \cdot 3 \cdot 5 \cdot 5$
Welche Primzahlen sind Teiler von 150? Die Primfaktorzerlegung ist bis auf die Reihenfolge der Faktoren eindeutig.

Welche Produkte von 2 Primzahlen sind Teiler von 150?

Welche Produkte von 3 Primzahlen sind Teiler von 150?

150 hat also insgesamt _____ Teiler. Jede Auswahl von Primfaktoren liefert einen Teiler.

Teileranzahl

Für jede natürliche Zahl n kürzen wir mit $d(n)$ die Anzahl der Teiler von n ab. Zum Beispiel: $d(150) = 12$
Wenn p eine Primzahl ist, dann gilt:

$d(p) =$ _____ $\quad d(p \cdot p) =$ _____ $\quad d(p \cdot p \cdot p) =$ _____ $\quad d(p^k) = d(\underbrace{p \cdot p \cdot \ldots \cdot p}_{k \text{ Faktoren}}) =$ _____

Wenn die natürlichen Zahlen m und n *keinen* gemeinsamen Teiler außer 1 haben, dann gilt:

$$d(m \cdot n) = d(m) \cdot d(n)$$

Um einen Teiler von $m \cdot n$ zu erhalten, kannst du *jeden* Teiler von m mit *jedem* Teiler von n multiplizieren. Dafür gibt es $d(m) \cdot d(n)$ Möglichkeiten. Jeder Teiler von $m \cdot n$ kann eindeutig in einen Teiler von m und einen Teiler von n zerlegt werden.

Aus der Primfaktorzerlegung $150 = 2 \cdot 3 \cdot 5 \cdot 5$ können wir damit schneller die Teileranzahl berechnen:

$$d(150) = d(2 \cdot 3 \cdot 5 \cdot 5) = d(2) \cdot d(3 \cdot 5 \cdot 5) = \underbrace{d(2)}_{\square} \cdot \underbrace{d(3)}_{\square} \cdot \underbrace{d(5 \cdot 5)}_{\square} = \text{_____}$$

Primfaktorzerlegung

Schreibe die Zahlen 180, 350 und 495 jeweils als Produkt von Primfaktoren.

180	2
90	2
45	3
15	3
5	5
1	

$\Longrightarrow 180 = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5$

350	2
175	5
35	5
7	7
1	

$\Longrightarrow 350 = 2 \cdot 5 \cdot 5 \cdot 7$

495	3
165	3
55	5
11	11
1	

$\Longrightarrow 495 = 3 \cdot 3 \cdot 5 \cdot 11$

Alle Teiler

Wie viele Teiler hat die Zahl 150? Du könntest alle Zahlen von 1 bis 150 durchprobieren. Das dauert aber lange.
Die Zahlen 1 und 150 sind Teiler von 150.
Um die anderen Teiler zu ermitteln, zerlegen wir 150 in Primfaktoren: $150 = 2 \cdot 3 \cdot 5 \cdot 5$
Welche Primzahlen sind Teiler von 150? Die Primfaktorzerlegung ist bis auf die Reihenfolge der Faktoren eindeutig.
2, 3, 5

Welche Produkte von 2 Primzahlen sind Teiler von 150?
$2 \cdot 3 = 6 \quad 2 \cdot 5 = 10 \quad 3 \cdot 5 = 15 \quad 5 \cdot 5 = 25$

Welche Produkte von 3 Primzahlen sind Teiler von 150?
$2 \cdot 3 \cdot 5 = 30 \quad 2 \cdot 5 \cdot 5 = 50 \quad 3 \cdot 5 \cdot 5 = 75$

150 hat also insgesamt 12 Teiler. Jede Auswahl von Primfaktoren liefert einen Teiler.

Teileranzahl

Für jede natürliche Zahl n kürzen wir mit $d(n)$ die Anzahl der Teiler von n ab. Zum Beispiel: $d(150) = 12$
Wenn p eine Primzahl ist, dann gilt:

$$d(p) = 2 \qquad d(p \cdot p) = 3 \qquad d(p \cdot p \cdot p) = 4 \qquad d(p^k) = d(\underbrace{p \cdot p \cdot \ldots \cdot p}_{k \text{ Faktoren}}) = k + 1$$

Wenn die natürlichen Zahlen m und n *keinen* gemeinsamen Teiler außer 1 haben, dann gilt:

$$d(m \cdot n) = d(m) \cdot d(n)$$

Um einen Teiler von $m \cdot n$ zu erhalten, kannst du *jeden* Teiler von m mit *jedem* Teiler von n multiplizieren. Dafür gibt es $d(m) \cdot d(n)$ Möglichkeiten. Jeder Teiler von $m \cdot n$ kann eindeutig in einen Teiler von m und einen Teiler von n zerlegt werden.

Aus der Primfaktorzerlegung $150 = 2 \cdot 3 \cdot 5 \cdot 5$ können wir damit schneller die Teileranzahl berechnen:

$$d(150) = d(2 \cdot 3 \cdot 5 \cdot 5) = d(2) \cdot d(3 \cdot 5 \cdot 5) = \underbrace{d(2)}_{2} \cdot \underbrace{d(3)}_{2} \cdot \underbrace{d(5 \cdot 5)}_{3} = 12$$

Größter gemeinsamer Teiler

Der **größte gemeinsame Teiler** zweier natürlichen Zahlen m und n ist die größte natürliche Zahl, die beide Zahlen m und n teilt. Wir schreiben dafür kurz $\mathbf{ggT(m, n)}$.
Wenn $\mathbf{ggT(m, n) = 1}$ gilt, dann sagen wir auch: „m und n sind **teilerfremd**“.

ggT berechnen

Was ist der größte gemeinsame Teiler von 270 und 315?

Du könntest alle Zahlen von 1 bis 270 durchprobieren. Das dauert aber lange.

Aus den Primfaktorzerlegungen von 270 und 315 können wir gemeinsame Teiler ablesen:

270	2	315	3
135	3	105	3
45	3	35	5
15	3	7	7
5	5	1	
1			

$\Longrightarrow\ 270 = 2 \cdot \textcircled{3} \cdot \textcircled{3} \cdot 3 \cdot \textcircled{5}$

$\Longrightarrow\ 315 = \textcircled{3} \cdot \textcircled{3} \cdot \textcircled{5} \cdot 7$

Jede Auswahl von Primfaktoren liefert einen Teiler. $3 \cdot 3 \cdot 5$ ist also ein Teiler von 270 und ein Teiler von 315. Ansonsten gibt es keine gemeinsamen Primfaktoren.

Der größte gemeinsame Teiler von 270 und 315 ist also $\text{ggT}(270, 315) = 3 \cdot 3 \cdot 5 = 45$.

ggT berechnen

Berechne $\text{ggT}(180, 210)$, $\text{ggT}(180, 420)$ und den größten gemeinsamen Teiler von 180, 210 und 420.

$\text{ggT}(180, 210, 420)$

Kleinstes gemeinsames Vielfache

Wenn a ein Teiler von b ist, dann nennen wir umgekehrt b ein **Vielfaches** von a.
Das **kleinste gemeinsame Vielfache** zweier natürlichen Zahlen m und n ist die kleinste natürliche Zahl, die ein Vielfaches von beiden Zahlen m und n ist. Wir schreiben dafür kurz $\mathbf{kgV(m, n)}$.

Zum Beispiel: Vielfache von 4: $\{4, 8, 12, 16, 20, 24, 28, 32, 36, \ldots\}$
Vielfache von 6: $\{6, 12, 18, 24, 30, 36, \ldots\}$

Gemeinsame Vielfache von 4 und 6: $\{12, 24, 36, \ldots\}$
$\Longrightarrow\ \text{kgV}(4, 6) = 12$

kgV berechnen

Was ist das kleinste gemeinsame Vielfache von 420 und 450? Wir berechnen die Primfaktorzerlegungen:

420	2	450	2
210	2	225	3
105	3	75	3
35	5	25	5
7	7	5	5
1		1	

$\Longrightarrow\ 420 = \textcircled{2} \cdot \textcircled{2} \cdot 3 \cdot 5 \cdot \textcircled{7}$

$\Longrightarrow\ 450 = 2 \cdot \textcircled{3} \cdot \textcircled{3} \cdot \textcircled{5} \cdot \textcircled{5}$

Eine Zahl ist ein Vielfaches von 420, wenn ihre PFZ *zumindest* die Faktoren $2 \cdot 2 \cdot 3 \cdot 5 \cdot 7$ enthält.

Eine Zahl ist ein Vielfaches von 450, wenn ihre PFZ *zumindest* die Faktoren $2 \cdot 3 \cdot 3 \cdot 5 \cdot 5$ enthält.

Die Zahl $2 \cdot 2 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7$ ist die *kleinste* natürliche Zahl, die ein Vielfaches von 420 und von 450 ist.

Das kleinste gemeinsame Vielfache von 420 und 450 ist also $\text{kgV}(420, 450) = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 = 6300$.

Größter gemeinsamer Teiler

Der **größte gemeinsame Teiler** zweier natürlichen Zahlen m und n ist die größte natürliche Zahl, die beide Zahlen m und n teilt. Wir schreiben dafür kurz $\mathbf{ggT(m, n)}$.
Wenn $\mathbf{ggT(m, n) = 1}$ gilt, dann sagen wir auch: „m und n sind **teilerfremd**“.

ggT berechnen

Was ist der größte gemeinsame Teiler von 270 und 315?

Du könntest alle Zahlen von 1 bis 270 durchprobieren. Das dauert aber lange.

Aus den Primfaktorzerlegungen von 270 und 315 können wir gemeinsame Teiler ablesen:

270	2	315	3
135	3	105	3
45	3	35	5
15	3	7	7
5	5	1	
1			

$\Longrightarrow 270 = 2 \cdot \textcircled{3} \cdot \textcircled{3} \cdot 3 \cdot \textcircled{5}$

$\Longrightarrow 315 = \textcircled{3} \cdot \textcircled{3} \cdot \textcircled{5} \cdot 7$

Jede Auswahl von Primfaktoren liefert einen Teiler. $3 \cdot 3 \cdot 5$ ist also ein Teiler von 270 und ein Teiler von 315. Ansonsten gibt es keine gemeinsamen Primfaktoren.

Der größte gemeinsame Teiler von 270 und 315 ist also $\text{ggT}(270, 315) = 3 \cdot 3 \cdot 5 = 45$.

ggT berechnen

Berechne $\text{ggT}(180, 210)$, $\text{ggT}(180, 420)$ und den größten gemeinsamen Teiler von 180, 210 und 420.

$\text{ggT}(180, 210, 420)$

180	2	210	2	420	2
90	2	105	3	210	2
45	3	35	5	105	3
15	3	7	7	35	5
5	5	1		7	7
1				1	

$\Longrightarrow 180 = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5$

$\Longrightarrow 210 = 2 \cdot 3 \cdot 5 \cdot 7$

$\Longrightarrow 420 = 2 \cdot 2 \cdot 3 \cdot 5 \cdot 7$

$\text{ggT}(180, 210) = 2 \cdot 3 \cdot 5 = 30$ $\quad \text{ggT}(180, 420) = 2 \cdot 2 \cdot 3 \cdot 5 = 60$ $\quad \text{ggT}(180, 210, 420) = 2 \cdot 3 \cdot 5 = 30$

Kleinstes gemeinsames Vielfache

Wenn a ein Teiler von b ist, dann nennen wir umgekehrt b ein **Vielfaches** von a.
Das **kleinste gemeinsame Vielfache** zweier natürlichen Zahlen m und n ist die kleinste natürliche Zahl, die ein Vielfaches von beiden Zahlen m und n ist. Wir schreiben dafür kurz $\mathbf{kgV(m, n)}$.

Zum Beispiel:
Vielfache von 4: $\{4, 8, 12, 16, 20, 24, 28, 32, 36, \ldots\}$
Vielfache von 6: $\{6, 12, 18, 24, 30, 36, \ldots\}$
Gemeinsame Vielfache von 4 und 6: $\{12, 24, 36, \ldots\}$
$\Longrightarrow \text{kgV}(4, 6) = 12$

kgV berechnen

Was ist das kleinste gemeinsame Vielfache von 420 und 450? Wir berechnen die Primfaktorzerlegungen:

420	2	450	2
210	2	225	3
105	3	75	3
35	5	25	5
7	7	5	5
1		1	

$\Longrightarrow 420 = \textcircled{2} \cdot \textcircled{2} \cdot 3 \cdot 5 \cdot \textcircled{7}$

$\Longrightarrow 450 = 2 \cdot \textcircled{3} \cdot \textcircled{3} \cdot \textcircled{5} \cdot \textcircled{5}$

Eine Zahl ist ein Vielfaches von 420, wenn ihre PFZ *zumindest* die Faktoren $2 \cdot 2 \cdot 3 \cdot 5 \cdot 7$ enthält.

Eine Zahl ist ein Vielfaches von 450, wenn ihre PFZ *zumindest* die Faktoren $2 \cdot 3 \cdot 3 \cdot 5 \cdot 5$ enthält.

Die Zahl $2 \cdot 2 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7$ ist die *kleinste* natürliche Zahl, die ein Vielfaches von 420 und von 450 ist.

Das kleinste gemeinsame Vielfache von 420 und 450 ist also $\text{kgV}(420, 450) = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 = 6300$.

kgV berechnen

Berechne $\mathrm{kgV}(60, 63)$, $\mathrm{kgV}(60, 294)$ und das kleinste gemeinsame Vielfache von 60, 63 und 294.

$\mathrm{kgV}(60, 63, 294)$

$\mathrm{ggT}(m, n) \cdot \mathrm{kgV}(m, n) = m \cdot n$

Berechne $\mathrm{kgV}(42, 140)$ und $\mathrm{ggT}(42, 140)$.

$\mathrm{ggT}(42, 70) \cdot \mathrm{kgV}(42, 70) =$ _____ $42 \cdot 140 =$ _____ Warum gilt $\mathrm{ggT}(m, n) \cdot \mathrm{kgV}(m, n) = m \cdot n$?

Sieb des Eratosthenes

Das **Sieb des Eratosthenes** ist ein Verfahren, um alle Primzahlen bis zu einer bestimmten Zahl zu ermitteln. Verwende das Sieb des Eratosthenes, um alle Primzahlen bis 100 zu ermitteln:

1) Die Zahl 1 hat nur einen Teiler.
Sie ist also keine Primzahl. Streiche die Zahl 1 rechts durch.

2) Die Zahl 2 hat genau zwei Teiler.
Sie ist also eine Primzahl. Kreise die Zahl 2 rechts ein.
Alle Vielfachen von 2 sind keine Primzahlen.
Streiche sie durch.

3) Suche die kleinste Zahl, die weder durchgestrichen noch eingekreist ist. Diese Zahl ist ein Primzahl. Kreise sie ein.
Alle Vielfachen dieser Zahl sind keine Primzahlen.
Streiche sie durch.
Wiederhole Schritt **3)**, bis alle Zahlen markiert sind.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Kannst du erklären, warum jede eingekreiste Zahl *sicher* eine Primzahl ist?

kgV berechnen

Berechne kgV(60, 63), kgV(60, 294) und das kleinste gemeinsame Vielfache von 60, 63 und 294.

kgV(60, 63, 294)

60	2
30	2
15	3
5	5
1	

$\Longrightarrow 60 = 2 \cdot 2 \cdot 3 \cdot 5$

63	3
21	3
7	7
1	

$\Longrightarrow 63 = 3 \cdot 3 \cdot 7$

294	2
147	3
49	7
7	7
1	

$\Longrightarrow 294 = 2 \cdot 3 \cdot 7 \cdot 7$

$\text{kgV}(60, 63) = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5 \cdot 7 = 1260$ $\quad \text{kgV}(60, 294) = 2 \cdot 2 \cdot 3 \cdot 5 \cdot 7 \cdot 7 = 2940$

$\text{kgV}(60, 63, 294) = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5 \cdot 7 \cdot 7 = 8820$

$\text{ggT}(m, n) \cdot \text{kgV}(m, n) = m \cdot n$

Berechne kgV(42, 140) und ggT(42, 140).

42	2
21	3
7	7
1	

$\Longrightarrow 42 = 2 \cdot 3 \cdot 7$

140	2
70	2
35	5
7	7
1	

$\Longrightarrow 140 = 2 \cdot 2 \cdot 5 \cdot 7$

$\text{ggT}(42, 140) = 2 \cdot 7 = 14$ $\quad \text{kgV}(42, 140) = 2 \cdot 2 \cdot 3 \cdot 5 \cdot 7 = 420$

$\text{ggT}(42, 70) \cdot \text{kgV}(42, 70) = 5880$ $\quad 42 \cdot 140 = 5880$

Warum gilt $\text{ggT}(m, n) \cdot \text{kgV}(m, n) = m \cdot n$?

Sieb des Eratosthenes

Das **Sieb des Eratosthenes** ist ein Verfahren, um alle Primzahlen bis zu einer bestimmten Zahl zu ermitteln. Verwende das Sieb des Eratosthenes, um alle Primzahlen bis 100 zu ermitteln:

1) Die Zahl 1 hat nur einen Teiler.
Sie ist also keine Primzahl. Streiche die Zahl 1 rechts durch.

2) Die Zahl 2 hat genau zwei Teiler.
Sie ist also eine Primzahl. Kreise die Zahl 2 rechts ein.
Alle Vielfachen von 2 sind keine Primzahlen.
Streiche sie durch.

3) Suche die kleinste Zahl, die weder durchgestrichen noch eingekreist ist. Diese Zahl ist ein Primzahl. Kreise sie ein.
Alle Vielfachen dieser Zahl sind keine Primzahlen.
Streiche sie durch.
Wiederhole Schritt **3)**, bis alle Zahlen markiert sind.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Kannst du erklären, warum jede eingekreiste Zahl *sicher* eine Primzahl ist?

2.2 Euklidischer Algorithmus

Die größte *bekannte* Primzahl (Stand: Oktober 2019, [27]) ist $2^{82\,589\,933} - 1$ und besteht aus 24 862 048 Ziffern. Wir wissen, dass es unendlich viele Primzahlen gibt, die größer sind ([37], Book IX, Prop. 20). Allerdings ist mit den technisch verfügbaren Mitteln bis jetzt kein Verfahren bekannt, um die Primfaktorzerlegung von sehr großen natürlichen Zahlen *effizient* zu berechnen. In [4] befindet sich eine Übersicht zur geschichtlichen Entwicklung und den gängigen Primzahltests sowie Faktorisierungsverfahren.

In der Schule lernen wir, wie man den größten gemeinsamen Teiler zweier natürlicher Zahlen aus deren Primfaktorzerlegung berechnen kann. Mit dem Euklidischen Algorithmus ([37], Book VII, Prop. 1, Prop. 2) können wir den größten gemeinsamen Teiler zweier Zahlen auch ohne die Primfaktorzerlegungen berechnen.

Ein möglicher Einstieg zu diesem Thema mit der Programmiersprache Python wird in [22] beschrieben. In [3] befindet sich ein weiterer Zugang, bei dem die Komplexität des Euklidischen Algorithmus im Fokus liegt. Was passiert, wenn der Algorithmus mit zwei aufeinander folgenden Gliedern der Fibonacci-Folge aufgerufen wird? Inwiefern ist das der worst-case für die Anzahl der benötigten Schritte?

Es folgt das Arbeitsblatt – Euklidischer Algorithmus und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Arbeitsblatt – Teilbarkeit und Primfaktorzerlegung

Lernziele:

✓ Wie kann man den größten gemeinsamen Teiler zweier natürlicher Zahlen mit dem **Euklidischen Algorithmus** berechnen?

✓ Für alle positiven natürlichen Zahlen a und b gibt es ganze Zahlen r und s mit:

$$\mathrm{ggT}(a, b) = a \cdot r + b \cdot s$$

Wie kann man r und s mit dem erweiterten Euklidischen Algorithmus berechnen?

Euklidischer Algorithmus

Wir können den größten gemeinsamen Teiler zweier Zahlen mithilfe der Primfaktorzerlegung berechnen. Jetzt berechnen wir ggT(603, 114) mit dem **Euklidischen Algorithmus**:

1) „Wie oft geht 114 in 603? 5 Mal, 33 Rest.“ $603 = 114 \cdot \boxed{5} + \boxed{33}$

2) „Wie oft geht 33 in 114? 3 Mal, 15 Rest.“ $114 = \boxed{33} \cdot \boxed{3} + \boxed{15}$

3) „Wie oft geht 15 in 33? 2 Mal, 3 Rest.“ $\boxed{33} = \boxed{15} \cdot \boxed{2} + \boxed{\mathbf{3}} \implies \text{ggT}(603, 114) = 3$

4) „Wie oft geht 3 in 15? 5 Mal, 0 Rest.“ $\boxed{15} = \boxed{\mathbf{3}} \cdot \boxed{5} + \boxed{0}$

Der letzte Rest $\neq 0$ ist der gesuchte größte gemeinsame Teiler.

Euklidischer Algorithmus

Berechne ggT(630, 282) und ggT(876, 612) mit dem Euklidischen Algorithmus.

$630 = 282 \cdot \square + \square$

$282 = \square \cdot \square + \square$

$\square = \square \cdot \square + \square$

$\square = \square \cdot \square + \square$

$\square = \square \cdot \square + \square$

$\implies \text{ggT}(630, 282) =$ _____

$876 = 612 \cdot \square + \square$

$612 = \square \cdot \square + \square$

$\square = \square \cdot \square + \square$

$\square = \square \cdot \square + \square$

$\implies \text{ggT}(876, 612) =$ _____

Warum funktioniert der Euklidische Algorithmus?

Um zu erklären, warum der Euklidische Algorithmus tatsächlich den größten gemeinsamen Teiler liefert, verwenden wir die folgenden 3 Eigenschaften:

1) Für positive natürlichen Zahlen a und b mit $a \mid b$ gilt: $\text{ggT}(a, b) =$ _____

2) $\text{ggT}(a, b) = \text{ggT}(b, a)$

3) $\text{ggT}(a, b) = \text{ggT}(a, b + k \cdot a)$ mit $k \in \mathbb{Z}$

t ist ein Teiler von a. $\iff$ Es gibt eine ganze Zahl c mit $a = c \cdot t$.
Erkläre: Aus $t \mid a$ und $t \mid b$ folgt $t \mid (b + k \cdot a)$.
Erkläre: Aus $t \mid a$ und $t \mid (b + k \cdot a)$ folgt $t \mid b$.

Rechts siehst du die Schritte des Euklidischen Algorithmus in umgekehrter Reihenfolge.

Erkläre mit den 3 Rechenregeln, warum der letzte Rest $\neq 0$ tatsächlich der größte gemeinsame Teiler ist.

$15 = \boxed{\mathbf{3}} \cdot 5 + 0 \overset{1)}{\implies} \text{ggT}(15, 3) =$ _____

$33 = 15 \cdot 2 + \boxed{\mathbf{3}} \overset{2,3)}{\implies} \text{ggT}(33, 15) =$ _____

$114 = 33 \cdot 3 + 15 \overset{2,3)}{\implies} \text{ggT}(114, 33) =$ _____

$603 = 114 \cdot 5 + 33 \overset{2,3)}{\implies} \text{ggT}(603, 114) =$ _____

Euklidischer Algorithmus

Wir können den größten gemeinsamen Teiler zweier Zahlen mithilfe der Primfaktorzerlegung berechnen. Jetzt berechnen wir $\mathrm{ggT}(603, 114)$ mit dem **Euklidischen Algorithmus**:

1) „Wie oft geht 114 in 603? 5 Mal, 33 Rest.“ $603 = 114 \cdot \boxed{5} + \boxed{33}$

2) „Wie oft geht 33 in 114? 3 Mal, 15 Rest.“ $114 = \boxed{33} \cdot \boxed{3} + \boxed{15}$

3) „Wie oft geht 15 in 33? 2 Mal, 3 Rest.“ $\boxed{33} = \boxed{15} \cdot \boxed{2} + \boxed{\mathbf{3}} \quad \Longrightarrow \mathrm{ggT}(603, 114) = 3$

4) „Wie oft geht 3 in 15? 5 Mal, 0 Rest.“ $\boxed{15} = \boxed{\mathbf{3}} \cdot \boxed{5} + \boxed{0}$

Der letzte Rest $\neq 0$ ist der gesuchte größte gemeinsame Teiler.

Euklidischer Algorithmus

Berechne $\mathrm{ggT}(630, 282)$ und $\mathrm{ggT}(876, 612)$ mit dem Euklidischen Algorithmus.

$630 = 282 \cdot \boxed{2} + \boxed{66}$

$282 = \boxed{66} \cdot \boxed{4} + \boxed{18}$

$\boxed{66} = \boxed{18} \cdot \boxed{3} + \boxed{12}$

$\boxed{18} = \boxed{12} \cdot \boxed{1} + \boxed{6}$

$\boxed{12} = \boxed{6} \cdot \boxed{2} + \boxed{0}$

$\Longrightarrow \mathrm{ggT}(630, 282) = 6$

$876 = 612 \cdot \boxed{1} + \boxed{264}$

$612 = \boxed{264} \cdot \boxed{2} + \boxed{84}$

$\boxed{264} = \boxed{84} \cdot \boxed{3} + \boxed{12}$

$\boxed{84} = \boxed{12} \cdot \boxed{7} + \boxed{0}$

$\Longrightarrow \mathrm{ggT}(876, 612) = 12$

Warum funktioniert der Euklidische Algorithmus?

Um zu erklären, warum der Euklidische Algorithmus tatsächlich den größten gemeinsamen Teiler liefert, verwenden wir die folgenden 3 Eigenschaften:

1) Für positive natürlichen Zahlen a und b mit $a \mid b$ gilt: $\mathrm{ggT}(a, b) = a$

2) $\mathrm{ggT}(a, b) = \mathrm{ggT}(b, a)$

3) $\mathrm{ggT}(a, b) = \mathrm{ggT}(a, b + k \cdot a)$ mit $k \in \mathbb{Z}$

t ist ein Teiler von a. $\Longleftrightarrow$ Es gibt eine ganze Zahl c mit $a = c \cdot t$.
Erkläre: Aus $t \mid a$ und $t \mid b$ folgt $t \mid (b + k \cdot a)$.
Erkläre: Aus $t \mid a$ und $t \mid (b + k \cdot a)$ folgt $t \mid b$.

Rechts siehst du die Schritte des Euklidischen Algorithmus in umgekehrter Reihenfolge.

Erkläre mit den 3 Rechenregeln, warum der letzte Rest $\neq 0$ tatsächlich der größte gemeinsame Teiler ist.

$15 = \boxed{\mathbf{3}} \cdot \boxed{5} + \boxed{0} \quad \overset{1)}{\Longrightarrow} \mathrm{ggT}(15, 3) = \underline{\mathbf{3}}$

$33 = \boxed{15} \cdot \boxed{2} + \boxed{\mathbf{3}} \quad \overset{2,3)}{\Longrightarrow} \mathrm{ggT}(33, 15) = \underline{\mathbf{3}}$

$114 = \boxed{33} \cdot \boxed{3} + \boxed{15} \quad \overset{2,3)}{\Longrightarrow} \mathrm{ggT}(114, 33) = \underline{\mathbf{3}}$

$603 = 114 \cdot \boxed{5} + \boxed{33} \quad \overset{2,3)}{\Longrightarrow} \mathrm{ggT}(603, 114) = \underline{\mathbf{3}}$

Ganzzahlige Linearkombinationen

Findest du zwei *ganze* Zahlen für die Lücken, damit die folgende Gleichung stimmt?

$$42 \cdot \square + 15 \cdot \square = 3$$

Warum können keine *ganzen* Zahlen in die Lücken passen, damit die folgende Gleichung gilt?

$$14 \cdot \square + 6 \cdot \square = 3$$

Erweiterter Euklidischer Algorithmus

Der größte gemeinsame Teiler von a und b kann stets als ganzzahlige Linearkombination von a und b dargestellt werden. Es gibt also ganze Zahlen r und s mit

$$a \cdot \boxed{r} + b \cdot \boxed{s} = \text{ggT}(a, b).$$

Lemma von Bézout

Der **Euklidische Algorithmus** hilft uns diese Koeffizienten r und s zu finden.

Zum Beispiel: $\text{ggT}(603, 114) = \mathbf{3}$
Gesucht sind ganze Zahlen r und s mit

$$603 \cdot r + 114 \cdot s = \mathbf{3}.$$

1) Wir formen die Zeilen vom Euklidischen Algorithmus jeweils auf den Rest um.

2) Wir setzen rückwärts ein, um die ganzen Zahlen r und s zu ermitteln:

$$\begin{aligned}
603 &= 114 \cdot \boxed{5} + \boxed{33} &&\Longrightarrow\ 33 = 603 - 114 \cdot 5\\
114 &= \boxed{33} \cdot \boxed{3} + \boxed{15} &&\Longrightarrow\ 15 = 114 - 33 \cdot 3\\
\boxed{33} &= \boxed{15} \cdot \boxed{2} + \boxed{\mathbf{3}} &&\Longrightarrow\ \mathbf{3} = 33 - 15 \cdot 2\\
\boxed{15} &= \boxed{\mathbf{3}} \cdot \boxed{5} + \boxed{0}
\end{aligned}$$

$$\begin{aligned}
\mathbf{3} &= 33 - 15 \cdot 2 = 33 - (114 - 33 \cdot 3) \cdot 2 = 33 \cdot 7 - 114 \cdot 2 =\\
&= (603 - 114 \cdot 5) \cdot 7 - 114 \cdot 2 = 603 \cdot 7 - 114 \cdot 37
\end{aligned}$$

$$\Longrightarrow\ 603 \cdot \square + 114 \cdot \square = \mathbf{3}$$

Erweiterter Euklidischer Algorithmus

Ermittle $\text{ggT}(700, 297)$ sowie ganze Zahlen r und s mit $700 \cdot r + 297 \cdot s = \text{ggT}(700, 297)$.

Ganzzahlige Linearkombinationen

Findest du zwei *ganze* Zahlen für die Lücken, damit die folgende Gleichung stimmt?

$$42 \cdot (-1) + 15 \cdot 3 = 3$$

Warum können keine *ganzen* Zahlen in die Lücken passen, damit die folgende Gleichung gilt?

$$14 \cdot \square + 6 \cdot \square = 3$$

Die linke Seite ist immer eine gerade Zahl.

Erweiterter Euklidischer Algorithmus

Der größte gemeinsame Teiler von a und b kann stets als ganzzahlige Linearkombination von a und b dargestellt werden. Es gibt also ganze Zahlen r und s mit

$$a \cdot \boxed{r} + b \cdot \boxed{s} = \text{ggT}(a, b).$$

Lemma von Bézout

Der **Euklidische Algorithmus** hilft uns diese Koeffizienten r und s zu finden.

Zum Beispiel: $\text{ggT}(603, 114) = \mathbf{3}$
Gesucht sind ganze Zahlen r und s mit

$$603 \cdot r + 114 \cdot s = \mathbf{3}.$$

1) Wir formen die Zeilen vom Euklidischen Algorithmus jeweils auf den Rest um.

2) Wir setzen rückwärts ein, um die ganzen Zahlen r und s zu ermitteln:

$$603 = 114 \cdot \boxed{5} + \boxed{33} \implies 33 = 603 - 114 \cdot 5$$
$$114 = \boxed{33} \cdot \boxed{3} + \boxed{15} \implies 15 = 114 - 33 \cdot 3$$
$$\boxed{33} = \boxed{15} \cdot \boxed{2} + \boxed{\mathbf{3}} \implies \mathbf{3} = 33 - 15 \cdot 2$$
$$\boxed{15} = \boxed{\mathbf{3}} \cdot \boxed{5} + \boxed{0}$$

$$\mathbf{3} = 33 - 15 \cdot 2 = 33 - (114 - 33 \cdot 3) \cdot 2 = 33 \cdot 7 - 114 \cdot 2 =$$
$$= (603 - 114 \cdot 5) \cdot 7 - 114 \cdot 2 = 603 \cdot 7 - 114 \cdot 37$$
$$\implies 603 \cdot 7 + 114 \cdot (-37) = \mathbf{3}$$

Erweiterter Euklidischer Algorithmus

Ermittle $\text{ggT}(700, 297)$ sowie ganze Zahlen r und s mit $700 \cdot r + 297 \cdot s = \text{ggT}(700, 297)$.

$$700 = 297 \cdot \boxed{2} + \boxed{106} \implies 106 = 700 - 297 \cdot 2$$
$$297 = \boxed{106} \cdot \boxed{2} + \boxed{85} \implies 85 = 297 - 106 \cdot 2$$
$$\boxed{106} = \boxed{85} \cdot \boxed{1} + \boxed{21} \implies 21 = 106 - 85 \cdot 1$$
$$\boxed{85} = \boxed{21} \cdot \boxed{4} + \boxed{1} \implies 1 = 85 - 21 \cdot 4$$
$$\boxed{21} = \boxed{1} \cdot \boxed{21} + \boxed{0}$$

$$\implies \text{ggT}(700, 297) = 1 = 85 - 21 \cdot 4 = 85 - (106 - 85 \cdot 1) \cdot 4 = 85 \cdot 5 - 106 \cdot 4 =$$
$$= (297 - 106 \cdot 2) \cdot 5 - 106 \cdot 4 = 297 \cdot 5 - 106 \cdot 14 =$$
$$= 297 \cdot 5 - (700 - 297 \cdot 2) \cdot 14 = 700 \cdot (-14) + 297 \cdot 33$$

2.3 Fundamentalsatz der Arithmetik

Das Lemma von Euklid ([37], Book VII, Prop. 30) besagt:
„Wenn eine Primzahl ein Produkt von zwei natürlichen Zahlen teilt, dann muss diese Primzahl einen der beiden Faktoren teilen.“
Auf den ersten Blick ist die Aussage offensichtlich: Wenn die Primzahl in der Primfaktorzerlegung des Produkts vorkommt, dann muss sie in der Primfaktorzerlegung von einem der beiden Faktoren vorkommen. Schließlich lernen wir ja schon in der 5. Schulstufe, wie wir *die* Primfaktorzerlegung jeder natürlichen Zahl berechnen können.

Tatsächlich zeigte Carl Friedrich Gauß vor rund 200 Jahren in [15], dass die (bis auf Reihenfolge der Faktoren) eindeutige Primfaktorzerlegung jeder natürlichen Zahl eine *Folgerung* aus dem Lemma von Euklid ist. Das ist der Fundamentalsatz der Arithmetik. Das Lemma von Euklid aus der Eindeutigkeit der Primfaktorzerlegung zu folgern, wäre also ein Zirkelschluss. Auf dem folgenden Arbeitsblatt ist eine Möglichkeit aufbereitet, wie man das Lemma von Euklid mit dem Euklidischen Algorithmus begründen kann.

Es folgt das Arbeitsblatt – Fundamentalsatz der Arithmetik und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Arbeitsblatt – Teilbarkeit und Primfaktorzerlegung

– Arbeitsblatt – Euklidischer Algorithmus

– A und B sind Aussagen. Was bedeutet $A \implies B$? Was bedeutet $A \iff B$?

– Was ist ein indirekter Beweis?

Lernziele:

✓ Was sagt der **Fundamentalsatz der Arithmetik** aus? Wie kann man ihn beweisen?

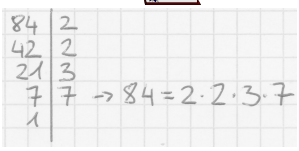
✓ Die Primfaktorzerlegungen von a und b sind gegeben:

$$a = 2^{v_1} \cdot 3^{v_2} \cdot 5^{v_3} \cdot 7^{v_4} \cdot 11^{v_5} \cdot 13^{v_6} \cdot \ \cdots \qquad b = 2^{w_1} \cdot 3^{w_2} \cdot 5^{w_3} \cdot 7^{w_4} \cdot 11^{w_5} \cdot 13^{w_6} \cdot \ \cdots$$

- Wie kann man damit prüfen, ob a ein Teiler von b ist?
- Wie kann man damit prüfen, ob a und b teilerfremd sind?
- Wie kann man damit $\mathrm{ggT}(a, b)$ bzw. $\mathrm{kgV}(a, b)$ berechnen?
- Warum gilt $\mathrm{ggT}(a, b) \cdot \mathrm{kgV}(a, b) = a \cdot b$?

✓ Was sagt das **Lemma von Euklid** aus?
Wie kann man es aus dem Euklidischen Algorithmus folgern?

Fundamentalsatz der Arithmetik



Als Kinder lernen wir so oder so ähnlich die Primfaktorzerlegung:

Tatsächlich steckt hinter dieser Methode
der **Fundamentalsatz der Arithmetik**:

1) Jede natürliche Zahl $n \geq 2$ kann als Produkt von Primzahlen geschrieben werden.

Existenz der Primfaktorzerlegung

2) Die Primfaktorzerlegung von n ist eindeutig bis auf die Reihenfolge der Faktoren.

Eindeutigkeit der Primfaktorzerlegung

Jedes Produkt von Primzahlen, das *nicht* genau aus den vier Faktoren 2, 2, 3 und 7 besteht, ist also *ungleich* 84.

Einen Beweis für den Fundamentalsatz der Arithmetik findest du am Ende des Arbeitsblatts.

Vielfachheiten

In der sogenannten *kanonischen* Primfaktorzerlegung sortieren wir die Primfaktoren aufsteigend und schreiben gleiche Primfaktoren als Potenz.

Zum Beispiel: $1176 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 7 \cdot 7 = 2^3 \cdot 3^1 \cdot 7^2$

Wir sagen auch: „Die Vielfachheit der Primzahl 2 in 1176 ist 3."

Folge von Vielfachheiten

Die Primfaktorzerlegung jeder natürlichen Zahl $n \geq 1$ hat eine Folge $\langle v_1, v_2, v_3, \ldots \rangle$ von Vielfachheiten:

$$n = 2^{v_1} \cdot 3^{v_2} \cdot 5^{v_3} \cdot 7^{v_4} \cdot 11^{v_5} \cdot 13^{v_6} \cdot \ \cdots$$

v_i ist die Vielfachheit der i-ten Primzahl.

Die Vielfachheiten sind ganze Zahlen $v_i \geq 0$, wobei nur *endlich* viele $v_i \neq 0$ sind.

Zum Beispiel: $1176 = 2^3 \cdot 3^1 \cdot 5^0 \cdot 7^2 \cdot 11^0 \cdot 13^0 \cdot \ \cdots \implies \langle v_n \rangle = \langle 3, 1, 0, 2, 0, 0, \ldots \rangle$

Umgekehrt liefert jede solche Folge die Primfaktorzerlegung einer natürlichen Zahl:

$$\langle v_n \rangle = \langle 3, 1, 0, 2, 0, 0, \ldots \rangle \implies 2^3 \cdot 3^1 \cdot 5^0 \cdot 7^2 \cdot 11^0 \cdot 13^0 \cdot \ \cdots = 1176$$

Wir schreiben dafür auch kurz: $1176 \cong \langle 3, 1, 0, 2, 0, 0, \ldots \rangle$

Codierung & Decodierung

a) Zerlege 1400 in Primfaktoren. Gib die zugehörige Folge von Vielfachheiten an.

b) Welche natürliche Zahl steckt hinter der Folge $\langle 0, 4, 2, 0, 0, \ldots \rangle$ von Vielfachheiten?

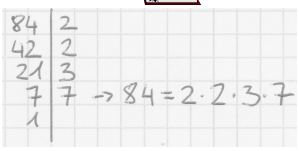
Vielfachheiten & Multiplikation

Die natürliche Zahl $a \geq 1$ hat die Folge von Vielfachheiten $\langle v_1, v_2, v_3, \ldots \rangle$. $a \cong \langle v_1, v_2, v_3, \ldots \rangle$
Die natürliche Zahl $b \geq 1$ hat die Folge von Vielfachheiten $\langle w_1, w_2, w_3, \ldots \rangle$. $b \cong \langle w_1, w_2, w_3, \ldots \rangle$

Welche Folge von Vielfachheiten hat die Zahl $a \cdot b$?

Fundamentalsatz der Arithmetik

Als Kinder lernen wir so oder so ähnlich die Primfaktorzerlegung:



Tatsächlich steckt hinter dieser Methode
der **Fundamentalsatz der Arithmetik**:

1) Jede natürliche Zahl $n \geq 2$ kann als Produkt von Primzahlen geschrieben werden.

