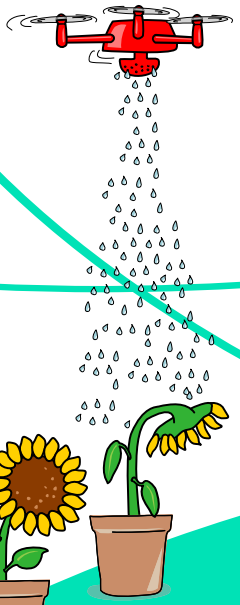


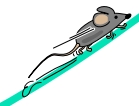


INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

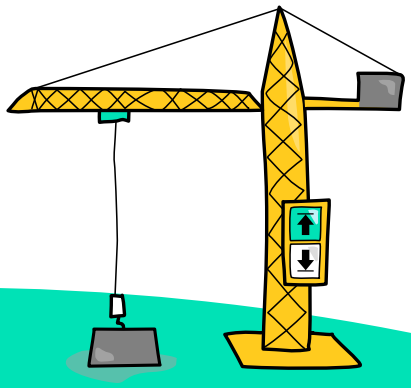
Aufgaben und Lösungen 2026

Schuljahre 7/8



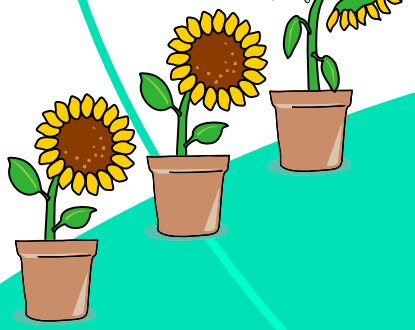


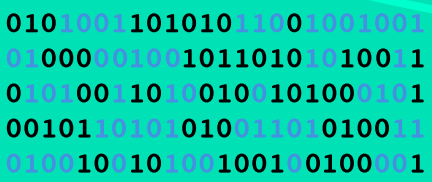
https://www.informatik-biber.ch/



Herausgeber:

Susanne Falkenberg-Thut, Nora A. Escherle,
Jean-Philippe Pellet

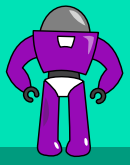




SVIA

www.svia-ssie-ssii.ch
schweizerischer verein für informatik in
ausbildung // société suisse pour l'infor
matique dans l'enseignement // società sviz
zera per l'informatica nell'insegnamento







HASLERSTIFTUNG ETH zürich | ABZ AUSBILDUNGS- UND BERATUNGSZENTRUM FÜR INFORMATIKUNTERRICHT | Scuola universitaria professionale della Svizzera italiana SUPSI | hep/ | PH LUZERN PÄDAGOGISCHE HOCHSCHULE

Mitarbeit Informatik-Biber 2026

Masiar Babazadeh, Susanne Falkenberg-Thut, Gabriel Parriaux, Jean-Philippe Pellet, Andrea Maria Schmid, Giovanni Serafini

Projektleitung: Nora A. Escherle, Janine Blattner

Herzlichen Dank für die Aufgabenentwicklung für den Schweizer Wettbewerb an:
Juraj Hromkovič, Regula Lacher: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Christian Datzko, Gymnasium Liestal
Nadine Fröhlich, Universität Basel
Urs Meier, Pädagogische Hochschule Luzern
Bernadette Spieler, Pädagogische Hochschule Zürich
Nils Böhnke, Mohadesa Ekhlassi, Dirk Schmerenbeck, Andreas Schneider, Jacqueline Staub und Alexander Wiß: Universität Trier, Deutschland

Die Aufgabenauswahl wurde erstellt in Zusammenarbeit mit den Organisatoren von Bebras in Deutschland, Österreich, Ungarn und Luxembourg. Besonders danken wir:
Wolfgang Pohl, Karsten Schulz, Margaretha Schlüter, Kirsten Schlüter, Michael Weigend: Bundesweite Informatikwettbewerbe (BWINF), Deutschland
Wilfried Baumann, Liam Baumann: Österreichische Computer Gesellschaft
Gerald Futschek, Lukas Lehner: Technische Universität Wien
Zsuzsa Pluhár, Bence Gaal: ELTE Informatikai Kar, Ungarn