Existenz der Primfaktorzerlegung

2) Die Primfaktorzerlegung von n ist eindeutig bis auf die Reihenfolge der Faktoren.

Eindeutigkeit der Primfaktorzerlegung

Jedes Produkt von Primzahlen, das *nicht* genau aus den vier Faktoren 2, 2, 3 und 7 besteht, ist also *ungleich* 84.

Einen Beweis für den Fundamentalsatz der Arithmetik findest du am Ende des Arbeitsblatts.

Vielfachheiten

In der sogenannten *kanonischen* Primfaktorzerlegung sortieren wir die Primfaktoren aufsteigend und schreiben gleiche Primfaktoren als Potenz.

Zum Beispiel: $1176 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 7 \cdot 7 = 2^3 \cdot 3^1 \cdot 7^2$

Wir sagen auch: „Die Vielfachheit der Primzahl 2 in 1176 ist 3.“

Folge von Vielfachheiten

Die Primfaktorzerlegung jeder natürlichen Zahl $n \geq 1$ hat eine Folge $\langle v_1, v_2, v_3, \ldots \rangle$ von Vielfachheiten:

$$n = 2^{v_1} \cdot 3^{v_2} \cdot 5^{v_3} \cdot 7^{v_4} \cdot 11^{v_5} \cdot 13^{v_6} \cdot \ \cdots$$

v_i ist die Vielfachheit der i-ten Primzahl.

Die Vielfachheiten sind ganze Zahlen $v_i \geq 0$, wobei nur *endlich* viele $v_i \neq 0$ sind.

Zum Beispiel: $1176 = 2^3 \cdot 3^1 \cdot 5^0 \cdot 7^2 \cdot 11^0 \cdot 13^0 \cdot \ \cdots \implies \langle v_n \rangle = \langle 3, 1, 0, 2, 0, 0, \ldots \rangle$

Umgekehrt liefert jede solche Folge die Primfaktorzerlegung einer natürlichen Zahl:

$$\langle v_n \rangle = \langle 3, 1, 0, 2, 0, 0, \ldots \rangle \implies 2^3 \cdot 3^1 \cdot 5^0 \cdot 7^2 \cdot 11^0 \cdot 13^0 \cdot \ \cdots = 1176$$

Wir schreiben dafür auch kurz: $1176 \cong \langle 3, 1, 0, 2, 0, 0, \ldots \rangle$

Codierung & Decodierung

a) Zerlege 1400 in Primfaktoren. Gib die zugehörige Folge von Vielfachheiten an.

$$1400 = 2 \cdot 2 \cdot 2 \cdot 5 \cdot 5 \cdot 7 = 2^3 \cdot 5^2 \cdot 7^1 \implies 1400 \cong \langle 3, 0, 2, 1, 0, 0, \ldots \rangle$$

b) Welche natürliche Zahl steckt hinter der Folge $\langle 0, 4, 2, 0, 0, \ldots \rangle$ von Vielfachheiten?

$$2^0 \cdot 3^4 \cdot 5^2 \cdot 7^0 \cdot 11^0 \cdot \ \cdots = 2025$$

Vielfachheiten & Multiplikation

Die natürliche Zahl $a \geq 1$ hat die Folge von Vielfachheiten $\langle v_1, v_2, v_3, \ldots \rangle$. $a \cong \langle v_1, v_2, v_3, \ldots \rangle$
Die natürliche Zahl $b \geq 1$ hat die Folge von Vielfachheiten $\langle w_1, w_2, w_3, \ldots \rangle$. $b \cong \langle w_1, w_2, w_3, \ldots \rangle$

Welche Folge von Vielfachheiten hat die Zahl $a \cdot b$?

$$a \cdot b = 2^{v_1} \cdot 2^{w_1} \cdot 3^{v_2} \cdot 3^{w_2} \cdot 5^{v_3} \cdot 5^{w_3} \cdot \ \cdots = 2^{v_1+w_1} \cdot 3^{v_2+w_2} \cdot 5^{v_3+w_3} \cdot \ \cdots$$

$$\implies a \cdot b \cong \langle v_1 + w_1, v_2 + w_2, v_3 + w_3, \ldots \rangle$$

Vielfachheiten & Teilbarkeit

Es gilt $a \cong \langle v_1, v_2, v_3, \ldots \rangle$ und $b \cong \langle w_1, w_2, w_3, \ldots \rangle$.

Mit den beiden Folgen können wir unmittelbar prüfen, ob a ein Teiler von b ist:

$$\boldsymbol{a \mid b \iff v_1 \leq w_1 \text{ und } v_2 \leq w_2 \text{ und } v_3 \leq w_3 \text{ und } \cdots}$$

In diesem Fall gibt es eine natürliche Zahl k mit $b = a \cdot k$. Welche Folge von Vielfachheiten hat k?

Vielfachheiten & ggT/kgV

Ermittle den größten gemeinsamen Teiler und das kleinste gemeinsame Vielfache von a und b.

a) $a = 2^3 \cdot 3^5 \cdot 7^1, \quad b = 2^2 \cdot 3^1 \cdot 5^4$

$\text{ggT}(a,b) =$ _____

$\text{kgV}(a,b) =$ _____

b) $a = 3^5 \cdot 11^2, \quad b = 2^3 \cdot 5^2 \cdot 7^4$

$\text{ggT}(a,b) =$ _____

$\text{kgV}(a,b) =$ _____

Vielfachheiten & ggT/kgV

Es gilt $a \cong \langle v_1, v_2, v_3, \ldots \rangle$ und $b \cong \langle w_1, w_2, w_3, \ldots \rangle$.

Mit den beiden Folgen können wir $\text{ggT}(a,b)$ und $\text{kgV}(a,b)$ unmittelbar berechnen:

1) $\boldsymbol{\text{ggT}(a,b) \cong \langle \min\{v_1, w_1\}, \min\{v_2, w_2\}, \min\{v_3, w_3\}, \ldots \rangle}$

$\min\{v_1, w_1\}$ ist die kleinere der beiden Zahlen. („Minimum“)

2) $\boldsymbol{\text{kgV}(a,b) \cong \langle \max\{v_1, w_1\}, \max\{v_2, w_2\}, \max\{v_3, w_3\}, \ldots \rangle}$

$\max\{v_1, w_1\}$ ist die größere der beiden Zahlen. („Maximum“)

$\text{ggT}(a,b) \cdot \text{kgV}(a,b)$

Erkläre, warum $\boldsymbol{\text{ggT}(a,b) \cdot \text{kgV}(a,b) = a \cdot b}$ gilt.

Vielfachheiten & Teilerfremdheit

Es gilt $a \cong \langle v_1, v_2, v_3, \ldots \rangle$ und $b \cong \langle w_1, w_2, w_3, \ldots \rangle$.

Wie kannst du mit den beiden Folgen unmittelbar prüfen, ob a und b *teilerfremd* sind. $\text{ggT}(a,b) = 1$

Vielfachheiten & Teilbarkeit

Es gilt $a \cong \langle v_1, v_2, v_3, \ldots \rangle$ und $b \cong \langle w_1, w_2, w_3, \ldots \rangle$.

Mit den beiden Folgen können wir unmittelbar prüfen, ob a ein Teiler von b ist:

$$\boldsymbol{a \mid b \iff v_1 \leq w_1 \quad \text{und} \quad v_2 \leq w_2 \quad \text{und} \quad v_3 \leq w_3 \quad \text{und} \quad \cdots}$$

In diesem Fall gibt es eine natürliche Zahl k mit $b = a \cdot k$. Welche Folge von Vielfachheiten hat k?

$k \cong \langle w_1 - v_1, w_2 - v_2, w_3 - v_3, \ldots \rangle \implies a \cdot k \cong \langle w_1, w_2, w_3, \ldots \rangle \cong b$ ✓

Vielfachheiten & ggT/kgV

Ermittle den größten gemeinsamen Teiler und das kleinste gemeinsame Vielfache von a und b.

a) $a = 2^3 \cdot 3^5 \cdot 7^1, \quad b = 2^2 \cdot 3^1 \cdot 5^4$

$\text{ggT}(a, b) = 2^2 \cdot 3^1$

$\text{kgV}(a, b) = 2^3 \cdot 3^5 \cdot 5^4 \cdot 7^1$

b) $a = 3^5 \cdot 11^2, \quad b = 2^3 \cdot 5^2 \cdot 7^4$

$\text{ggT}(a, b) = 1$

$\text{kgV}(a, b) = 2^3 \cdot 3^5 \cdot 5^2 \cdot 7^4 \cdot 11^2$

Vielfachheiten & ggT/kgV

Es gilt $a \cong \langle v_1, v_2, v_3, \ldots \rangle$ und $b \cong \langle w_1, w_2, w_3, \ldots \rangle$.

Mit den beiden Folgen können wir $\text{ggT}(a, b)$ und $\text{kgV}(a, b)$ unmittelbar berechnen:

1) $\boldsymbol{\text{ggT}(a, b) \cong \langle \min\{v_1, w_1\}, \min\{v_2, w_2\}, \min\{v_3, w_3\}, \ldots \rangle}$

$\min\{v_1, w_1\}$ ist die kleinere der beiden Zahlen. („Minimum“)

2) $\boldsymbol{\text{kgV}(a, b) \cong \langle \max\{v_1, w_1\}, \max\{v_2, w_2\}, \max\{v_3, w_3\}, \ldots \rangle}$

$\max\{v_1, w_1\}$ ist die größere der beiden Zahlen. („Maximum“)

$\text{ggT}(a, b) \cdot \text{kgV}(a, b)$

Erkläre, warum $\boldsymbol{\text{ggT}(a, b) \cdot \text{kgV}(a, b) = a \cdot b}$ gilt.

$a \cong \langle v_1, v_2, v_3, \ldots \rangle \quad b \cong \langle w_1, w_2, w_3, \ldots \rangle$

Allgemein gilt: $\min\{v_i, w_i\} + \max\{v_i, w_i\} = v_i + w_i$

$\text{ggT}(a, b) \cdot \text{kgV}(a, b)$ und $a \cdot b$ haben also die gleiche Folge von Vielfachheiten und sind somit gleich.

Vielfachheiten & Teilerfremdheit

Es gilt $a \cong \langle v_1, v_2, v_3, \ldots \rangle$ und $b \cong \langle w_1, w_2, w_3, \ldots \rangle$.

Wie kannst du mit den beiden Folgen unmittelbar prüfen, ob a und b *teilerfremd* sind. $\text{ggT}(a, b) = 1$

a und b sind genau dann teilerfremd, wenn $\min\{v_i, w_i\} = 0$ für alle i gilt.

Es darf also *nicht dieselbe* Primzahl in *beiden* Primfaktorzerlegungen vorkommen.

Gemeinsame Faktoren kürzen

Erkläre, warum die beiden Zahlen $\frac{a}{\text{ggT}(a,b)}$ und $\frac{b}{\text{ggT}(a,b)}$ teilerfremd sind.

Wo sind die Primfaktoren?

Begründe die folgende Aussage: $\left.\begin{array}{l} n \mid a \cdot b \\ \text{ggT}(n,a) = 1 \end{array}\right\} \implies n \mid b$

Division

Begründe die folgenden beiden Äquivalenzen:

$$\boldsymbol{a \mid b \cdot c} \overset{\textbf{1)}}{\iff} \frac{a}{\text{ggT}(a,b)} \mid \frac{b}{\text{ggT}(a,b)} \cdot c \overset{\textbf{2)}}{\iff} \boldsymbol{\frac{a}{\text{ggT}(a,b)} \mid c}$$

Diese Eigenschaft wird sich am Arbeitsblatt – Kongruenz und Restklassen als nützlich erweisen.

Lemma von Euklid

Das **Lemma von Euklid** ist eine grundlegende Aussage der Zahlentheorie:
„Wenn eine Primzahl ein Produkt von zwei Zahlen teilt, dann teilt sie mindestens einen der Faktoren."

$$p \mid a \cdot b \implies p \mid a \quad \text{oder} \quad p \mid b \quad \text{für alle Primzahlen } p \text{ und } a, b \in \mathbb{Z}$$

Das Lemma von Euklid gilt dann auch für Produkte mit endlich vielen Faktoren:

$$p \mid a \cdot \underbrace{(b \cdot c)}_{\in \mathbb{Z}} \implies p \mid a \quad \text{oder} \quad p \mid b \cdot c \implies p \mid a \quad \text{oder} \quad p \mid b \quad \text{oder} \quad p \mid c$$

„Wenn eine Primzahl p ein Produkt teilt, dann teilt p mindestens einen der Faktoren."

Gemeinsame Faktoren kürzen

Erkläre, warum die beiden Zahlen $\frac{a}{\text{ggT}(a,b)}$ und $\frac{b}{\text{ggT}(a,b)}$ teilerfremd sind.

$a \cong \langle v_1, v_2, v_3, \ldots \rangle \quad b \cong \langle w_1, w_2, w_3, \ldots \rangle$
$\implies \frac{a}{\text{ggT}(a,b)} \cong \langle v_1 - \min\{v_1, w_1\}, v_2 - \min\{v_2, w_2\}, \ldots \rangle$
$\implies \frac{b}{\text{ggT}(a,b)} \cong \langle w_1 - \min\{v_1, w_1\}, w_2 - \min\{v_2, w_2\}, \ldots \rangle$

Für alle i gilt $v_i - \min\{v_i, w_i\} = 0$ oder $w_i - \min\{v_i, w_i\} = 0$.
Die beiden Zahlen sind also teilerfremd.

Wo sind die Primfaktoren?

Begründe die folgende Aussage: $\left.\begin{array}{l} n \mid a \cdot b \\ \text{ggT}(n, a) = 1 \end{array}\right\} \implies n \mid b$

$a \cong \langle v_1, v_2, v_3, \ldots \rangle \quad b \cong \langle w_1, w_2, w_3, \ldots \rangle \quad n \cong \langle x_1, x_2, x_3, \ldots \rangle$

$$\implies \left.\begin{array}{rl} x_i \leq v_i + w_i & \text{für alle } i \\ \min\{x_i, v_i\} = 0 & \text{für alle } i \end{array}\right\} \overset{(\star)}{\implies} x_i \leq w_i \quad \text{für alle } i$$

$(\star)$ ist korrekt, weil $x_i = 0$ oder $v_i = 0$ gilt. In beiden Fällen folgt $x_i \leq w_i$.

Division

Begründe die folgenden beiden Äquivalenzen:

$$\boldsymbol{a \mid b \cdot c} \overset{1)}{\iff} \frac{a}{\text{ggT}(a,b)} \mid \frac{b}{\text{ggT}(a,b)} \cdot c \overset{2)}{\iff} \boldsymbol{\frac{a}{\text{ggT}(a,b)} \mid c}$$

Diese Eigenschaft wird sich am Arbeitsblatt – Kongruenz und Restklassen als nützlich erweisen.

1) Es gilt: $b \cdot c = a \cdot k \iff \frac{b}{\text{ggT}(a,b)} \cdot c = \frac{a}{\text{ggT}(a,b)} \cdot k$

2) $\Longrightarrow$
$\frac{a}{\text{ggT}(a,b)}$ und $\frac{b}{\text{ggT}(a,b)}$ sind teilerfremd.

Aus $\frac{a}{\text{ggT}(a,b)} \mid \frac{b}{\text{ggT}(a,b)} \cdot c$ folgt deshalb $\frac{a}{\text{ggT}(a,b)} \mid c$.

$\Longleftarrow$
Aus $A \mid B$ folgt immer $A \mid B \cdot C$.

Lemma von Euklid

Das **Lemma von Euklid** ist eine grundlegende Aussage der Zahlentheorie:
„Wenn eine Primzahl ein Produkt von zwei Zahlen teilt, dann teilt sie mindestens einen der Faktoren."

$$p \mid a \cdot b \implies p \mid a \quad \text{oder} \quad p \mid b \quad \text{für alle Primzahlen } p \text{ und } a, b \in \mathbb{Z}$$

Das Lemma von Euklid gilt dann auch für Produkte mit endlich vielen Faktoren:

$$p \mid a \cdot \underbrace{(b \cdot c)}_{\in \mathbb{Z}} \implies p \mid a \quad \text{oder} \quad p \mid b \cdot c \implies p \mid a \quad \text{oder} \quad p \mid b \quad \text{oder} \quad p \mid c$$

„Wenn eine Primzahl p ein Produkt teilt, dann teilt p mindestens einen der Faktoren."

Lemma von Euklid – Beweis

Warum folgt aus $p \mid a \cdot b$, dass die Primzahl p eine der Zahlen a oder b teilen muss?

Wir beweisen jetzt das **Lemma von Euklid** ohne die Eindeutigkeit der Primfaktorzerlegung zu verwenden.
Tatsächlich ist die Eindeutigkeit der Primfaktorzerlegung eine Folgerung aus dem Lemma von Euklid.

Wenn p ein Teiler von a ist, dann ist nichts mehr zu begründen.

Wenn p *kein* Teiler von a ist, dann gilt $\text{ggT}(p, a) = 1$. Denn p ist eine Primzahl.

1) Der Euklidische Algorithmus liefert ganze Zahlen r und s mit $p \cdot r + a \cdot s = \underbrace{\text{ggT}(p, a)}_{=1}$.

2) Wir multiplizieren mit b auf beiden Seiten der Gleichung: $p \cdot r \cdot b + a \cdot b \cdot s = b$

3) Da p ein Teiler von $a \cdot b$ gibt es eine ganze Zahl k mit $a \cdot b = k \cdot p$.

Wir setzen ein: $p \cdot r \cdot b + k \cdot p \cdot s = b \implies p \cdot \underbrace{(r \cdot b + k \cdot s)}_{\in \mathbb{Z}} = b$

Wenn p kein Teiler von a ist, dann muss also p ein Teiler von b sein. □

Existenz der Primfaktorzerlegung – Beweis

Warum kann jede natürliche Zahl $n \geq 2$ als Produkt vom Primfaktoren geschrieben werden?
Indirekter Beweis: Angenommen, es gibt eine natürliche Zahl ≥ 2, die keine Primfaktorzerlegung hat.

1) Wähle unter allen solchen Zahlen die *kleinste* Zahl m, die *keine* Primfaktorzerlegung (PFZ) hat.

2) Dann kann m *keine* Primzahl sein. Denn jede Primzahl hat eine PFZ, nämlich sich selbst.

3) Also gibt es natürliche Zahlen a und b mit $m = a \cdot b$ und $1 < a, b < m$. Sonst wäre m eine Primzahl.

4) Dann müssen aber a und b jeweils eine PFZ haben. m ist ja die *kleinste* Zahl ≥ 2 ohne PFZ.

5) Dann hat aber auch m eine PFZ: $m = a \cdot b = \underbrace{(\text{PFZ von } a) \cdot (\text{PFZ von } b)}_{\text{PFZ von } m}$ ↯ Widerspruch zu **1)**

Die Annahme war also falsch. *Jede* natürliche Zahl $n \geq 2$ muss eine Primfaktorzerlegung haben. □

Eindeutigkeit der Primfaktorzerlegung – Beweis

Warum ist die PFZ jeder natürlichen Zahl $n \geq 2$ eindeutig bis auf die Reihenfolge der Faktoren?
Indirekter Beweis: Angenommen es gibt eine natürliche Zahl ≥ 2 mit verschiedenen PFZ.

1) Wähle unter allen solchen Zahlen die *kleinste* Zahl m, die verschiedene PFZ hat:

$$m = p_1 \cdot p_2 \cdot \ldots \cdot p_r = P_1 \cdot P_2 \cdot \ldots \cdot P_s$$

2) Dann kann m *keine* Primzahl sein. Denn jede Primzahl hat eine *eindeutige* PFZ, nämlich sich selbst.
Beide Zerlegungen haben also mindestens zwei Primfaktoren. $(r, s \geq 2)$

3) Es kann keine Primzahl in beiden Zerlegungen vorkommen.
Wenn eine Primzahl p in beiden Zerlegungen vorkommt, dann ist $\frac{m}{p}$ eine *kleinere* Zahl als m, die verschiedene PFZ hat. ↯

4) $p_1 \mid m \implies p_1 \mid P_1 \cdot P_2 \cdot \ldots \cdot P_s \implies p_1 \mid P_1$ oder $p_1 \mid P_2$ oder $\cdots$ oder $p_1 \mid P_s$ Lemma von Euklid

5) Wenn eine Primzahl eine Primzahl teilt, dann sind sie gleich. Sonst wäre die größere Zahl keine Primzahl.

6) Die Primzahl p_1 kommt also in beiden PFZ vor. ↯ Widerspruch zu **3)**

Die Annahme war also falsch.
Die PFZ *jeder* natürlichen Zahl $n \geq 2$ muss bis auf die Reihenfolge der Faktoren eindeutig sein. □

Lemma von Euklid – Beweis

Warum folgt aus $p \mid a \cdot b$, dass die Primzahl p eine der Zahlen a oder b teilen muss?

Wir beweisen jetzt das **Lemma von Euklid** ohne die Eindeutigkeit der Primfaktorzerlegung zu verwenden.
Tatsächlich ist die Eindeutigkeit der Primfaktorzerlegung eine Folgerung aus dem Lemma von Euklid.

Wenn p ein Teiler von a ist, dann ist nichts mehr zu begründen.

Wenn p *kein* Teiler von a ist, dann gilt $\text{ggT}(p, a) = 1$. Denn p ist eine Primzahl.

1) Der Euklidische Algorithmus liefert ganze Zahlen r und s mit $p \cdot r + a \cdot s = \underbrace{\text{ggT}(p, a)}_{=1}$.

2) Wir multiplizieren mit b auf beiden Seiten der Gleichung: $p \cdot r \cdot b + a \cdot b \cdot s = b$

3) Da p ein Teiler von $a \cdot b$ gibt es eine ganze Zahl k mit $a \cdot b = k \cdot p$.

Wir setzen ein: $p \cdot r \cdot b + k \cdot p \cdot s = b \implies p \cdot \underbrace{(r \cdot b + k \cdot s)}_{\in \mathbb{Z}} = b$

Wenn p kein Teiler von a ist, dann muss also p ein Teiler von b sein. □

Existenz der Primfaktorzerlegung – Beweis

Warum kann jede natürliche Zahl $n \geq 2$ als Produkt vom Primfaktoren geschrieben werden?
Indirekter Beweis: Angenommen, es gibt eine natürliche Zahl ≥ 2, die keine Primfaktorzerlegung hat.

1) Wähle unter allen solchen Zahlen die *kleinste* Zahl m, die *keine* Primfaktorzerlegung (PFZ) hat.

2) Dann kann m *keine* Primzahl sein. Denn jede Primzahl hat eine PFZ, nämlich sich selbst.

3) Also gibt es natürliche Zahlen a und b mit $m = a \cdot b$ und $1 < a, b < m$. Sonst wäre m eine Primzahl.

4) Dann müssen aber a und b jeweils eine PFZ haben. m ist ja die *kleinste* Zahl ≥ 2 ohne PFZ.

5) Dann hat aber auch m eine PFZ: $m = a \cdot b = \underbrace{(\text{PFZ von } a) \cdot (\text{PFZ von } b)}_{\text{PFZ von } m}$ ↯ Widerspruch zu **1)**

Die Annahme war also falsch. *Jede* natürliche Zahl $n \geq 2$ muss eine Primfaktorzerlegung haben. □

Eindeutigkeit der Primfaktorzerlegung – Beweis

Warum ist die PFZ jeder natürlichen Zahl $n \geq 2$ eindeutig bis auf die Reihenfolge der Faktoren?
Indirekter Beweis: Angenommen es gibt eine natürliche Zahl ≥ 2 mit verschiedenen PFZ.

1) Wähle unter allen solchen Zahlen die *kleinste* Zahl m, die verschiedene PFZ hat:

$$m = p_1 \cdot p_2 \cdot \ldots \cdot p_r = P_1 \cdot P_2 \cdot \ldots \cdot P_s$$

2) Dann kann m *keine* Primzahl sein. Denn jede Primzahl hat eine *eindeutige* PFZ, nämlich sich selbst.
Beide Zerlegungen haben also mindestens zwei Primfaktoren. $(r, s \geq 2)$

3) Es kann keine Primzahl in beiden Zerlegungen vorkommen.
Wenn eine Primzahl p in beiden Zerlegungen vorkommt, dann ist $\frac{m}{p}$ eine *kleinere* Zahl als m, die verschiedene PFZ hat. ↯

4) $p_1 \mid m \implies p_1 \mid P_1 \cdot P_2 \cdot \ldots \cdot P_s \implies p_1 \mid P_1$ oder $p_1 \mid P_2$ oder $\cdots$ oder $p_1 \mid P_s$ Lemma von Euklid

5) Wenn eine Primzahl eine Primzahl teilt, dann sind sie gleich. Sonst wäre die größere Zahl keine Primzahl.

6) Die Primzahl p_1 kommt also in beiden PFZ vor. ↯ Widerspruch zu **3)**

Die Annahme war also falsch.
Die PFZ *jeder* natürlichen Zahl $n \geq 2$ muss bis auf die Reihenfolge der Faktoren eindeutig sein. □

2.4 Kongruenz und Restklassen

Als Carl Friedrich Gauß vor mehr als 200 Jahren in [15] die Schreibweise $a \equiv b \mod m$ einführte, hat er wohl nicht damit gerechnet, dass diese einmal für Verschlüsselungen in einem weltweiten Netzwerk verwendet werden würde. Hinter dem im Jahr 1978 publizierten und heute noch verwendeten RSA-Verfahren [2] steckt nämlich genau die mathematische Theorie der Kongruenzen und Restklassen sowie der Satz von Euler.

Eine Einführung zu Kongruenzen und der Zahlentheorie wird zum Beispiel in [5] gegeben. Eine Übersicht zu den Definitionen und wichtigsten Sätzen (nicht nur der Zahlentheorie) ist im „Handbook of Discrete and Combinatorial Mathematics“ [28] zu finden.

Wir versuchen auf den nächsten Arbeitsblättern alle notwendigen Definitionen und Rechenregeln so aufzubereiten, dass schließlich die gesamte Mathematik hinter dem RSA-Verfahren erfasst werden kann.

Es folgt das Arbeitsblatt – Kongruenz und Restklassen und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Arbeitsblatt – Fundamentalsatz der Arithmetik

Lernziele:

✓ Was sind die **Teiler** bzw. **Vielfache** einer ganzen Zahl?

✓ Was sind **Kongruenzen**? Was bedeutet die Schreibweise $\boldsymbol{a \equiv b \mod m}$?

✓ Welche Rechenregeln gelten für Kongruenzen?

✓ Warum stimmen die **Teilbarkeitsregeln** für 2, 3, 5, 9 und 10?

✓ Wie berechnet man die Additions- und Multiplikationstabellen modulo n?

✓ Warum stimmt die folgende Kürzungsregel für Kongruenzen?

$$\boldsymbol{a \cdot c \equiv b \cdot c \mod m \iff a \equiv b \mod \frac{m}{\mathrm{ggT}(c,m)}}$$

Teilbarkeit ganzer Zahlen

a und b sind ganze Zahlen mit $a \neq 0$. $a, b \in \mathbb{Z}, a \neq 0$

Gibt es eine ganze Zahl k mit $\boldsymbol{b = a \cdot k}$, dann nennen wir a einen **Teiler** von b.

Wir schreiben dafür kurz $\boldsymbol{a \mid b}$. Bei der Division $b : a = k$ bleibt also kein Rest.

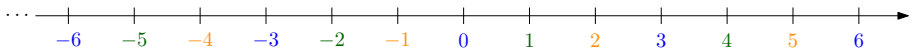
Gibt es *keine* solche ganze Zahl k, dann schreiben wir $a \nmid b$.

Teilbarkeit ganzer Zahlen

Entscheide jeweils, ob $\mid$ oder $\nmid$ stimmt: **a)** $4 \square -24$ **b)** $-5 \square 20$ **c)** $6 \square -3$ **d)** $-3 \square -15$ **e)** $2 \square 0$

Vielfache & Restklassen

Die **Vielfachen** einer ganzen Zahl m sind alle Zahlen der Form $m \cdot k$ mit $k \in \mathbb{Z}$.
Zum Beispiel: Die ganze Zahl 3 hat die Vielfachen $\{\ldots, -9, -6, -3, 0, 3, 6, 9, \ldots\}$.



Dividiert man ein Vielfaches von 3 durch 3, dann bleibt 0 Rest. Wir sagen auch:

Die Zahlenmenge $\overline{0} = \{\ldots, -9, -6, -3, 0, 3, 6, 9, \ldots\}$ ist die **Restklasse 0 modulo 3**.
Jede Zahl in der Restklasse 0 können wir in der Form $3 \cdot k + 0$ mit $k \in \mathbb{Z}$ schreiben.

Die Zahlenmenge $\overline{1} = \{\ldots, -8, -5, -2, 1, 4, 7, 10, \ldots\}$ ist die **Restklasse 1 modulo 3**.
Jede Zahl in der Restklasse 1 können wir in der Form $3 \cdot k + 1$ mit $k \in \mathbb{Z}$ schreiben.

Die Zahlenmenge $\overline{2} = \{\ldots, -7, -4, -1, 2, 5, 8, 11, \ldots\}$ ist die **Restklasse 2 modulo 3**.
Jede Zahl in der Restklasse 2 können wir in der Form $3 \cdot k + 2$ mit $k \in \mathbb{Z}$ schreiben.

Jede Zahl einer Restklasse heißt auch **Repräsentant** dieser Restklasse.

Meistens verwenden wir die m Zahlen 0, 1, 2, …, $m - 1$ als Repräsentanten ihrer Restklassen modulo m.

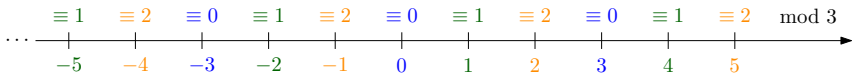
Kongruenz

Zwei Zahlen a und b liegen in der gleichen Restklasse modulo m.

Dann sagen wir: „a und b sind **kongruent** modulo m“ bzw. „a ist **kongruent** zu b modulo m“

Dann schreiben wir kurz: $\boldsymbol{a \equiv b \mod m}$ Der Ausdruck $a \equiv b \mod m$ ist keine Gleichung, sondern eine **Kongruenz**.

Zum Beispiel:

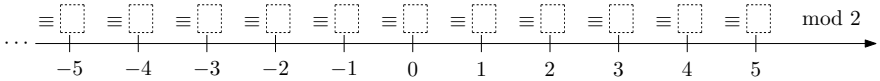


$3 \equiv 0 \mod 3$
$-5 \equiv 1 \mod 3$
$5 \equiv 2 \mod 3$
$-4 \equiv 5 \mod 3$

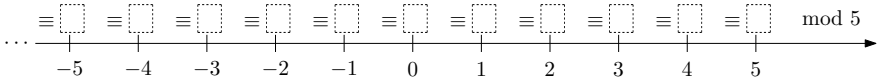
Kongruenz

Fülle in die Lücke jeweils jene Zahl aus $\{0, 1, 2, \ldots, m - 1\}$, zu der die Zahl modulo m kongruent ist.

a) $m = 2$



b) $m = 5$



Teilbarkeit ganzer Zahlen

a und b sind ganze Zahlen mit $a \neq 0$. $a, b \in \mathbb{Z}, a \neq 0$

Gibt es eine ganze Zahl k mit $\boldsymbol{b = a \cdot k}$, dann nennen wir a einen **Teiler** von b.

Wir schreiben dafür kurz $\boldsymbol{a \mid b}$. Bei der Division $b : a = k$ bleibt also kein Rest.

Gibt es *keine* solche ganze Zahl k, dann schreiben wir $a \nmid b$.

Teilbarkeit ganzer Zahlen

Entscheide jeweils, ob $\mid$ oder $\nmid$ stimmt: **a)** $4 \mid -24$ **b)** $-5 \mid 20$ **c)** $6 \nmid -3$ **d)** $-3 \mid -15$ **e)** $2 \mid 0$

Vielfache & Restklassen

Die **Vielfachen** einer ganzen Zahl m sind alle Zahlen der Form $m \cdot k$ mit $k \in \mathbb{Z}$.
Zum Beispiel: Die ganze Zahl 3 hat die Vielfachen $\{\ldots, -9, -6, -3, 0, 3, 6, 9, \ldots\}$.

Dividiert man ein Vielfaches von 3 durch 3, dann bleibt 0 Rest. Wir sagen auch:

Die Zahlenmenge $\overline{0} = \{\ldots, -9, -6, -3, 0, 3, 6, 9, \ldots\}$ ist die **Restklasse 0 modulo 3**.
Jede Zahl in der Restklasse 0 können wir in der Form $3 \cdot k + 0$ mit $k \in \mathbb{Z}$ schreiben.

Die Zahlenmenge $\overline{1} = \{\ldots, -8, -5, -2, 1, 4, 7, 10, \ldots\}$ ist die **Restklasse 1 modulo 3**.
Jede Zahl in der Restklasse 1 können wir in der Form $3 \cdot k + 1$ mit $k \in \mathbb{Z}$ schreiben.

Die Zahlenmenge $\overline{2} = \{\ldots, -7, -4, -1, 2, 5, 8, 11, \ldots\}$ ist die **Restklasse 2 modulo 3**.
Jede Zahl in der Restklasse 2 können wir in der Form $3 \cdot k + 2$ mit $k \in \mathbb{Z}$ schreiben.

Jede Zahl einer Restklasse heißt auch **Repräsentant** dieser Restklasse.

Meistens verwenden wir die m Zahlen 0, 1, 2, …, $m - 1$ als Repräsentanten ihrer Restklassen modulo m.

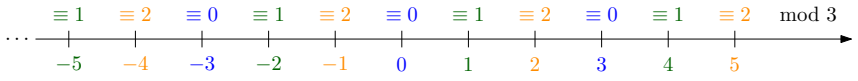
Kongruenz

Zwei Zahlen a und b liegen in der gleichen Restklasse modulo m.

Dann sagen wir: „a und b sind **kongruent** modulo m“ bzw. „a ist **kongruent** zu b modulo m“

Dann schreiben wir kurz: $\boldsymbol{a \equiv b \mod m}$ Der Ausdruck $a \equiv b \mod m$ ist keine Gleichung, sondern eine **Kongruenz**.

Zum Beispiel:

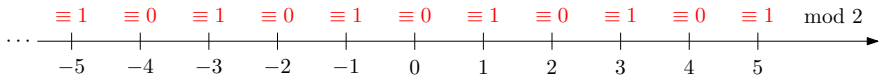


$3 \equiv 0 \mod 3$
$-5 \equiv 1 \mod 3$
$5 \equiv 2 \mod 3$
$-4 \equiv 5 \mod 3$

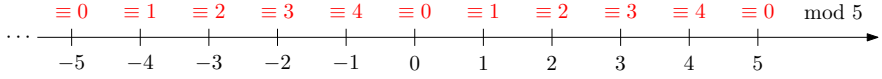
Kongruenz

Fülle in die Lücke jeweils jene Zahl aus $\{0, 1, 2, \ldots, m - 1\}$, zu der die Zahl modulo m kongruent ist.

a) $m = 2$



b) $m = 5$



Kongruenz?

Entscheide, ob die Zahlen kongruent ($\equiv$) oder nicht kongruent ($\not\equiv$) sind.

a) $27 \square 42 \mod 5$ **b)** $-3 \square 39 \mod 7$ **c)** $981273 \square 48276 \mod 2$ **d)** $846 \square 14322 \mod 3$

Äquivalente Aussagen

Zwei Zahlen a und b liegen genau dann in der gleichen Restklasse modulo m, wenn $a = b + k \cdot m$ mit einer passenden ganzen Zahl k gilt. a und b unterscheiden sich um ein Vielfaches von m.
Erkläre damit die folgende Aussage:

$$\boldsymbol{a \equiv b \mod m \iff m \mid a - b}$$

a und b liegen genau dann in der gleichen Restklasse modulo m, wenn m ein Teiler von $a - b$ ist.

Rechenregeln für Kongruenzen

Erkläre die folgenden Rechenregeln für Kongruenzen:

1) $\left.\begin{array}{l} a_1 \equiv b_1 \mod m \\ a_2 \equiv b_2 \mod m \end{array}\right\} \implies a_1 + a_2 \equiv b_1 + b_2 \mod m$

Wir dürfen zwei Kongruenzen modulo m addieren.

2) $a \equiv b \mod m \implies a \cdot c \equiv b \cdot c \mod m$

Wir dürfen beide Seiten einer Kongruenz mit der gleichen ganzen Zahl multiplizieren.

3) $\left.\begin{array}{l} a_1 \equiv b_1 \mod m \\ a_2 \equiv b_2 \mod m \end{array}\right\} \implies a_1 \cdot a_2 \equiv b_1 \cdot b_2 \mod m$

Wir dürfen zwei Kongruenzen modulo m multiplizieren.

Division

Wir dürfen beide Seiten einer Kongruenz *nicht* einfach durch einen gemeinsamen Teiler dividieren.
Zum Beispiel: $\underbrace{18}_{=3\cdot 6} \equiv \underbrace{30}_{=5\cdot 6} \mod 4$, aber $3 \not\equiv 5 \mod 4$.

Die richtige Rechenregel zum Kürzen gemeinsamer Teiler ist am Ende des Arbeitsblatts.

Kongruenz?

Entscheide, ob die Zahlen kongruent ($\equiv$) oder nicht kongruent ($\not\equiv$) sind.

a) $27 \equiv 42 \mod 5$ **b)** $-3 \equiv 39 \mod 7$ **c)** $981273 \not\equiv 48276 \mod 2$ **d)** $846 \equiv 14322 \mod 3$

Äquivalente Aussagen

Zwei Zahlen a und b liegen genau dann in der gleichen Restklasse modulo m, wenn $a = b + k \cdot m$ mit einer passenden ganzen Zahl k gilt. a und b unterscheiden sich um ein Vielfaches von m.
Erkläre damit die folgende Aussage:

$$\boldsymbol{a \equiv b \mod m \iff m \mid a - b}$$

a und b liegen genau dann in der gleichen Restklasse modulo m, wenn m ein Teiler von $a - b$ ist.

$$a \equiv b \mod m \iff a = b + k \cdot m \iff a - b = k \cdot m \iff m \mid a - b$$

Rechenregeln für Kongruenzen

Erkläre die folgenden Rechenregeln für Kongruenzen:

1) $\left.\begin{array}{l} a_1 \equiv b_1 \mod m \\ a_2 \equiv b_2 \mod m \end{array}\right\} \implies a_1 + a_2 \equiv b_1 + b_2 \mod m$

Wir dürfen zwei Kongruenzen modulo m addieren.

$$\left.\begin{array}{l} a_1 - b_1 = k_1 \cdot m \\ a_2 - b_2 = k_2 \cdot m \end{array}\right\} \implies (a_1 + a_2) - (b_1 + b_2) = \underbrace{(k_1 + k_2)}_{\in \mathbb{Z}} \cdot m \implies a_1 + a_2 \equiv b_1 + b_2 \mod m \checkmark$$

2) $a \equiv b \mod m \implies a \cdot c \equiv b \cdot c \mod m$

Wir dürfen beide Seiten einer Kongruenz mit der gleichen ganzen Zahl multiplizieren.

$$m \mid a - b \implies m \mid \underbrace{(a - b) \cdot c}_{a \cdot c - b \cdot c} \checkmark$$

3) $\left.\begin{array}{l} a_1 \equiv b_1 \mod m \\ a_2 \equiv b_2 \mod m \end{array}\right\} \implies a_1 \cdot a_2 \equiv b_1 \cdot b_2 \mod m$

Wir dürfen zwei Kongruenzen modulo m multiplizieren.

$$\left.\begin{array}{lll} a_1 \equiv b_1 \mod m & \implies & a_1 \cdot a_2 \equiv b_1 \cdot a_2 \mod m \\ a_2 \equiv b_2 \mod m & \implies & b_1 \cdot b_2 \equiv b_1 \cdot a_2 \mod m \end{array}\right\} \implies a_1 \cdot a_2 \equiv b_1 \cdot b_2 \mod m$$

Division

Wir dürfen beide Seiten einer Kongruenz *nicht* einfach durch einen gemeinsamen Teiler dividieren.
Zum Beispiel: $\underbrace{18}_{=3 \cdot 6} \equiv \underbrace{30}_{=5 \cdot 6} \mod 4$, aber $3 \not\equiv 5 \mod 4$.

Die richtige Rechenregel zum Kürzen gemeinsamer Teiler ist am Ende des Arbeitsblatts.

Division mit Rest

Gegeben sind zwei ganze Zahlen z und m.
Gesucht ist jene Zahl r aus $\{0, 1, 2, \ldots, m-1\}$, für die $z \equiv r \mod m$ gilt.
Zum Beispiel: $z = 208$, $m = 12$

1) Wir führen die Division mit Rest $z : m$ durch.

$$208 : 12 = 17 + \frac{4}{12}$$

„Wie oft geht 12 in 208? 17 Mal, 4 Rest“

2) Wir multiplizieren die Gleichung mit m.

$$208 = 17 \cdot 12 + 4$$

3) Aus den Rechenregeln für Kongruenzen folgt, dass der Rest $r = 4$ die gesuchte Zahl ist.

$$208 = \underbrace{17 \cdot 12}_{\equiv 0 \mod 12} + 4 \implies 208 \equiv 4 \mod 12$$

Division mit Rest

Ermittle jene Zahl r aus $\{0, 1, 2, \ldots, m-1\}$, für die $z \equiv r \mod m$ gilt, und fülle die Lücken aus.

a) $42 = \_\_\_ \cdot 8 + \_\_\_ \implies 42 \equiv \_\_\_ \mod 8$

b) $96 = \_\_\_ \cdot 7 + \_\_\_ \implies 96 \equiv \_\_\_ \mod 7$

c) $129 = \_\_\_ \cdot 2 + \_\_\_ \implies 129 \equiv \_\_\_ \mod 2$

d) $358 = \_\_\_ \cdot 6 + \_\_\_ \implies 358 \equiv \_\_\_ \mod 6$

Einerziffer entscheidet

a) *„Jede natürliche Zahl ist modulo* 10 *in der gleichen Restklasse wie ihre Einerziffer.“*
Wir erklären die Aussage anhand der Zahl $4723 = 4 \cdot 1000 + 7 \cdot 100 + 2 \cdot 10 + 3 \cdot 1$:

$$4723 = \underbrace{4 \cdot 100 \cdot 10}_{\equiv 0 \mod 10} + \underbrace{7 \cdot 10 \cdot 10}_{\equiv 0 \mod 10} + \underbrace{2 \cdot 10}_{\equiv 0 \mod 10} + \underbrace{3 \cdot 1}_{\equiv 3 \mod 10} \equiv 3 \mod 10$$

Eine natürliche Zahl ist also genau dann durch 10 teilbar, wenn ihre Einerziffer ___ ist.

b) *„Jede natürliche Zahl ist modulo* 5 *in der gleichen Restklasse wie ihre Einerziffer.“*
Erkläre die Aussage anhand der Zahl 4723:

$4723 =$

Eine natürliche Zahl ist also genau dann durch 5 teilbar, wenn ihre Einerziffer ___ oder ___ ist.

c) *„Jede natürliche Zahl ist modulo* 2 *in der gleichen Restklasse wie ihre Einerziffer.“*
Erkläre die Aussage anhand der Zahl 4723:

$4723 =$

Eine natürliche Zahl ist also genau dann durch 2 teilbar, wenn ihre Einerziffer ___, ___, ___, ___ oder ___ ist.

Division mit Rest

Gegeben sind zwei ganze Zahlen z und m.
Gesucht ist jene Zahl r aus $\{0, 1, 2, \ldots, m-1\}$, für die $z \equiv r \mod m$ gilt.
Zum Beispiel: $z = 208$, $m = 12$

1) Wir führen die Division mit Rest $z : m$ durch.

$$208 : 12 = 17 + \frac{4}{12}$$

„Wie oft geht 12 in 208? 17 Mal, 4 Rest“

2) Wir multiplizieren die Gleichung mit m.

$$208 = 17 \cdot 12 + 4$$

3) Aus den Rechenregeln für Kongruenzen folgt, dass der Rest $r = 4$ die gesuchte Zahl ist.

$$208 = \underbrace{17 \cdot 12}_{\equiv 0 \mod 12} + 4 \implies 208 \equiv 4 \mod 12$$

Division mit Rest

Ermittle jene Zahl r aus $\{0, 1, 2, \ldots, m-1\}$, für die $z \equiv r \mod m$ gilt, und fülle die Lücken aus.

a) $42 = 5 \cdot 8 + 2 \implies 42 \equiv 2 \mod 8$

b) $96 = 13 \cdot 7 + 5 \implies 96 \equiv 5 \mod 7$

c) $129 = 64 \cdot 2 + 1 \implies 129 \equiv 1 \mod 2$

d) $358 = 59 \cdot 6 + 4 \implies 358 \equiv 4 \mod 6$

Einerziffer entscheidet

a) *„Jede natürliche Zahl ist modulo* 10 *in der gleichen Restklasse wie ihre Einerziffer.“*
Wir erklären die Aussage anhand der Zahl $4723 = 4 \cdot 1000 + 7 \cdot 100 + 2 \cdot 10 + 3 \cdot 1$:

$$4723 = \underbrace{4 \cdot 100 \cdot 10}_{\equiv 0 \mod 10} + \underbrace{7 \cdot 10 \cdot 10}_{\equiv 0 \mod 10} + \underbrace{2 \cdot 10}_{\equiv 0 \mod 10} + \underbrace{3 \cdot 1}_{\equiv 3 \mod 10} \equiv 3 \mod 10$$

Eine natürliche Zahl ist also genau dann durch 10 teilbar, wenn ihre Einerziffer 0 ist.

b) *„Jede natürliche Zahl ist modulo* 5 *in der gleichen Restklasse wie ihre Einerziffer.“*
Erkläre die Aussage anhand der Zahl 4723:

$$4723 = \underbrace{4 \cdot 200 \cdot 5}_{\equiv 0 \mod 5} + \underbrace{7 \cdot 20 \cdot 5}_{\equiv 0 \mod 5} + \underbrace{2 \cdot 2 \cdot 5}_{\equiv 0 \mod 5} + \underbrace{3 \cdot 1}_{\equiv 3 \mod 5} \equiv 3 \mod 5$$

Eine natürliche Zahl ist also genau dann durch 5 teilbar, wenn ihre Einerziffer 0 oder 5 ist.

c) *„Jede natürliche Zahl ist modulo* 2 *in der gleichen Restklasse wie ihre Einerziffer.“*
Erkläre die Aussage anhand der Zahl 4723:

$$4723 = \underbrace{4 \cdot 500 \cdot 2}_{\equiv 0 \mod 2} + \underbrace{7 \cdot 50 \cdot 2}_{\equiv 0 \mod 2} + \underbrace{2 \cdot 5 \cdot 2}_{\equiv 0 \mod 2} + \underbrace{3 \cdot 1}_{\equiv 3 \mod 2} \equiv 3 \mod 2$$

Eine natürliche Zahl ist also genau dann durch 2 teilbar, wenn ihre Einerziffer 0, 2, 4, 6 oder 8 ist.

Ziffernsumme entscheidet

a) *„Jede natürliche Zahl ist modulo 9 in der gleichen Restklasse wie ihre Ziffernsumme."*
Wir erklären die Aussage anhand der Zahl $4723 = 4 \cdot (999+1) + 7 \cdot (99+1) + 2 \cdot (9+1) + 3 \cdot 1$:

$$4723 = \underbrace{4 \cdot 9 \cdot 111}_{\equiv 0 \bmod 9} + \mathbf{4} + \underbrace{7 \cdot 9 \cdot 11}_{\equiv 0 \bmod 9} + \mathbf{7} + \underbrace{2 \cdot 9 \cdot 1}_{\equiv 0 \bmod 9} + \mathbf{2} + \mathbf{3} \cdot 1 \equiv \mathbf{4+7+2+3} \mod 9$$

Eine natürliche Zahl ist also genau dann durch 9 teilbar, wenn ihre Ziffernsumme durch 9 teilbar ist.

b) *„Jede natürliche Zahl ist modulo 3 in der gleichen Restklasse wie ihre Ziffernsumme."*
Erkläre die Aussage anhand der Zahl 4723:

$4723 =$

Eine natürliche Zahl ist also genau dann durch 3 teilbar, wenn ihre Ziffernsumme durch 3 teilbar ist.

Addition und Multiplikation modulo 4

1) Fülle in den beiden Tabellen die Reste bei Division durch 4 aus.

Rest von $(\square + \square) : 4$				
$+$	0	1	2	3
0				
1				
2				
3				

Rest von $(\square \cdot \square) : 4$				
$\cdot$	0	1	2	3
0				
1				
2				
3				

2) Ermittle jene Zahl r aus $\{0,1,2,3\}$, für die $23 \equiv r \mod 4$ gilt, und fülle die Lücken aus.

$23 = \_\_\_ \cdot 4 + \_\_\_ \implies 23 \equiv \_\_\_ \mod 4$

Ermittle jene Zahl r aus $\{0,1,2,3\}$, für die $42 \equiv r \mod 4$ gilt, und fülle die Lücken aus.

$42 = \_\_\_ \cdot 4 + \_\_\_ \implies 42 \equiv \_\_\_ \mod 4$

3) Berechne $23+42 \mod 4$ und $23 \cdot 42 \mod 4$.

$23+42 \equiv \_\_\_ + \_\_\_ \equiv \_\_\_ \mod 4$ $\qquad$ $23 \cdot 42 \equiv \_\_\_ \cdot \_\_\_ \equiv \_\_\_ \mod 4$

4) Wie viele Lösungen hat die Kongruenz $2 \cdot x \equiv 2 \mod 4$ über der Grundmenge $\{0,1,2,3\}$?

5) Wie viele Lösungen hat die Kongruenz $2 \cdot x \equiv 1 \mod 4$ über der Grundmenge $\{0,1,2,3\}$?

Ziffernsumme entscheidet

a) *„Jede natürliche Zahl ist modulo 9 in der gleichen Restklasse wie ihre Ziffernsumme."*
Wir erklären die Aussage anhand der Zahl $4723 = 4 \cdot (999 + 1) + 7 \cdot (99 + 1) + 2 \cdot (9 + 1) + 3 \cdot 1$:

$$4723 = \underbrace{4 \cdot 9 \cdot 111}_{\equiv 0 \bmod 9} + \mathbf{4} + \underbrace{7 \cdot 9 \cdot 11}_{\equiv 0 \bmod 9} + \mathbf{7} + \underbrace{2 \cdot 9 \cdot 1}_{\equiv 0 \bmod 9} + \mathbf{2} + \mathbf{3} \cdot 1 \equiv \mathbf{4 + 7 + 2 + 3} \mod 9$$

Eine natürliche Zahl ist also genau dann durch 9 teilbar, wenn ihre Ziffernsumme durch 9 teilbar ist.

b) *„Jede natürliche Zahl ist modulo 3 in der gleichen Restklasse wie ihre Ziffernsumme."*
Erkläre die Aussage anhand der Zahl 4723:

$$4723 = \underbrace{4 \cdot 3 \cdot 333}_{\equiv 0 \bmod 3} + \mathbf{4} + \underbrace{7 \cdot 3 \cdot 33}_{\equiv 0 \bmod 3} + \mathbf{7} + \underbrace{2 \cdot 3 \cdot 3}_{\equiv 0 \bmod 3} + \mathbf{2} + \mathbf{3} \cdot 1 \equiv \mathbf{4 + 7 + 2 + 3} \mod 3$$

Eine natürliche Zahl ist also genau dann durch 3 teilbar, wenn ihre Ziffernsumme durch 3 teilbar ist.

Addition und Multiplikation modulo 4

1) Fülle in den beiden Tabellen die Reste bei Division durch 4 aus.

Rest von ($\square$ + $\square$) : 4				
+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Rest von ($\square$ · $\square$) : 4				
·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

2) Ermittle jene Zahl r aus $\{0, 1, 2, 3\}$, für die $23 \equiv r \mod 4$ gilt, und fülle die Lücken aus.

$23 = 5 \cdot 4 + 3 \implies 23 \equiv 3 \mod 4$

Ermittle jene Zahl r aus $\{0, 1, 2, 3\}$, für die $42 \equiv r \mod 4$ gilt, und fülle die Lücken aus.

$42 = 10 \cdot 4 + 2 \implies 42 \equiv 2 \mod 4$

3) Berechne $23 + 42 \mod 4$ und $23 \cdot 42 \mod 4$.

$23 + 42 \equiv 3 + 2 \equiv 1 \mod 4$ $\qquad$ $23 \cdot 42 \equiv 3 \cdot 2 \equiv 2 \mod 4$

4) Wie viele Lösungen hat die Kongruenz $2 \cdot x \equiv 2 \mod 4$ über der Grundmenge $\{0, 1, 2, 3\}$?
2 Lösungen: $x = 1$ und $x = 3$

5) Wie viele Lösungen hat die Kongruenz $2 \cdot x \equiv 1 \mod 4$ über der Grundmenge $\{0, 1, 2, 3\}$?
Keine Lösung.

42

Addition und Multiplikation modulo 5

1) Fülle in den beiden Tabellen die Reste bei Division durch 5 aus.

Rest von (□ + □) : 5					
+	0	1	2	3	4
0					
1					
2					
3					
4					

Rest von (□ · □) : 5					
·	0	1	2	3	4
0					
1					
2					
3					
4					

2) Was fällt dir in der Tabelle zum Multiplizieren auf?

3) a und b sind jeweils Zahlen aus $\{1, 2, 3, 4\}$.
Wie viele Lösungen hat die Kongruenz $a \cdot x \equiv b \mod 5$ über der Grundmenge $\{0, 1, 2, 3, 4\}$?

Addition und Multiplikation modulo 6

1) Fülle in den beiden Tabellen die Reste bei Division durch 6 aus.

Rest von (□ + □) : 6						
+	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

Rest von (□ · □) : 6						
·	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

2) Welche Zeilen in der Multiplikations-Tabelle enthalten alle Zahlen aus $\{0, 1, 2, 3, 4, 5\}$ genau einmal?
Es gibt einen Zusammenhang mit dem Divisor 6. Hast du eine Vermutung?

Addition und Multiplikation modulo 5

1) Fülle in den beiden Tabellen die Reste bei Division durch 5 aus.

Rest von ($\square$ + $\square$) : 5					
+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Rest von ($\square$ · $\square$) : 5					
·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

2) Was fällt dir in der Tabelle zum Multiplizieren auf?
Bis auf in der ersten Zeile enthält jede Zeile alle Zahlen aus $\{0, 1, 2, 3, 4\}$ genau einmal.

3) a und b sind jeweils Zahlen aus $\{1, 2, 3, 4\}$.
Wie viele Lösungen hat die Kongruenz $a \cdot x \equiv b \mod 5$ über der Grundmenge $\{0, 1, 2, 3, 4\}$?
Genau eine Lösung.

Addition und Multiplikation modulo 6

1) Fülle in den beiden Tabellen die Reste bei Division durch 6 aus.

Rest von ($\square$ + $\square$) : 6						
+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Rest von ($\square$ · $\square$) : 6						
·	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

2) Welche Zeilen in der Multiplikations-Tabelle enthalten alle Zahlen aus $\{0, 1, 2, 3, 4, 5\}$ genau einmal?

Es gibt einen Zusammenhang mit dem Divisor 6. Hast du eine Vermutung?

Die Zeile $1 \cdot x$ und die Zeile $5 \cdot x$.
1 und 5 sind genau die Zahlen aus $\{1, 2, 3, 4, 5\}$, die zu 6 teilerfremd sind.

Kürzungsregel für Kongruenzen

Wir dürfen beide Seiten einer Kongruenz modulo m durch einen gemeinsamen Teiler c dividieren. Bei der neuen Kongruenz müssen wir aber modulo $\frac{m}{\mathrm{ggT}(c,m)}$ rechnen:

$$\boldsymbol{a \cdot c \equiv b \cdot c \mod m \overset{(\star)}{\iff} a \equiv b \mod \frac{m}{\mathrm{ggT}(c,m)}}$$

Zum Beispiel: $\underbrace{18}_{=3\cdot 6} \equiv \underbrace{30}_{=5\cdot 6} \mod 4 \iff 3 \equiv 5 \mod 2$, weil $\mathrm{ggT}(6,4) = 2$.

Auf dem AB – Fundamentalsatz der Arithmetik haben wir $m \mid c \cdot N \iff \frac{m}{\mathrm{ggT}(c,m)} \mid N$ gezeigt. Damit können wir $(\star)$ begründen:

$$\begin{aligned}
a \cdot c \equiv b \cdot c \mod m &\iff \underbrace{a \cdot c - b \cdot c}_{c\cdot(a-b)} \equiv 0 \mod m \iff \\
&\iff m \mid c \cdot (a-b) \overset{N=a-b}{\iff} \\
&\iff \tfrac{m}{\mathrm{ggT}(c,m)} \mid a-b \iff \\
&\iff a - b \equiv 0 \mod \tfrac{m}{\mathrm{ggT}(c,m)} \iff \\
&\iff a \equiv b \mod \tfrac{m}{\mathrm{ggT}(c,m)}
\end{aligned}$$

Addition und Multiplikation modulo 7

1) Fülle in den beiden Tabellen die Reste bei Division durch 7 aus.

Rest von (□ + □) : 7							
+	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

Rest von (□ · □) : 7							
·	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

2) c ist eine Zahl aus $\{1,2,3,4,5,6\}$.
Die Spalte $x \cdot c \mod 7$ enthält jede der Zahlen $\{0,1,2,3,4,5,6\}$ genau einmal.
Begründe, warum das aus der Rechenregel für die Division folgt.

Kürzungsregel für Kongruenzen

Wir dürfen beide Seiten einer Kongruenz modulo m durch einen gemeinsamen Teiler c dividieren. Bei der neuen Kongruenz müssen wir aber modulo $\frac{m}{\mathrm{ggT}(c,m)}$ rechnen:

$$\boldsymbol{a \cdot c \equiv b \cdot c \mod m \overset{(\star)}{\iff} a \equiv b \mod \frac{m}{\mathrm{ggT}(c,m)}}$$

Zum Beispiel: $\underbrace{18}_{=3\cdot 6} \equiv \underbrace{30}_{=5\cdot 6} \mod 4 \iff 3 \equiv 5 \mod 2$, weil $\mathrm{ggT}(6,4) = 2$.

Auf dem AB – Fundamentalsatz der Arithmetik haben wir $m \mid c \cdot N \iff \frac{m}{\mathrm{ggT}(c,m)} \mid N$ gezeigt. Damit können wir $(\star)$ begründen:

$$\begin{aligned}
a \cdot c \equiv b \cdot c \mod m &\iff \underbrace{a \cdot c - b \cdot c}_{c\cdot(a-b)} \equiv 0 \mod m \iff \\
&\iff m \mid c \cdot (a-b) \overset{N=a-b}{\iff} \\
&\iff \tfrac{m}{\mathrm{ggT}(c,m)} \mid a-b \iff \\
&\iff a - b \equiv 0 \mod \tfrac{m}{\mathrm{ggT}(c,m)} \iff \\
&\iff a \equiv b \mod \tfrac{m}{\mathrm{ggT}(c,m)}
\end{aligned}$$

Addition und Multiplikation modulo 7

1) Fülle in den beiden Tabellen die Reste bei Division durch 7 aus.

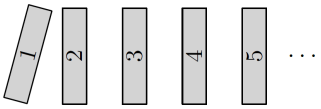
Rest von (□ + □) : 7							
+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Rest von (□ · □) : 7							
·	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

2) c ist eine Zahl aus $\{1,2,3,4,5,6\}$.
Die Spalte $x \cdot c \mod 7$ enthält jede der Zahlen $\{0,1,2,3,4,5,6\}$ genau einmal.
Begründe, warum das aus der Rechenregel für die Division folgt.
7 ist eine Primzahl, also ist $\mathrm{ggT}(c,7) = 1$
Aus $a \cdot c \equiv b \cdot c \mod 7$ folgt mit der Rechenregel $a \equiv b \mod 7$.
Die Zahlen $\{0,1,2,3,4,5,6\}$ sind in verschiedenen Restklassen modulo 7.
Also müssen $0 \cdot c$, $1 \cdot c$, $2 \cdot c$, …, $6 \cdot c$ in verschiedenen Restklassen modulo 7 sein.

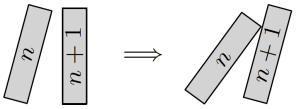
2.5 Vollständige Induktion

Zu jeder natürlichen Zahl $n \geq 1$ haben wir einen Dominostein, der mit n beschriftet ist. Wir stellen die Dominosteine in aufsteigender Reihenfolge auf.
Angenommen, wir haben die folgenden beiden Informationen:



1) Der Dominostein mit der Zahl 1 fällt um.

2) Für jede Zahl $n = 1, 2, 3, 4, 5, \ldots$ gilt:
Wenn der Dominostein mit der Zahl n umfällt,
dann fällt auch der Dominostein mit der Zahl $n + 1$ um.



Dann wissen wir damit sicher, dass jeder dieser Dominosteine schließlich umfällt.

Diese Grundidee hinter der Beweistechnik „Vollständige Induktion“ ist so elementar, dass sie wohl schon weit früher implizit angewandt wurde [1], bevor sie Blaise Pascal im 17. Jahrhundert im Buch „Traité du triangle arithmétique“ [23] explizit ausformuliert und verwendet hat.

Am folgenden Arbeitsblatt sind das Beweisprinzip und einige klassische Induktionsbeweise aufbereitet. Mit vollständiger Induktion können wir auf den darauffolgenden Arbeitsblättern auch den Binomischen Lehrsatz und den Kleinen Satz von Fermat beweisen.

Wenn eine Aussage für aufeinanderfolgende ganze Zahlen zu zeigen ist, dann liefert die vollständige Induktion einen ersten Ansatz, mit dem man einen Beweis versuchen kann. Freilich garantiert ein gelungener Beweis mit vollständiger Induktion *nicht*, dass es keinen kürzeren oder eleganteren Beweis geben kann. Der Binomische Lehrsatz und der Kleine Satz von Fermat sind gute Beispiele dafür.

Es folgt das Arbeitsblatt – Vollständige Induktion und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Termrechnung auf Komplexitätsniveau der 9. Schulstufe

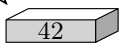
Lernziele:

✓ Was ist die Grundidee hinter **vollständiger Induktion**?

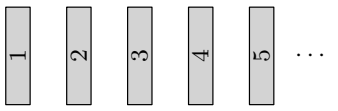
✓ Wie kann man diese Beweistechnik anwenden?

Dominoeffekt

Zu jeder natürlichen Zahl $n \geq 1$ haben wir einen Dominostein, der mit n beschriftet ist.

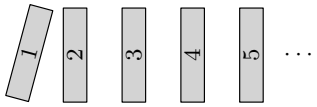


Wir stellen die Dominosteine in aufsteigender Reihenfolge auf.
Angenommen, wir haben die folgenden beiden Informationen:

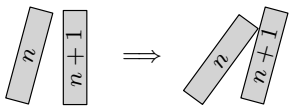


1) Der Dominostein mit der Zahl 1 fällt um.

2) Für jede Zahl $n = 1, 2, 3, 4, 5, \ldots$ gilt:
Wenn der Dominostein mit der Zahl n umfällt,
dann fällt auch der Dominostein mit der Zahl $n + 1$ um.



Erkläre, warum dann der Dominostein mit der Zahl 42 sicher umfällt.
Schließlich fällt jeder Dominostein um.



Vollständige Induktion

Die **vollständige Induktion** ist eine Beweistechnik, die auf diesem Dominoeffekt aufbaut.
Wir wollen zum Beispiel beweisen, dass die Formel

$$\underbrace{1 + 2 + 3 + \cdots + n}_{n \text{ Summanden}} = \frac{n \cdot (n+1)}{2}$$

Zum Beispiel: $\underbrace{1+2+3+4}_{=10} = \underbrace{\frac{4 \cdot 5}{2}}_{=10}$ ✓

für alle natürlichen Zahlen $n = 1, 2, 3, \ldots$ gilt. Dazu überprüfen wir Folgendes:

1) Induktionsanfang: Die Formel gilt für $n = 1$. Der erste Dominostein fällt um.

2) Induktionsschritt: Wenn die Formel für n gilt, dann gilt auch die Formel für $n + 1$. $n \geq 1$
Wenn der Dominostein mit der Zahl n umfällt, dann fällt auch der nächste Dominostein mit der Zahl $n + 1$ um.

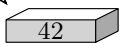
1) Überprüfe den **Induktionsanfang** für $n = 1$:

2) Überprüfe den **Induktionsschritt** $n \to n + 1$:
Du darfst also verwenden, dass $1 + 2 + 3 + \cdots + n = \frac{n \cdot (n+1)}{2}$ gilt. **„Induktionsvoraussetzung“**
Daraus musst du folgern, dass auch $1 + 2 + 3 + \cdots + n + (n + 1) = \frac{(n+1) \cdot (n+2)}{2}$ gilt.

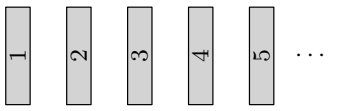
Die Formel gilt also für alle natürlichen Zahlen $n = 1, 2, 3, \ldots$

Dominoeffekt

Zu jeder natürlichen Zahl $n \geq 1$ haben wir einen Dominostein, der mit n beschriftet ist.

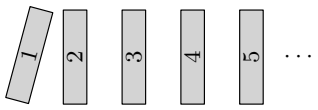


Wir stellen die Dominosteine in aufsteigender Reihenfolge auf.
Angenommen, wir haben die folgenden beiden Informationen:

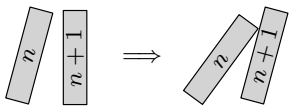


1) Der Dominostein mit der Zahl 1 fällt um.

2) Für jede Zahl $n = 1, 2, 3, 4, 5, \ldots$ gilt:
Wenn der Dominostein mit der Zahl n umfällt,
dann fällt auch der Dominostein mit der Zahl $n + 1$ um.



Erkläre, warum dann der Dominostein mit der Zahl 42 sicher umfällt.



Schließlich fällt jeder Dominostein um.

Der erste Stein fällt um.
Deshalb fällt der zweite Stein um. $n = 1$
Deshalb fällt der dritte Stein um. $n = 2$
⋮
Deshalb fällt der Stein mit der Zahl 41 um. $n = 40$
Deshalb fällt der Stein mit der Zahl 42 um. $n = 41$

Vollständige Induktion

Die **vollständige Induktion** ist eine Beweistechnik, die auf diesem Dominoeffekt aufbaut.
Wir wollen zum Beispiel beweisen, dass die Formel

$$\underbrace{1 + 2 + 3 + \cdots + n}_{n \text{ Summanden}} = \frac{n \cdot (n+1)}{2}$$

Zum Beispiel: $\underbrace{1+2+3+4}_{=10} = \underbrace{\frac{4 \cdot 5}{2}}_{=10}$ ✓

für alle natürlichen Zahlen $n = 1, 2, 3, \ldots$ gilt. Dazu überprüfen wir Folgendes:

1) Induktionsanfang: Die Formel gilt für $n = 1$. Der erste Dominostein fällt um.

2) Induktionsschritt: Wenn die Formel für n gilt, dann gilt auch die Formel für $n + 1$. $n \geq 1$
Wenn der Dominostein mit der Zahl n umfällt, dann fällt auch der nächste Dominostein mit der Zahl $n + 1$ um.

1) Überprüfe den **Induktionsanfang** für $n = 1$:

$$1 \overset{?}{=} \underbrace{\frac{1 \cdot 2}{2}}_{=1} \quad ✓$$

2) Überprüfe den **Induktionsschritt** $n \to n + 1$:

Du darfst also verwenden, dass $1 + 2 + 3 + \cdots + n = \frac{n \cdot (n+1)}{2}$ gilt. **„Induktionsvoraussetzung“**

Daraus musst du folgern, dass auch $1 + 2 + 3 + \cdots + n + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$ gilt.

$$1 + 2 + 3 + \cdots + n + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = (n+1) \cdot \left(\underbrace{\frac{n}{2} + 1}_{=\frac{n+2}{2}} \right) = \frac{(n+1) \cdot (n+2)}{2} \quad ✓$$

Die Formel gilt also für alle natürlichen Zahlen $n = 1, 2, 3, \ldots$

Summe der ersten n ungeraden Zahlen

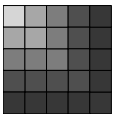
Zeige mit vollständiger Induktion, dass die Formel

$$\underbrace{1+3+5+\cdots+(2\cdot n-1)}_{n \text{ Summanden}} = n^2$$

für alle natürlichen Zahlen $n \geq 1$ gilt.

Zum Beispiel: $\underbrace{1+3+5+7+9}_{=25} = \underbrace{5^2}_{=25}$ ✓

Alternativer geometrischer Beweis:



Summe der ersten n Quadratzahlen

Zeige mit vollständiger Induktion, dass die Formel

$$\underbrace{1^2+2^2+3^2+\cdots+n^2}_{n \text{ Summanden}} = \frac{n\cdot(n+1)\cdot(n+\frac{1}{2})}{3}$$

für alle natürlichen Zahlen $n \geq 1$ gilt.

Summe der ersten n ungeraden Zahlen

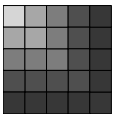
Zeige mit vollständiger Induktion, dass die Formel

$$\underbrace{1+3+5+\cdots+(2\cdot n-1)}_{n \text{ Summanden}} = n^2$$

für alle natürlichen Zahlen $n \geq 1$ gilt.

Zum Beispiel: $\underbrace{1+3+5+7+9}_{=25} = \underbrace{5^2}_{=25}$ ✓

Alternativer geometrischer Beweis:



1) Induktionsanfang $n = 1$:

$$1 \stackrel{?}{=} 1^2 \checkmark$$

2) Induktionsschritt $n \to n+1$:

$$1+3+5+\cdots+(2\cdot n-1)+(2\cdot n+1) = n^2+2\cdot n+1 = (n+1)^2 \checkmark$$

Summe der ersten n Quadratzahlen

Zeige mit vollständiger Induktion, dass die Formel

$$\underbrace{1^2+2^2+3^2+\cdots+n^2}_{n \text{ Summanden}} = \frac{n\cdot(n+1)\cdot(n+\frac{1}{2})}{3}$$

für alle natürlichen Zahlen $n \geq 1$ gilt.

1) Induktionsanfang $n = 1$:

$$1^2 \stackrel{?}{=} \underbrace{\frac{1\cdot 2\cdot 1{,}5}{3}}_{=1} \checkmark$$

2) Induktionsschritt $n \to n+1$:

$$1^2+2^2+3^2+\cdots+n^2+(n+1)^2 = \frac{n\cdot(n+1)\cdot(n+\frac{1}{2})}{3}+(n+1)^2 = (n+1)\cdot\left[\frac{n\cdot(n+\frac{1}{2})}{3}+n+1\right]$$

Zu zeigen ist also, dass $\frac{n\cdot(n+\frac{1}{2})}{3}+n+1 = \frac{(n+2)\cdot(n+\frac{3}{2})}{3}$ gilt.

$$\iff n\cdot\left(n+\frac{1}{2}\right)+3\cdot n+3 = (n+2)\cdot\left(n+\frac{3}{2}\right) \iff$$

$$\iff n^2+\underbrace{\frac{n}{2}+3\cdot n}_{=\frac{7\cdot n}{2}}+3 = n^2+\underbrace{\frac{3\cdot n}{2}+2\cdot n}_{=\frac{7\cdot n}{2}}+3 \checkmark$$

Rekursive Darstellung → Explizite Darstellung

Die Folge $\langle a_n \rangle$ ist rekursiv definiert:

$$a_1 = 0 \quad \text{und} \quad a_{n+1} = 4 \cdot a_n + 2, \qquad n \geq 1$$

Zeige mit vollständiger Induktion, dass

$$a_n = \frac{4^n - 4}{6}$$

für alle natürlichen Zahlen $n \geq 1$ gilt.

Es gilt also:

$a_2 = 4 \cdot a_1 + 2 = 2$
$a_3 = 4 \cdot a_2 + 2 = 10$
$a_4 = 4 \cdot a_3 + 2 = 42$
$\vdots$

Fibonacci-Folge

Die Fibonacci-Folge $\langle f_n \rangle$ ist rekursiv definiert:

$$f_1 = 1,\ f_2 = 1 \quad \text{und} \quad f_{n+2} = f_{n+1} + f_n, \qquad n \geq 1$$

Zeige mit vollständiger Induktion, dass

$$f_{n+1} \cdot f_{n-1} - f_n^2 = (-1)^n$$

für alle natürlichen Zahlen $n \geq 2$ gilt.

Es gilt also:

$f_3 = f_2 + f_1 = 2$
$f_4 = f_3 + f_2 = 3$
$f_5 = f_4 + f_3 = 5$
$\vdots$

Der Induktionsanfang ist in dieser Aufgabe also bei $n = 2$.

Rekursive Darstellung → Explizite Darstellung

Die Folge $\langle a_n \rangle$ ist rekursiv definiert:

$$a_1 = 0 \quad \text{und} \quad a_{n+1} = 4 \cdot a_n + 2, \qquad n \geq 1$$

Es gilt also:

$a_2 = 4 \cdot a_1 + 2 = 2$
$a_3 = 4 \cdot a_2 + 2 = 10$
$a_4 = 4 \cdot a_3 + 2 = 42$
$\vdots$

Zeige mit vollständiger Induktion, dass

$$a_n = \frac{4^n - 4}{6}$$

für alle natürlichen Zahlen $n \geq 1$ gilt.

1) Induktionsanfang $n = 1$:

$$\underbrace{a_1}_{=0} \stackrel{?}{=} \underbrace{\frac{4^1 - 4}{6}}_{=0} \quad \checkmark$$

2) Induktionsschritt $n \to n + 1$:

$$a_{n+1} = 4 \cdot a_n + 2 = 4 \cdot \frac{4^n - 4}{6} + 2 = \frac{4^{n+1} - 16}{6} + \frac{12}{6} = \frac{4^{n+1} - 4}{6} \quad \checkmark$$

Fibonacci-Folge

Die Fibonacci-Folge $\langle f_n \rangle$ ist rekursiv definiert:

$$f_1 = 1,\ f_2 = 1 \quad \text{und} \quad f_{n+2} = f_{n+1} + f_n, \qquad n \geq 1$$

Es gilt also:

$f_3 = f_2 + f_1 = 2$
$f_4 = f_3 + f_2 = 3$
$f_5 = f_4 + f_3 = 5$
$\vdots$

Zeige mit vollständiger Induktion, dass

$$f_{n+1} \cdot f_{n-1} - f_n^2 = (-1)^n$$

für alle natürlichen Zahlen $n \geq 2$ gilt.

Der Induktionsanfang ist in dieser Aufgabe also bei $n = 2$.

1) Induktionsanfang $n = 2$:

$$\underbrace{f_3 \cdot f_1 - f_2^2}_{=2 \cdot 1 - 1^2 = 1} \stackrel{?}{=} \underbrace{(-1)^2}_{=1} \quad \checkmark$$

2) Induktionsschritt $n \to n + 1$:

$$\begin{aligned} f_{n+2} \cdot f_n - f_{n+1}^2 &= (f_{n+1} + f_n) \cdot f_n - f_{n+1} \cdot (f_n + f_{n-1}) = \\ &= \cancel{f_{n+1} \cdot f_n} + f_n^2 - \cancel{f_{n+1} \cdot f_n} - f_{n+1} \cdot f_{n-1} = \\ &= f_n^2 - f_{n+1} \cdot f_{n-1} = \\ &= (-1) \cdot (f_{n+1} \cdot f_{n-1} - f_n^2) = \\ &= (-1) \cdot (-1)^n = (-1)^{n+1} \quad \checkmark \end{aligned}$$

Ungleichung

Zeige mit vollständiger Induktion, dass die Ungleichung

$$3^n \geq n^3$$

für alle natürlichen Zahlen $n \geq 3$ gilt.

Hinweis: Überlege im Induktionsschritt, warum für $n \geq 3$ die Ungleichungen $3 \cdot n^2 \leq n^3$ und $3 \cdot n + 1 \leq n^3$ gelten.

Bernoulli-Ungleichung

Zeige mit vollständiger Induktion, dass die **Bernoulli-Ungleichung**

$$\boldsymbol{(1+x)^n \geq 1 + n \cdot x}, \qquad x \geq -1$$

für alle Zahlen $n = 0, 1, 2, 3, \ldots$ gilt.

An welcher Stelle im Beweis verwendest du die Voraussetzung $x \geq -1$?

Ungleichung

Zeige mit vollständiger Induktion, dass die Ungleichung

$$3^n \geq n^3$$

für alle natürlichen Zahlen $n \geq 3$ gilt.

Hinweis: Überlege im Induktionsschritt, warum für $n \geq 3$ die Ungleichungen $3 \cdot n^2 \leq n^3$ und $3 \cdot n + 1 \leq n^3$ gelten.

1) Induktionsanfang $n = 3$:

$$3^3 \overset{?}{\geq} 3^3 \checkmark$$

2) Induktionsschritt $n \to n + 1$:

$$(n+1)^3 = n^3 + 3 \cdot n^2 + 3 \cdot n + 1$$

Es gilt $3 \cdot n^2 \leq n^3$, weil $3 \leq n$.
Es gilt $3 \cdot n + 1 \leq 4 \cdot n \leq n^3$, weil $4 \leq n^2$.

$$\Longrightarrow (n+1)^3 \leq n^3 + n^3 + n^3 = 3 \cdot n^3 \leq 3 \cdot 3^n = 3^{n+1}$$

Bernoulli-Ungleichung

Zeige mit vollständiger Induktion, dass die **Bernoulli-Ungleichung**

$$\boldsymbol{(1+x)^n \geq 1 + n \cdot x}, \qquad x \geq -1$$

für alle Zahlen $n = 0, 1, 2, 3, \ldots$ gilt.

1) Induktionsanfang $n = 0$:

$$\underbrace{(1+x)^0}_{=1} \overset{?}{\geq} \underbrace{1 + 0 \cdot x}_{=1} \checkmark$$

2) Induktionsschritt $n \to n + 1$:

$$\begin{aligned}(1+x)^{n+1} &= (1+x) \cdot (1+x)^n \geq \\ &\geq (1+x) \cdot (1 + n \cdot x) = \\ &= 1 + n \cdot x + x + \underbrace{n \cdot x^2}_{\geq 0} \geq \\ &\geq 1 + (n+1) \cdot x \checkmark\end{aligned}$$

An welcher Stelle im Beweis verwendest du die Voraussetzung $x \geq -1$?

2.6 Binomischer Lehrsatz

Der Binomische Lehrsatz

$$(a+b)^2 = a^2 + 2 \cdot a \cdot b + b^2$$

für den Exponenten $n = 2$ war bereits Euklid bekannt:
„If a straight line be cut at random, the square on the whole is equal to the squares on the segments and twice the rectangle contained by the segments.“ ([36], Book II, Prop. 4)

Der allgemeine Fall für natürliche Exponenten

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot a^k \cdot b^{n-k}$$

und dessen Zusammenhang mit den Zahlen im „Pascalschen Dreieck“ waren bereits mehr als 500 Jahre vor dem im 17. Jahrhundert wirkenden Blaise Pascal im arabischen Raum bekannt ([38]).

Auf dem nächsten Arbeitsblatt ist ein Beweis für den Binomischen Lehrsatz mit vollständiger Induktion aufbereitet. Ein kürzerer kombinatorischer Beweis befindet sich auf dem Arbeitsblatt – Pascalsches Dreieck II.

Es folgt das Arbeitsblatt – Binomischer Lehrsatz und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

- Termrechnung auf Komplexitätsniveau der 9. Schulstufe
- Operieren mit dem Summenzeichen $\sum_{k=1}^{n} f(k) = f(1) + f(2) + \cdots + f(n)$
- Arbeitsblatt – Vollständige Induktion

Lernziele:

- ✓ Was ist der **Binomialkoeffizient** $\binom{\boldsymbol{n}}{\boldsymbol{k}}$?
- ✓ Was ist das **Pascalsche Dreieck**?
- ✓ Was sagt der **Binomische Lehrsatz** aus?
  Wie kann man ihn mit vollständiger Induktion beweisen?