Die Online-Version des Wettbewerbs wurde auf cuttle.org realisiert. Für die gute Zusammenarbeit danken wir:
Eljakim Schrijvers, Justina Oostendorp, Alieke Stijf, Kyra Willekes: cuttle.org, Niederlande
Andrew Csizmadia: Raspberry Pi Foundation, Vereinigtes Königreich

Die Programmieraufgaben wurden speziell für die Online-Plattform erstellt und entwickelt. Wir danken herzlich für die Initiative:
Jacqueline Staub: Universität Trier, Deutschland
Dirk Schmerenbeck: Universität Trier, Deutschland
Dave Oostendorp: cuttle.org, Niederlande

Für den Support während der Wettbewerbswochen danken wir:
Valérie Verdan: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht

Für die Organisation und Durchführung des Finales 2025 an der ETH danken wir:
Dennis Komm, Hans-Joachim Böckenhauer, Angélica Herrera Loyo, Andre Macejko, Adeline Pittet, Valérie Verdan: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Für die Korrektur der Finalaufgaben:
Niu Daniel, Jay Forrer, Alessandra Frank, Oliver Graf, Kassak Jakuk, Simon Krause, Adeline Richter, Colin Rüegg, Benjamin Schlaich, Alexander Skodinis, Emanuel Skodinis, Moritz Stocker, Philip

© Informatik-Biber 2026, SVIA

Whittington, Ursus Wigger, Guiheng Zheng, Mateo Zuluaga
Für die Übersetzung der Finalaufgaben ins Französische:
Jean-Philippe Pellet: Haute école pédagogique du canton de Vaud
Workshop-Leitende für die Finalisten:
Nora A. Escherle, Susanne Falkenberg-Thut, Jean-Philippe Pellet, Gloria Pittet, Clemens Pohle, Hannah Rauterberg, Peter Rossmanith, Dirk Schmerenbeck, Benjamin Schmid, Giovanni Serafini

Christoph Frei: Chragokyberneticks (Logo Informatik-Biber Schweiz)
Andrea Leu, Sarah Beyeler, Lea Hasler, Maggie Winter: Senarclens Leu + Partner AG

Ganz besonderen Dank gilt unseren grossen Förderern Juraj Hromkovič, Dennis Komm, Gabriel Parriaux und der Haslerstiftung. Ohne sie würde es diesen Wettbewerb nicht geben.

Die deutschsprachige Fassung der Aufgaben wurde ähnlich auch in Deutschland, Österreich und Luxemburg verwendet. Die französischsprachige Version wurde ebenfalls in Luxemburg verwendet.

Die französischsprachige Übersetzung wurde von Elsa Pellet und die italienischsprachige Übersetzung von Christian Giang erstellt.

© Informatik-Biber 2026, SVIA

INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Der Informatik-Biber 2026 wurde vom Schweizerischen Verein für Informatik in der Ausbildung (SVIA) durchgeführt und massgeblich und grosszügig von der Hasler Stiftung unterstützt. Weitere Partner*innen und Wettbewerbssponsoren, die den Wettbewerb finanziell unterstützt haben, sind das Amt für Kindergarten, Volksschule und Beratung (AKVB) des Kantons Bern, Amt für Wirtschaft AWI des Kantons Zürich, die Basellandschaftliche Gebäudeversicherung, die CYON AG, die Informatik Stiftung Schweiz, sowie die UBS.

Folgende Akademischen Partner unterstützen uns bei der Aufgabenerstellung: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht, Haute école pédagogique du canton de Vaud, La Scuola universitaria professionale della Svizzera italiana, Pädagogische Hochschule Luzern und die Universität Trier.

Dieses Aufgabenheft wurde am 23. November 2026 mit dem Textsatzsystem LaTeX erstellt. Wir bedanken uns bei Christian Datzko für die Entwicklung und langjährige Pflege des Systems zum Generieren der 36 Versionen dieser Broschüre (nach Sprachen und Schulstufen). Das System wurde analog zum Vorgänger-System neu programmiert, welches ab 2014 gemeinsam mit Ivo Blöchliger entwickelt wurde. Jean-Philippe Pellet danken wir für die Entwicklung der `bebras` Toolchain, die seit 2020 für die automatisierte Konvertierung der Markdown- und YAML-Quelldokumente verwendet wird.