Binomialkoeffizient

Der **Binomialkoeffizient** $\binom{n}{k}$ kann folgendermaßen berechnet werden:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \quad \text{für alle } k, n \in \mathbb{N} \quad \text{mit} \quad 0 \leq k \leq n.$$

Dabei ist $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$ und $0! = 1$.

Sprechweisen: $\binom{n}{k} \leftrightarrow$ „n über k“
$n! \leftrightarrow$ „n Fakultät“ bzw. „n Faktorielle“

Wie viele Möglichkeiten gibt es, um aus n Personen eine Gruppe von k Personen auszuwählen?
Die Lösung dieses Abzählproblems ist der Binomialkoeffizient $\binom{n}{k}$. Eine Erklärung dafür findest du auf dem AB – Kombinatorik.

Pascalsches Dreieck

Das links dargestellte **Pascalsche Dreieck** ist durch folgende beide Regeln eindeutig festgelegt:

1) Jede Zahl am linken Rand und am rechten Rand ist 1.

2) Jede Zahl im Inneren ist die Summe der beiden benachbarten Zahlen in der Zeile darüber.

Die Zahlen im Pascalschen Dreieck sind genau die rechts dargestellten Binomialkoeffizienten:

$$\begin{array}{ccccccccccc} &&&&&1&&&&& \\ &&&&1&&1&&&& \\ &&&1&&2&&1&&& \\ &&1&&3&&3&&1&& \\ &1&&4&&6&&4&&1& \\ 1&&5&&10&&10&&5&&1 \end{array} \quad = \quad \begin{array}{ccccccccccc} &&&&&\binom{0}{0}&&&&& \\ &&&&\binom{1}{0}&&\binom{1}{1}&&&& \\ &&&\binom{2}{0}&&\binom{2}{1}&&\binom{2}{2}&&& \\ &&\binom{3}{0}&&\binom{3}{1}&&\binom{3}{2}&&\binom{3}{3}&& \\ &\binom{4}{0}&&\binom{4}{1}&&\binom{4}{2}&&\binom{4}{3}&&\binom{4}{4}& \\ \binom{5}{0}&&\binom{5}{1}&&\binom{5}{2}&&\binom{5}{3}&&\binom{5}{4}&&\binom{5}{5} \end{array}$$

Eine kombinatorische Erklärung dafür findest du auf dem AB – Pascalsches Dreieck II.

Wir können die Gleichheit auch rechnerisch überprüfen:

1) Rechne nach, dass $\binom{n}{0} = 1$ und $\binom{n}{n} = 1$ für alle $n \in \mathbb{N}$ gilt.

2) Rechne nach, dass $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ für alle $0 < k < n$ gilt.

Binomialkoeffizient

Der **Binomialkoeffizient** $\binom{n}{k}$ kann folgendermaßen berechnet werden:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \quad \text{für alle } k, n \in \mathbb{N} \quad \text{mit} \quad 0 \leq k \leq n.$$

Dabei ist $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$ und $0! = 1$.

Sprechweisen: $\binom{n}{k} \leftrightarrow$ „n über k“
$n! \leftrightarrow$ „n Fakultät“ bzw. „n Faktorielle“

Wie viele Möglichkeiten gibt es, um aus n Personen eine Gruppe von k Personen auszuwählen?
Die Lösung dieses Abzählproblems ist der Binomialkoeffizient $\binom{n}{k}$. Eine Erklärung dafür findest du auf dem AB – Kombinatorik.

Pascalsches Dreieck

Das links dargestellte **Pascalsche Dreieck** ist durch folgende beide Regeln eindeutig festgelegt:

1) Jede Zahl am linken Rand und am rechten Rand ist 1.

2) Jede Zahl im Inneren ist die Summe der beiden benachbarten Zahlen in der Zeile darüber.

Die Zahlen im Pascalschen Dreieck sind genau die rechts dargestellten Binomialkoeffizienten:

$$\begin{matrix} & & & & & 1 & & & & & \\ & & & & 1 & & 1 & & & & \\ & & & 1 & & 2 & & 1 & & & \\ & & 1 & & 3 & & 3 & & 1 & & \\ & 1 & & 4 & & \color{blue}{6} & & \color{blue}{4} & & 1 & \\ 1 & & 5 & & 10 & & \color{blue}{10} & & 5 & & 1 \end{matrix} \quad = \quad \begin{matrix} & & & & & \binom{0}{0} & & & & & \\ & & & & \binom{1}{0} & & \binom{1}{1} & & & & \\ & & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & & \\ & & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & & \\ & \binom{4}{0} & & \binom{4}{1} & & \color{blue}{\binom{4}{2}} & & \color{blue}{\binom{4}{3}} & & \binom{4}{4} & \\ \binom{5}{0} & & \binom{5}{1} & & \binom{5}{2} & & \color{blue}{\binom{5}{3}} & & \binom{5}{4} & & \binom{5}{5} \end{matrix}$$

Eine kombinatorische Erklärung dafür findest du auf dem AB – Pascalsches Dreieck II.

Wir können die Gleichheit auch rechnerisch überprüfen:

1) Rechne nach, dass $\binom{n}{0} = 1$ und $\binom{n}{n} = 1$ für alle $n \in \mathbb{N}$ gilt.

$$\binom{n}{0} = \frac{n!}{0! \cdot n!} = 1 \checkmark \qquad \binom{n}{n} = \frac{n!}{n! \cdot 0!} = 1 \checkmark$$

2) Rechne nach, dass $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ für alle $0 < k < n$ gilt.

$$\begin{aligned} \binom{n-1}{k-1} + \binom{n-1}{k} &= \frac{(n-1)!}{(k-1)! \cdot (n-k)!} + \frac{(n-1)!}{k! \cdot (n-1-k)!} \\ &= \frac{(n-1)! \cdot k}{k! \cdot (n-k)!} + \frac{(n-1)! \cdot (n-k)}{k! \cdot (n-k)!} \\ &= \frac{(n-1)! \cdot (k+n-k)}{k! \cdot (n-k)!} \\ &= \frac{n!}{k! \cdot (n-k)!} = \binom{n}{k} \checkmark \end{aligned}$$

$k \cdot (k-1)! = k!$

Binomischer Lehrsatz

Der **Binomische Lehrsatz**

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k} \qquad \text{mit } n = 0, 1, 2, 3, \ldots$$

hilft beim Ausmultiplizieren von Binomen der Form $(x+y)^n$. Zum Beispiel:

$$\begin{aligned}(x+y)^4 &= \binom{4}{0} \cdot x^0 \cdot y^4 + \binom{4}{1} \cdot x^1 \cdot y^3 + \binom{4}{2} \cdot x^2 \cdot y^2 + \binom{4}{3} \cdot x^3 \cdot y^1 + \binom{4}{4} \cdot x^4 \cdot y^0 \\ &= 1 \cdot y^4 + 4 \cdot x \cdot y^3 + 6 \cdot x^2 \cdot y^2 + 4 \cdot x^3 \cdot y + 1 \cdot x^4\end{aligned}$$

Eine kombinatorische Erklärung für den Binomischen Lehrsatz findest du auf dem AB – Pascalsches Dreieck II.

Jetzt beweisen wir den Binomischen Lehrsatz mit vollständiger Induktion nach n:

Überprüfe den Binomischen Lehrsatz für $n = 0$:

Als Nächstes überprüfen wir den Induktionsschritt $n \to n+1$.

Wir dürfen also verwenden, dass $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k}$ gilt.

Daraus müssen wir folgern, dass $(x+y)^{n+1} = \sum_{k=0}^{n+1} \binom{n+1}{k} \cdot x^k \cdot y^{n+1-k}$ gilt.

Wir ziehen zunächst die Summanden $k=0$ und $k=n+1$ aus der Summe, um $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$ verwenden zu können.

$$\begin{aligned}&\sum_{k=0}^{n+1} \binom{n+1}{k} \cdot x^k \cdot y^{n+1-k} = \\ &= y^{n+1} + x^{n+1} + \sum_{k=1}^{n} \left[\binom{n}{k-1} + \binom{n}{k} \right] \cdot x^k \cdot y^{n+1-k} = && \text{Ausmultiplizieren, Summe aufteilen} \\ &= \underbrace{y^{n+1}}_{k=0} + \underbrace{x^{n+1}}_{k=n} + \left[\sum_{k=0}^{n-1} \binom{n}{k} \cdot x^{k+1} \cdot y^{n-k} \right] + \left[\sum_{k=1}^{n} \binom{n}{k} \cdot x^k \cdot y^{n+1-k} \right] = && \sum_{k=1}^{n} f(k) = \sum_{k=0}^{n-1} f(k+1) \\ &= x \cdot \left[\sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k} \right] + y \cdot \left[\sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k} \right] = \\ &= x \cdot (x+y)^n + y \cdot (x+y)^n = (x+y) \cdot (x+y)^n = (x+y)^{n+1} \checkmark\end{aligned}$$

Zeilensumme im Pascalschen Dreieck

Berechne mit dem Binomischen Lehrsatz die n-te Zeilensumme im Pascalschen Dreieck:

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} =$$

Alternierende Zeilensumme im Pascalschen Dreieck

Berechne mit dem Binomischen Lehrsatz für $n \geq 1$:

$$\sum_{k=0}^{n} \binom{n}{k} \cdot (-1)^k =$$

Binomischer Lehrsatz

Der **Binomische Lehrsatz**

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k} \qquad \text{mit } n = 0, 1, 2, 3, \ldots$$

hilft beim Ausmultiplizieren von Binomen der Form $(x+y)^n$. Zum Beispiel:

$$\begin{aligned}(x+y)^4 &= \binom{4}{0} \cdot x^0 \cdot y^4 + \binom{4}{1} \cdot x^1 \cdot y^3 + \binom{4}{2} \cdot x^2 \cdot y^2 + \binom{4}{3} \cdot x^3 \cdot y^1 + \binom{4}{4} \cdot x^4 \cdot y^0 \\ &= 1 \cdot y^4 + 4 \cdot x \cdot y^3 + 6 \cdot x^2 \cdot y^2 + 4 \cdot x^3 \cdot y + 1 \cdot x^4\end{aligned}$$

Eine kombinatorische Erklärung für den Binomischen Lehrsatz findest du auf dem AB – Pascalsches Dreieck II.

Jetzt beweisen wir den Binomischen Lehrsatz mit vollständiger Induktion nach n:

Überprüfe den Binomischen Lehrsatz für $n = 0$:

$$(x+y)^0 = 1 \qquad \sum_{k=0}^{0} \binom{0}{k} \cdot x^k \cdot y^{0-k} = \binom{0}{0} \cdot x^0 \cdot y^0 = 1 \checkmark$$

Als Nächstes überprüfen wir den Induktionsschritt $n \to n+1$.

Wir dürfen also verwenden, dass $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k}$ gilt.

Daraus müssen wir folgern, dass $(x+y)^{n+1} = \sum_{k=0}^{n+1} \binom{n+1}{k} \cdot x^k \cdot y^{n+1-k}$ gilt.

Wir ziehen zunächst die Summanden $k = 0$ und $k = n+1$ aus der Summe, um $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$ verwenden zu können.

$$\begin{aligned}&\sum_{k=0}^{n+1} \binom{n+1}{k} \cdot x^k \cdot y^{n+1-k} = \\ &= y^{n+1} + x^{n+1} + \sum_{k=1}^{n} \left[\binom{n}{k-1} + \binom{n}{k} \right] \cdot x^k \cdot y^{n+1-k} = && \text{Ausmultiplizieren, Summe aufteilen} \\ &= \underbrace{y^{n+1}}_{k=0} + \underbrace{x^{n+1}}_{k=n} + \left[\sum_{k=0}^{n-1} \binom{n}{k} \cdot x^{k+1} \cdot y^{n-k} \right] + \left[\sum_{k=1}^{n} \binom{n}{k} \cdot x^k \cdot y^{n+1-k} \right] = && \sum_{k=1}^{n} f(k) = \sum_{k=0}^{n-1} f(k+1) \\ &= x \cdot \left[\sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k} \right] + y \cdot \left[\sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k} \right] = \\ &= x \cdot (x+y)^n + y \cdot (x+y)^n = (x+y) \cdot (x+y)^n = (x+y)^{n+1} \checkmark\end{aligned}$$

Zeilensumme im Pascalschen Dreieck

Berechne mit dem Binomischen Lehrsatz die n-te Zeilensumme im Pascalschen Dreieck:

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} = \sum_{k=0}^{n} \binom{n}{k} \cdot 1^k \cdot 1^{n-k} = (1+1)^n = 2^n$$

Alternierende Zeilensumme im Pascalschen Dreieck

Berechne mit dem Binomischen Lehrsatz für $n \geq 1$:

$$\sum_{k=0}^{n} \binom{n}{k} \cdot (-1)^k = \sum_{k=0}^{n} \binom{n}{k} \cdot (-1)^k \cdot 1^{n-k} = (-1+1)^n = 0^n = 0$$

2.7 Kleiner Satz von Fermat

Im Jahr 1640 schrieb der französische Mathematiker Pierre de Fermat einen Brief an Frenicle de Bessy. Darin formuliert er die folgende Behauptung:

„Without exception, every prime number measures [i.e. divides] *one of the powers* -1 *of any progression whatever, and the exponent of the said power is a sub-multiple of the given prime number* -1*. Also, after one has found the first power that satisfies the problem, all those of which the exponents are multiples of the exponent of the first will similarly satisfy the problem. This proposition is generally true for all series and for all prime numbers; I would send you a demonstration of it, if I did not fear going on too long.“* ([7])

Fermat illustrierte diese Behauptung anhand von Beispielen, gab aber keinen Beweis dafür. Publiziert wurde der erste Beweis für diesen „Kleinen Satz von Fermat“ im Jahr 1749 von Leonhard Euler [34]. 14 Jahre später veröffentlichte Euler eine Verallgemeinerung dieses Satzes („Satz von Euler“) in [12].

Es folgt das Arbeitsblatt – Kleiner Satz von Fermat und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Arbeitsblatt – Kongruenz und Restklassen

– Arbeitsblatt – Vollständige Induktion

– Arbeitsblatt – Binomischer Lehrsatz

Lernziele:

✓ Was sagt der **Kleine Satz von Fermat** aus?
Wie kann man ihn mit vollständiger Induktion beweisen?

✓ Was ist die **Eulersche Phi-Funktion** φ?
p und q sind verschiedene Primzahlen. Warum gilt dann $\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q)$?

✓ Was sagt der **Satz von Euler** aus?
Wie kann man ihn beweisen?

Kürzungsregel für Kongruenzen

Erinnere dich an die folgende Rechenregel für Kongruenzen:

$$\boldsymbol{a \cdot c \equiv b \cdot c \mod m \iff a \equiv b \mod \frac{m}{\mathrm{ggT}(c,m)}}$$

Eine Erklärung für die Rechenregel findest du am Arbeitsblatt – Kongruenz und Restklassen.

Wenn c und m teilerfremd sind, dann gilt also:

$$a \cdot c \equiv b \cdot c \mod m \iff a \equiv b \mod \_\_\_\_$$

Kleiner Satz von Fermat (1640)

Für jede ganze Zahl a und jede Primzahl p gilt der **Kleine Satz von Fermat**:

$$\boldsymbol{a^p \equiv a \mod p}$$

Zum Beispiel: $\underbrace{4^5}_{=1024} \equiv 4 \mod 5$

Wenn a kein Vielfaches von p ist, dann gilt $\mathrm{ggT}(a,p) = \_\_\_\_$ und damit:

$$\boldsymbol{a^{p-1} \equiv 1 \mod p}$$

Pascalsches Dreieck & Primzahlen

p ist eine Primzahl. Rechts siehst du die obersten 12 Reihen vom Pascalschen Dreieck:

In jeder „Primzahlreihe“ sind alle Zahlen im Inneren jeweils Vielfache dieser Primzahl.

Zum Beispiel: Die Zahlen

7, 21, 35, 35, 27, 7

sind alle durch 7 teilbar.

Das ist kein Zufall.

					1						
				1		1					
$p=2$			1		2		1				
$p=3$		1		3		3		1			
	1		4		6		4		1		
$p=5$		1		5		10		10		5		1
	1		6		15		20		15		6		1
$p=7$		1		7		21		35		35		21		7		1
	1		8		28		56		70		56		28		8		1
		1		9		36		84		126		126		84		36		9		1
	1		10		45		120		210		252		210		120		45		10		1
$p=11$		1		11		55		165		330		462		462		330		165		55		11		1

Die Zahlen in Reihe p sind die folgenden Binomialkoeffizienten:

$$\binom{p}{0}, \binom{p}{1}, \binom{p}{2}, \binom{p}{3}, \cdots, \binom{p}{p-1}, \binom{p}{p} \quad \text{mit} \quad \binom{p}{k} = \frac{\overbrace{p \cdot (p-1) \cdot (p-2) \cdot \ldots \cdot (p-k+1)}^{k \text{ Faktoren}}}{k!}$$

Erkläre, warum die Primzahl p den Binomialkoeffizienten $\binom{p}{k}$ für alle k mit $1 \leq k \leq p-1$ teilt.

Kürzungsregel für Kongruenzen

Erinnere dich an die folgende Rechenregel für Kongruenzen:

$$\boldsymbol{a \cdot c \equiv b \cdot c \mod m \iff a \equiv b \mod \frac{m}{\mathrm{ggT}(c,m)}}$$

Eine Erklärung für die Rechenregel findest du am Arbeitsblatt – Kongruenz und Restklassen.

Wenn c und m teilerfremd sind, dann gilt also:

$$a \cdot c \equiv b \cdot c \mod m \iff a \equiv b \mod m$$

Kleiner Satz von Fermat (1640)

Für jede ganze Zahl a und jede Primzahl p gilt der **Kleine Satz von Fermat**:

$$\boldsymbol{a^p \equiv a \mod p}$$

Zum Beispiel: $\underbrace{4^5}_{=1024} \equiv 4 \mod 5$

Wenn a kein Vielfaches von p ist, dann gilt $\mathrm{ggT}(a,p) = 1$ und damit:

$$\boldsymbol{a^{p-1} \equiv 1 \mod p}$$

Pascalsches Dreieck & Primzahlen

p ist eine Primzahl. Rechts siehst du die obersten 12 Reihen vom Pascalschen Dreieck:

In jeder „Primzahlreihe“ sind alle Zahlen im Inneren jeweils Vielfache dieser Primzahl.

Zum Beispiel: Die Zahlen

7, 21, 35, 35, 27, 7

sind alle durch 7 teilbar.

Das ist kein Zufall.

					1						
				1		1					
$p=2$			1		2		1				
$p=3$		1		3		3		1			
			1		4		6		4		1
$p=5$		1		5		10		10		5		1
			1		6		15		20		15		6		1
$p=7$		1		7		21		35		35		21		7		1
			1		8		28		56		70		56		28		8		1
				1		9		36		84		126		126		84		36		9		1
			1		10		45		120		210		252		210		120		45		10		1
$p=11$		1		11		55		165		330		462		462		330		165		55		11		1

Die Zahlen in Reihe p sind die folgenden Binomialkoeffizienten:

$$\binom{p}{0}, \binom{p}{1}, \binom{p}{2}, \binom{p}{3}, \cdots, \binom{p}{p-1}, \binom{p}{p} \quad \text{mit} \quad \binom{p}{k} = \frac{\overbrace{p \cdot (p-1) \cdot (p-2) \cdot \ldots \cdot (p-k+1)}^{k \text{ Faktoren}}}{k!}$$

Erkläre, warum die Primzahl p den Binomialkoeffizienten $\binom{p}{k}$ für alle k mit $1 \leq k \leq p-1$ teilt.

Im Zähler von $\binom{p}{k}$ kommt der Faktor p vor, weil $k \geq 1$.

Der Nenner $k!$ und p sind teilerfremd, weil p eine Primzahl ist und $k \leq p-1$.

Das Ergebnis $\binom{p}{k}$ der Division ist eine ganze Zahl, deren Primfaktorzerlegung also p enthalten muss.

Kleiner Satz von Fermat – Beweis ($a \geq 0$)

Wir beweisen jetzt $\boldsymbol{a^p \equiv a \mod p}$ für alle Primzahlen p mit vollständiger Induktion nach $a \geq 0$.
Dabei verwenden wir die beiden folgenden Aussagen:

- Binomischer Lehrsatz: $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k}$
- $\binom{p}{k} \equiv 0 \mod p$ für alle Primzahlen p und $1 \leq k \leq p-1$.

1) Überprüfe den **Induktionsanfang** für $a = 0$:

2) Überprüfe den **Induktionsschritt** $a \to a+1$:
Du darfst also verwenden, dass $a^p \equiv a \mod p$ gilt.
Daraus musst du folgern, dass auch $(a+1)^p \equiv a+1 \mod p$ gilt.

□

Kleiner Satz von Fermat – Beweis ($a < 0$)

1) Für die gerade Primzahl $p = 2$ können wir $\boldsymbol{a^2 \equiv a \mod 2}$ für alle ganzen Zahlen a direkt beweisen:

- Wenn a gerade ist, dann ist a^2 eine gerade / ungerade Zahl.
- Wenn a ungerade ist, dann ist a^2 eine gerade / ungerade Zahl.

Streiche jeweils durch.

Die ganzen Zahlen a und a^2 liegen also in der gleichen Restklasse modulo 2.

2) Für jede ungerade Primzahl p und *negative* ganze Zahl a haben wir $(-a)^p \equiv (-a) \mod p$ gezeigt.
Folgere daraus, dass auch $a^p \equiv a \mod p$ gilt.

Also gilt $a^p \equiv a \mod p$ auch für alle negativen ganzen Zahlen a.

Leonhard Euler (1707-1783) bewies, dass eine Verallgemeinerung von $a^{p-1} \equiv 1 \mod p$ nicht nur für Primzahlen p stimmt, sondern für beliebige natürliche Zahlen.
Als Nächstes sehen wir uns diesen **Satz von Euler** und einen eleganten Beweis dafür an.
Der Satz von Euler spielt eine wichtige Rolle in der modernen Verschlüsselung.

Mehr dazu erfährst du am Arbeitsblatt – RSA-Verfahren.

Kleiner Satz von Fermat – Beweis ($a \geq 0$)

Wir beweisen jetzt $\boldsymbol{a^p \equiv a \mod p}$ für alle Primzahlen p mit vollständiger Induktion nach $a \geq 0$.
Dabei verwenden wir die beiden folgenden Aussagen:

- Binomischer Lehrsatz: $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot y^{n-k}$
- $\binom{p}{k} \equiv 0 \mod p$ für alle Primzahlen p und $1 \leq k \leq p-1$.

1) Überprüfe den **Induktionsanfang** für $a = 0$:

$0^p \equiv 0 \mod p$ ✓

2) Überprüfe den **Induktionsschritt** $a \to a+1$:
Du darfst also verwenden, dass $a^p \equiv a \mod p$ gilt.
Daraus musst du folgern, dass auch $(a+1)^p \equiv a+1 \mod p$ gilt.

$$(a+1)^p = \sum_{k=0}^{p} \binom{p}{k} \cdot a^k \cdot 1^{p-k} = \underbrace{\binom{p}{p} \cdot a^p \cdot 1^0}_{=a^p \ (k=p)} + \underbrace{\binom{p}{0} \cdot a^0 \cdot 1^p}_{=1 \ (k=0)} + \sum_{k=1}^{p-1} \underbrace{\binom{p}{k} \cdot a^k \cdot 1^{p-k}}_{\equiv 0 \bmod p} \equiv a^p + 1 \mod p$$

$$\Longrightarrow (a+1)^p \equiv a^p + 1 \equiv a + 1 \mod p \ ✓$$

□

Kleiner Satz von Fermat – Beweis ($a < 0$)

1) Für die gerade Primzahl $p = 2$ können wir $\boldsymbol{a^2 \equiv a \mod 2}$ für alle ganzen Zahlen a direkt beweisen:

- Wenn a gerade ist, dann ist a^2 eine gerade / ~~ungerade~~ Zahl.
- Wenn a ungerade ist, dann ist a^2 eine ~~gerade~~ / ungerade Zahl.

Streiche jeweils durch.

Die ganzen Zahlen a und a^2 liegen also in der gleichen Restklasse modulo 2.

2) Für jede ungerade Primzahl p und *negative* ganze Zahl a haben wir $(-a)^p \equiv (-a) \mod p$ gezeigt.
Folgere daraus, dass auch $a^p \equiv a \mod p$ gilt.

$(-a)^p = -a^p$, weil p ungerade ist.

$$(-a)^p \equiv (-a) \mod p \Longrightarrow -a^p \equiv (-a) \mod p \overset{\cdot(-1)}{\Longrightarrow} a^p \equiv a \mod p$$

Also gilt $a^p \equiv a \mod p$ auch für alle negativen ganzen Zahlen a.

Leonhard Euler (1707-1783) bewies, dass eine Verallgemeinerung von $a^{p-1} \equiv 1 \mod p$ nicht nur für Primzahlen p stimmt, sondern für beliebige natürliche Zahlen.
Als Nächstes sehen wir uns diesen **Satz von Euler** und einen eleganten Beweis dafür an.
Der Satz von Euler spielt eine wichtige Rolle in der modernen Verschlüsselung.

Mehr dazu erfährst du am Arbeitsblatt – RSA-Verfahren.

Eulersche Phi-Funktion

Die **Eulersche Phi-Funktion** φ ist für jede natürliche Zahl $n \geq 1$ definiert.
$\varphi(n)$ ist die Anzahl natürlicher Zahlen a, für die $1 \leq a \leq n$ und $\text{ggT}(a, n) = 1$ gilt. Kurz:

$$\boldsymbol{\varphi(n) = |\{a \in \mathbb{N} \mid 1 \leq a \leq n \wedge \text{ggT}(a, n) = 1\}|}$$

Eulersche Phi-Funktion

Ermittle die folgenden Funktionswerte der Eulerschen Phi-Funktion.

1) $\varphi(1) =$ _____ **4)** $\varphi(4) =$ _____ **7)** $\varphi(7) =$ _____ **10)** $\varphi(10) =$ _____

2) $\varphi(2) =$ _____ **5)** $\varphi(5) =$ _____ **8)** $\varphi(8) =$ _____ **11)** $\varphi(11) =$ _____

3) $\varphi(3) =$ _____ **6)** $\varphi(6) =$ _____ **9)** $\varphi(9) =$ _____ **12)** $\varphi(12) =$ _____

Eulersche Phi-Funktion

p und q sind verschiedene Primzahlen.

1) Für welche natürlichen Zahlen a mit $1 \leq a \leq p$ gilt $\text{ggT}(a, p) = 1$?

_____ $\Longrightarrow$ $\varphi(p) =$ _____

2) Für welche natürlichen Zahlen a mit $1 \leq a \leq p^k$ gilt *nicht* $\text{ggT}(a, p^k) = 1$?

_____ $\Longrightarrow$ $\varphi(p^k) =$ _____

3) Für welche natürlichen Zahlen a mit $1 \leq a \leq p \cdot q$ gilt *nicht* $\text{ggT}(a, p \cdot q) = 1$?

$\Longrightarrow$ $\varphi(p \cdot q) =$ _____ $= \varphi(p) \cdot \varphi(q)$

Multiplikativität der Eulerschen Phi-Funktion

Tatsächlich ist die Eulersche Phi-Funktion für alle *teilerfremden* Zahlen m und n multiplikativ.

Aus $\text{ggT}(m, n) = 1$ folgt also $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$.

Primfaktorzerlegung von $n \rightsquigarrow \varphi(n)$

Zerlege n in Primfaktoren und berechne $\varphi(n)$.

a) $n = 42$ **b)** $n = 360$

Eulersche Phi-Funktion

Die **Eulersche Phi-Funktion** φ ist für jede natürliche Zahl $n \geq 1$ definiert.
$\varphi(n)$ ist die Anzahl natürlicher Zahlen a, für die $1 \leq a \leq n$ und $\text{ggT}(a,n) = 1$ gilt. Kurz:

$$\boldsymbol{\varphi(n) = |\{a \in \mathbb{N} \mid 1 \leq a \leq n \wedge \text{ggT}(a,n) = 1\}|}$$

Eulersche Phi-Funktion

Ermittle die folgenden Funktionswerte der Eulerschen Phi-Funktion.

1) $\varphi(1) = 1$ **4)** $\varphi(4) = 2$ **7)** $\varphi(7) = 6$ **10)** $\varphi(10) = 4$

2) $\varphi(2) = 1$ **5)** $\varphi(5) = 4$ **8)** $\varphi(8) = 4$ **11)** $\varphi(11) = 10$

3) $\varphi(3) = 2$ **6)** $\varphi(6) = 2$ **9)** $\varphi(9) = 6$ **12)** $\varphi(12) = 4$

Eulersche Phi-Funktion

p und q sind verschiedene Primzahlen.

1) Für welche natürlichen Zahlen a mit $1 \leq a \leq p$ gilt $\text{ggT}(a,p) = 1$?

$1, 2, 3, \ldots, p-1 \implies \varphi(p) = p - 1$

2) Für welche natürlichen Zahlen a mit $1 \leq a \leq p^k$ gilt *nicht* $\text{ggT}(a,p^k) = 1$?

$1 \cdot p, 2 \cdot p, 3 \cdot p \ldots, p^{k-1} \cdot p \implies \varphi(p^k) = p^k - p^{k-1} = p^{k-1} \cdot (p-1)$

3) Für welche natürlichen Zahlen a mit $1 \leq a \leq p \cdot q$ gilt *nicht* $\text{ggT}(a, p \cdot q) = 1$?

Vielfache von p, die $\leq p \cdot q$ sind:

$\underbrace{1 \cdot p, 2 \cdot p, 3 \cdot p, \ldots, (q-1) \cdot p, q \cdot p}_{q \text{ Zahlen}}$

Vielfache von q, die $\leq p \cdot q$ sind:

$\underbrace{1 \cdot q, 2 \cdot q, 3 \cdot q, \ldots, (p-1) \cdot q, p \cdot q}_{p \text{ Zahlen}}$

$$\implies \varphi(p \cdot q) = p \cdot q - (p + q - 1) = (p-1) \cdot (q-1) = \varphi(p) \cdot \varphi(q)$$

Multiplikativität der Eulerschen Phi-Funktion

Tatsächlich ist die Eulersche Phi-Funktion für alle *teilerfremden* Zahlen m und n multiplikativ.

Aus $\text{ggT}(m,n) = 1$ folgt also $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$.

Primfaktorzerlegung von $n \rightsquigarrow \varphi(n)$

Zerlege n in Primfaktoren und berechne $\varphi(n)$.

a) $n = 42$

$42 = 2 \cdot 3 \cdot 7$

$\implies \varphi(42) = \varphi(2) \cdot \varphi(3) \cdot \varphi(7)$
$= 1 \cdot 2 \cdot 6 = 12$

b) $n = 360$

$360 = 2^3 \cdot 3^2 \cdot 5$

$\implies \varphi(360) = \varphi(2^3) \cdot \varphi(3^2) \cdot \varphi(5)$
$= 4 \cdot 6 \cdot 4 = 96$

Satz von Euler (1736)

Wenn $\text{ggT}(a,n) = 1$ ist, dann gilt der **Satz von Euler**:

$$\boldsymbol{a^{\varphi(n)} \equiv 1 \mod n}$$

Wenn n eine Primzahl ist, dann ist das der Satz von Fermat.

mod 100

Ermittle mit dem Satz von Euler die letzten beiden Dezimalstellen von 23^{42}.

Kongruenz & ggT

Für den Beweis vom Satz von Euler verwenden wir zwei Eigenschaften:

1) $\boldsymbol{a \equiv b \mod n \implies \text{ggT}(a,n) = \text{ggT}(b,n)}$

Alle Zahlen, die in der gleichen Restklasse modulo n sind, haben mit n den gleichen größten gemeinsamen Teiler.

Begründe die Aussage: Hinweis: $\text{ggT}(a + k \cdot n, n) = \text{ggT}(a,n)$

2) Wenn $\text{ggT}(a,n) = 1$, dann gilt: $\boldsymbol{x \cdot a \equiv y \cdot a \mod n \iff x \equiv y \mod n}$

Satz von Euler – Beweis

Damit beweisen wir jetzt $\boldsymbol{a^{\varphi(n)} \equiv 1 \mod n}$ für alle Zahlen a mit $\text{ggT}(a,n) = 1$.

Unter den Zahlen $\{1, 2, 3, \ldots, n\}$ gibt es $\varphi(n)$ verschiedene Zahlen, die zu n teilerfremd sind.
Wir geben diesen $\varphi(n)$ verschiedenen Zahlen jeweils einen Namen: $\{a_1, a_2, a_3, \ldots, a_{\varphi(n)}\}$

Aus $\text{ggT}(a_i, n) = 1$ und $\text{ggT}(a,n) = 1$ folgt $\text{ggT}(a_i \cdot a, n) =$ _____ .

Wegen **1)** muss $a_i \cdot a$ in der gleichen Restklasse modulo n wie eine der Zahlen $\{a_1, a_2, \ldots, a_{\varphi(n)}\}$ sein.
Wegen **2)** sind die Zahlen $a_1 \cdot a$, $a_2 \cdot a$, … , $a_{\varphi(n)} \cdot a$ alle in verschiedenen Restklassen modulo n, denn:

$$a_i \cdot a \equiv a_j \cdot a \mod n \overset{\textbf{2)}}{\implies} a_i \equiv a_j \mod n \implies a_i = a_j$$

„Indirekter Beweis“

Die Multiplikation mit a vertauscht also nur die Reihenfolge der $\varphi(n)$ verschiedenen Restklassen.

$$\implies a_1 \cdot a_2 \cdot \ldots \cdot a_{\varphi(n)} \equiv (a_1 \cdot a) \cdot (a_2 \cdot a) \cdot \ldots \cdot (a_{\varphi(n)} \cdot a) \mod n$$
$$\implies a_1 \cdot a_2 \cdot \ldots \cdot a_{\varphi(n)} \equiv a_1 \cdot a_2 \cdot \ldots \cdot a_{\varphi(n)} \cdot a^{\varphi(n)} \mod n$$
$$\overset{\textbf{2)}}{\implies} 1 \equiv a^{\varphi(n)} \mod n \qquad \square$$

Satz von Euler (1736)

Wenn $\text{ggT}(a,n) = 1$ ist, dann gilt der **Satz von Euler**:

$$\boldsymbol{a^{\varphi(n)} \equiv 1 \mod n}$$

Wenn n eine Primzahl ist, dann ist das der Satz von Fermat.

mod 100

Ermittle mit dem Satz von Euler die letzten beiden Dezimalstellen von 23^{42}.

$$n = 100 = 2^2 \cdot 5^2 \implies \varphi(100) = \varphi(2^2) \cdot \varphi(5^2) = 2 \cdot 20 = 40$$

Der Satz von Euler garantiert also $a^{40} \equiv 1 \mod 100$ für alle Zahlen a mit $\text{ggT}(a,100) = 1$.

$$\text{ggT}(23,100) = 1 \implies 23^{42} = \underbrace{23^2}_{=529} \cdot \underbrace{23^{40}}_{\equiv 1 \mod 100} \equiv 529 \mod 100$$

Die beiden letzten Stellen von 23^{42} sind also 29.

Kongruenz & ggT

Für den Beweis vom Satz von Euler verwenden wir zwei Eigenschaften:

1) $\boldsymbol{a \equiv b \mod n \implies \text{ggT}(a,n) = \text{ggT}(b,n)}$

Alle Zahlen, die in der gleichen Restklasse modulo n sind, haben mit n den gleichen größten gemeinsamen Teiler.

Begründe die Aussage: Hinweis: $\text{ggT}(a + k \cdot n, n) = \text{ggT}(a,n)$

$$a \equiv b \mod n \implies b = a + k \cdot n \text{ mit einer ganzen Zahl } k$$
$$\implies \text{ggT}(b,n) = \text{ggT}(a + k \cdot n, n) = \text{ggT}(a,n)$$

2) Wenn $\text{ggT}(a,n) = 1$, dann gilt: $\boldsymbol{x \cdot a \equiv y \cdot a \mod n \iff x \equiv y \mod n}$

Satz von Euler – Beweis

Damit beweisen wir jetzt $\boldsymbol{a^{\varphi(n)} \equiv 1 \mod n}$ für alle Zahlen a mit $\text{ggT}(a,n) = 1$.

Unter den Zahlen $\{1,2,3,\ldots,n\}$ gibt es $\varphi(n)$ verschiedene Zahlen, die zu n teilerfremd sind.
Wir geben diesen $\varphi(n)$ verschiedenen Zahlen jeweils einen Namen: $\{a_1, a_2, a_3, \ldots, a_{\varphi(n)}\}$

Aus $\text{ggT}(a_i,n) = 1$ und $\text{ggT}(a,n) = 1$ folgt $\text{ggT}(a_i \cdot a, n) = 1$.

Wegen **1)** muss $a_i \cdot a$ in der gleichen Restklasse modulo n wie eine der Zahlen $\{a_1, a_2, \ldots, a_{\varphi(n)}\}$ sein.
Wegen **2)** sind die Zahlen $a_1 \cdot a,\ a_2 \cdot a,\ \ldots,\ a_{\varphi(n)} \cdot a$ alle in verschiedenen Restklassen modulo n, denn:

$$a_i \cdot a \equiv a_j \cdot a \mod n \overset{2)}{\implies} a_i \equiv a_j \mod n \implies a_i = a_j$$

„Indirekter Beweis“

Die Multiplikation mit a vertauscht also nur die Reihenfolge der $\varphi(n)$ verschiedenen Restklassen.

$$\implies a_1 \cdot a_2 \cdot \ldots \cdot a_{\varphi(n)} \equiv (a_1 \cdot a) \cdot (a_2 \cdot a) \cdot \ldots \cdot (a_{\varphi(n)} \cdot a) \mod n$$
$$\implies a_1 \cdot a_2 \cdot \ldots \cdot a_{\varphi(n)} \equiv a_1 \cdot a_2 \cdot \ldots \cdot a_{\varphi(n)} \cdot a^{\varphi(n)} \mod n$$
$$\overset{2)}{\implies} 1 \equiv a^{\varphi(n)} \mod n \qquad \square$$

2.8 RSA-Verfahren

Im Jahr 1978 publizierten Ron **R**ivest, Adi **S**hamir und Leonard **A**dleman mit dem RSA-Verfahren eine praktische Umsetzung für das theoretische Modell von asymmmetrischer Verschlüsselung [2]. Bei der Berechnung des privaten Schlüssels kommt der Euklidische Algorithmus zum Einsatz. Grundlage für einen Beweis, dass das Verfahren tatsächlich funktioniert, ist der Satz von Euler.

Auf dem folgenden Arbeitsblatt wird zunächst der Unterschied zwischen symmetrischer und asymmetrischer Verschlüsselung behandelt. Anschließend wird auf Grundlage der der ersten vier Arbeitsblätter erklärt, *wie* das RSA-Verfahren funktioniert. Gemeinsam mit dem Arbeitsblatt zum Kleinen Satz von Fermat kann abschließend auch verstanden werden, *warum* das RSA-Verfahren funktioniert.

In [14] wird ein spielerischer Zugang zum Thema Kryptographie beschrieben. Eine mögliche Implementierung des RSA-Verfahrens mit der Programmiersprache Python befindet sich in [22]. Zusätzliche Informationen zum RSA-Verfahren und weiteren Verschlüsselungsverfahren sind im „Handbook of Discrete and Combinatorial Mathematics“ [28] aufgelistet.

Es folgt das Arbeitsblatt – RSA-Verfahren und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Arbeitsblatt – Euklidischer Algorithmus

– Arbeitsblatt – Kongruenz und Restklassen

– Arbeitsblatt – Kleiner Satz von Fermat

Lernziele:

✓ Was ist das Grundprinzip von **symmetrischer Verschlüsselung**?

✓ Was ist das Grundprinzip von **asymmetrischer Verschlüsselung**?

✓ Wie funktioniert das **RSA-Verfahren**?
Warum werden dafür große Primzahlen benötigt?

✓ Wie kann man die Korrektheit des RSA-Verfahrens beweisen?

Symmetrische Verschlüsselung

Bob möchte eine verschlüsselte Nachricht von Alice empfangen.
Dazu machen sie sich ein geheimes Passwort aus („Privater Schlüssel“).