Hinweis: Alle Links wurden am 20. November 2026 geprüft.

Die Aufgaben sind lizenziert unter einer Creative Commons Namensnennung – Nicht-kommerziell – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz. Die Autoren sind auf S. 77 genannt.

Vorwort

Der Wettbewerb «Informatik-Biber», der in verschiedenen Ländern der Welt schon seit über 20 Jahren bestens etabliert ist, will das Interesse von Kindern und Jugendlichen an der Informatik wecken. Der Wettbewerb wird in der Schweiz auf Deutsch, Französisch und Italienisch vom Schweizerischen Verein für Informatik in der Ausbildung SVIA durchgeführt und von der Hasler Stiftung unterstützt.

Der Informatik-Biber ist der Schweizer Partner der Wettbewerbs-Initiative «Bebras International Challenge on Informatics and Computational Thinking» (`https://www.bebras.org/`), die in Litauen ins Leben gerufen wurde.

Der Wettbewerb wurde 2010 zum ersten Mal in der Schweiz durchgeführt. 2012 wurde zum ersten Mal der «Kleine Biber» (Stufen 3 und 4) angeboten.

Der Informatik-Biber regt Schülerinnen und Schüler an, sich aktiv mit Themen der Informatik auseinander zu setzen. Er will Berührungsängste mit dem Schulfach Informatik abbauen und das Interesse an Fragestellungen dieses Fachs wecken. Der Wettbewerb setzt keine Anwenderkenntnisse im Umgang mit dem Computer voraus – ausser dem «Surfen» im Internet, denn der Wettbewerb findet online am Computer statt. Für die Fragen ist strukturiertes und logisches Denken, aber auch Phantasie notwendig. Die Aufgaben sind bewusst für eine weiterführende Beschäftigung mit Informatik über den Wettbewerb hinaus angelegt.

Der Informatik-Biber 2026 wurde in fünf Altersgruppen durchgeführt:

- Stufen 3 und 4
- Stufen 5 und 6
- Stufen 7 und 8
- Stufen 9 und 10
- Stufen 11 bis 13

Jede Altersgruppe erhält Aufgaben in drei Schwierigkeitsstufen: leicht, mittel und schwierig. In den Altersgruppen 3 und 4 waren 9 Aufgaben zu lösen, mit je drei Aufgaben in jeder der drei Schwierigkeitsstufen. Für die Altersklassen 5 und 6 waren es je vier Aufgaben aus jeder Schwierigkeitsstufe, also 12 insgesamt. Für die restlichen Altersklassen waren es 15 Aufgaben, also fünf Aufgaben pro Schwierigkeitsstufe.

Für jede richtige Antwort wurden Punkte gutgeschrieben, für jede falsche Antwort wurden Punkte abgezogen. Wurde die Frage nicht beantwortet, blieb das Punktekonto unverändert. Je nach Schwierigkeitsgrad wurden unterschiedlich viele Punkte gutgeschrieben beziehungsweise abgezogen:

	leicht	**mittel**	**schwer**
richtige Antwort	6 Punkte	9 Punkte	12 Punkte
falsche Antwort	−2 Punkte	−3 Punkte	−4 Punkte

© Informatik-Biber 2026, SVIA

Dieses international angewandte System zur Punkteverteilung soll den Anreiz zum blossen Erraten der Lösung eliminieren.

Jede Teilnehmerin und jeder Teilnehmer hatte zu Beginn 45 Punkte (Stufen 3 und 4: 27 Punkte; Stufen 5 und 6: 36 Punkte) auf dem Punktekonto.

Damit waren maximal 180 Punkte (Stufen 3 und 4: 108 Punkte; Stufen 5 und 6: 144 Punkte) zu erreichen, das minimale Ergebnis betrug 0 Punkte.

Bei vielen Aufgaben wurden die Antwortalternativen am Bildschirm in zufälliger Reihenfolge angezeigt. Manche Aufgaben wurden in mehreren Altersgruppen gestellt. Diese Aufgaben hatten folglich in den verschiedenen Altersgruppen unterschiedliche Schwierigkeitsstufen.

Einige Aufgaben werden für bestimmte Altersgruppen als «Bonus» angegeben: sie haben keinen Einfluss auf die Berechnung der Gesamtpunktzahl. Diese Übungen dienen vielmehr dazu, bei mehreren TeilnehmerInnen mit identischer Punktzahl zu entscheiden, wer sich für eine mögliche nächste Runde qualifiziert.

Für weitere Informationen:

Schweizerischer Verein für Informatik in der Ausbildung
SVIA-SSIE-SSII
Informatik-Biber
Janine Blattner

`https://www.informatik-biber.ch/kontaktieren/`
`https://www.informatik-biber.ch/`

© Informatik-Biber 2026, SVIA

Inhaltsverzeichnis

Mitarbeit Informatik-Biber 2026 i

Vorwort iv

Inhaltsverzeichnis vi

1. Laptops 2026-CH-03a 1

2. Spaziergang im Park 2026-DE-01 5

3. Eingangshalle 2026-DE-07 9

4. Schneemobil-Taxi 2026-AT-01 13

5. Azulejos 2026-CH-06 17

6. Kopf oder Zahl 2026-IN-05 21

7. Quadrat einfärben 2026-DE-09 23

8. Auto-Fabrik 2026-JP-03 27

9. Geralds Garten 2026-LT-01a 31

10. Video-Kontakte 2026-HU-01b 35

11. Feuergeist 2026-CZ-02 39

12. Muddys Methode 2026-DE-10 43

13. Aika und Benji 2026-JP-05 47

14. Operation Giesskanne 2026-JP-02a 51

15. Murmelmusik 2026-DE-05 55

16. Münzspiel 2026-AT-02 59

17. Sprung-Spiel 2026-NL-01 63

18. Versteckspiel 2026-US-03b 67

19. Quartierhilfe 2026-DE-90 71

A. Aufgabenautoren 77

B. Akademische Partner 79

C. Sponsoring 80

© Informatik-Biber 2026, SVIA

1. Laptops

4 Kinder besuchen den Informatikkurs. Jedes Kind benutzt immer den gleichen Laptop.

Nach dem Unterricht legen sie ihre Laptops in 2 Kisten. Sie stapeln die Laptops so, dass jeder seinen Laptop am nächsten Tag direkt von oben herausnehmen kann, ohne die anderen Laptops umräumen zu müssen.

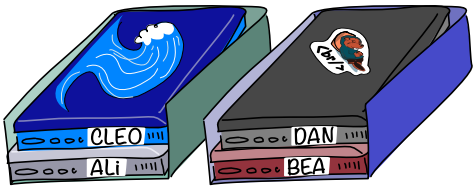
Heute verlassen sie den Raum nach dem Unterricht in dieser Reihenfolge:

1. Ali
2. Bea
3. Cleo
4. Dan

Wegen dem Stundenplan wissen sie bereits, dass sie morgen in dieser Reihenfolge zum Unterricht kommen:

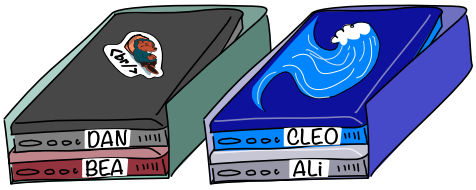
1. Bea
2. Ali
3. Dan
4. Cleo

Wie stapeln die Kinder heute die Laptops, wenn sie den Raum verlassen?

A) 

B) 

C) 

D) 

Lösung

Antwort B ist richtig.

Wir prüfen die vier Antworten, ob sie zu den Reihenfolgen heute und morgen passen.

Antwort B: Heute können die Kinder die Laptops so stapeln wie in Antwort B: Ali und Bea legen ihre Laptops nacheinander in die linke Kiste, Cleo und Dan ihre Laptops nacheinander in die rechte Kiste.

Wenn die Kinder morgen ihre Laptops nehmen wollen, nehmen Bea und Ali ihre Laptops nacheinander aus der linken Kiste und Dan und Cleo ihre Laptops nacheinander aus der rechten Kiste. Antwort B ist also richtig.