1) Alice verschlüsselt ihre Nachricht mit diesem Schlüssel.

2) Alice schickt die verschlüsselte Nachricht an Bob.

3) Bob entschlüsselt die empfangene Nachricht mit diesem Schlüssel.

Das ist das Grundprinzip von **symmetrischer Verschlüsselung**. Sie bringt Probleme mit sich:

⚠ Jede Person, die den Schlüssel kennt, kann die Nachricht auch entschlüsseln.

Alice und Bob müssen also *beide* sorgsam mit diesem privaten Schlüssel umgehen.

⚠ Wie vereinbaren Alice und Bob den privaten Schlüssel? Alice wohnt in Australien. Bob wohnt in Brasilien.

Caesar-Verschlüsselung

Ein historisches Beispiel für symmetrische Verschlüsselung ist die Caesar-Verschlüsselung.
Dabei wird jeder Buchstabe in der Nachricht nach folgendem Muster ersetzt:

Klartext:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Geheimtext:	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

G A L L I E N → J D O O L H Q

Der private Schlüssel bei diesem Verfahren ist eine natürliche Zahl von 1 bis 26.
Diese Zahl gibt an, um wie viele Stellen das Alphabet verschoben wird.

Gaius Julius Caesar soll – wie im Beispiel oben – für militärische Nachrichten den Schlüssel 3 verwendet haben.

Zum Entschlüsseln verschiebt man das Alphabet um den privaten Schlüssel in die andere Richtung:

Geheimtext:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Klartext:	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

J D O O L H Q → G A L L I E N

Welche Schwachstelle hat diese Verschlüsselung, auch wenn Alice und Bob den Schlüssel geheim halten?

Monoalphabetische Verschlüsselung

Bei monoalphabetischen Verschlüsselungen werden die Buchstaben im Alphabet beliebig vertauscht.
Ein privater Schlüssel zum Verschlüsseln und Entschlüsseln von Nachrichten kann dann so aussehen:

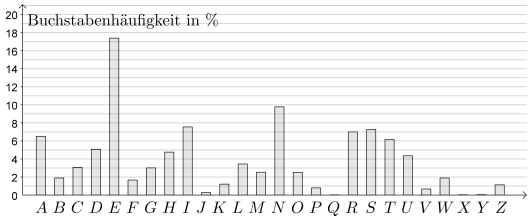
Klartext:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Geheimtext:	L	Q	B	D	Z	R	T	E	J	V	C	I	M	P	H	X	O	K	W	F	Y	A	U	N	S	G

G A L L I E N → T L I I J Z P

Wie viele solche Schlüssel ermöglichen die 26 Buchstaben im deutschen Alphabet?

Mehr dazu findest du am Arbeitsblatt – Kombinatorik.

Trotz der großen Schlüsselanzahl hat jede monoalphabetische Verschlüsselung eine Schwachstelle.
Rechts siehst du die prozentuellen Häufigkeiten der Buchstaben in typischen deutschsprachigen Texten:
Du fängst eine lange verschlüsselte Nachricht ab.
Wie würdest du versuchen, den Code zu knacken?



Unter diesem Link kannst du deine Fertigkeiten beim Codeknacken auf die Probe stellen.

Symmetrische Verschlüsselung

Bob möchte eine verschlüsselte Nachricht von Alice empfangen.
Dazu machen sie sich ein geheimes Passwort aus („Privater Schlüssel“).

1) Alice verschlüsselt ihre Nachricht mit diesem Schlüssel.

2) Alice schickt die verschlüsselte Nachricht an Bob.

3) Bob entschlüsselt die empfangene Nachricht mit diesem Schlüssel.

Das ist das Grundprinzip von **symmetrischer Verschlüsselung**. Sie bringt Probleme mit sich:

⚠ Jede Person, die den Schlüssel kennt, kann die Nachricht auch entschlüsseln.

Alice und Bob müssen also *beide* sorgsam mit diesem privaten Schlüssel umgehen.

⚠ Wie vereinbaren Alice und Bob den privaten Schlüssel? Alice wohnt in Australien. Bob wohnt in Brasilien.

Caesar-Verschlüsselung

Ein historisches Beispiel für symmetrische Verschlüsselung ist die Caesar-Verschlüsselung.
Dabei wird jeder Buchstabe in der Nachricht nach folgendem Muster ersetzt:

Klartext:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Geheimtext:	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

G	A	L	L	I	E	N
J	D	O	O	L	H	Q

Der private Schlüssel bei diesem Verfahren ist eine natürliche Zahl von 1 bis 26.
Diese Zahl gibt an, um wie viele Stellen das Alphabet verschoben wird.

Gaius Julius Caesar soll – wie im Beispiel oben – für militärische Nachrichten den Schlüssel 3 verwendet haben.

Zum Entschlüsseln verschiebt man das Alphabet um den privaten Schlüssel in die andere Richtung:

Geheimtext:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Klartext:	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

J	D	O	O	L	H	Q
G	A	L	L	I	E	N

Welche Schwachstelle hat diese Verschlüsselung, auch wenn Alice und Bob den Schlüssel geheim halten?
Es gibt nur 26 verschiedene Schlüssel.
Man kann also den richtigen Schlüssel durch Probieren relativ schnell finden. („Brute Force“)

Monoalphabetische Verschlüsselung

Bei monoalphabetischen Verschlüsselungen werden die Buchstaben im Alphabet beliebig vertauscht.
Ein privater Schlüssel zum Verschlüsseln und Entschlüsseln von Nachrichten kann dann so aussehen:

Klartext:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Geheimtext:	L	Q	B	D	Z	R	T	E	J	V	C	I	M	P	H	X	O	K	W	F	Y	A	U	N	S	G

G	A	L	L	I	E	N
T	L	I	I	J	Z	P

Wie viele solche Schlüssel ermöglichen die 26 Buchstaben im deutschen Alphabet?

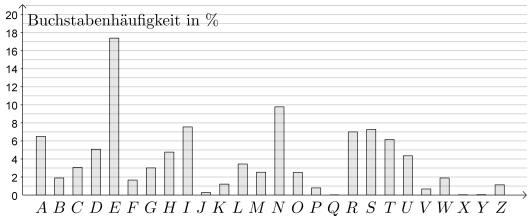
$26! = 26 \cdot 25 \cdot 24 \cdot \ldots \cdot 2 \cdot 1 \approx 4 \cdot 10^{26}$

Mehr dazu findest du am Arbeitsblatt – Kombinatorik.

Trotz der großen Schlüsselanzahl hat jede monoalphabetische Verschlüsselung eine Schwachstelle.
Rechts siehst du die prozentuellen Häufigkeiten der Buchstaben in typischen deutschsprachigen Texten:

Du fängst eine lange verschlüsselte Nachricht ab.
Wie würdest du versuchen, den Code zu knacken?

Man kann z.B. die Häufigkeiten der Buchstaben im Geheimtext mit den Buchstabenhäufigkeiten vergleichen, um Rückschlüsse zu ziehen.



Unter diesem Link kannst du deine Fertigkeiten beim Codeknacken auf die Probe stellen.

Asymmetrische Verschlüsselung

Bob möchte eine verschlüsselte Nachricht von Alice empfangen.
Dazu erstellt Bob vorher zwei verschiedene Schlüssel:

- **Öffentlicher Schlüssel:** Dieser Schlüssel ist *nicht* geheim. „Public Key“
  Jede Person kann ihn verwenden, um an Bob verschlüsselte Nachrichten zu senden.
  Verschlüsselte Nachrichten können mit dem öffentlichen Schlüssel *nicht* entschlüsselt werden.
- **Privater Schlüssel:** Diesen Schlüssel muss *nur* Bob geheim halten. „Private Key“
  Bob kann mit *seinem* privaten Schlüssel jede Nachricht entschlüsseln,
  die mit *seinem* öffentlichen Schlüssel verschlüsselt wurde.

Das ist das Grundprinzip von **asymmetrischer Verschlüsselung**. „Public-Key-Verschlüsselung“
Asymmetrische Verschlüsselung hat Vorteile gegenüber symmetrischer Verschlüsselung:

1) Nur der Empfänger Bob muss *seinen* privaten Schlüssel geheim halten.

2) Es muss *kein* geheimer Schlüssel ausgetauscht werden.

In der Praxis sind symmetrische Verfahren schneller als asymmetrische Verfahren. Deshalb wird hybride Verschlüsselung verwendet:
Zuerst tauschen Alice und Bob einen privaten Schlüssel mit *asymmetrischer* Verschlüsselung aus.
Danach verwendet Alice diesen privaten Schlüssel, um Nachrichten an Bob mit *symmetrischer* Verschlüsselung zu senden.

RSA-Verfahren – Schlüsselpaar erzeugen

Das **RSA-Verfahren** ist ein asymmetrisches Verschlüsselungsverfahren.

Der Algorithmus wurde im Jahr 1977 von **R**ivest, **S**hamir und **A**dleman veröffentlicht.

Einen öffentlichen Schlüssel und den zugehörigen privaten Schlüssel kannst du so berechnen:

1) Wähle (geheim) zwei verschiedene Primzahlen p und q.

2) Berechne $n = p \cdot q$.

3) Berechne $\varphi(n) = (p-1) \cdot (q-1)$. Mehr zur Eulerschen φ-Funktion findest du am AB – Kleiner Satz von Fermat.

4) Wähle eine Zahl $e > 1$, die zu $\varphi(n)$ teilerfremd ist, also $\text{ggT}(e, \varphi(n)) = 1$.
Berechne mit dem Euklidischen Algorithmus eine ganze Zahl $d > 1$ mit $e \cdot d \equiv 1 \mod \varphi(n)$.

Die beiden Zahlen $\boldsymbol{n}$ und $\boldsymbol{e}$ sind der **öffentliche Schlüssel**. Die Zahl $\boldsymbol{d}$ ist der **private Schlüssel**.

RSA-Verfahren – Schlüsselpaar erzeugen

Erzeuge den öffentlichen und den privaten RSA-Schlüssel mit $p = 7$, $q = 13$ und $e = 11$.

$n =$ _____ $\varphi(n) =$ _____

Euklidischer Algorithmus:

$72 = 11 \cdot \square + \square$

$11 = \square \cdot \square + \square$

$\square = \square \cdot \square + \square$

$\square = \square \cdot \square + \square$

Öffentlicher Schlüssel: _____ Privater Schlüssel: _____

Asymmetrische Verschlüsselung

Bob möchte eine verschlüsselte Nachricht von Alice empfangen.
Dazu erstellt Bob vorher zwei verschiedene Schlüssel:

- **Öffentlicher Schlüssel:** Dieser Schlüssel ist *nicht* geheim. „Public Key“
  Jede Person kann ihn verwenden, um an Bob verschlüsselte Nachrichten zu senden.
  Verschlüsselte Nachrichten können mit dem öffentlichen Schlüssel *nicht* entschlüsselt werden.
- **Privater Schlüssel:** Diesen Schlüssel muss *nur* Bob geheim halten. „Private Key“
  Bob kann mit *seinem* privaten Schlüssel jede Nachricht entschlüsseln,
  die mit *seinem* öffentlichen Schlüssel verschlüsselt wurde.

Das ist das Grundprinzip von **asymmetrischer Verschlüsselung**. „Public-Key-Verschlüsselung“
Asymmetrische Verschlüsselung hat Vorteile gegenüber symmetrischer Verschlüsselung:

1) Nur der Empfänger Bob muss *seinen* privaten Schlüssel geheim halten.

2) Es muss *kein* geheimer Schlüssel ausgetauscht werden.

In der Praxis sind symmetrische Verfahren schneller als asymmetrische Verfahren. Deshalb wird hybride Verschlüsselung verwendet:
Zuerst tauschen Alice und Bob einen privaten Schlüssel mit *asymmetrischer* Verschlüsselung aus.
Danach verwendet Alice diesen privaten Schlüssel, um Nachrichten an Bob mit *symmetrischer* Verschlüsselung zu senden.

RSA-Verfahren – Schlüsselpaar erzeugen

Das **RSA-Verfahren** ist ein asymmetrisches Verschlüsselungsverfahren.

Der Algorithmus wurde im Jahr 1977 von **R**ivest, **S**hamir und **A**dleman veröffentlicht.

Einen öffentlichen Schlüssel und den zugehörigen privaten Schlüssel kannst du so berechnen:

1) Wähle (geheim) zwei verschiedene Primzahlen p und q.

2) Berechne $n = p \cdot q$.

3) Berechne $\varphi(n) = (p-1) \cdot (q-1)$. Mehr zur Eulerschen φ-Funktion findest du am AB – Kleiner Satz von Fermat.

4) Wähle eine Zahl $e > 1$, die zu $\varphi(n)$ teilerfremd ist, also $\text{ggT}(e, \varphi(n)) = 1$.
Berechne mit dem Euklidischen Algorithmus eine ganze Zahl $d > 1$ mit $e \cdot d \equiv 1 \mod \varphi(n)$.

Die beiden Zahlen $\boldsymbol{n}$ und $\boldsymbol{e}$ sind der **öffentliche Schlüssel**. Die Zahl $\boldsymbol{d}$ ist der **private Schlüssel**.

RSA-Verfahren – Schlüsselpaar erzeugen

Erzeuge den öffentlichen und den privaten RSA-Schlüssel mit $p = 7$, $q = 13$ und $e = 11$.

$n = 7 \cdot 13 = 91$ $\quad \varphi(n) = 6 \cdot 12 = 72$

$$\begin{aligned} 1 &= 6 - 5 \cdot 1 = 6 - (11 - 6 \cdot 1) \cdot 1 = \\ &= 6 \cdot 2 - 11 \cdot 1 = (72 - 11 \cdot 6) \cdot 2 - 11 \cdot 1 = \\ &= 72 \cdot 2 - 11 \cdot 13 \end{aligned}$$

$\implies 1 \equiv -11 \cdot 13 \mod 72 \implies d = -13 \equiv 59 \mod 72$

Euklidischer Algorithmus:

$$\begin{aligned} 72 &= 11 \cdot 6 + 6 \\ 11 &= 6 \cdot 1 + 5 \\ 6 &= 5 \cdot 1 + 1 \\ 5 &= 1 \cdot 5 + 0 \end{aligned}$$

Öffentlicher Schlüssel: $n = 91$, $e = 11$ Privater Schlüssel: $d = 59$

RSA-Verfahren – Nachrichten senden

Alice möchte an Bob eine verschlüsselte Nachricht senden.

1) Alice erhält den öffentlichen Schlüssel (n, e) von Bob.

2) Alice kann damit jede natürliche Zahl m mit $1 \leq m < n$ an Bob übertragen.
Alice berechnet $m^e \mod n$ und sendet das Ergebnis c an Bob. $1 \leq c < n$

Jede Nachricht ist im Computer als Binärzahl – zum Beispiel 100101110101 – gespeichert. Text versenden ist wie Zahlen versenden. Wenn die Nachricht zu lange ist, kann sie vorher in Teile zerlegt werden. In der Praxis werden jeder Nachricht vor der Verschlüsselung noch zufällige Teile hinzugefügt, damit *gleiche unverschlüsselte* Nachrichten *verschiedene verschlüsselte* Nachrichten ergeben.

RSA-Verfahren – Nachrichten senden

Bob hat den öffentlichen Schlüssel $n = 91$, $e = 11$. Verschlüssle damit die Nachricht $m = 42$.
Wir berechnen $42^{11} \mod 91$ schrittweise: Dazu zerlegen wir den Exponenten in Zweier-Potenzen: $11 = 2^3 + 2^1 + 2^0$

1) $42^2 = 19 \cdot 91 + 35 \equiv \square \mod 91$

2) $42^4 = (42^2)^2 \equiv 35^2 = 13 \cdot 91 + 42 \equiv \square \mod 91$

3) $42^8 = (42^4)^2 \equiv 42^2 \equiv \square \mod 91$

$$\Longrightarrow 42^{11} = 42^8 \cdot 42^2 \cdot 42^1 \equiv 35 \cdot 35 \cdot 42 = 565 \cdot 91 + 35 \equiv 35 \mod 91 \Longrightarrow c = \square$$

RSA-Verfahren – Nachrichten empfangen

Bob möchte die verschlüsselte Nachricht c entschlüsseln.

1) Bob berechnet $c^d \mod n$ mit seinem privaten Schlüssel d.

2) Das Ergebnis ist die unverschlüsselte Nachricht m. Der Beweis dafür ist auf der nächsten Seite.

RSA-Verfahren – Nachrichten empfangen

Bob hat den öffentlichen Schlüssel $n = 91$, $e = 11$ und den privaten Schlüssel $d = 59$.
Entschlüssle damit die Nachricht $c = 35$.

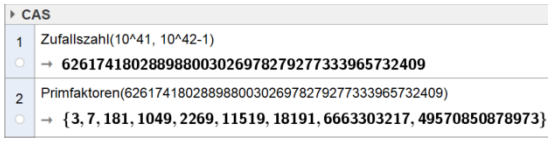
Was sind *große* Primzahlen?

Die Sicherheit des RSA-Verfahrens beruht darauf, dass wir mit unseren aktuellen technischen Mitteln aus einer großen Zahl $n = p \cdot q$ die beiden Primfaktoren p und q *nicht effizient* berechnen können.

Sonst könnte jede Person aus dem öffentlichen Schlüssel auch den privaten Schlüssel berechnen und die Nachrichten entschlüsseln.

42-stellige Zahlen zerlegt GeoGebra im Bruchteil einer Sekunde in ihre Primfaktoren:

626 174 180 288 988 003 026 978 279 277 333 965 732 409 =
3·7·181·1049·2269·11 519·18 191·6 663 303 217·49 570 850 878 973



In der Praxis werden deshalb beim RSA-Verfahren Primzahlen mit rund 300 Stellen verwendet.

RSA-Verfahren – Nachrichten senden

Alice möchte an Bob eine verschlüsselte Nachricht senden.

1) Alice erhält den öffentlichen Schlüssel (n, e) von Bob.

2) Alice kann damit jede natürliche Zahl m mit $1 \leq m < n$ an Bob übertragen.
Alice berechnet $m^e \mod n$ und sendet das Ergebnis c an Bob. $1 \leq c < n$

Jede Nachricht ist im Computer als Binärzahl – zum Beispiel 100101110101 – gespeichert. Text versenden ist wie Zahlen versenden. Wenn die Nachricht zu lange ist, kann sie vorher in Teile zerlegt werden. In der Praxis werden jeder Nachricht vor der Verschlüsselung noch zufällige Teile hinzugefügt, damit *gleiche unverschlüsselte* Nachrichten *verschiedene verschlüsselte* Nachrichten ergeben.

RSA-Verfahren – Nachrichten senden

Bob hat den öffentlichen Schlüssel $n = 91$, $e = 11$. Verschlüssle damit die Nachricht $m = 42$.
Wir berechnen $42^{11} \mod 91$ schrittweise: Dazu zerlegen wir den Exponenten in Zweier-Potenzen: $11 = 2^3 + 2^1 + 2^0$

1) $42^2 = 19 \cdot 91 + 35 \equiv 35 \mod 91$

2) $42^4 = (42^2)^2 \equiv 35^2 = 13 \cdot 91 + 42 \equiv 42 \mod 91$

3) $42^8 = (42^4)^2 \equiv 42^2 \equiv 35 \mod 91$

$$\Longrightarrow 42^{11} = 42^8 \cdot 42^2 \cdot 42^1 \equiv 35 \cdot 35 \cdot 42 = 565 \cdot 91 + 35 \equiv 35 \mod 91 \Longrightarrow c = 35$$

RSA-Verfahren – Nachrichten empfangen

Bob möchte die verschlüsselte Nachricht c entschlüsseln.

1) Bob berechnet $c^d \mod n$ mit seinem privaten Schlüssel d.

2) Das Ergebnis ist die unverschlüsselte Nachricht m. Der Beweis dafür ist auf der nächsten Seite.

RSA-Verfahren – Nachrichten empfangen

Bob hat den öffentlichen Schlüssel $n = 91$, $e = 11$ und den privaten Schlüssel $d = 59$.
Entschlüssle damit die Nachricht $c = 35$.

1) $35^2 = 91 \cdot 13 + 42 \equiv 42 \mod 91$

2) $35^4 = (35^2)^2 \equiv 42^2 = 91 \cdot 19 + 35 \equiv 35 \mod 91$

3) $35^8 = (35^4)^2 \equiv 35^2 \equiv 42 \mod 91$

4) $35^{16} = (35^8)^2 \equiv 42^2 \equiv 35 \mod 91$

5) $35^{32} = (35^{16})^2 \equiv 35^2 \equiv 42 \mod 91$

$$\Longrightarrow 35^{59} = 35^{32} \cdot 35^{16} \cdot 35^8 \cdot 35^2 \cdot 35^1 \equiv 42^3 \cdot 35^2 \equiv 42^4 \equiv 42 \mod 91 \Longrightarrow m = 42$$

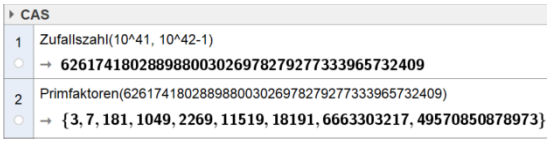
Was sind *große* Primzahlen?

Die Sicherheit des RSA-Verfahrens beruht darauf, dass wir mit unseren aktuellen technischen Mitteln aus einer großen Zahl $n = p \cdot q$ die beiden Primfaktoren p und q *nicht effizient* berechnen können.

Sonst könnte jede Person aus dem öffentlichen Schlüssel auch den privaten Schlüssel berechnen und die Nachrichten entschlüsseln.

42-stellige Zahlen zerlegt GeoGebra im Bruchteil einer Sekunde in ihre Primfaktoren:

626 174 180 288 988 003 026 978 279 277 333 965 732 409 =
3·7·181·1049·2269·11 519·18 191·6 663 303 217·49 570 850 878 973



In der Praxis werden deshalb beim RSA-Verfahren Primzahlen mit rund 300 Stellen verwendet.

Primfaktorzerlegung

p und q sind *verschiedene* Primzahlen.
Erkläre, warum dann die folgende Äquivalenz für alle ganzen Zahlen a und b gilt:

$$a \equiv b \mod p \cdot q \iff a \equiv b \mod p \quad \text{und} \quad a \equiv b \mod q$$

RSA-Verfahren – Beweis

Warum funktioniert das **RSA-Verfahren**?
Wir müssen für alle Zahlen m mit $1 \leq m < n$ die folgende Identität zeigen:

$$\boldsymbol{(m^e)^d \equiv m \mod n} \tag{1}$$

Der private Schlüssel d wurde so gewählt, dass $e \cdot d \equiv 1 \mod \varphi(n)$ gilt.
Es gibt also eine ganze Zahl k mit $e \cdot d = 1 + k \cdot \varphi(n)$. Statt (1) können wir somit auch zeigen:

$$m \cdot m^{k \cdot \varphi(n)} \equiv m \mod n \tag{2}$$

Fall 1: m und n sind teilerfremde Zahlen.
Erkläre, warum dann (2) aus dem Satz von Euler ($m^{\varphi(n)} \equiv 1 \mod n$) folgt:

Fall 2: m und $n = p \cdot q$ sind *nicht* teilerfremd.
Da p und q verschiedene Primzahlen sind, können wir statt (2) auch zeigen, dass

$$m \cdot m^{k \cdot \varphi(n)} \equiv m \mod p \quad \text{und} \quad m \cdot m^{k \cdot \varphi(n)} \equiv m \mod q \quad \text{gilt.} \tag{3}$$

Die *kleinste* positive natürliche Zahl s, für die $\text{ggT}(s, p) > 1$ und $\text{ggT}(s, q) > 1$ gilt, ist $s =$ _____ .

Da $m < p \cdot q$ und $\text{ggT}(m, p \cdot q) > 1$ gilt, bleiben also nur 2 Möglichkeiten:

a) $\text{ggT}(m, q) = 1$ und $p \mid m$ oder **b)** $\text{ggT}(m, p) = 1$ und $q \mid m$

Wir zeigen jetzt (3) mit Hilfe von **a)**. Mit **b)** funktioniert es genauso. p und q vertauschen nur ihre Rollen.

$$p \mid m \implies \underbrace{m \cdot m^{k \cdot \varphi(n)}}_{\equiv 0} \equiv \underbrace{m}_{\equiv 0} \mod p \checkmark$$

Auf dem Arbeitsblatt – Kleiner Satz von Fermat haben wir erklärt, warum $\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q)$ gilt.
Wegen $\text{ggT}(m, q) = 1$ können wir den Satz von Euler verwenden: $m^{\varphi(q)} \equiv 1 \mod q$
Erkläre damit, warum $m \cdot m^{k \cdot \varphi(n)} \equiv m \mod q$ gilt:

RSA-Signatur

Das RSA-Verfahren kann wegen $(m^d)^e = (m^e)^d \equiv m \mod n$ auch umgekehrt verwendet werden:

1) Bob verschlüsselt seine Nachricht mit seinem privaten Schlüssel d. Er *unterschreibt* damit die Nachricht.

2) Alice prüft mit dem öffentlichen Schlüssel e von Bob, ob die Nachricht tatsächlich von ihm kommt.
Wenn ja, dann ist $(m^d)^e \equiv m \mod n$. Wenn nein, dann sollte die entschlüsselte Nachricht $(m^d)^e$ keinen Sinn ergeben.

Primfaktorzerlegung

p und q sind *verschiedene* Primzahlen.
Erkläre, warum dann die folgende Äquivalenz für alle ganzen Zahlen a und b gilt:

$$a \equiv b \mod p \cdot q \iff a \equiv b \mod p \quad \text{und} \quad a \equiv b \mod q$$

$a \equiv b \mod p \cdot q \iff p \cdot q \mid a - b$
Links und rechts steht: „Die Primfaktorzerlegung von $a - b$ enthält beide Primzahlen p und q."

RSA-Verfahren – Beweis

Warum funktioniert das **RSA-Verfahren**?
Wir müssen für alle Zahlen m mit $1 \leq m < n$ die folgende Identität zeigen:

$$\boldsymbol{(m^e)^d \equiv m \mod n} \tag{1}$$

Der private Schlüssel d wurde so gewählt, dass $e \cdot d \equiv 1 \mod \varphi(n)$ gilt.
Es gibt also eine ganze Zahl k mit $e \cdot d = 1 + k \cdot \varphi(n)$. Statt (1) können wir somit auch zeigen:

$$m \cdot m^{k \cdot \varphi(n)} \equiv m \mod n \tag{2}$$

Fall 1: m und n sind teilerfremde Zahlen.
Erkläre, warum dann (2) aus dem Satz von Euler ($m^{\varphi(n)} \equiv 1 \mod n$) folgt:

$$m \cdot m^{k \cdot \varphi(n)} = m \cdot \left(m^{\varphi(n)}\right)^k \equiv m \cdot 1^k = m \mod n \checkmark$$

Fall 2: m und $n = p \cdot q$ sind *nicht* teilerfremd.
Da p und q verschiedene Primzahlen sind, können wir statt (2) auch zeigen, dass

$$m \cdot m^{k \cdot \varphi(n)} \equiv m \mod p \quad \text{und} \quad m \cdot m^{k \cdot \varphi(n)} \equiv m \mod q \quad \text{gilt.} \tag{3}$$

Die *kleinste* positive natürliche Zahl s, für die $\text{ggT}(s, p) > 1$ und $\text{ggT}(s, q) > 1$ gilt, ist $s = p \cdot q$.

Da $m < p \cdot q$ und $\text{ggT}(m, p \cdot q) > 1$ gilt, bleiben also nur 2 Möglichkeiten:

a) $\text{ggT}(m, q) = 1$ und $p \mid m$ oder **b)** $\text{ggT}(m, p) = 1$ und $q \mid m$

Wir zeigen jetzt (3) mit Hilfe von **a)**. Mit **b)** funktioniert es genauso. p und q vertauschen nur ihre Rollen.

$$p \mid m \implies \underbrace{m \cdot m^{k \cdot \varphi(n)}}_{\equiv 0} \equiv \underbrace{m}_{\equiv 0} \mod p \checkmark$$

Auf dem Arbeitsblatt – Kleiner Satz von Fermat haben wir erklärt, warum $\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q)$ gilt.
Wegen $\text{ggT}(m, q) = 1$ können wir den Satz von Euler verwenden: $m^{\varphi(q)} \equiv 1 \mod q$
Erkläre damit, warum $m \cdot m^{k \cdot \varphi(n)} \equiv m \mod q$ gilt:

$$m \cdot m^{k \cdot \varphi(n)} = m \cdot \left(m^{\varphi(q)}\right)^{k \cdot \varphi(p)} \equiv m \cdot 1^{k \cdot \varphi(p)} = m \mod q \checkmark$$

RSA-Signatur

Das RSA-Verfahren kann wegen $(m^d)^e = (m^e)^d \equiv m \mod n$ auch umgekehrt verwendet werden:

1) Bob verschlüsselt seine Nachricht mit seinem privaten Schlüssel d. Er *unterschreibt* damit die Nachricht.

2) Alice prüft mit dem öffentlichen Schlüssel e von Bob, ob die Nachricht tatsächlich von ihm kommt.
Wenn ja, dann ist $(m^d)^e \equiv m \mod n$. Wenn nein, dann sollte die entschlüsselte Nachricht $(m^d)^e$ keinen Sinn ergeben.

3 Algorithmen

In der 9. Schulstufe sollen SchülerInnen im Informatik-Unterricht *„Algorithmen erklären, entwerfen, darstellen und in einer Programmiersprache implementieren“* können sowie *„Grundprinzipien von Automaten, Algorithmen, Datenstrukturen und Programmen erklären“* können (AHS, [29]) bzw. *„Ablaufalgorithmen entwerfen und Berechnungsschritte systematisch angeben“* können (BHS, HTL, [31]).
Für die 10. Schulstufe sieht der Lehrplan von höheren technischen Lehranstalten weiters vor ([31]):

- *„Kommentare, Konstanten und Variablen in einer Programmiersprache darstellen und Befehlsstrukturen einer Programmiersprache anwenden“*
- *„Datenstrukturen und Objekte aus einfachen Datentypen zusammensetzen und einfache Befehlsstrukturen erstellen“*
- *„Programme mit Verzweigungen, Schleifen und Datentypen; Dateizugriff; Anwendungen auf einfache Algorithmen“*
- *„die wichtigsten Datentypen unterscheiden und ihre Einsatzbereiche anführen;“*
- *„Datenstrukturen und Objekte aus einfachen Datentypen zusammensetzen und komplexe Befehlsstrukturen erstellen“*
- *„Anwendungen auf komplexe Algorithmen“*

Algorithmen und deren Analyse bilden eine Schnittstelle zwischen der Mathematik und der Informatik. Die folgenden vier Arbeitsblätter sollen Möglichkeiten aufzeigen, wie diese Verzahnung auch im Schulunterricht stattfinden kann. Mögliche Einsatzgebiete für die Arbeitsblätter sind zum Beispiel der Informatik-Unterricht, ein Wahlpflichtfach Mathematik oder fächerübergreifende Projekte.

Am ersten Arbeitsblatt werden in einer (mathematischen) Einführung in Algorithmen die elementaren Strukturen beim Programmieren behandelt. Auf den darauffolgenden Arbeitsblättern zu Sortieralgorithmen, dem Kruskal-Algorithmus sowie dem Dijkstra-Algorithmus sind Themen aufbereitet, die sich meines Erachtens durch eine geringe Einstiegshürde bei gleichzeitig hoher Praxisrelevanz auszeichnen. Diese drei Arbeitsblätter können unabhängig voneinander in beliebiger Reihenfolge behandelt werden.

3.1 Einführung in Algorithmen

In [3] befindet sich eine umfangreiche Einführung zu Algorithmen und deren Analyse: Nach einer geschichtlichen Einbettung werden klassische Probleme und zugehörige Algorithmen vorgestellt, deren Effizienz im Anschluss mit der $\mathcal{O}$-Notation analysiert wird. Schließlich wird eine Einführung in die Komplexitätstheorie, Turing-Maschinen sowie das P-NP-Problem gegeben. Letzteres gehört zu jenen ungelösten Millennium-Problemen, für dessen Lösung im Jahr 2000 vom Clay Mathematics Institute 1 Million US-$ ausgeschrieben wurde [16].

Die Ansprüche als Lehrperson in Informatik müssen natürlich an die vorhandene Unterrichtszeit angepasst werden. Die folgenden Arbeitsblättern sollen die Einführung zu Algorithmen unterstützen, deren Analyse am Beispiel von Sortieralgorithmen greifbar machen und erste Einblicke in Algorithmen im Bereich der diskreten Mathematik geben.

Am ersten Arbeitsblatt werden die grundlegenden Begriffe und Ablaufstrukturen zur Programmierung eingeführt und anhand von Beispielen geübt. Um die Einsatzmöglichkeiten des Arbeitsblatts nicht einzuschränken, sind die Algorithmen in Pseudocode angegeben. Parallel zur Bearbeitung des Arbeitsblatts bietet sich damit auch die Implementierung in einer konkreten Programmiersprache wie zum Beispiel Snap ([20]) oder Python ([13]) an.

Es folgt das Arbeitsblatt – Einführung in Algorithmen und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

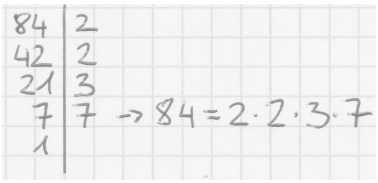
Lernziele:

- ✓ Was ist ein **Algorithmus**?
- ✓ Was sind **Variablen** und **Wertzuweisungen**?
- ✓ Was sind **if-Abfragen**?
- ✓ Was sind **for-Schleifen**?
- ✓ Was sind **repeat until-Schleifen**?
- ✓ Was sind **while-Schleifen**?

Was ist ein Algorithmus?

Als Kind hast du ein Verfahren gelernt, um jede natürliche Zahl als Produkt von Primfaktoren zu schreiben. Zum Beispiel: $84 = 2 \cdot 2 \cdot 3 \cdot 7$

1) Dividiere so oft wie möglich durch 2 ohne Rest.
2) Dividiere so oft wie möglich durch 3 ohne Rest.
3) Dividiere so oft wie möglich durch 5 ohne Rest.
4) Setze mit den weiteren Primzahlen fort, bis das Ergebnis 1 ist.

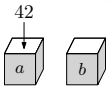


*„Ein **Algorithmus** ist [...] eine wohldefinierte Rechenvorschrift, die eine Größe oder eine Menge von Größen als Eingabe verwendet und eine Größe oder eine Menge von Größen als Ausgabe erzeugt. Somit ist ein Algorithmus eine Folge von Rechenschritten, die die Eingabe in die Ausgabe umwandeln.“*

Quelle: Algorithmen - Eine Einführung, Cormen, Thomas H. / Leiserson, Charles E. / Rivest, Ronald / Stein, Clifford

Variablen & Wertzuweisungen

Mit **Variablen** können wir Werte abspeichern.
Dafür verwenden wir die folgende Schreibweise:



1: $a \leftarrow 42$ — In Zeile 1 erhält die Variable a den Wert 42.
2: $b \leftarrow a$ — In Zeile 2 erhält die Variable b den aktuellen Wert von a.
3: $a \leftarrow a + 2$ — In Zeile 3 erhält die Variable a den aktuellen Wert von a plus 2.
4: $b \leftarrow \frac{a+b}{2}$ — In Zeile 4 erhält die Variable b das arithmetische Mittel von a und b.

Zeile	1	2	3	4
a	42	42	44	44
b	–	42	42	43

In der Tabelle links siehst du welchen Wert die Variablen a und b in jeder Zeile nach Durchführung der Wertzuweisung haben.

Variablen & Wertzuweisungen

Trage in die Tabelle jene Werte ein, die die Variablen jeweils nach Durchführung der Zeile haben.

1: $x \leftarrow 23$
2: $x \leftarrow x - 5$
3: $y \leftarrow \frac{x}{2}$
4: $x \leftarrow y - x$
5: $y \leftarrow -x - 3$

Zeile	1	2	3	4	5
x					
y					

if-Abfrage

In der Praxis hängen unsere Entscheidungen von der aktuellen Situation ab.

Falls es kalt ist, ziehe ich mich warm an. *Falls* es regnet, nehme ich einen Regenschirm.

Bei Algorithmen heißen solche bedingten Entscheidungen **if**-Abfragen. Zum Beispiel:

1: $T \leftarrow 8$
2: $K \leftarrow 5$
3: **if** $T \leq 10$ **then**
4: $\quad K \leftarrow 10$
5: $\quad T \leftarrow T + 4$
6: **end if**
7: $T \leftarrow 20$

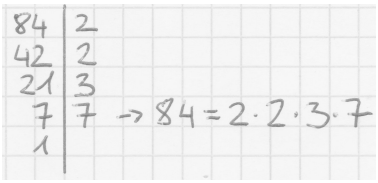
In Zeile 3 wird überprüft, ob der Wert von T höchstens 10 ist.
Falls diese Bedingung erfüllt ist, werden die eingerückten Befehle bis zum **end if** durchgeführt.
Falls sie *nicht* erfüllt ist, werden die eingerückten Befehle bis zum **end if** *nicht* durchgeführt.

Zeile	1	2	3	4	5	6	7
T	8	8	**8**	8	12	12	20
K	–	5	5	10	10	10	10

Was ist ein Algorithmus?

Als Kind hast du ein Verfahren gelernt, um jede natürliche Zahl als Produkt von Primfaktoren zu schreiben. Zum Beispiel: $84 = 2 \cdot 2 \cdot 3 \cdot 7$

1) Dividiere so oft wie möglich durch 2 ohne Rest.
2) Dividiere so oft wie möglich durch 3 ohne Rest.
3) Dividiere so oft wie möglich durch 5 ohne Rest.
4) Setze mit den weiteren Primzahlen fort, bis das Ergebnis 1 ist.

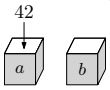


*„Ein **Algorithmus** ist [...] eine wohldefinierte Rechenvorschrift, die eine Größe oder eine Menge von Größen als Eingabe verwendet und eine Größe oder eine Menge von Größen als Ausgabe erzeugt. Somit ist ein Algorithmus eine Folge von Rechenschritten, die die Eingabe in die Ausgabe umwandeln.“*

Quelle: Algorithmen - Eine Einführung, Cormen, Thomas H. / Leiserson, Charles E. / Rivest, Ronald / Stein, Clifford

Variablen & Wertzuweisungen

Mit **Variablen** können wir Werte abspeichern.
Dafür verwenden wir die folgende Schreibweise:



1: $a \leftarrow 42$ — In Zeile 1 erhält die Variable a den Wert 42.
2: $b \leftarrow a$ — In Zeile 2 erhält die Variable b den aktuellen Wert von a.
3: $a \leftarrow a + 2$ — In Zeile 3 erhält die Variable a den aktuellen Wert von a plus 2.
4: $b \leftarrow \frac{a+b}{2}$ — In Zeile 4 erhält die Variable b das arithmetische Mittel von a und b.

Zeile	1	2	3	4
a	42	42	44	44
b	–	42	42	43

In der Tabelle links siehst du welchen Wert die Variablen a und b in jeder Zeile nach Durchführung der Wertzuweisung haben.

Variablen & Wertzuweisungen

Trage in die Tabelle jene Werte ein, die die Variablen jeweils nach Durchführung der Zeile haben.

1: $x \leftarrow 23$
2: $x \leftarrow x - 5$
3: $y \leftarrow \frac{x}{2}$
4: $x \leftarrow y - x$
5: $y \leftarrow -x - 3$

Zeile	1	2	3	4	5
x	23	18	18	−9	−9
y	–	–	9	9	6

if-Abfrage

In der Praxis hängen unsere Entscheidungen von der aktuellen Situation ab.

Falls es kalt ist, ziehe ich mich warm an. *Falls* es regnet, nehme ich einen Regenschirm.

Bei Algorithmen heißen solche bedingten Entscheidungen **if**-Abfragen. Zum Beispiel:

1: $T \leftarrow 8$
2: $K \leftarrow 5$
3: **if** $T \leq 10$ **then**
4: $\quad K \leftarrow 10$
5: $\quad T \leftarrow T + 4$
6: **end if**
7: $T \leftarrow 20$

In Zeile 3 wird überprüft, ob der Wert von T höchstens 10 ist.
Falls diese Bedingung erfüllt ist, werden die eingerückten Befehle bis zum **end if** durchgeführt.
Falls sie *nicht* erfüllt ist, werden die eingerückten Befehle bis zum **end if** *nicht* durchgeführt.