Antwort A Heute können die Kinder die Laptops so stapeln wie in Antwort A: Ali legt seinen Laptop in die linke Kiste, Bea ihren in die rechte, dann Cleo ihren in die linke und Dan seinen in die rechte Kiste.

Aber wenn Bea morgen als erste ihren Laptop nehmen will, muss sie vorher Dans Laptop aus der rechten Kiste holen. Antwort A ist also falsch.

Antwort C: Heute können die Kinder die Laptops *nicht* so stapeln wie in Antwort C: Wenn Ali als erster seinen Laptop in eine der Kisten legt, kann sein Laptop nur unten in der Kiste liegen; hier liegt der Laptop aber oben in der rechten Kiste. Antwort C ist also falsch.

Antwort D: Heute können die Kinder die Laptops so stapeln wie in Antwort D: Ali legt seinen Laptop in die rechte Kiste, Bea ihren in die linke, dann Cleo ihren in die rechte und Dan seinen in die linke Kiste. Das ist die gleiche Stapelung wie in Antwort A, nur mit vertauschten Kisten.

Und auch hier kann Bea morgen nicht als erste ihren Laptop nehmen, ohne vorher Dans Laptop aus der – in diesem Fall linken – Kiste zu holen. Antwort D ist also auch falsch.

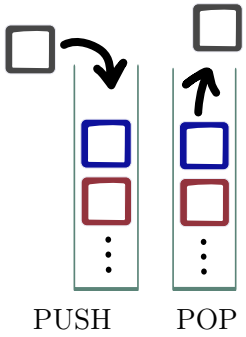
Dies ist Informatik!

Aus den Laptop-Kisten kann immer nur der oberste Laptop entnommen werden. Derjenige Laptop, der als letzter in die Kiste gelegt wird, muss zuerst herausgenommen werden. Die beiden Laptops in einer Kiste bilden nämlich einen Stapel.

Auch die Informatik kennt *Stapel*. Ein Informatik-Stapel, häufig auch mit dem englischen Wort «Stack» bezeichnet, ist eine nützliche Datenstruktur, wenn es darum geht, Daten während einer Berechnung vorübergehend zu speichern.

Bei der Verwendung eines Stacks kann ein Programm zwei Aktionen ausführen: Es kann ein Element auf den Stapel legen (engl.: push) oder das Element entnehmen (engl.: pop), das sich gerade oben auf dem Stapel befindet. So realisiert ein Stack das Speicherprinzip *LIFO* (kurz für engl.: last in, first out).

 © Informatik-Biber 2026, SVIA



Hätten die Kinder nur eine Laptop-Kiste, könnten die Laptops nur einen Stapel bilden. Dann müssten die Kinder nach dem LIFO-Prinzip beim Beginn des Informatikunterrichts die Laptops in genau der umgekehrten Reihenfolge nehmen, wie sie die Laptops am Ende der vorigen Stunde in die Kiste gelegt haben. Mit zwei Kisten haben die Kinder mehr Möglichkeiten, wie sie die Laptops stapeln und wieder nehmen können. Der zweite Stapel erhöht also die Flexibiltät, und so kann bei der Vereinbarung der Kinder in dieser Biberaufgabe die Herausnehme-Reihenfolge morgen eben anders sein als die umgekehrte Hineinlege-Reihenfolge heute.