Zeile	1	2	3	4	5	6	7
T	8	8	**8**	8	12	12	20
K	–	5	5	10	10	10	10

if-Abfrage

Trage in die Tabelle jene Werte ein, die die Variablen jeweils nach Durchführung der Zeile haben.

```
1: x ← −5
2: y ← 3
3: if x · y < 0 then
4:     x ← x · y
5:     y ← x
6: end if
7: if x ≠ y then
8:     x ← 42
9: end if
```

Zeile	1	2	3	4	5	6	7	8	9
x									
y									

if – else if – else

Bei einer Schularbeit hängt die Note von den erreichten Punkten ab.

Ein Programm soll die Punkteanzahl P in die entsprechende Ziffernote N (1, 2, 3, 4 oder 5) übersetzen.

Punkte	Note
30 oder mehr	Sehr gut
[26; 30[	Gut
[21; 26[	Befriedigend
[16; 21[	Genügend
weniger als 16	Nicht genügend

Wir können die Übersetzung mit fünf if-Abfragen lösen:

```
1: if P ≥ 30 then
2:     N ← 1
3: end if
4: if 26 ≤ P < 30 then
5:     N ← 2
6: end if
7: if 21 ≤ P < 26 then
8:     N ← 3
9: end if
10: if 16 ≤ P < 21 then
11:     N ← 4
12: end if
13: if P < 16 then
14:     N ← 5
15: end if
```

Es reicht aber auch eine einzige **if – else if – else**-Abfrage:

```
1: if P ≥ 30 then
2:     N ← 1
3: else if P ≥ 26 then
4:     N ← 2
5: else if P ≥ 21 then
6:     N ← 3
7: else if P ≥ 16 then
8:     N ← 4
9: else
10:     N ← 5
11: end if
```

Bei einer **if – else if – else**-Abfrage werden die Bedingungen von oben nach unten überprüft, bis zum *ersten Mal* eine Bedingung erfüllt ist.

Dann werden *ausschließlich* die Befehle von dieser *ersten* erfüllten Bedingung durchgeführt. Alle weiteren Befehle bis zum **end if** werden übersprungen.

Zeile 3 im Programm wird also nur dann erreicht, falls $P \geq 30$ *nicht* gilt.
Dann muss stattdessen $P < 30$ gelten.
Deshalb reicht es in Zeile 3 die Bedingung $P \geq 26$ abzufragen.

Falls *keine einzige* Bedingung erfüllt ist, werden die Befehle von **else** durchgeführt.

if – else if – else

1) Der Wert, den die Variable s nach der folgenden if – else if – else-Abfrage hat, hängt von den Werten der beiden Variablen a und b ab. Vervollständige die Tabelle:

```
1: if a · b > 0 then
2:     s ← 1
3: else if a · b < 0 then
4:     s ← −1
5: else
6:     s ← 0
7: end if
```

a	3	5	−4	2	0	−3
b	4	−2	−1	0	0	1
s						

2) Wie kannst du den Wert von s unmittelbar erkennen, ohne $a \cdot b$ zu berechnen?

if-Abfrage

Trage in die Tabelle jene Werte ein, die die Variablen jeweils nach Durchführung der Zeile haben.

1: $x \leftarrow -5$
2: $y \leftarrow 3$
3: **if** $x \cdot y < 0$ **then**
4: $\quad x \leftarrow x \cdot y$
5: $\quad y \leftarrow x$
6: **end if**
7: **if** $x \neq y$ **then**
8: $\quad x \leftarrow 42$
9: **end if**

Zeile	1	2	3	4	5	6	7	8	9
x	-5	-5	-5	-15	-15	-15	-15	-15	-15
y	–	3	3	3	-15	-15	-15	-15	-15

if – else if – else

Bei einer Schularbeit hängt die Note von den erreichten Punkten ab.

Ein Programm soll die Punkteanzahl P in die entsprechende Ziffernote N (1, 2, 3, 4 oder 5) übersetzen.

Punkte	Note
30 oder mehr	Sehr gut
$[26; 30[$	Gut
$[21; 26[$	Befriedigend
$[16; 21[$	Genügend
weniger als 16	Nicht genügend

Wir können die Übersetzung mit fünf if-Abfragen lösen:

1: **if** $P \geq 30$ **then**
2: $\quad N \leftarrow 1$
3: **end if**
4: **if** $26 \leq P < 30$ **then**
5: $\quad N \leftarrow 2$
6: **end if**
7: **if** $21 \leq P < 26$ **then**
8: $\quad N \leftarrow 3$
9: **end if**
10: **if** $16 \leq P < 21$ **then**
11: $\quad N \leftarrow 4$
12: **end if**
13: **if** $P < 16$ **then**
14: $\quad N \leftarrow 5$
15: **end if**

Es reicht aber auch eine einzige **if – else if – else**-Abfrage:

1: **if** $P \geq 30$ **then**
2: $\quad N \leftarrow 1$
3: **else if** $P \geq 26$ **then**
4: $\quad N \leftarrow 2$
5: **else if** $P \geq 21$ **then**
6: $\quad N \leftarrow 3$
7: **else if** $P \geq 16$ **then**
8: $\quad N \leftarrow 4$
9: **else**
10: $\quad N \leftarrow 5$
11: **end if**

Bei einer **if – else if – else**-Abfrage werden die Bedingungen von oben nach unten überprüft, bis zum *ersten Mal* eine Bedingung erfüllt ist.

Dann werden *ausschließlich* die Befehle von dieser *ersten* erfüllten Bedingung durchgeführt. Alle weiteren Befehle bis zum **end if** werden übersprungen.

Zeile 3 im Programm wird also nur dann erreicht, falls $P \geq 30$ *nicht* gilt.
Dann muss stattdessen $P < 30$ gelten.
Deshalb reicht es in Zeile 3 die Bedingung $P \geq 26$ abzufragen.

Falls *keine einzige* Bedingung erfüllt ist, werden die Befehle von **else** durchgeführt.

if – else if – else

1) Der Wert, den die Variable s nach der folgenden if – else if – else-Abfrage hat, hängt von den Werten der beiden Variablen a und b ab. Vervollständige die Tabelle:

1: **if** $a \cdot b > 0$ **then**
2: $\quad s \leftarrow 1$
3: **else if** $a \cdot b < 0$ **then**
4: $\quad s \leftarrow -1$
5: **else**
6: $\quad s \leftarrow 0$
7: **end if**

a	3	5	-4	2	0	-3
b	4	-2	-1	0	0	1
s	1	-1	1	0	0	-1

2) Wie kannst du den Wert von s unmittelbar erkennen, ohne $a \cdot b$ zu berechnen?

$s = 1 \iff a$ und b haben das gleiche Vorzeichen.
$s = -1 \iff a$ und b haben verschiedene Vorzeichen.
$s = 0 \iff a = 0$ und/oder $b = 0$.

for - Schleife

```
1: a ← 1
2: b ← 1
3: a ← a + b
4: b ← a + b
5: a ← a + b
6: b ← a + b
7: a ← a + b
8: b ← a + b
```

Trage in die Tabelle jene Werte ein, die die Variablen jeweils nach Durchführung der Zeile haben.

Zeile	1	2	3	4	5	6	7	8
a								
b								

Bei diesem Programm werden die Befehle $a \leftarrow a + b$, $b \leftarrow a + b$ dreimal hintereinander ausgeführt.

Mit einer **for**-Schleife können wir das gleiche Programm kürzer anschreiben:

```
1: a ← 1
2: b ← 1
3: for i ← 1 to 3 do
4:     a ← a + b
5:     b ← a + b
6: end for
```

In Zeile 3 wird die Zählvariable i definiert.
i erhält den Wert 1. Dann werden die Befehle bis zum **end for** werden ausgeführt.
i erhält den Wert 2. Dann werden die Befehle bis zum **end for** werden ausgeführt.
i erhält den Wert 3. Dann werden die Befehle bis zum **end for** werden ausgeführt.

Zeile	1	2	3	4	5	6	3	4	5	6	3	4	5	6
a	1	1	1	2	2	2	2	5	5	5	5	13	13	13
b	–	1	1	1	3	3	3	3	8	8	8	8	21	21
i	–	–	1	1	1	1	2	2	2	2	3	3	3	3

for - Schleife

Welchen Wert hat die Variable a schließlich?

a)
```
1: a ← 4
2: for i ← 1 to 5 do
3:     a ← a + 2
4: end for
```

b)
```
1: a ← 25
2: for i ← 3 to 9 do
3:     a ← a − 3
4: end for
```

c)
```
1: a ← 3
2: for i ← 1 to 10 do
3:     a ← a · 2
4: end for
```

for - Schleife

Fülle jene Zahl in die Lücke, damit die Variable a schließlich den Wert 42 hat.

a)
```
1: a ← ☐
2: for i ← 1 to 7 do
3:     a ← a − 3
4: end for
```

b)
```
1: a ← 6
2: for i ← 1 to ☐ do
3:     a ← a + 4
4: end for
```

c)
```
1: a ← 72
2: for i ← 4 to 13 do
3:     a ← a − ☐
4: end for
```

for - Schleife

Das folgende Programm verwendet in einer **for**-Schleife den Wert der Zählvariable i.

```
1: s ← 0
2: for i ← 1 to 9 do
3:     s ← s + i
4: end for
```

Wir dürfen i in der Schleife aber *nicht* neue Werte zuweisen.

Welchen Wert hat die Variable s nach Durchführung der **for**-Schleife?

for - Schleife

1: $a \leftarrow 1$
2: $b \leftarrow 1$
3: $a \leftarrow a + b$
4: $b \leftarrow a + b$
5: $a \leftarrow a + b$
6: $b \leftarrow a + b$
7: $a \leftarrow a + b$
8: $b \leftarrow a + b$

Trage in die Tabelle jene Werte ein, die die Variablen jeweils nach Durchführung der Zeile haben.

Zeile	1	2	3	4	5	6	7	8
a	1	1	2	2	5	5	13	13
b	–	1	1	3	3	8	8	21

Bei diesem Programm werden die Befehle $a \leftarrow a + b$, $b \leftarrow a + b$ dreimal hintereinander ausgeführt.

Mit einer **for**-Schleife können wir das gleiche Programm kürzer anschreiben:

1: $a \leftarrow 1$
2: $b \leftarrow 1$
3: **for** $i \leftarrow 1$ **to** 3 **do**
4: $\quad a \leftarrow a + b$
5: $\quad b \leftarrow a + b$
6: **end for**

In Zeile 3 wird die Zählvariable i definiert.
i erhält den Wert 1. Dann werden die Befehle bis zum **end for** werden ausgeführt.
i erhält den Wert 2. Dann werden die Befehle bis zum **end for** werden ausgeführt.
i erhält den Wert 3. Dann werden die Befehle bis zum **end for** werden ausgeführt.

Zeile	1	2	3	4	5	6	3	4	5	6	3	4	5	6
a	1	1	1	2	2	2	2	5	5	5	5	13	13	13
b	–	1	1	1	3	3	3	3	8	8	8	8	21	21
i	–	–	1	1	1	1	2	2	2	2	3	3	3	3

for - Schleife

Welchen Wert hat die Variable a schließlich?

a)
1: $a \leftarrow 4$
2: **for** $i \leftarrow 1$ **to** 5 **do**
3: $\quad a \leftarrow a + 2$
4: **end for**

$a = 4 + 5 \cdot 2 = 14$

b)
1: $a \leftarrow 25$
2: **for** $i \leftarrow 3$ **to** 9 **do**
3: $\quad a \leftarrow a - 3$
4: **end for**

$a = 25 - 7 \cdot 3 = 4$

c)
1: $a \leftarrow 3$
2: **for** $i \leftarrow 1$ **to** 10 **do**
3: $\quad a \leftarrow a \cdot 2$
4: **end for**

$a = 3 \cdot 2^{10} = 3072$

for - Schleife

Fülle jene Zahl in die Lücke, damit die Variable a schließlich den Wert 42 hat.

a)
1: $a \leftarrow 63$
2: **for** $i \leftarrow 1$ **to** 7 **do**
3: $\quad a \leftarrow a - 3$
4: **end for**

$x - 7 \cdot 3 = 42 \implies x = 63$

b)
1: $a \leftarrow 6$
2: **for** $i \leftarrow 1$ **to** 9 **do**
3: $\quad a \leftarrow a + 4$
4: **end for**

$6 + x \cdot 4 = 42 \implies x = 9$

c)
1: $a \leftarrow 72$
2: **for** $i \leftarrow 4$ **to** 13 **do**
3: $\quad a \leftarrow a - 3$
4: **end for**

$72 - 10 \cdot x = 42 \implies x = 3$

for - Schleife

Das folgende Programm verwendet in einer **for**-Schleife den Wert der Zählvariable i.

1: $s \leftarrow 0$
2: **for** $i \leftarrow 1$ **to** 9 **do**
3: $\quad s \leftarrow s + i$
4: **end for**

Wir dürfen i in der Schleife aber *nicht* neue Werte zuweisen.

Welchen Wert hat die Variable s nach Durchführung der **for**-Schleife?

$0 + 1 + 2 + 3 + \cdots + 9 = \frac{9 \cdot 10}{2} = 45$

repeat until - Schleife

Bei einer **for**-Schleife legt man *zu Beginn* fest, wie oft die Schleife ausgeführt werden soll.
Bei einer **repeat until**-Schleife legt man stattdessen *am Ende* eine Abbruchbedingung fest.
Die Schleife wird *so lange wiederholt*, *bis* die Abbruchbedingung erfüllt ist. *repeat until*

1: $a \leftarrow 1$
2: **repeat**
3: $\quad a \leftarrow 2 \cdot a$
4: **until** $a > 100$

Nach Durchlauf	1	2	3	4	5	6	7
a	2	4	8	16	32	64	128

Nach 7 Durchläufen ist die Abbruchbedingung $a > 100$ erstmals erfüllt.
Die **repeat until**-Schleife bricht also ab. Die Variable a hat danach den Wert 128.

repeat until - Schleife

Wie oft wird die **repeat until**-Schleife durchgeführt? Welchen Wert hat die Variable a schließlich?

a)
1: $a \leftarrow 13$
2: **repeat**
3: $\quad a \leftarrow a - 2$
4: **until** $a \leq 4$

b)
1: $a \leftarrow 5$
2: **repeat**
3: $\quad a \leftarrow 3 \cdot a$
4: **until** $a > 2306$

Endlosschleife

Was passiert bei der folgenden **repeat until**-Schleife?

1: $a \leftarrow 1$
2: **repeat**
3: $\quad a \leftarrow a + 2$
4: **until** $a = 10$

while - Schleife

Bei einer **while**-Schleife legt man *zu Beginn* der Schleife eine Bedingung fest.
Solange diese Bedingung erfüllt ist, wird die Schleife durchgeführt. *while*

1: $a \leftarrow 2$
2: **while** $a \leq 16$ **do**
3: $\quad a \leftarrow a \cdot a$
4: **end while**

Vor Durchlauf	1	2	3	4
a	2	4	16	256

Vor dem 4. Durchlauf ist die Bedingung $a \leq 16$ erstmals *nicht* erfüllt.
Die **while**-Schleife bricht also ab. Die Variable a hat danach den Wert 256.

while - Schleife

Wie oft wird die **while**-Schleife vollständig durchgeführt? Welchen Wert hat die Variable a schließlich?

a)
1: $a \leftarrow 25$
2: **while** $a > 1$ **do**
3: $\quad a \leftarrow a - 3$
4: **end while**

b)
1: $a \leftarrow 7$
2: **while** $a^2 \leq 42$ **do**
3: $\quad a \leftarrow a + 1$
4: **end while**

repeat until - Schleife

Bei einer **for**-Schleife legt man *zu Beginn* fest, wie oft die Schleife ausgeführt werden soll.
Bei einer **repeat until**-Schleife legt man stattdessen *am Ende* eine Abbruchbedingung fest.
Die Schleife wird *so lange wiederholt*, *bis* die Abbruchbedingung erfüllt ist. *repeat until*

```
1: a ← 1
2: repeat
3:     a ← 2 · a
4: until a > 100
```

Nach Durchlauf	1	2	3	4	5	6	7
a	2	4	8	16	32	64	128

Nach 7 Durchläufen ist die Abbruchbedingung $a > 100$ erstmals erfüllt.
Die **repeat until**-Schleife bricht also ab. Die Variable a hat danach den Wert 128.

repeat until - Schleife

Wie oft wird die **repeat until**-Schleife durchgeführt? Welchen Wert hat die Variable a schließlich?

a)
```
1: a ← 13
2: repeat
3:     a ← a − 2
4: until a ≤ 4
```

$13 - 2 \cdot x \leq 4 \implies x \geq 4{,}5$
5 Durchführungen, $a = 13 - 2 \cdot 5 = 3$

b)
```
1: a ← 5
2: repeat
3:     a ← 3 · a
4: until a > 2306
```

$5 \cdot 3^x > 2306 \implies x > \underbrace{\log_3\left(\frac{2306}{5}\right)}_{=5{,}58\ldots}$
6 Durchführungen, $a = 5 \cdot 3^6 = 3645$

Endlosschleife

Was passiert bei der folgenden **repeat until**-Schleife?

```
1: a ← 1
2: repeat
3:     a ← a + 2
4: until a = 10
```

Die Abbruchbedingung wird *nie* erfüllt.
$(1 \to 3 \to 5 \to 7 \to 9 \to 11 \to 13 \to \cdots)$
Die Schleife wird also *nie* abgebrochen. („Endlosschleife“)

while - Schleife

Bei einer **while**-Schleife legt man *zu Beginn* der Schleife eine Bedingung fest.
Solange diese Bedingung erfüllt ist, wird die Schleife durchgeführt. *while*

```
1: a ← 2
2: while a ≤ 16 do
3:     a ← a · a
4: end while
```

Vor Durchlauf	1	2	3	4
a	2	4	16	256

Vor dem 4. Durchlauf ist die Bedingung $a \leq 16$ erstmals *nicht* erfüllt.
Die **while**-Schleife bricht also ab. Die Variable a hat danach den Wert 256.

while - Schleife

Wie oft wird die **while**-Schleife vollständig durchgeführt? Welchen Wert hat die Variable a schließlich?

a)
```
1: a ← 25
2: while a > 1 do
3:     a ← a − 3
4: end while
```

$25 - 3 \cdot x > 1 \implies x < 8$
8 vollständige Durchführungen, $a = 1$

b)
```
1: a ← 7
2: while a² ≤ 42 do
3:     a ← a + 1
4: end while
```

0 vollständige Durchführungen, $a = 7$

3.2 Sortieralgorithmen

Schon vor der großflächigen Verbreitung des PCs und Internets sowie der Prägung des Begriffs „big data“ war das effiziente Sortieren und Suchen in großen Datenmengen ein Forschungsgebiet der Informatik und Mathematik [17].

Dieses Thema bietet sich meines Erachtens auch in der Schule in vielerlei Hinsicht an:

- SchülerInnen kennen die Aufgabenstellung aus Situationen des Alltags.
  Zum Beispiel: Suchergebnisse nach Bewertung/Preis/Relevanz etc. sortieren lassen.
- Die Aufgabenstellung lässt sich leicht greifbar machen.
  Zum Beispiel: Spielkarten der Größe nach sortieren lassen.
- Der Unterschied zwischen Mensch und Maschine lässt sich leicht sichtbar machen:
  Im Gegensatz zum Computer sehen wir als Menschen „auf einen Blick“, welche von 10 Zufallszahlen zwischen 1 und 100 die kleinste Zahl ist.
  Der Computer benötigt stattdessen konkrete Handlungsanweisungen, wie er auf die Speicherplätze zugreifen und die Zahlen vergleichen soll.
- Die gleiche Aufgabenstellung kann auf verschiedene Arten gelöst werden.
  Die Frage, wie man dann diese Methoden vergleichen kann, liegt auf der Hand.
  Welchen Unterschied kann hier ein besserer Algorithmus ausmachen?

Auf dem nächsten Arbeitsblatt werden die beiden Sortieralgorithmen *Selection Sort* – als Vertreter der intuitiven, aber langsamen Verfahren – und *Merge Sort* – als Vertreter der schnellen Verfahren – aufbereitet und verglichen.

Es folgt das Arbeitsblatt – Sortieralgorithmen und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Arbeitsblatt – Einführung in Algorithmen

– Arbeitsblatt – Vollständige Induktion

Lernziele:

✓ Wie funktioniert der Algorithmus **Selection Sort**?
Wie viele Vergleiche benötigt er zum Sortieren von n Zahlen?

✓ Wie funktioniert der Algorithmus **Merge Sort**?
Wie viele Vergleiche benötigt er ungefähr zum Sortieren von n Zahlen?

✓ Was ist ein **rekursiver Algorithmus**?
Wie hängen rekursive Algorithmen und vollständige Induktion zusammen?

Selection Sort

Der Algorithmus **Selection Sort** sortiert jede Liste mit n Zahlen in n Durchläufen aufsteigend:

1) Im ersten Durchlauf durchsucht er die Liste nach der kleinsten Zahl und tauscht sie mit der ersten Zahl.

Dann ist die kleinste Zahl also *sicher* an der richtigen Stelle.

2) Im zweiten Durchlauf durchsucht er die Liste ab der zweiten Zahl nach der kleinsten Zahl und tauscht sie mit der zweiten Zahl.

Dann sind die kleinsten 2 Zahlen also *sicher* an der richtigen Stelle.

Durchlauf 1	17	11	**2**	19	3	13	7	5
Durchlauf 2	2	11	17	19	**3**	13	7	5
Durchlauf 3	2	3	17	19	11	13	7	**5**
Durchlauf 4	2	3	5	19	11	13	**7**	17
Durchlauf 5	2	3	5	7	**11**	13	19	17
Durchlauf 6	2	3	5	7	11	**13**	19	17
Durchlauf 7	2	3	5	7	11	13	19	**17**
Durchlauf 8	2	3	5	7	11	13	17	**19**
	2	3	5	7	11	13	17	19

3) Im dritten Durchlauf durchsucht er die Liste ab der dritten Zahl nach der kleinsten Zahl und tauscht sie mit der dritten Zahl.

Dann sind die kleinsten 3 Zahlen also *sicher* an der richtigen Stelle.

⋮

Nach n Durchläufen ist die Liste aufsteigend sortiert.

Den letzten Durchlauf können wir auch weglassen.

Selection Sort

Sortiere die Liste $\langle 42, 23, 6, 26, 3, 87, 30\rangle$ mit dem Sortieralgorithmus **Selection Sort** aufsteigend.

	42	23	6	26	3	87	30
Durchlauf 1							
Durchlauf 2							
Durchlauf 3							
Durchlauf 4							
Durchlauf 5							
Durchlauf 6							
Durchlauf 7							

Algorithmus – Selection Sort

Algorithmus: SelectionSort
Input: Liste L mit n Zahlen
Output: Liste L mit aufsteigend sortierten Zahlen

```
1:  function SelectionSort(L)
2:      for i ← 1 to n − 1 do
3:          min ← L[i]
4:          minPos ← i
5:          for j ← i + 1 to n do
6:              if L[j] < min then
7:                  min ← L[j]
8:                  minPos ← j
9:              end if
10:         end for
11:         t ← L[i]
12:         L[i] ← min
13:         L[minPos] ← t
14:     end for
15:     return L
16: end function
```

Die Zählvariable i gibt die aktuelle Durchlaufnummer an.
Den letzten Durchlauf $i = n$ lassen wir hier weg. (Zeile 2)

Im Durchlauf i durchsuchen wir $L[i]$, $L[i+1]$, $L[i+2]$, …, $L[n]$ von links nach rechts nach der kleinsten Zahl. (Zeilen 3–9)

Dafür verwenden wir zwei Hilfsvariablen:
Am Ende von Durchlauf i soll die Variable min die kleinste Zahl unter den Zahlen $L[i]$, $L[i+1]$, $L[i+2]$, …, $L[n]$ enthalten.
Die Variable $minPos$ speichert die Position („Index“) dieser kleinsten Zahl.

Zu Beginn von Durchlauf i ist $L[i]$ die kleinste gefundene Zahl. (Zeilen 3–4)
Jedes Mal, wenn wir eine kleinere Zahl finden, speichern wir diese Zahl und ihre Position ab. (Zeilen 5–9)

Zum Abschluss von Durchlauf i vertauschen wir die Zahlen an den Positionen i und $minPos$. Dafür ist eine Hilfsvariable zum Zwischenspeichern eines Werts notwendig. (Zeilen 11–13)

Nach den $n-1$ Durchläufen sind die Zahlen in L aufsteigend sortiert.
L wird als Output zurückgegeben. (Zeile 15)

Selection Sort

Der Algorithmus **Selection Sort** sortiert jede Liste mit n Zahlen in n Durchläufen aufsteigend:

1) Im ersten Durchlauf durchsucht er die Liste nach der kleinsten Zahl und tauscht sie mit der ersten Zahl.

Dann ist die kleinste Zahl also *sicher* an der richtigen Stelle.

2) Im zweiten Durchlauf durchsucht er die Liste ab der zweiten Zahl nach der kleinsten Zahl und tauscht sie mit der zweiten Zahl.

Dann sind die kleinsten 2 Zahlen also *sicher* an der richtigen Stelle.

Durchlauf 1	17	11	**2**	19	3	13	7	5
Durchlauf 2	2	11	17	19	**3**	13	7	5
Durchlauf 3	2	3	17	19	11	13	7	**5**
Durchlauf 4	2	3	5	19	11	13	**7**	17
Durchlauf 5	2	3	5	7	**11**	13	19	17
Durchlauf 6	2	3	5	7	11	**13**	19	17
Durchlauf 7	2	3	5	7	11	13	19	**17**
Durchlauf 8	2	3	5	7	11	13	17	**19**

3) Im dritten Durchlauf durchsucht er die Liste ab der dritten Zahl nach der kleinsten Zahl und tauscht sie mit der dritten Zahl.

Dann sind die kleinsten 3 Zahlen also *sicher* an der richtigen Stelle.

⋮

Nach n Durchläufen ist die Liste aufsteigend sortiert.

Den letzten Durchlauf können wir auch weglassen.

Selection Sort

Sortiere die Liste $\langle 42, 23, 6, 26, 3, 87, 30\rangle$ mit dem Sortieralgorithmus **Selection Sort** aufsteigend.

Durchlauf 1	42	23	6	26	**3**	87	30
Durchlauf 2	3	23	**6**	26	42	87	30
Durchlauf 3	3	6	**23**	26	42	87	30
Durchlauf 4	3	6	23	**26**	42	87	30
Durchlauf 5	3	6	23	26	42	87	**30**
Durchlauf 6	3	6	23	26	30	87	**42**
Durchlauf 7	3	6	23	26	30	42	**87**
	3	6	23	26	30	42	87

Algorithmus – Selection Sort

Algorithmus: SelectionSort
Input: Liste L mit n Zahlen
Output: Liste L mit aufsteigend sortierten Zahlen

```
1:  function SelectionSort(L)
2:      for i ← 1 to n − 1 do
3:          min ← L[i]
4:          minPos ← i
5:          for j ← i + 1 to n do
6:              if L[j] < min then
7:                  min ← L[j]
8:                  minPos ← j
9:              end if
10:         end for
11:         t ← L[i]
12:         L[i] ← min
13:         L[minPos] ← t
14:     end for
15:     return L
16: end function
```

Die Zählvariable i gibt die aktuelle Durchlaufnummer an.
Den letzten Durchlauf $i = n$ lassen wir hier weg. (Zeile 2)

Im Durchlauf i durchsuchen wir $L[i]$, $L[i+1]$, $L[i+2]$, …, $L[n]$ von links nach rechts nach der kleinsten Zahl. (Zeilen 3–9)

Dafür verwenden wir zwei Hilfsvariablen:
Am Ende von Durchlauf i soll die Variable min die kleinste Zahl unter den Zahlen $L[i]$, $L[i+1]$, $L[i+2]$, …, $L[n]$ enthalten.
Die Variable $minPos$ speichert die Position („Index“) dieser kleinsten Zahl.

Zu Beginn von Durchlauf i ist $L[i]$ die kleinste gefundene Zahl. (Zeilen 3–4)
Jedes Mal, wenn wir eine kleinere Zahl finden, speichern wir diese Zahl und ihre Position ab. (Zeilen 5–9)

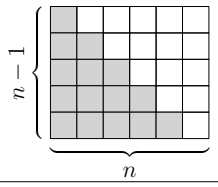
Zum Abschluss von Durchlauf i vertauschen wir die Zahlen an den Positionen i und $minPos$. Dafür ist eine Hilfsvariable zum Zwischenspeichern eines Werts notwendig. (Zeilen 11–13)

Nach den $n-1$ Durchläufen sind die Zahlen in L aufsteigend sortiert.
L wird als Output zurückgegeben. (Zeile 15)

Selection Sort – Anzahl Vergleiche

Wie viele Vergleiche benötigt der Algorithmus **Selection Sort** zum Sortieren einer Liste mit n Zahlen?

Jede Abfrage der Form „Ist $a < b$?“, „Ist $a = b$?“, „Ist $a \leq b$?“ usw. zählt als Vergleich.



Zwei sortierte Listen verschmelzen

Aktivität:	Zwei *sortierte* Listen verschmelzen
Benötigtes Material:	≈ 30 Karten, die möglichst groß mit jeweils einer Zahl beschriftet sind.
Anzahl Personen:	2
Vorbereitung:	Mit den Karten 2 etwa gleich große Stapel bilden, die jeweils aufsteigend sortiert sind. Die beiden Personen erhalten jeweils einen Stapel und zeigen die kleinste Zahl in ihrem Stapel her.
Aufgabenstellung:	Die beiden Personen sollen aus den sortierten Stapeln einen gemeinsamen sortierten Stapel erzeugen.

Sortierte Listen verschmelzen

Gegeben sind zwei sortierte Zahlenlisten L_1 und L_2.
Die beiden Listen sollen zu einer sortierten Gesamtliste L verschmolzen werden.
Dazu vergleichen wir immer wieder die kleinsten Zahlen der beiden Listen.
Die kleinere Zahl verschieben wir an das Ende der sortierten Gesamtliste L:

Sortierte Liste L_1	Sortierte Liste L_2		Sortierte Gesamtliste L
2 11 17 19	**3** 5 7 13	⟶	2
11 17 19	**3** 5 7 13	⟶	2 3
11 17 19	**5** 7 13	⟶	2 3 5
11 17 19	**7** 13	⟶	2 3 5 7
11 17 19	**13**	⟶	2 3 5 7 11
17 19	**13**	⟶	2 3 5 7 11 13
17 19		⟶	2 3 5 7 11 13 17 19

Sobald L_1 oder L_2 leer ist, können wir den Rest der anderen Liste an die Gesamtliste anhängen.
Angenommen, die beiden sortierten Listen enthalten zusammen n Zahlen.
Erkläre, warum damit zum Verschmelzen der Listen weniger als n Vergleiche notwendig sind.

Merge Sort – Iterative Lösung

Aktivität:	Merge Sort „merge“ ist englisch für „verschmelzen“.
Benötigtes Material:	≈ 30 Karten, die möglichst groß mit jeweils einer Zahl bedruckt sind.
Anzahl Personen:	beliebig
Vorbereitung:	Jede Person erhält 1 Karte. Jede Person hat also einen sortierten „Stapel“.
Aufgabenstellung:	Das Ziel ist am Ende einen einzigen sortierten Kartenstapel zu haben. Dazu treffen sich immer wieder 2 Personen und verschmelzen wie zuvor ihre Stapel. Diese Treffen können *gleichzeitig* stattfinden, bis nur noch ein sortierter Stapel übrig ist. Parallelisierung

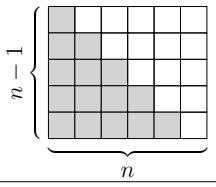
Selection Sort – Anzahl Vergleiche

Wie viele Vergleiche benötigt der Algorithmus **Selection Sort** zum Sortieren einer Liste mit n Zahlen?

Jede Abfrage der Form „Ist $a < b$?“, „Ist $a = b$?“, „Ist $a \leq b$?“ usw. zählt als Vergleich.

In Durchlauf i finden $n - i$ Vergleiche statt.

Insgesamt werden also $1 + 2 + 3 + \cdots + (n-1) = \frac{n \cdot (n-1)}{2}$ Vergleiche benötigt.



Zwei sortierte Listen verschmelzen

Aktivität:	Zwei *sortierte* Listen verschmelzen
Benötigtes Material:	≈ 30 Karten, die möglichst groß mit jeweils einer Zahl beschriftet sind.
Anzahl Personen:	2
Vorbereitung:	Mit den Karten 2 etwa gleich große Stapel bilden, die jeweils aufsteigend sortiert sind. Die beiden Personen erhalten jeweils einen Stapel und zeigen die kleinste Zahl in ihrem Stapel her.
Aufgabenstellung:	Die beiden Personen sollen aus den sortierten Stapeln einen gemeinsamen sortierten Stapel erzeugen.

Sortierte Listen verschmelzen

Gegeben sind zwei sortierte Zahlenlisten L_1 und L_2.
Die beiden Listen sollen zu einer sortierten Gesamtliste L verschmolzen werden.
Dazu vergleichen wir immer wieder die kleinsten Zahlen der beiden Listen.
Die kleinere Zahl verschieben wir an das Ende der sortierten Gesamtliste L:

Sortierte Liste L_1	Sortierte Liste L_2		Sortierte Gesamtliste L
2 11 17 19	**3** 5 7 13	⟶	2
11 17 19	**3** 5 7 13	⟶	2 3
11 17 19	**5** 7 13	⟶	2 3 5
11 17 19	**7** 13	⟶	2 3 5 7
11 17 19	**13**	⟶	2 3 5 7 11
17 19	**13**	⟶	2 3 5 7 11 13
17 19		⟶	2 3 5 7 11 13 17 19

Sobald L_1 oder L_2 leer ist, können wir den Rest der anderen Liste an die Gesamtliste anhängen.
Angenommen, die beiden sortierten Listen enthalten zusammen n Zahlen.
Erkläre, warum damit zum Verschmelzen der Listen weniger als n Vergleiche notwendig sind.
In jeder Zeile findet genau ein Vergleich statt.
Spätestens nach $n - 1$ Vergleichen ist eine der beiden Listen leer.

Merge Sort – Iterative Lösung

Aktivität:	Merge Sort „merge“ ist englisch für „verschmelzen“.
Benötigtes Material:	≈ 30 Karten, die möglichst groß mit jeweils einer Zahl bedruckt sind.
Anzahl Personen:	beliebig
Vorbereitung:	Jede Person erhält 1 Karte. Jede Person hat also einen sortierten „Stapel“.
Aufgabenstellung:	Das Ziel ist am Ende einen einzigen sortierten Kartenstapel zu haben. Dazu treffen sich immer wieder 2 Personen und verschmelzen wie zuvor ihre Stapel. Diese Treffen können *gleichzeitig* stattfinden, bis nur noch ein sortierter Stapel übrig ist. Parallelisierung

Merge Sort – Anzahl Vergleiche

Der Algorithmus **Merge Sort** sortiert jede Liste mit $n = 2^k$ Zahlen in k Durchläufen:
Rechts siehst du ein Beispiel mit $k = 3$ und $n = 2^3 = 8$.

Wir starten mit 8 sortierten Listen mit jeweils einer Zahl.
Nach Durchlauf 1 enthalten die 4 sortierten Listen jeweils 2 Zahlen.
Nach Durchlauf 2 enthalten die 2 sortierten Listen jeweils 4 Zahlen.
Nach Durchlauf 3 enthält die eine sortierte Liste alle 8 Zahlen.

Durchlauf 1	17	11	2	19	3	13	7	5
Durchlauf 2	11	17	2	19	3	13	5	7
Durchlauf 3	2	11	17	19	3	5	7	13
	2	3	5	7	11	13	17	19

In jedem Durchlauf sind weniger als $n = 8$ Vergleiche notwendig. Zum Sortieren von $n = 2^k$ Zahlen benötigt **Merge Sort** also weniger als $n \cdot k = n \cdot \log_2(n)$ Vergleiche. $n = 2^k \iff k = \log_2(n)$

Bei $n = 9, 10, \ldots, 16$ Zahlen sind dann 4 Durchläufe notwendig. Allgemein benötigt Merge Sort weniger als $n \cdot \lceil \log_2(n) \rceil$ Vergleiche.

Analyse von Algorithmen

Wenn zwei Algorithmen das gleiche Problem lösen, können wir sie nach mehreren Kriterien vergleichen:

- Wie viele Operationen benötigt der Algorithmus? Wertzuweisungen ($x \leftarrow 42$) Vergleiche („Ist $a < b$?“)
- Wie viel Speicherplatz ist insgesamt notwendig?

Die Antworten auf diese Fragen können dabei von der Problemgröße abhängen.
Bei Sortieralgorithmen ist die Problemgröße die Anzahl n der Zahlen, die sortiert werden sollen.
Bei der Analyse von Algorithmen wollen wir Antworten auf diese Fragen für *großes* n finden.

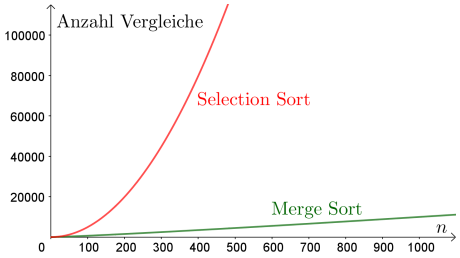
Selection Sort vs. Merge Sort

Eine Liste mit n Zahlen soll sortiert werden.
Dafür benötigt der Algorithmus …

… **Selection Sort** insgesamt $\frac{n \cdot (n-1)}{2}$ Vergleiche.

… **Merge Sort** insgesamt rund $n \cdot \log_2(n)$ Vergleiche.

Die Graphen der zugehörigen Funktionen siehst du rechts.



Bei $n = 2^{10} = 1024$ Zahlen benötigt der Algorithmus …

… **Selection Sort** insgesamt _____ Vergleiche.

… **Merge Sort** insgesamt rund _____ Vergleiche.

Die Analyse von Algorithmen und die Suche nach effizienten Algorithmen sind Disziplinen, die eine Schnittstelle zwischen Informatik und Mathematik bilden.

Merge Sort – Rekursive Lösung

Algorithmus: MergeSort
Input: Liste L mit n Zahlen
Output: Liste L mit aufsteigend sortierten Zahlen

```
1:  function MergeSort(L)
2:      n ← Length(L)
3:      if n < 2 then
4:          return L
5:      else
6:          L1 ← MergeSort(FirstHalf(L))
7:          L2 ← MergeSort(SecondHalf(L))
8:          return MergeSortedLists(L1, L2)
9:      end if
10: end function
```

Falls die Input-Liste L weniger als 2 Zahlen enthält, ist sie automatisch sortiert und kann sofort als Output zurückgegeben werden. (Zeilen 2–4)

Ansonsten teilen wir L in zwei Hälften FirstHalf(L) und SecondHalf(L).
Diese beiden kürzeren Listen lassen wir vom gleichen Algorithmus MergeSort sortieren. (Zeilen 6–7)
Warum das funktioniert, erfährst du gleich.

Die beiden sortierten Listen L_1 und L_2 verschmelzen wir wie zuvor zu einer sortierten Gesamtliste und geben sie als Output zurück. (Zeile 8)

Merge Sort – Anzahl Vergleiche

Der Algorithmus **Merge Sort** sortiert jede Liste mit $n = 2^k$ Zahlen in k Durchläufen:
Rechts siehst du ein Beispiel mit $k = 3$ und $n = 2^3 = 8$.

Wir starten mit 8 sortierten Listen mit jeweils einer Zahl.
Nach Durchlauf 1 enthalten die 4 sortierten Listen jeweils 2 Zahlen.
Nach Durchlauf 2 enthalten die 2 sortierten Listen jeweils 4 Zahlen.
Nach Durchlauf 3 enthält die eine sortierte Liste alle 8 Zahlen.