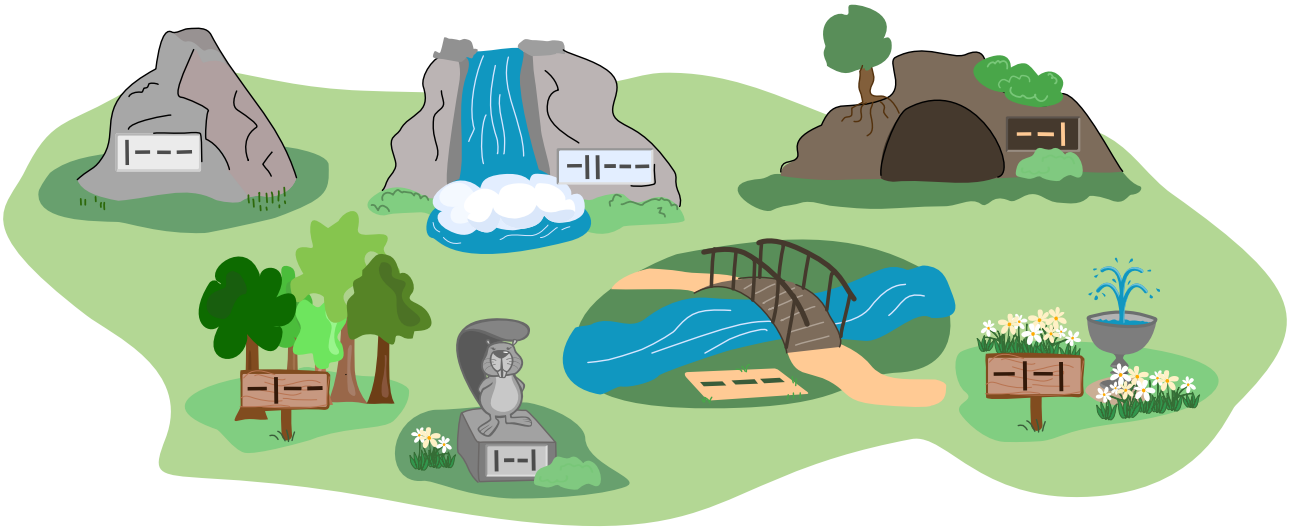
Stichwörter und Webseiten

- *Stack (Stapelspeicher, LIFO)*: `https://de.wikipedia.org/wiki/Stapelspeicher`

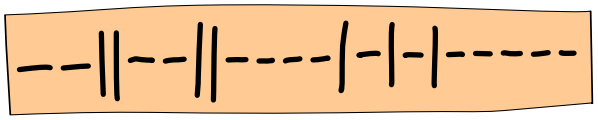
© Informatik-Biber 2026, SVIA

2. Spaziergang im Park

Im Biberpark gibt es mehrere besondere Orte. Jeder dieser Orte hat eine Markierung, und zwar aus den Zeichen - und |.



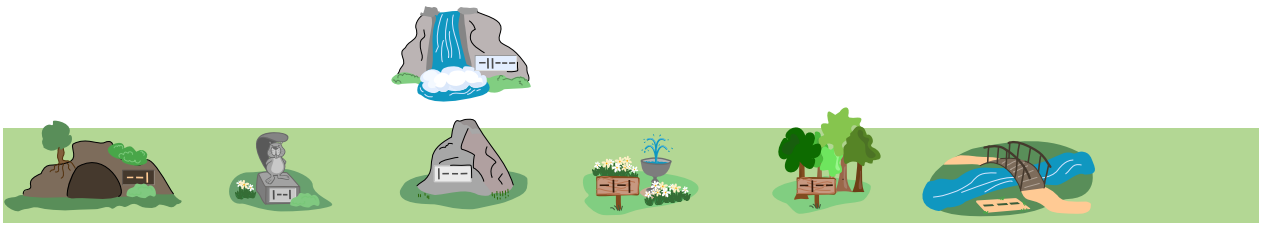
Simon geht im Biberpark spazieren. Immer wenn er an einem besonderen Ort vorbeikommt, schreibt er seine Markierung auf ein Blatt, eine nach der anderen. Am Ende hat er diese Zeichen aufgeschrieben:



An welchen Orten ist Simon in welcher Reihenfolge vorbeigekommen?

Lösung

So ist es richtig:



Das hat Simon auf seinem Spaziergang notiert:

`--||--||----|-|-|-----`

Die Markierungen der Orte sind unterschiedlich viele Zeichen lang. Daher wissen wir zunächst nicht, wo genau in Simons Notiz eine Markierung endet und eine neue beginnt.

Um die richtige Antwort zu finden, gehen wir die Notiz von links nach rechts durch, also in der Reihenfolge, wie Simon die Markierungen aufgeschrieben hat. Dabei prüfen wir, ob ein oder mehrere aufeinanderfolgende Zeichen dem Beginn einer Markierung entsprechen.

Mit dem ersten Zeichen `-` beginnen mehrere Markierungen. Wir können daher noch nicht sagen, welche Markierung Simon zuerst notiert und an welchem Ort er daher zuerst vorbeigekommen ist.

Mit den ersten beiden Zeichen `--` beginnen genau zwei Markierungen: die der Höhle `--|` und die der Brücke `---`. Wir wissen also auch an dieser Stelle noch nicht, welche Markierung er als erste notiert und an welchem Ort er daher zuerst vorbeigekommen ist.

Mit den ersten drei Zeichen `--|` beginnt nur die Markierung der Höhle. Wir wissen jetzt, dass Simon zuerst an der Höhle vorbeigekommen ist und deren Markierung notiert hat. Zudem wissen wir, dass nun eine neue Markierung beginnt.

Das nächste Zeichen, das Simon notiert hat, ist `|`. Nur die jeweils vier Zeichen langen Markierungen der Statue `|--|` und des Felsens `|---` beginnen damit. Die nächsten vier Zeichen aus Simons Notiz `|--|` bilden genau die Markierung der Statue. Die Statue ist also der zweite Ort, an dem Simon vorbeigekommen ist.

In der gleichen Weise gehen wir nun die weiteren Zeichen Schritt für Schritt durch. So können wir feststellen, dass Simon insgesamt an diesen besonderen Orten in dieser Reihenfolge vorbeigekommen ist:

1. Höhle `--|`
2. Statue `|--|`
3. Felsen `|---`
4. Springbrunnen `-|-|`
5. Baumgruppe `-|--`
6. Brücke `---`

 © Informatik-Biber 2026, SVIA

Simons Notiz enthält auch die Markierung des Wasserfalls `-||---`. Es gibt aber weder eine einzelne Markierung noch mehrere Markierungen hintereinander, die mit dem davor stehenden Teil `--||-` der Notiz übereinstimmen. Simon ist also nicht am Wasserfall vorbeigekommen.

Dies ist Informatik!

Du hast soeben eine Folge von Codewörtern – die Markierungen der besonderen Orten – aus einer zusammenhängenden Kodierung ohne Trennzeichen zwischen den Codewörtern dekodiert.

In dieser Biberaufgabe hast du eine Kodierung – Simons Notiz – in eine Folge von Codewörtern zerlegt. Dadurch konntest du die Kodierung dekodieren, und zwar in die Reihenfolge der Orte.

Die Zerlegung und damit die Dekodierung war vor allem deshalb möglich, weil der Code (die Menge der aller Codewörter) in dieser Bebras-Aufgabe *präfixfrei* ist: Kein Codewort ist ein Präfix eines anderen; das heisst, kein Codewort beginnt mit einem anderen Codewort.

Präfixfreie Codes spielen in der Informatik eine wichtige Rolle, vor allem bei der *Datenkompression*. Ihre definierende Eigenschaft garantiert eine eindeutige Dekodierbarkeit: Wenn eine Folge von Codesymbolen – in der Informatik sind Codesymbole sehr oft Bits – der Reihe nach gelesen wird, sind die Grenzen zwischen den Codewörtern eindeutig. An jeder Stelle in der Folge ist klar, wo ein Codewort endet und das nächste beginnt, ohne dass Trennzeichen oder ein Vorausschauen erforderlich sind.

Ein bekanntes Beispiel für die Erzeugung präfixfreier Codes ist die *Huffman-Kodierung*, die 1952 von David A. Huffman entwickelt wurde. Diese weit verbreitete Komprimierungsmethode ordnet häufigeren Zeichen kürzere Codewörter und selteneren Zeichen längere Codewörter zu.

Huffman-Kodierungen sind so konstruiert, dass sie präfixfrei sind. Dadurch wird sichergestellt, dass komprimierte Daten zuverlässig und effizient dekodiert werden können.