	17	11	2	19	3	13	7	5
Durchlauf 1	11	17	2	19	3	13	5	7
Durchlauf 2	2	11	17	19	3	5	7	13
Durchlauf 3	2	3	5	7	11	13	17	19

In jedem Durchlauf sind weniger als $n = 8$ Vergleiche notwendig. Zum Sortieren von $n = 2^k$ Zahlen benötigt **Merge Sort** also weniger als $n \cdot k = n \cdot \log_2(n)$ Vergleiche. $n = 2^k \iff k = \log_2(n)$

Bei $n = 9, 10, \ldots, 16$ Zahlen sind dann 4 Durchläufe notwendig. Allgemein benötigt Merge Sort weniger als $n \cdot \lceil \log_2(n) \rceil$ Vergleiche.

Analyse von Algorithmen

Wenn zwei Algorithmen das gleiche Problem lösen, können wir sie nach mehreren Kriterien vergleichen:

- Wie viele Operationen benötigt der Algorithmus? Wertzuweisungen ($x \leftarrow 42$) Vergleiche („Ist $a < b$?“)
- Wie viel Speicherplatz ist insgesamt notwendig?

Die Antworten auf diese Fragen können dabei von der Problemgröße abhängen.
Bei Sortieralgorithmen ist die Problemgröße die Anzahl n der Zahlen, die sortiert werden sollen.
Bei der Analyse von Algorithmen wollen wir Antworten auf diese Fragen für *großes* n finden.

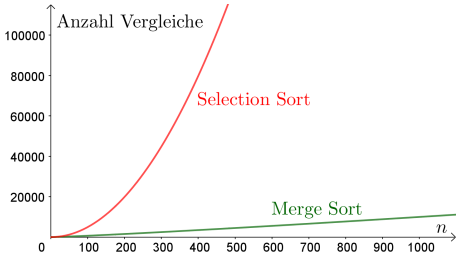
Selection Sort vs. Merge Sort

Eine Liste mit n Zahlen soll sortiert werden.
Dafür benötigt der Algorithmus …

… **Selection Sort** insgesamt $\frac{n \cdot (n-1)}{2}$ Vergleiche.

… **Merge Sort** insgesamt rund $n \cdot \log_2(n)$ Vergleiche.

Die Graphen der zugehörigen Funktionen siehst du rechts.



Bei $n = 2^{10} = 1024$ Zahlen benötigt der Algorithmus …

… **Selection Sort** insgesamt 523776 Vergleiche.

… **Merge Sort** insgesamt rund 10240 Vergleiche.

Die Analyse von Algorithmen und die Suche nach effizienten Algorithmen sind Disziplinen, die eine Schnittstelle zwischen Informatik und Mathematik bilden.

Merge Sort – Rekursive Lösung

Algorithmus: MergeSort
Input: Liste L mit n Zahlen
Output: Liste L mit aufsteigend sortierten Zahlen

```
1:  function MergeSort(L)
2:      n ← Length(L)
3:      if n < 2 then
4:          return L
5:      else
6:          L1 ← MergeSort(FirstHalf(L))
7:          L2 ← MergeSort(SecondHalf(L))
8:          return MergeSortedLists(L1, L2)
9:      end if
10: end function
```

Falls die Input-Liste L weniger als 2 Zahlen enthält, ist sie automatisch sortiert und kann sofort als Output zurückgegeben werden. (Zeilen 2–4)

Ansonsten teilen wir L in zwei Hälften FirstHalf(L) und SecondHalf(L).
Diese beiden kürzeren Listen lassen wir vom gleichen Algorithmus MergeSort sortieren. (Zeilen 6–7)
Warum das funktioniert, erfährst du gleich.

Die beiden sortierten Listen L_1 und L_2 verschmelzen wir wie zuvor zu einer sortierten Gesamtliste und geben sie als Output zurück. (Zeile 8)

Rekursive Algorithmen

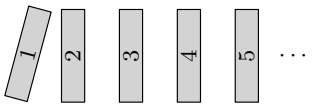
Dieser Algorithmus MergeSort löst ein Problem, indem er sich selbst aufruft.
In der Informatik nennt man solche Algorithmen **rekursiv**. „recur“ ist englisch für „wiederkehren“.

Warum funktioniert der rekursive Merge Sort – Algorithmus?

Hinter rekursiven Algorithmen steckt das mathematische Prinzip der vollständigen Induktion.
Wir zeigen, dass MergeSort jede Liste mit $n = 1, 2, 3, 4, 5, \ldots$ Zahlen richtig sortiert.

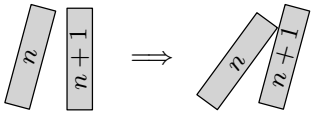
1) Induktionsanfang:
Erkläre, warum MergeSort jede Liste mit einer einzigen Zahl richtig sortiert.

Der erste Dominostein fällt um.



2) Induktionsschritt:
Du darfst verwenden, dass MergeSort jede Liste mit *höchstens* n Zahlen richtig sortiert. $n \geq 1$
Erkläre, warum MergeSort dann auch jede Liste mit $n+1$ Zahlen richtig sortiert.

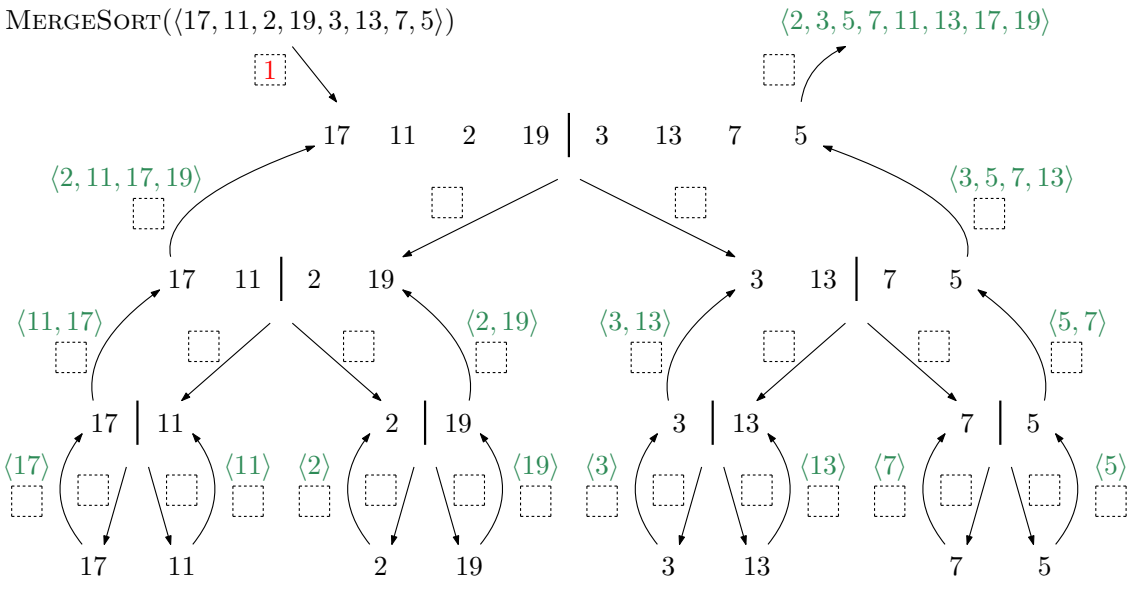
Wenn die ersten n Dominosteine umfallen, dann fällt auch der $(n+1)$-te Dominostein um.



MergeSort sortiert also jede noch so lange Liste richtig. Jeder Dominostein fällt schließlich um.

Programmablauf

Jeder Pfeil nach unten ist ein MergeSort-Aufruf.
Jeder Pfeil nach oben ist mit dem zugehörigen Output beschriftet.
In welcher Reihenfolge werden diese Schritte beim rekursiven MergeSort-Algorithmus durchgeführt?
Fülle in die Lücken die richtige Reihenfolge $1, 2, 3, 4, 5, 6, \ldots$ ein:



Rekursive Algorithmen

Dieser Algorithmus MergeSort löst ein Problem, indem er sich selbst aufruft.
In der Informatik nennt man solche Algorithmen **rekursiv**. „recur“ ist englisch für „wiederkehren“.

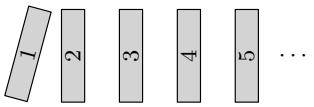
***Warum* funktioniert der rekursive Merge Sort – Algorithmus?**

Hinter rekursiven Algorithmen steckt das mathematische Prinzip der vollständigen Induktion.
Wir zeigen, dass MergeSort jede Liste mit $n = 1, 2, 3, 4, 5, \ldots$ Zahlen richtig sortiert.

1) Induktionsanfang:
Erkläre, warum MergeSort jede Liste mit einer einzigen Zahl richtig sortiert.

Der erste Dominostein fällt um.

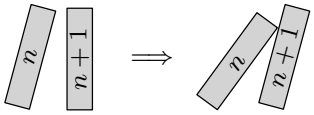
Eine Liste mit nur einer Zahl ist automatisch sortiert. In Zeile 4 bricht MergeSort ab und gibt die Liste unverändert zurück.



2) Induktionsschritt:
Du darfst verwenden, dass MergeSort jede Liste mit *höchstens* n Zahlen richtig sortiert. $n \geq 1$
Erkläre, warum MergeSort dann auch jede Liste mit $n+1$ Zahlen richtig sortiert.

Wenn die ersten n Dominosteine umfallen, dann fällt auch der $(n+1)$-te Dominostein um.

FirstHalf(L) und SecondHalf(L) enthalten jeweils höchstens n Zahlen. In Zeile 6 und 7 liefert MergeSort also sortierte Listen L_1 und L_2. Die beiden sortierten Listen werden in Zeile 8 zu einer sortierten Liste verschmolzen und zurückgegeben.

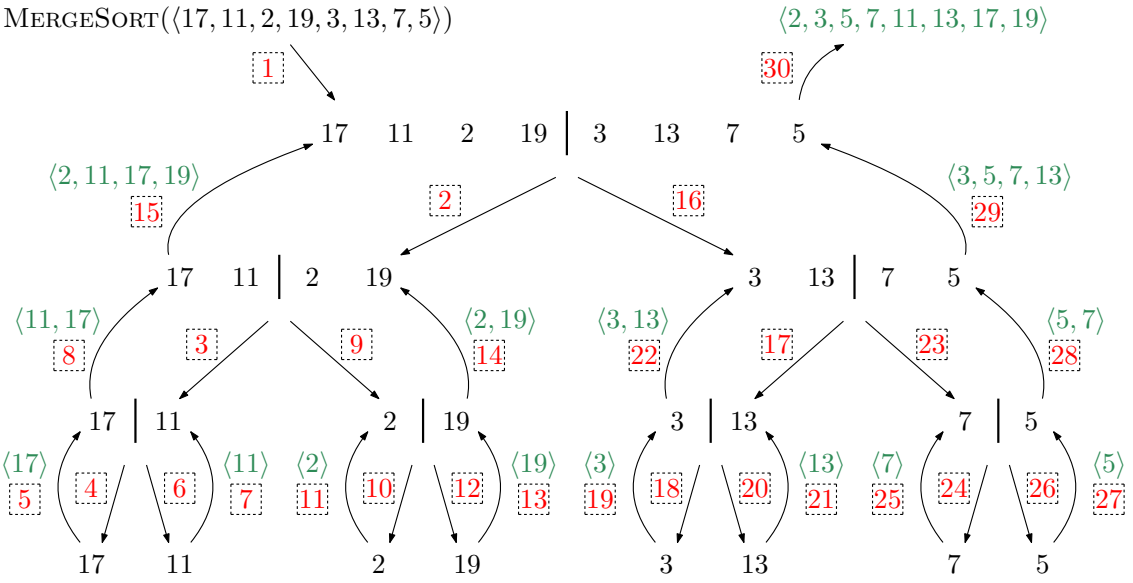


MergeSort sortiert also jede noch so lange Liste richtig.

Jeder Dominostein fällt schließlich um.

Programmablauf

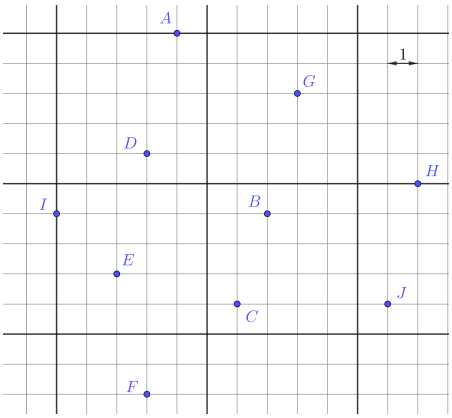
Jeder Pfeil nach unten ist ein MergeSort-Aufruf.
Jeder Pfeil nach oben ist mit dem zugehörigen Output beschriftet.
In welcher Reihenfolge werden diese Schritte beim rekursiven MergeSort-Algorithmus durchgeführt?
Fülle in die Lücken die richtige Reihenfolge $1, 2, 3, 4, 5, 6, \ldots$ ein:



3.3 Kruskal-Algorithmus

„Im Bild rechts sind 10 Gitterpunkte eingezeichnet. Du darfst mit einem Stift Verbindungen zwischen den Punkten einzeichnen. Schließlich soll jeder Punkt mit allen anderen Punkten verbunden sein – entweder direkt oder indirekt über andere Punkte.

Die Gesamtlänge aller eingezeichneten Verbindungen soll dabei so klein wie möglich sein.“



Das „minimum spanning tree problem“ ist mit dieser geringen Einstiegshürde aus meiner Sicht bestens für den Schulunterricht geeignet. Ein möglicher Aufbau zur Behandlung dieser und weiterer graphentheoretischer Fragestellungen im Unterricht wird in [19] anhand von lebensnahen Aufgaben beschrieben.

Für das „minimum spanning tree problem“ gibt es mehrere Verfahren, mit denen stets eine beste Lösung gefunden werden kann. Joseph Kruskal veröffentlichte einen solchen Algorithmus im Jahr 1956 ([18]). Dieser nach ihm benannte Algorithmus sowie der Beweis, dass er stets eine optimale Lösung findet, sind auf dem nächsten Arbeitsblatt aufbereitet.

Es folgt das Arbeitsblatt – Kruskal-Algorithmus und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Arbeitsblatt – Einführung in Algorithmen

– Arbeitsblatt – Vollständige Induktion

Lernziele:

✓ Was ist ein **kantengewichteter Graph**?

✓ Was sind die **Zusammenhangskomponenten** eines Graphen?

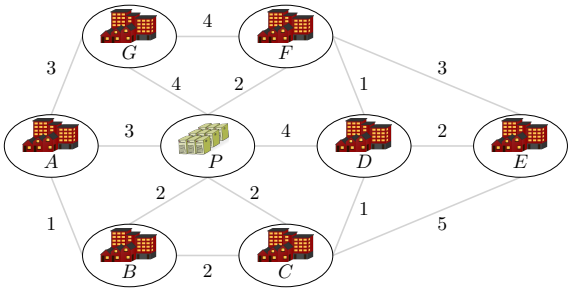
✓ Welche Graphen sind **Bäume**?

✓ Wie konstruiert der **Kruskal-Algorithmus** einen aufspannenden Baum?

✓ Warum liefert der Algorithmus einen **minimal aufspannenden Baum**?

Minimum spanning tree problem

Ein Internet-Provider möchte 7 Städte mit Glasfaserkabeln in sein Netzwerk anbinden.
Die Anbindung kann entweder direkt mit dem Provider oder indirekt über Städte erfolgen.
Alle möglichen Verbindungen und die Kosten für die Kabelverlegung (in Mio. €) sind dargestellt.



Wie viele Verbindungen sind mindestens notwendig, um alle Städte anzubinden?

Versuche ein Netzwerk zu finden, bei dem die Gesamtkosten so klein wie möglich sind.

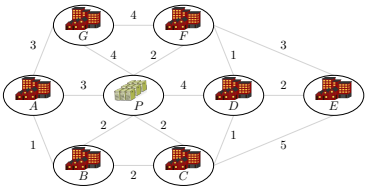
Kantengewichteter Graph

In der Graphentheorie sprechen wir von **kantengewichteten Graphen** und nennen ...

... die Orte A, B, C, D, E, F, G, P auch **Knoten**.

... die Strecken zwischen 2 Knoten auch **Kanten**.

... die Zahlen neben den Kanten auch **Gewichte**.



Die positiven Gewichte geben in diesem Kontext die Kosten zur Verbindung zweier Orte an.

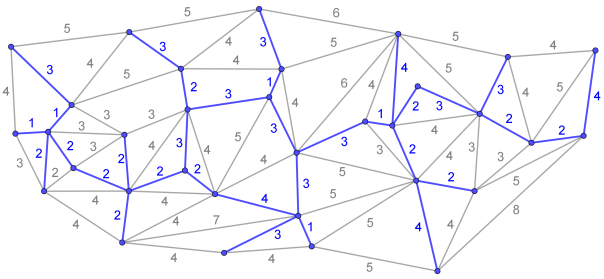
Kreise

Im rechts dargestellten Netzwerk sind alle Städte direkt oder indirekt miteinander verbunden.

Die Gesamtkosten sind aber sicher *nicht* so klein wie möglich, weil es einen **Kreis** gibt.

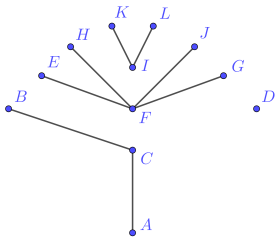
Zeichne ihn rechts ein.

Erkläre, um wieviel du die Gesamtkosten sicher reduzieren kannst.

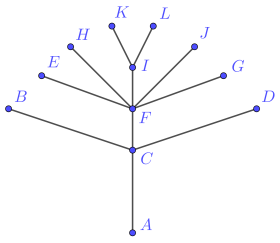


Zusammenhangskomponenten & Bäume

Alle Knoten eines Graphen, die direkt oder indirekt miteinander verbunden sind, bilden eine sogenannte **Zusammenhangskomponente** (kurz: **Komponente**).
Zum Beispiel hat der Graph links die 4 Komponenten $\{A, B, C\}$, $\{D\}$, $\{E, F, G, H, J\}$ und $\{I, K, L\}$.



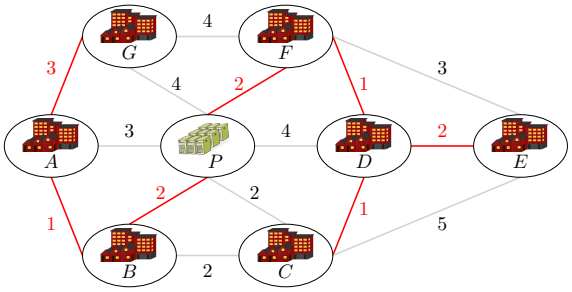
Der Graph rechts hat nur eine Komponente.
Solche Graphen heißen **zusammenhängend**.



Ein zusammenhängender Graph, der keinen Kreis enthält, heißt **Baum**.

Minimum spanning tree problem

Ein Internet-Provider möchte 7 Städte mit Glasfaserkabeln in sein Netzwerk anbinden.
Die Anbindung kann entweder direkt mit dem Provider oder indirekt über Städte erfolgen.
Alle möglichen Verbindungen und die Kosten für die Kabelverlegung (in Mio. €) sind dargestellt.



Wie viele Verbindungen sind mindestens notwendig, um alle Städte anzubinden?

7 Verbindungen

Versuche ein Netzwerk zu finden, bei dem die Gesamtkosten so klein wie möglich sind.

Minimale Gesamtkosten: 12 Mio. €

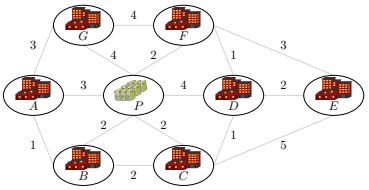
Kantengewichteter Graph

In der Graphentheorie sprechen wir von **kantengewichteten Graphen** und nennen ...

... die Orte A, B, C, D, E, F, G, P auch **Knoten**.

... die Strecken zwischen 2 Knoten auch **Kanten**.

... die Zahlen neben den Kanten auch **Gewichte**.



Die positiven Gewichte geben in diesem Kontext die Kosten zur Verbindung zweier Orte an.

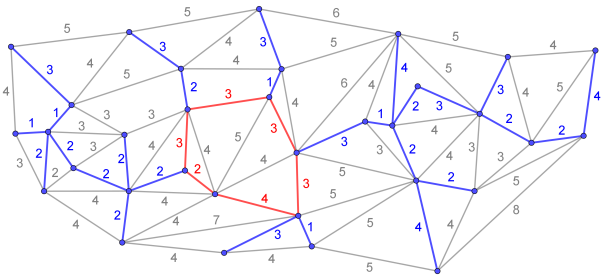
Kreise

Im rechts dargestellten Netzwerk sind alle Städte direkt oder indirekt miteinander verbunden.

Die Gesamtkosten sind aber sicher *nicht* so klein wie möglich, weil es einen **Kreis** gibt.

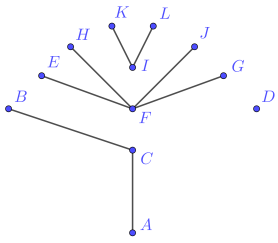
Zeichne ihn rechts ein.

Erkläre, um wieviel du die Gesamtkosten sicher reduzieren kannst.
Entfernt man im Kreis die Verbindung mit Kosten 4, bleiben alle Städte verbunden.

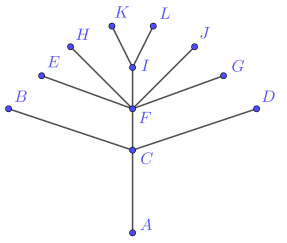


Zusammenhangskomponenten & Bäume

Alle Knoten eines Graphen, die direkt oder indirekt miteinander verbunden sind, bilden eine sogenannte **Zusammenhangskomponente** (kurz: **Komponente**).
Zum Beispiel hat der Graph links die 4 Komponenten $\{A, B, C\}$, $\{D\}$, $\{E, F, G, H, J\}$ und $\{I, K, L\}$.



Der Graph rechts hat nur eine Komponente.
Solche Graphen heißen **zusammenhängend**.



Ein zusammenhängender Graph, der keinen Kreis enthält, heißt **Baum**.

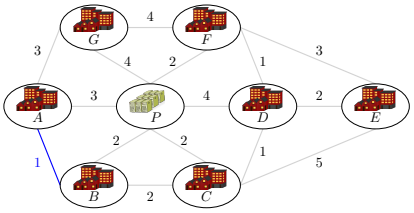
Kruskal-Algorithmus (1956)

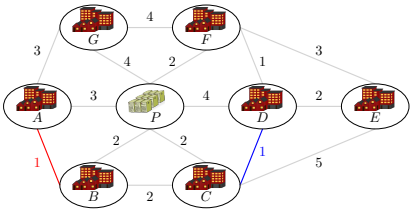
Der **Kruskal-Algorithmus** findet in jedem zusammenhängenden Graphen mit positiven Kantengewichten einen **minimal aufspannenden Baum**.

Dieser Baum verbindet also alle Knoten des Graphen so, dass die Gesamtkosten so klein wie möglich sind.

Wenn der Graph n Knoten enthält, dann benötigt der Algorithmus dafür $n-1$ Durchläufe.
In jedem Durchlauf werden zwei Komponenten durch Hinzufügen einer Kante verschmolzen:

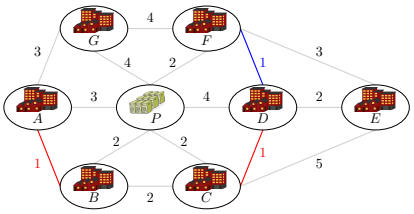
Durchlauf 1:
Komponenten: $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{F\}$, $\{G\}$, $\{P\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.

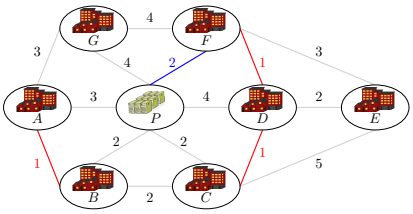




Durchlauf 2:
Komponenten: $\{A,B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{F\}$, $\{G\}$, $\{P\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.

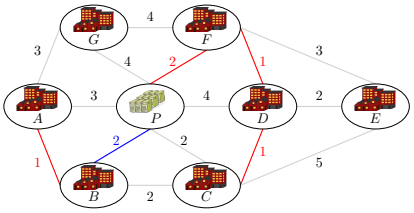
Durchlauf 3:
Komponenten: $\{A,B\}$, $\{C,D\}$, $\{E\}$, $\{F\}$, $\{G\}$, $\{P\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.

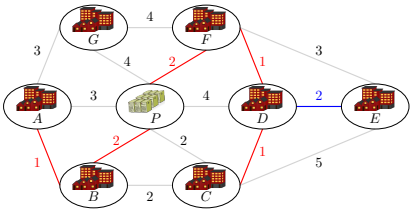




Durchlauf 4:
Komponenten: $\{A,B\}$, $\{C,D,F\}$, $\{E\}$, $\{G\}$, $\{P\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.

Durchlauf 5:
Komponenten: $\{A,B\}$, $\{C,D,F,P\}$, $\{E\}$, $\{G\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.

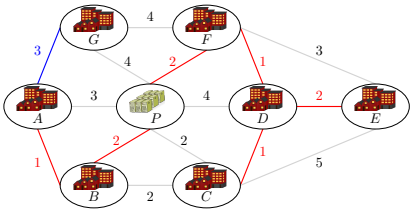




Durchlauf 6:
Komponenten: $\{A,B,C,D,F,P\}$, $\{E\}$, $\{G\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.

Durchlauf 7:
Komponenten: $\{A,B,C,D,E,F,P\}$, $\{G\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.

Beachte, dass die günstigeren Kanten keine Komponenten verbinden.



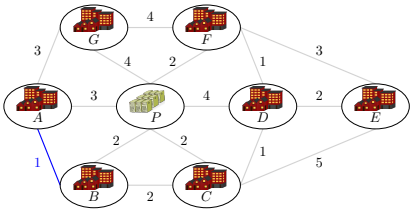
Der Kruskal-Algorithmus ist ein sogenannter **Greedy-Algorithmus**. „greedy“ ist Englisch für gierig.
Er trifft in jedem Durchlauf jene Entscheidung, die zum aktuellen Zeitpunkt am besten erscheint.

Kruskal-Algorithmus (1956)

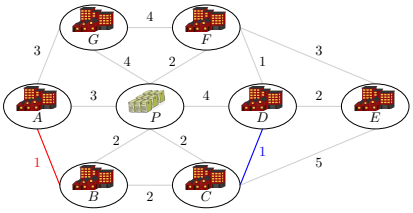
Der **Kruskal-Algorithmus** findet in jedem zusammenhängenden Graphen mit positiven Kantengewichten einen **minimal aufspannenden Baum**.

Dieser Baum verbindet also alle Knoten des Graphen so, dass die Gesamtkosten so klein wie möglich sind.

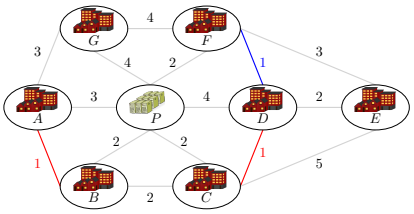
Wenn der Graph n Knoten enthält, dann benötigt der Algorithmus dafür $n-1$ Durchläufe.
In jedem Durchlauf werden zwei Komponenten durch Hinzufügen einer Kante verschmolzen:



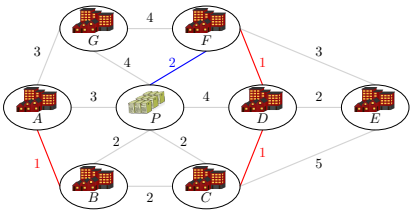
Durchlauf 1:
Komponenten: $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{F\}$, $\{G\}$, $\{P\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.



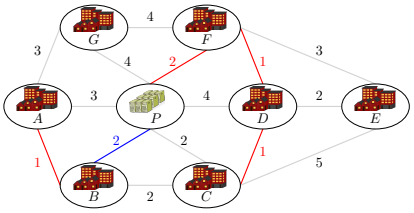
Durchlauf 2:
Komponenten: $\{A,B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{F\}$, $\{G\}$, $\{P\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.



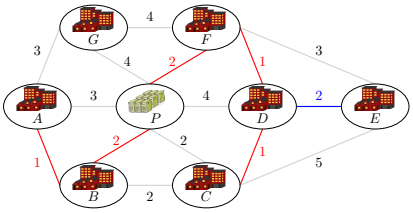
Durchlauf 3:
Komponenten: $\{A,B\}$, $\{C,D\}$, $\{E\}$, $\{F\}$, $\{G\}$, $\{P\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.



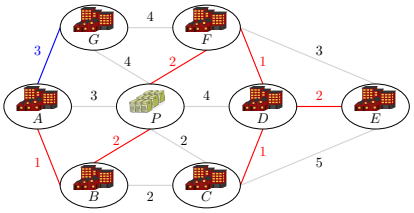
Durchlauf 4:
Komponenten: $\{A,B\}$, $\{C,D,F\}$, $\{E\}$, $\{G\}$, $\{P\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.



Durchlauf 5:
Komponenten: $\{A,B\}$, $\{C,D,F,P\}$, $\{E\}$, $\{G\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.



Durchlauf 6:
Komponenten: $\{A,B,C,D,F,P\}$, $\{E\}$, $\{G\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.



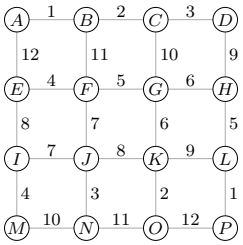
Durchlauf 7:
Komponenten: $\{A,B,C,D,E,F,P\}$, $\{G\}$
Wähle unter allen Kanten, die zwei Komponenten verbinden, eine günstigste Kante aus.

Beachte, dass die günstigeren Kanten keine Komponenten verbinden.

Der Kruskal-Algorithmus ist ein sogenannter **Greedy-Algorithmus**. „greedy“ ist Englisch für gierig.
Er trifft in jedem Durchlauf jene Entscheidung, die zum aktuellen Zeitpunkt am besten erscheint.

Kruskal-Algorithmus

Rechts ist ein zusammenhängender Graph mit positiven Kantengewichten dargestellt.
Ermittle einen minimal aufspannenden Baum mit dem Kruskal-Algorithmus.
Welches Gesamtgewicht hat dieser Baum?



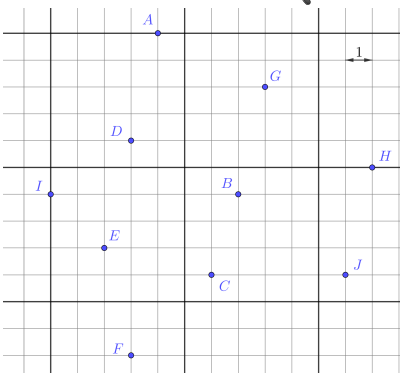
Euklidischer Abstand

Im quadratischen Raster rechts sind
10 Gitterpunkte eingezeichnet.

1) Trage die Entfernungen zwischen den Punkten in der Tabelle unten ein.

2) Die Entfernungen sind die Kantengewichte. Ermittle einen minimal aufspannden Baum, und zeichne ihn rechts ein.

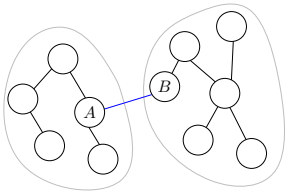
3) Welches Gesamtgewicht hat dieser Baum?



	A	B	C	D	E	F	G	H	I	J
A	–	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$
B	–	–	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$
C	–	–	–	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$
D	–	–	–	–	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$
E	–	–	–	–	–	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$
F	–	–	–	–	–	–	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$
G	–	–	–	–	–	–	–	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$
H	–	–	–	–	–	–	–	–	$\sqrt{}$	$\sqrt{}$
I	–	–	–	–	–	–	–	–	–	$\sqrt{}$
J	–	–	–	–	–	–	–	–	–	–

Warum liefert Kruskal einen aufspannenden Baum?

Ein Graph enthält keinen Kreis.
Die Knoten A und B sind in verschiedenen Komponenten.
Eine Kante zwischen A und B wird hinzugefügt.
Erkläre, warum der Graph dann *keinen* Kreis enthalten kann.



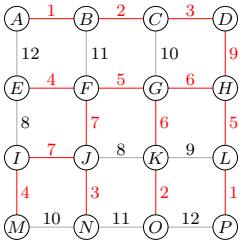
Der Kruskal-Algorithmus liefert also einen Graphen, der alle Knoten enthält, aber keinen Kreis.

Ein solcher Graph heißt **aufspannender Baum**.

Kruskal-Algorithmus

Rechts ist ein zusammenhängender Graph mit positiven Kantengewichten dargestellt.
Ermittle einen minimal aufspannenden Baum mit dem Kruskal-Algorithmus.
Welches Gesamtgewicht hat dieser Baum?

$$2 \cdot (1+2+3+4+5+6+7)+9 = 65$$



Euklidischer Abstand

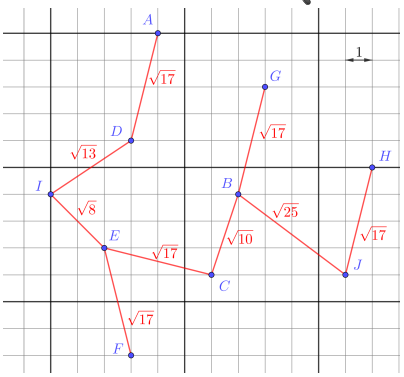
Im quadratischen Raster rechts sind
10 Gitterpunkte eingezeichnet.

1) Trage die Entfernungen zwischen den Punkten in der Tabelle unten ein.

2) Die Entfernungen sind die Kantengewichte. Ermittle einen minimal aufspannden Baum, und zeichne ihn rechts ein.

3) Welches Gesamtgewicht hat dieser Baum?

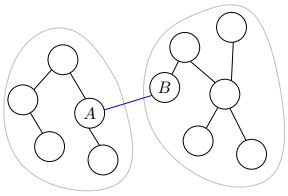
$$\sqrt{8}+\sqrt{10}+\sqrt{13}+5\cdot\sqrt{17}+\sqrt{25} = 35{,}21\ldots$$



	A	B	C	D	E	F	G	H	I	J
A	–	$\sqrt{45}$	$\sqrt{85}$	$\sqrt{17}$	$\sqrt{68}$	$\sqrt{145}$	$\sqrt{20}$	$\sqrt{89}$	$\sqrt{52}$	$\sqrt{130}$
B	–	–	$\sqrt{10}$	$\sqrt{20}$	$\sqrt{29}$	$\sqrt{52}$	$\sqrt{17}$	$\sqrt{26}$	$\sqrt{49}$	$\sqrt{25}$
C	–	–	–	$\sqrt{34}$	$\sqrt{17}$	$\sqrt{18}$	$\sqrt{53}$	$\sqrt{52}$	$\sqrt{45}$	$\sqrt{25}$
D	–	–	–	–	$\sqrt{17}$	$\sqrt{64}$	$\sqrt{29}$	$\sqrt{82}$	$\sqrt{13}$	$\sqrt{89}$
E	–	–	–	–	–	$\sqrt{17}$	$\sqrt{72}$	$\sqrt{109}$	$\sqrt{8}$	$\sqrt{82}$
F	–	–	–	–	–	–	$\sqrt{125}$	$\sqrt{130}$	$\sqrt{45}$	$\sqrt{73}$
G	–	–	–	–	–	–	–	$\sqrt{25}$	$\sqrt{80}$	$\sqrt{58}$
H	–	–	–	–	–	–	–	–	$\sqrt{145}$	$\sqrt{17}$
I	–	–	–	–	–	–	–	–	–	$\sqrt{130}$
J	–	–	–	–	–	–	–	–	–	–

Warum liefert Kruskal einen aufspannenden Baum?

Ein Graph enthält keinen Kreis.
Die Knoten A und B sind in verschiedenen Komponenten.
Eine Kante zwischen A und B wird hinzugefügt.
Erkläre, warum der Graph dann *keinen* Kreis enthalten kann.
Indirekte Begründung: Angenommen, der Graph enthält einen Kreis.
Dann gibt es aber auch ohne die neue Kante einen Weg von A zu B.
Die Knoten A und B sind aber in verschiedenen Komponenten. ↯



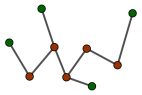
Der Kruskal-Algorithmus liefert also einen Graphen, der alle Knoten enthält, aber keinen Kreis.

Ein solcher Graph heißt **aufspannender Baum**.

Bäume haben Blätter.

Erkläre, warum es in jedem Baum mit mindestens 2 Knoten einen Knoten geben muss, von dem nur eine Kante wegführt.

Ein solcher Knoten heißt auch **Blatt**.



Kantenanzahl in Bäumen

Begründe mit vollständiger Induktion, warum jeder Baum mit $n \geq 2$ Knoten genau $n-1$ Kanten hat.

Warum liefert Kruskal einen *minimal* aufspannenden Baum?

Gegeben ist ein zusammenhängender Graph mit $n \geq 2$ Knoten und positiven Kantengewichten. Dann liefert der Kruskal-Algorithmus einen aufspannenden Baum mit minimalen Gesamtkosten. Wir geben dafür eine indirekte Begründung:
Angenommen, es gibt eine beste Lösung mit *weniger* Gesamtkosten als die Kruskal-Lösung.

1) Warum kann diese beste Lösung keinen Kreis enthalten?

2) Die beste Lösung muss also ein Baum sein, der _____ Kanten hat.

3) Fügt man einem Graphen eine Kante hinzu, wird die Anzahl der Komponenten höchstens um _____ kleiner. Damit ein Graph mit n Knoten und $n-1$ Kanten schließlich zusammenhängend ist, muss also *jede* Kante jeweils 2 Komponenten verschmelzen.

4) Der Kruskal-Algorithmus wählt $n-1$ Kanten mit aufsteigenden Gewichten $k_1 \leq k_2 \leq \cdots \leq k_{n-1}$. Wir sortieren die Kanten von der besten Lösung auch nach den Gewichten $b_1 \leq b_2 \leq \cdots \leq b_{n-1}$. Warum muss es einen Durchlauf d geben mit $k_d > b_d$?

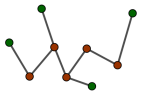
5) Behauptung: Die Kruskal-Lösung hat *vor* Durchlauf d höchstens so viele Komponenten wie die beste Lösung *nach* Durchlauf d. Das ist schließlich ein Widerspruch zu **3)**. ↯

Es kann also keine günstigere Lösung als die Kruskal-Lösung geben.

Wir begründen die Behauptung. Dazu nehmen wir die Knoten einer beliebigen Komponente der besten Lösung *nach* Durchlauf d. Erkläre, warum die Kruskal-Lösung bereits *vor* Durchlauf d eine Komponente haben muss, die alle diese Knoten enthält.

Bäume haben Blätter.

Erkläre, warum es in jedem Baum mit mindestens 2 Knoten einen Knoten geben muss, von dem nur eine Kante wegführt.



Ein solcher Knoten heißt auch **Blatt**.

Indirekte Begründung: Wenn von jedem Knoten mindestens 2 Kanten wegführen, dann könnte man ausgehend von einem Knoten immer wieder zu einem Nachbar weitergehen. Spätestens nach n Schritten landet man bei einem Knoten, den man schon besucht hat. Der Graph hat also einen Kreis und ist damit kein Baum. ↯

Kantenanzahl in Bäumen

Begründe mit vollständiger Induktion, warum jeder Baum mit $n \geq 2$ Knoten genau $n-1$ Kanten hat.

1) Induktionsanfang $n = 2$: •—• ✓

2) Induktionsschritt $n \to n+1$: Entferne vom Baum mit $n+1$ Knoten ein Blatt und die zugehörige Kante. Der neue Baum hat n Knoten. Aus der Induktionsvoraussetzung folgt, dass der neue Baum $n-1$ Kanten hat. Der ursprüngliche Baum hat also n Kanten. ✓

Warum liefert Kruskal einen *minimal* aufspannenden Baum?

Gegeben ist ein zusammenhängender Graph mit $n \geq 2$ Knoten und positiven Kantengewichten. Dann liefert der Kruskal-Algorithmus einen aufspannenden Baum mit minimalen Gesamtkosten.
Wir geben dafür eine indirekte Begründung:
Angenommen, es gibt eine beste Lösung mit *weniger* Gesamtkosten als die Kruskal-Lösung.

1) Warum kann diese beste Lösung keinen Kreis enthalten?
Wenn sie einen Kreis, enthält, könnte man eine beliebige Kante vom Kreis entfernen. Der neue Graph ist auch zusammenhängend und hat weniger Gesamtkosten.

2) Die beste Lösung muss also ein Baum sein, der $n-1$ Kanten hat.

3) Fügt man einem Graphen eine Kante hinzu, wird die Anzahl der Komponenten höchstens um 1 kleiner. Damit ein Graph mit n Knoten und $n-1$ Kanten schließlich zusammenhängend ist, muss also *jede* Kante jeweils 2 Komponenten verschmelzen.

4) Der Kruskal-Algorithmus wählt $n-1$ Kanten mit aufsteigenden Gewichten $k_1 \leq k_2 \leq \cdots \leq k_{n-1}$. Wir sortieren die Kanten von der besten Lösung auch nach den Gewichten $b_1 \leq b_2 \leq \cdots \leq b_{n-1}$. Warum muss es einen Durchlauf d geben mit $k_d > b_d$?
Laut Annahme hat die beste Lösung weniger Gesamtkosten als die Kruskal-Lösung.

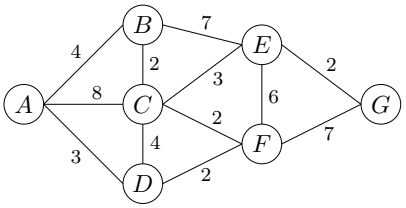
5) Behauptung: Die Kruskal-Lösung hat *vor* Durchlauf d höchstens so viele Komponenten wie die beste Lösung *nach* Durchlauf d. Das ist schließlich ein Widerspruch zu **3)**. ↯

Es kann also keine günstigere Lösung als die Kruskal-Lösung geben.

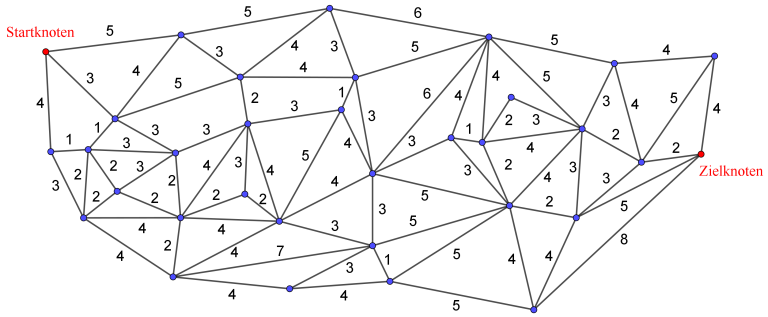
Wir begründen die Behauptung. Dazu nehmen wir die Knoten einer beliebigen Komponente der besten Lösung *nach* Durchlauf d. Erkläre, warum die Kruskal-Lösung bereits *vor* Durchlauf d eine Komponente haben muss, die alle diese Knoten enthält.
Für jede Kante dieser beliebigen Komponente gilt:
Die Kosten sind kleiner als k_d, aber Kruskal wählt sie in Durchlauf d nicht aus.
Die beiden Knoten müssen also schon vor Durchlauf d in der gleichen Kruskal-Komponente liegen.
Alle Knoten dieser beliebigen Komponente sind also in einer Kruskal-Komponente enthalten.

3.4 Dijkstra-Algorithmus

„Der kürzeste Weg von A nach B hat die Länge 4.
Der kürzeste Weg von A nach C hat die Länge 6
und führt über B.
Wie lang ist der kürzeste Weg von A nach G?“



Durch geschicktes Probieren können wir in diesem kleinen Beispiel schließlich einen Weg mit Länge 11 von A nach G finden. Aber gibt es *sicher* keinen kürzeren Weg?
Und wie können wir bei größeren Graphen systematisch vorgehen?



Auch dieses „shortest path problem“ zeichnet sich nicht nur durch eine niedrige Einstiegshürde, sondern auch eine hohe Praxisrelevanz zum Beispiel in der Routenplanung aus. Mögliche Einstiege in den Unterricht und Übungsbeispiele befinden sich in [3, 14, 19].

Edsger W. Dijkstra veröffentlichte im Jahr 1959 einen Algorithmus [9], der stets einen kürzesten Weg findet. Dieser nach ihm benannte Algorithmus ist am folgenden Arbeitsblatt aufbereitet.

Es folgt das Arbeitsblatt – Dijkstra-Algorithmus und die Ausarbeitung.
Die aktuelle Version des Arbeitsblatts ist auf der Mathematik macht Freu(n)de-Website unter https://mmf.univie.ac.at/materialien verfügbar.

Benötigtes Vorwissen:

– Arbeitsblatt – Einführung in Algorithmen

Lernziele:

✓ Was ist ein **kantengewichteter Graph**?

✓ Gegeben ist ein zusammenhängender kantengewichteter Graph und ein Startknoten. Wie findet der **Dijkstra-Algorithmus** den jeweils schnellsten Weg zu allen anderen Knoten?

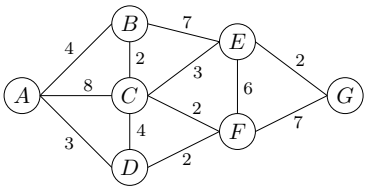
Shortest path problem

Dein Routenplaner soll den schnellsten Weg von einem Ort A zu einem anderen Ort G ermitteln.
Die Verbindungen benachbarter Orte sind rechts in einem **kantengewichteten Graphen** dargestellt:
In der Graphentheorie nennen wir ...

... die Orte A, B, C, D, E, F, G auch **Knoten**.

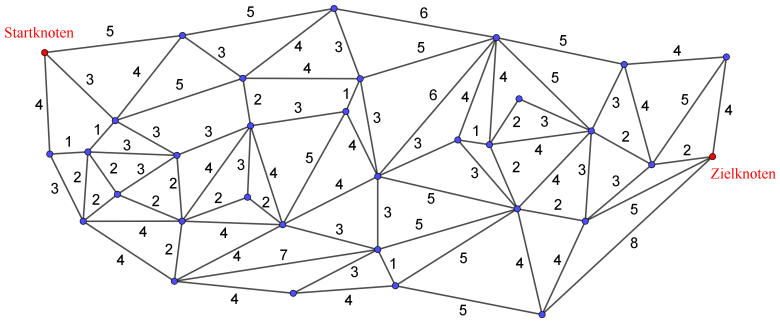
... die Strecken zwischen 2 Knoten auch **Kanten**.

... die Zahlen neben den Kanten auch **Gewichte**.



Die Gewichte geben in diesem Kontext die Fahrzeit zwischen den Orten an. Die Gewichte sind also positiv.
Ein möglicher Weg von A nach G ist $A \xrightarrow{3} D \xrightarrow{2} F \xrightarrow{7} G$ mit Fahrzeit $3+2+7=12$.

Findest du einen schnelleren Weg? Hast du *sicher* den schnellsten Weg gefunden?



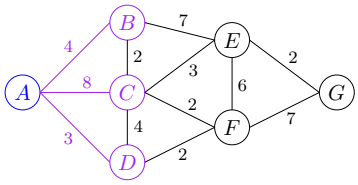
In der Praxis sind die Graphen deutlich größer.

Gesucht ist ein Verfahren, mit dem wir effizient einen kürzesten Weg finden.

Dijkstra-Algorithmus (1959)

Der **Dijkstra-Algorithmus** findet in jedem kantengewichteten Graphen mit positiven Kantengewichten ausgehend von einem Startknoten den jeweils **schnellsten Weg** zu allen anderen Knoten.
Wenn der Graph n Knoten enthält, dann benötigt der Algorithmus dafür n Durchläufe.
In jedem Durchlauf finden wir den schnellsten Weg vom Startknoten zu einem neuen Knoten:

1) In Durchlauf 1 finden wir den schnellsten Weg von A nach A. Alle Kantengewichte sind positiv.
Die Fahrzeit 0 speichern wir in der folgenden Tabelle ab.
Außerdem tragen wir die jeweiligen Fahrzeiten zu den benachbarten Knoten B, C und D ein.



	A	B	C	D	E	F	G
Durchlauf 1	0	4	8	3	∞	∞	∞

Zu den anderen Knoten haben wir noch keinen Weg gefunden.
Die Fahrzeit ist also noch „unendlich“ (∞) groß.

2) Dann suchen wir in der Tabelle unter den verbleibenden Knoten B, C, ..., G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 1 ist das Knoten D mit Fahrzeit 3.

Kürzester Weg

Warum kann kein Weg ausgehend von A über B oder C mit kleinerer Fahrzeit als 3 nach D führen?

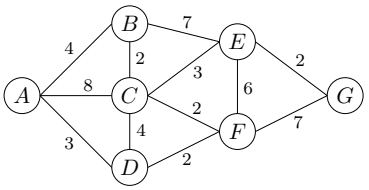
Shortest path problem

Dein Routenplaner soll den schnellsten Weg von einem Ort A zu einem anderen Ort G ermitteln.
Die Verbindungen benachbarter Orte sind rechts in einem **kantengewichteten Graphen** dargestellt:
In der Graphentheorie nennen wir ...

... die Orte A, B, C, D, E, F, G auch **Knoten**.

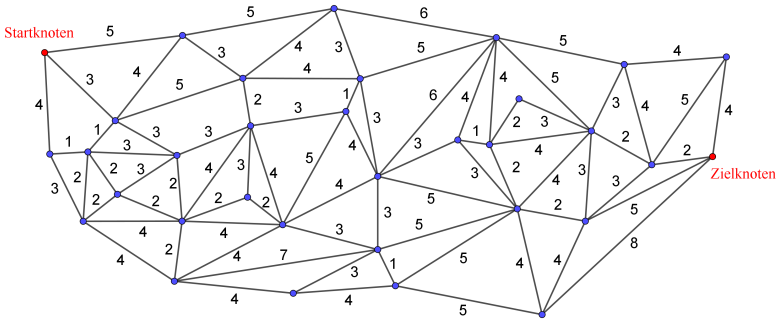
... die Strecken zwischen 2 Knoten auch **Kanten**.

... die Zahlen neben den Kanten auch **Gewichte**.



Die Gewichte geben in diesem Kontext die Fahrzeit zwischen den Orten an. Die Gewichte sind also positiv.
Ein möglicher Weg von A nach G ist $A \xrightarrow{3} D \xrightarrow{2} F \xrightarrow{7} G$ mit Fahrzeit $3+2+7=12$.

Findest du einen schnelleren Weg? Hast du *sicher* den schnellsten Weg gefunden?



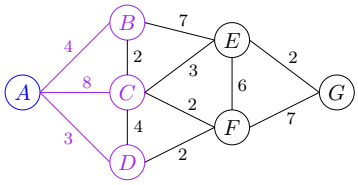
In der Praxis sind die Graphen deutlich größer.

Gesucht ist ein Verfahren, mit dem wir effizient einen kürzesten Weg finden.

Dijkstra-Algorithmus (1959)

Der **Dijkstra-Algorithmus** findet in jedem kantengewichteten Graphen mit positiven Kantengewichten ausgehend von einem Startknoten den jeweils **schnellsten Weg** zu allen anderen Knoten.
Wenn der Graph n Knoten enthält, dann benötigt der Algorithmus dafür n Durchläufe.
In jedem Durchlauf finden wir den schnellsten Weg vom Startknoten zu einem neuen Knoten:

1) In Durchlauf 1 finden wir den schnellsten Weg von A nach A. Alle Kantengewichte sind positiv.
Die Fahrzeit 0 speichern wir in der folgenden Tabelle ab.
Außerdem tragen wir die jeweiligen Fahrzeiten zu den benachbarten Knoten B, C und D ein.



	A	B	C	D	E	F	G
Durchlauf 1	0	4	8	3	∞	∞	∞

Zu den anderen Knoten haben wir noch keinen Weg gefunden.
Die Fahrzeit ist also noch „unendlich“ (∞) groß.

2) Dann suchen wir in der Tabelle unter den verbleibenden Knoten B, C, ..., G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 1 ist das Knoten D mit Fahrzeit 3.

Kürzester Weg

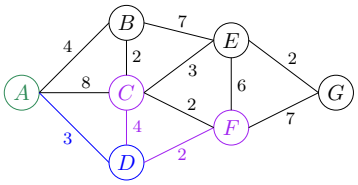
Warum kann kein Weg ausgehend von A über B oder C mit kleinerer Fahrzeit als 3 nach D führen?

Die Kantengewichte sind positiv.
Jeder Weg $A \xrightarrow{4} B \longrightarrow \cdots \longrightarrow D$ hat also mehr als Fahrzeit 4.
Jeder Weg $A \xrightarrow{8} C \longrightarrow \cdots \longrightarrow D$ hat also mehr als Fahrzeit 8.

Dijkstra-Algorithmus – Durchlauf 2

2) In Durchlauf 2 finden wir also den schnellsten Weg von A nach D. Er hat Fahrzeit 3.
Dann prüfen wir, ob es über D einen schnelleren Weg zu seinen benachbarten Knoten gibt:

C : $3 + 4 = 7 < 8$ ✓ F : $3 + 2 = 5 < \infty$ ✓ Zu A haben wir bereits den schnellsten Weg gefunden.



Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 1	0	4	8	3	∞	∞	∞
Durchlauf 2	0	4	7	3	∞	5	∞

3) Dann suchen wir in der Tabelle unter den verbleibenden Knoten B, C, E, F und G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 2 ist das Knoten B mit Fahrzeit 4.

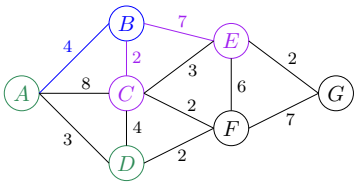
Kürzester Weg

Warum kann kein Weg ausgehend von A mit kleinerer Fahrzeit als 4 nach B führen?

Dijkstra-Algorithmus – Durchlauf 3

3) In Durchlauf 3 finden wir also den schnellsten Weg von A nach B. Er hat Fahrzeit 4.
Dann prüfen wir, ob es über B einen schnelleren Weg zu seinen benachbarten Knoten gibt:

C : $4 + 2 = 6 < 7$ ✓ E : $4 + 7 = 11 < \infty$ ✓ Zu A haben wir bereits den schnellsten Weg gefunden.



Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 2	0	4	7	3	∞	5	∞
Durchlauf 3	0	4	6	3	11	5	∞

4) Dann suchen wir in der Tabelle unter den verbleibenden Knoten C, E, F und G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 3 ist das Knoten F mit Fahrzeit 5.

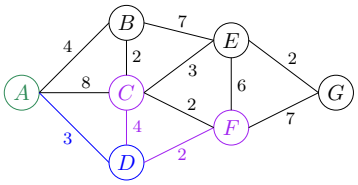
Kürzester Weg

Warum kann kein Weg ausgehend von A mit kleinerer Fahrzeit als 5 nach F führen?

Dijkstra-Algorithmus – Durchlauf 2

2) In Durchlauf 2 finden wir also den schnellsten Weg von A nach D. Er hat Fahrzeit 3.
Dann prüfen wir, ob es über D einen schnelleren Weg zu seinen benachbarten Knoten gibt:

$C: 3 + 4 = 7 < 8 \checkmark$ $F: 3 + 2 = 5 < \infty \checkmark$ Zu A haben wir bereits den schnellsten Weg gefunden.



Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 1	0	4	8	3	∞	∞	∞
Durchlauf 2	0	4	7	3	∞	5	∞

3) Dann suchen wir in der Tabelle unter den verbleibenden Knoten B, C, E, F und G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 2 ist das Knoten B mit Fahrzeit 4.

Kürzester Weg

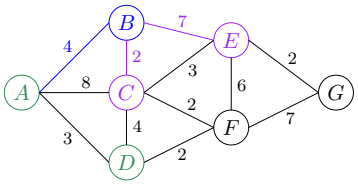
Warum kann kein Weg ausgehend von A mit kleinerer Fahrzeit als 4 nach B führen?

Zu den Knoten $\{A, D\}$ haben wir bereits den kürzesten Weg gefunden.
Angenommen, es gibt einen Weg mit kleinerer Fahrzeit als 4 nach B.
Suche den letzten Knoten aus $\{A, D\}$ auf diesem Weg. A ist sicher am Weg enthalten.
Die nächste Kante auf diesem Weg haben wir aber bereits untersucht. Der Weg hat mit dieser Kante schon mindestens Fahrzeit 4. Da die Gewichte positiv sind, kann der Weg nicht kürzer sein. ↯

Dijkstra-Algorithmus – Durchlauf 3

3) In Durchlauf 3 finden wir also den schnellsten Weg von A nach B. Er hat Fahrzeit 4.
Dann prüfen wir, ob es über B einen schnelleren Weg zu seinen benachbarten Knoten gibt:

$C: 4 + 2 = 6 < 7 \checkmark$ $E: 4 + 7 = 11 < \infty \checkmark$ Zu A haben wir bereits den schnellsten Weg gefunden.



Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 2	0	4	7	3	∞	5	∞
Durchlauf 3	0	4	6	3	11	5	∞

4) Dann suchen wir in der Tabelle unter den verbleibenden Knoten C, E, F und G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 3 ist das Knoten F mit Fahrzeit 5.

Kürzester Weg

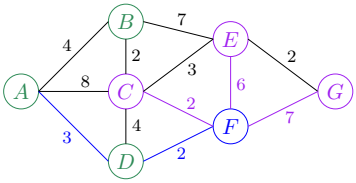
Warum kann kein Weg ausgehend von A mit kleinerer Fahrzeit als 5 nach F führen?

Zu den Knoten $\{A, B, D\}$ haben wir bereits den kürzesten Weg gefunden.
Angenommen, es gibt einen Weg mit kleinerer Fahrzeit als 5 nach F.
Suche den letzten Knoten aus $\{A, B, D\}$ auf diesem Weg. A ist sicher am Weg enthalten.
Die nächste Kante auf diesem Weg haben wir aber bereits untersucht. Der Weg hat mit dieser Kante schon mindestens Fahrzeit 5. Da die Gewichte positiv sind, kann der Weg nicht kürzer sein. ↯

4) In Durchlauf 4 finden wir also den schnellsten Weg von A nach F. Er hat Fahrzeit 5.
Dann prüfen wir, ob es über F einen schnelleren Weg zu seinen benachbarten Knoten gibt:

$C: 5 + 2 = 7 \geq 6$ ✗ $\quad E: 5 + 6 = 11 \geq 11$ ✗ $\quad G: 5 + 7 = 12 < \infty$ ✓

Zu D haben wir bereits den schnellsten Weg gefunden.



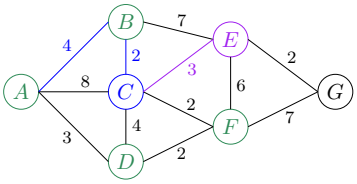
Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 3	0	4	6	3	11	5	∞
Durchlauf 4	0	4	6	3	11	5	12

5) Dann suchen wir in der Tabelle unter den verbleibenden Knoten C, E und G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 4 ist das Knoten C mit Fahrzeit 6.
In Durchlauf 5 finden wir also den schnellsten Weg von A nach C. Er hat Fahrzeit 6.
Dann prüfen wir, ob es über C einen schnelleren Weg zu seinen benachbarten Knoten gibt:

$E: 6 + 3 = 9 < 11$ ✓

Zu A, B, D und F haben wir bereits den schnellsten Weg gefunden.



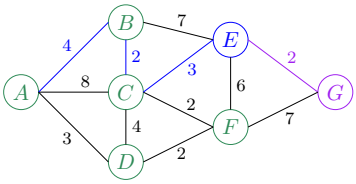
Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 4	0	4	6	3	11	5	12
Durchlauf 5	0	4	6	3	9	5	12

6) Dann suchen wir in der Tabelle unter den verbleibenden Knoten E und G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 5 ist das Knoten E mit Fahrzeit 9.
In Durchlauf 6 finden wir also den schnellsten Weg von A nach E. Er hat Fahrzeit 9.
Dann prüfen wir, ob es über E einen schnelleren Weg zu seinen benachbarten Knoten gibt:

$G: 9 + 2 = 11 < 12$ ✓

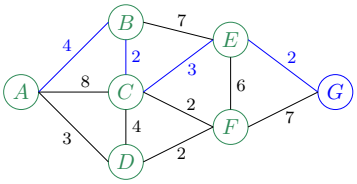
Zu B, C und F haben wir bereits den schnellsten Weg gefunden.



Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 5	0	4	6	3	9	5	12
Durchlauf 6	0	4	6	3	9	5	11

7) Schließlich finden wir in Durchlauf 7 den schnellsten Weg von A nach G mit Fahrzeit 11:



	A	B	C	D	E	F	G
Durchlauf 6	0	4	6	3	9	5	11
Durchlauf 7	0	4	6	3	9	5	11

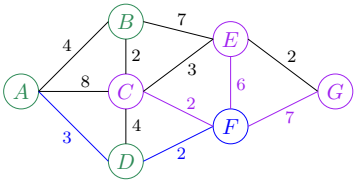
Der schnellste Weg von A nach G ist also $A \xrightarrow{4} B \xrightarrow{2} C \xrightarrow{3} E \xrightarrow{2} G$ mit Fahrzeit 11.

Der Dijkstra-Algorithmus findet vom Startknoten den schnellsten Weg zu *jedem* anderen Knoten.
Wenn nur der schnellste Weg zu einem bestimmten Knoten gesucht ist, kann man auch nach dem entsprechenden Durchlauf abbrechen.

4) In Durchlauf 4 finden wir also den schnellsten Weg von A nach F. Er hat Fahrzeit 5.
Dann prüfen wir, ob es über F einen schnelleren Weg zu seinen benachbarten Knoten gibt:

$C: 5 + 2 = 7 \geq 6$ ✗ $\quad E: 5 + 6 = 11 \geq 11$ ✗ $\quad G: 5 + 7 = 12 < \infty$ ✓

Zu D haben wir bereits den schnellsten Weg gefunden.



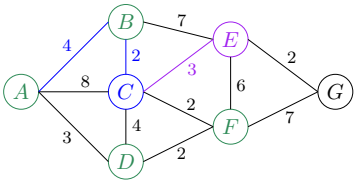
Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 3	0	4	6	3	11	5	∞
Durchlauf 4	0	4	6	3	11	5	12

5) Dann suchen wir in der Tabelle unter den verbleibenden Knoten C, E und G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 4 ist das Knoten C mit Fahrzeit 6.
In Durchlauf 5 finden wir also den schnellsten Weg von A nach C. Er hat Fahrzeit 6.
Dann prüfen wir, ob es über C einen schnelleren Weg zu seinen benachbarten Knoten gibt:

$E: 6 + 3 = 9 < 11$ ✓

Zu A, B, D und F haben wir bereits den schnellsten Weg gefunden.



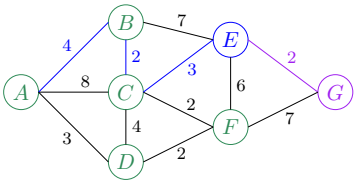
Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 4	0	4	6	3	11	5	12
Durchlauf 5	0	4	6	3	9	5	12

6) Dann suchen wir in der Tabelle unter den verbleibenden Knoten E und G jenen Knoten mit der *kleinsten* Fahrzeit. Nach Durchlauf 5 ist das Knoten E mit Fahrzeit 9.
In Durchlauf 6 finden wir also den schnellsten Weg von A nach E. Er hat Fahrzeit 9.
Dann prüfen wir, ob es über E einen schnelleren Weg zu seinen benachbarten Knoten gibt:

$G: 9 + 2 = 11 < 12$ ✓

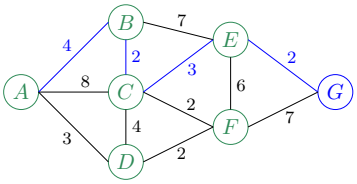
Zu B, C und F haben wir bereits den schnellsten Weg gefunden.



Die Ergebnisse speichern wir in der Tabelle ab:

	A	B	C	D	E	F	G
Durchlauf 5	0	4	6	3	9	5	12
Durchlauf 6	0	4	6	3	9	5	11

7) Schließlich finden wir in Durchlauf 7 den schnellsten Weg von A nach G mit Fahrzeit 11:



	A	B	C	D	E	F	G
Durchlauf 6	0	4	6	3	9	5	11
Durchlauf 7	0	4	6	3	9	5	11

Der schnellste Weg von A nach G ist also $A \xrightarrow{4} B \xrightarrow{2} C \xrightarrow{3} E \xrightarrow{2} G$ mit Fahrzeit 11.

Der Dijkstra-Algorithmus findet vom Startknoten den schnellsten Weg zu *jedem* anderen Knoten.
Wenn nur der schnellste Weg zu einem bestimmten Knoten gesucht ist, kann man auch nach dem entsprechenden Durchlauf abbrechen.

Rückverfolgung

In der folgenden Tabelle ist das Ergebnis des Dijkstra-Algorithmus vollständig dargestellt.
In jeder Zeile ist in blauer Farbe markiert, zu welchem Knoten der kürzeste Weg gefunden werde.
Wir können nur mit Hilfe der Tabelle den schnellsten Weg von A nach G rückverfolgen:

	A	B	C	D	E	F	G
Durchlauf 1	0	4	8	3	∞	∞	∞
Durchlauf 2	0	4	7	3	∞	5	∞
Durchlauf 3	0	4	6	3	11	5	∞
Durchlauf 4	0	4	6	3	11	5	12
Durchlauf 5	0	4	6	3	9	5	12
Durchlauf 6	0	4	6	3	9	5	11
Durchlauf 7	0	4	6	3	9	5	11

In Spalte G suchen wir das Update auf 11.
Nach G sind wir also von ____ gekommen.

In Spalte E suchen wir das Update auf 9.
Nach E sind wir also von ____ gekommen.

In Spalte C suchen wir das auf Update 6.
Nach C sind wir also von ____ gekommen.

In Spalte B suchen wir das auf Update 4.
Nach B sind wir also von ____ gekommen.

Der schnellste Weg von A nach G ist also $A \xrightarrow{4} B \xrightarrow{2} C \xrightarrow{3} E \xrightarrow{2} G$ mit Fahrzeit 11.

Vorgänger merken

Bei großen Graphen können wir Speicherplatz sparen, indem wir nicht die komplette Tabelle speichern.
Stattdessen merken wir uns bei jedem Update, von welchem Knoten der neue schnellere Weg kommt:

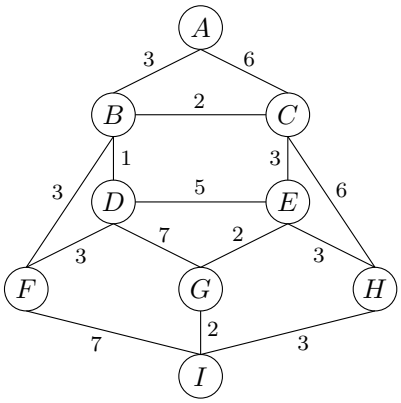
	A	B	C	D	E	F	G
Durchlauf 1	0	4	8	3	∞	∞	∞
Durchlauf 2	0	4	7	3	∞	5	∞
Durchlauf 3	0	4	6	3	11	5	∞
Durchlauf 4	0	4	6	3	11	5	12
Durchlauf 5	0	4	6	3	9	5	12
Durchlauf 6	0	4	6	3	9	5	11
Durchlauf 7	0	4	6	3	9	5	11

Vorgänger	A	B	C	D	E	F	G
Durchlauf 1	–	A	A	A	–	–	–
Durchlauf 2	–	A	D	A	–	D	–
Durchlauf 3	–	A	B	A	B	D	–
Durchlauf 4	–	A	B	A	B	D	F
Durchlauf 5	–	A	B	A	C	D	F
Durchlauf 6	–	A	B	A	C	D	E
Durchlauf 7	–	A	B	A	C	D	E

Erkläre, wie du nur mithilfe der letzten Zeile den schnellsten Weg von A nach G rückverfolgen kannst:

Dijkstra-Algorithmus

Ermittle mit dem Dijkstra-Algorithmus den schnellsten Weg von A nach I.



	A	B	C	D	E	F	G	H	I
Durchlauf 1									
Durchlauf 2									
Durchlauf 3									
Durchlauf 4									
Durchlauf 5									
Durchlauf 6									
Durchlauf 7									
Durchlauf 8									
Durchlauf 9									

Der schnellste Weg von A nach I ist also _____ mit Fahrzeit ____ .

Rückverfolgung

In der folgenden Tabelle ist das Ergebnis des Dijkstra-Algorithmus vollständig dargestellt.
In jeder Zeile ist in blauer Farbe markiert, zu welchem Knoten der kürzeste Weg gefunden werde.
Wir können nur mit Hilfe der Tabelle den schnellsten Weg von A nach G rückverfolgen:

	A	B	C	D	E	F	G
Durchlauf 1	0	4	8	3	∞	∞	∞
Durchlauf 2	0	4	7	3	∞	5	∞
Durchlauf 3	0	4	6	3	11	5	∞
Durchlauf 4	0	4	6	3	11	5	12
Durchlauf 5	0	4	6	3	9	5	12
Durchlauf 6	0	4	6	3	9	5	11
Durchlauf 7	0	4	6	3	9	5	11

In Spalte G suchen wir das Update auf 11.
Nach G sind wir also von E gekommen.

In Spalte E suchen wir das Update auf 9.
Nach E sind wir also von C gekommen.

In Spalte C suchen wir das auf Update 6.
Nach C sind wir also von B gekommen.

In Spalte B suchen wir das auf Update 4.
Nach B sind wir also von A gekommen.

Der schnellste Weg von A nach G ist also $A \xrightarrow{4} B \xrightarrow{2} C \xrightarrow{3} E \xrightarrow{2} G$ mit Fahrzeit 11.

Vorgänger merken

Bei großen Graphen können wir Speicherplatz sparen, indem wir nicht die komplette Tabelle speichern.
Stattdessen merken wir uns bei jedem Update, von welchem Knoten der neue schnellere Weg kommt:

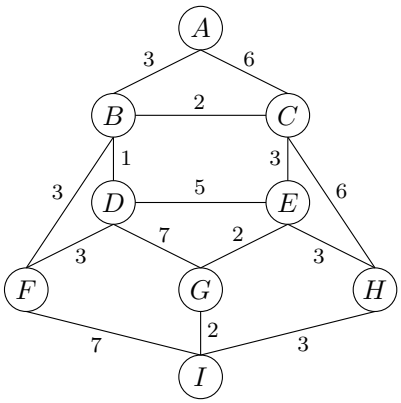
	A	B	C	D	E	F	G
Durchlauf 1	0	4	8	3	∞	∞	∞
Durchlauf 2	0	4	7	3	∞	5	∞
Durchlauf 3	0	4	6	3	11	5	∞
Durchlauf 4	0	4	6	3	11	5	12
Durchlauf 5	0	4	6	3	9	5	12
Durchlauf 6	0	4	6	3	9	5	11
Durchlauf 7	0	4	6	3	9	5	11

Vorgänger	A	B	C	D	E	F	G
Durchlauf 1	–	A	A	A	–	–	–
Durchlauf 2	–	A	D	A	–	D	–
Durchlauf 3	–	A	B	A	B	D	–
Durchlauf 4	–	A	B	A	B	D	F
Durchlauf 5	–	A	B	A	C	D	F
Durchlauf 6	–	A	B	A	C	D	E
Durchlauf 7	–	A	B	A	C	D	E

Erkläre, wie du nur mithilfe der letzten Zeile den schnellsten Weg von A nach G rückverfolgen kannst:
G hat den Vorgänger E. E hat den Vorgänger C. C hat den Vorgänger B. B hat den Vorgänger A.
Schnellster Weg: $A \longrightarrow B \longrightarrow C \longrightarrow E \longrightarrow G$

Dijkstra-Algorithmus

Ermittle mit dem Dijkstra-Algorithmus den schnellsten Weg von A nach I.



	A	B	C	D	E	F	G	H	I
Durchlauf 1	0	3	6	∞	∞	∞	∞	∞	∞
Durchlauf 2	0	3	5	4	∞	6	∞	∞	∞
Durchlauf 3	0	3	5	4	9	6	11	∞	∞
Durchlauf 4	0	3	5	4	8	6	11	11	∞
Durchlauf 5	0	3	5	4	8	6	11	11	13
Durchlauf 6	0	3	5	4	8	6	10	10	13
Durchlauf 7	0	3	5	4	8	6	10	10	12
Durchlauf 8	0	3	5	4	8	6	10	10	12
Durchlauf 9	0	3	5	4	8	6	10	10	12

Der schnellste Weg von A nach I ist also $A \xrightarrow{3} B \xrightarrow{2} C \xrightarrow{3} E \xrightarrow{2} G \xrightarrow{2} I$ mit Fahrzeit 12.

4 Ausblick

Seit 2016 werden im Rahmen des Projekts „Mathematik macht Freu(n)de“ Unterrichtsmaterialien entwickelt und unter https://mmf.univie.ac.at/materialien zur Verfügung gestellt. Bisher lag dabei der Fokus auf jenen Themen, die hohe Relevanz für die österreichischen zentralen Reifeprüfungen in Mathematik haben ([24], [32]). Die im Zuge dieser Diplomarbeit erstellten Materialien greifen nun auch Themen auf, die zwar schließlich nicht Stoff der Reifeprüfung sind, aber jedenfalls mit interessierten SchülerInnen beispielsweise in einem Wahlpflichtfach erarbeitet werden können.
In diesem Zusammenhang bieten sich auch zahlreiche andere Themen für weitere Materialien an, wie zum Beispiel:

- Aussagenlogik & Indirekte Beweise
  - Es gibt unendlich viele Primzahlen.
  - $\sqrt{2}$ ist eine irrationale Zahl.
- Vollständige Induktion in der Geometrie
  - Winkelsumme im n-Eck
  - Satz von Pick
  - Eulerscher Polyedersatz
- Algorithmen
  - Travelling Salesman Problem
  - Monte-Carlo-Simulationen
  - Gaußsches Eliminationsverfahren
  - Türme von Hanoi
- Mengenlehre & Zahlentheorie
  - Die Zahlenmengen $\mathbb{N}$, $\mathbb{Z}$ und $\mathbb{Q}$ sind gleich mächtig.
  - Die Zahlenmengen $\mathbb{N}$ und $\mathbb{R}$ sind *nicht* gleich mächtig.
  - Chinesischer Restsatz

Die in dieser Diplomarbeit vorgestellten Materialien befinden sich alle in ihrer ersten Version. Unter https://mmf.univie.ac.at/materialien werden in Zukunft die weiterentwickelten Versionen kostenlos zur Verfügung gestellt. Wir freuen uns über Anregungen, Feedback und Wünsche an mmf@univie.ac.at.

Literaturverzeichnis

[1] Fabio Acerbi. Plato: Parmenides 149a7-c3. a proof by complete induction? *Springer Verlag*, 2000.

[2] Ronald L. Rivest, Adi Shamir, Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[3] Armin P. Barth. *Algorithmik für Einsteiger – Für Studierende, Lehrer und Schüler in den Fächern Mathematik und Informatik (2. Auflage)*. Springer Spektrum, 2013.

[4] David M. Bressoud. *Factorization and Primality Testing*. Springer-Verlag, 1989.

[5] Peter Bundschuh. *Einführung in die Zahlentheorie*. Springer-Verlag, 2008.

[6] David Burghes. The use of discrete mathematics in the teaching of mathematics. *International Journal of Mathematical Education in Science and Technology*, 16:5:651–664, 1985.

[7] Bob Burn. *Fermat's Little Theorem: Proofs That Fermat Might Have Used*. The Mathematical Gazette, 2002.

[8] Georg Cantor. *Über eine elementare Frage der Mannigfaltigkeitslehre*. Teubner, 1892.

[9] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, 1959.

[10] Kathleen A. Dolgos. Discrete mathematics in the high school curriculum. *International Journal of Mathematical Education in Science and Technology*, 21:3:439–442, 1990.

[11] Joachim Engel. Die NCTM-Standards – Anstöße für den Mathematikunterricht nach TIMSS. *Zentralblatt für Didaktik der Mathematik*, 32:3:71–76, 2000.

[12] Leonhard Euler. *Theoremata arithmetica nova methodo demonstrata*. Novi Commentarii academiae scientiarum Petropolitanae 8, 1763.

[13] Python Software Foundation. Python language reference, version 3.8.0. http://www.python.org. Zugriff: 04.12.2019.

[14] Jens Gallenbacher. *Abenteuer Informatik – IT zum Anfassen – von Routenplaner bis Online-Banking (3. Auflage)*. Springer Spektrum, 2012.

[15] Carl F. Gauß. *Disquisitiones Arithmeticae*. Gerhard Fleischer, Leipzig, 1801.

[16] Clay Mathematics Institute. https://www.claymath.org/millennium-problems/p-vs-np-problem. Zugriff: 01.12.2019.

[17] Donald E. Knuth. *The art of computer programming 3: Sorting and searching*. Addison-Wesley, 1973.

[18] Joseph Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society, 7*, 1956.

[19] Stephan Hußmann, Brigitte Lutz-Westphal. *Kombinatorische Optimierung erleben – In Studium und Unterricht*. Vieweg & Sohn Verlag, 2007.

[20] Brian Harvey, Jens Mönig. Snap! reference manual. http://snap.berkeley.edu/SnapManual.pdf. Zugriff: 04.12.2019.

[21] National Council of Teachers of Mathematics. *Principles and Standards for School Mathematics*. 2000.

[22] Reinhard Oldenburg. *Mathematische Algorithmen im Unterricht*. Vieweg + Teubner, 2011.

[23] Blaise Pascal. *Traité du triangle arithmétique*. Chez Gvillavme Desprer, 1665.

[24] Lukas Riegler. *Mathematik macht Freu(n)de – Kompetenzmaterialien*. Schriftenreihe zur Didaktik der Mathematik der Österreichischen Mathematischen Gesellschaft (ÖMG) – Heft Nr. 51, 2018.

[25] Valerie A. DeBellis, Joseph G. Rosenstein. Discrete Mathematics in Primary and Secondary Schools in the United States. *ZDM*, 36:4:46–55, 2004.

[26] Valerie A. DeBellis, Joseph G. Rosenstein. Discrete Mathematics in the Schools: Experiences from the USA. *Mathematics in School*, 37:2:2–4, 2008.

[27] GIMPS – Great Internet Mersenne Prime Search. https://www.mersenne.org/primes. Zugriff: 11.11.2019.

[28] Kenneth H. Rosen, John G. Michaels, Jonathan L. Gross, Jerrold W. Grossman, Douglas R. Shier. *Handbook of Discrete and Combinatorial Mathematics*. CRC Press, 1999.

[29] Österreichisches Bundesrecht. *Lehrpläne – Allgemeinbildende Höhere Schulen*. RIS, 2019. https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10008568, Zugriff: 01.12.2019.

[30] Österreichisches Bundesrecht. *Lehrpläne – Neue Mittelschulen*. RIS, 2019. https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20007850, Zugriff: 01.12.2019.

[31] Österreichisches Bundesrecht. *Lehrpläne der Höheren technischen und gewerblichen Lehranstalten*. RIS, 2019. https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20009288, Zugriff: 01.12.2019.

[32] SRDP – Standardisierte Reife und Diplomprüfung. https://www.srdp.at. Zugriff: 01.12.2019.

[33] Ian Anderson, Bram van Asch, Jack van Lint. Discrete mathematics in the high school curriculum. *ZDM*, 36:3:105–116, 2004.

[34] F. W. Clarke, W. N. Everitt, L. L. Littlejohn, S. J. R. Vorster. H. J. S. Smith and the Fermat Two Squares Theorem. *The American Mathematical Monthly*, 106:7:652–665, 1999.

[35] Alexander Karp, Nicholas Wasserman. *Mathematics in middle and secondary school: A problem solving approach*. IAP, 2015.

[36] Euclid (Translated with introduction and commentary by Sir Thomas L. Heath). *The Thirteen Books of Euclid's Elements, Vol.* 1 *(Books* I *and* II*)*. Dover Publications Inc., 2000.

[37] Euclid (Translated with introduction and commentary by Sir Thomas L. Heath). *The Thirteen Books of Euclid's Elements, Vol.* 2 *(Books* III-IX*)*. Dover Publications Inc., 2000.

[38] Mohammad Yadegari. The binomial theorem: A widespread concept in medieval islamic mathematics. *Historia Mathematics 7*, 1980.