Stichwörter und Webseiten

- *präfixfreie Codes*: `https://de.wikipedia.org/wiki/Präfixcode`
- *Datenkompression*: `https://de.wikipedia.org/wiki/Datenkompression`
- *Huffman-Kodierung*: `https://de.wikipedia.org/wiki/Huffman-Kodierung`

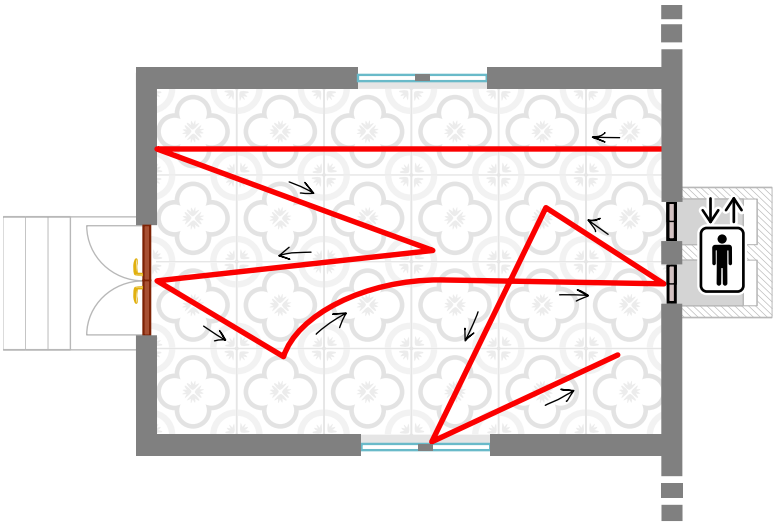


© Informatik-Biber 2026, SVIA

3. Eingangshalle

Ein Sicherheitsroboter fährt durch eine leere Eingangshalle.
Wenn nichts Besonderes passiert, fährt er geradeaus.
Es gibt drei Ausnahme-Regeln:

1. Wenn der Roboter ein Hindernis berührt, ändert er seine Richtung zufällig.
2. Wenn er eine Bewegung erkennt, ändert er seine Richtung zum Ort der Bewegung hin.
3. Wenn er ein Geräusch hört, ändert er seine Richtung zum Geräusch hin.



Dies ist der Grundriss der Eingangshalle. Die Wände, die Tür links, die Fenster (oben und unten) und die Aufzüge (rechts) sind Hindernisse für den Roboter.
Die rote Line zeigt, wie der Roboter heute gefahren ist.

Während seiner Fahrt heute sind drei Ereignisse passiert:



Ein Ball trifft ein Fenster.

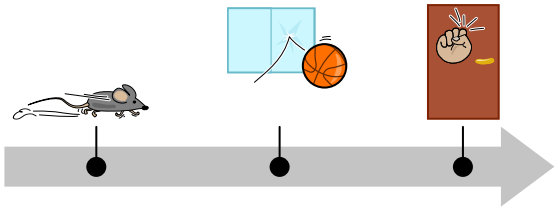


Jemand klopft an die Tür.



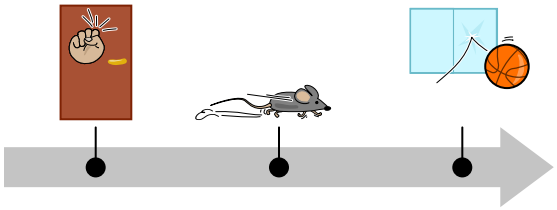
Eine Maus rennt durch die Eingangshalle.

In welcher Reihenfolge sind die Ereignisse passiert?

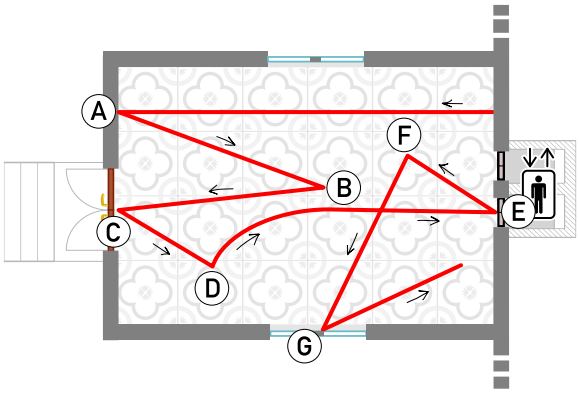


Lösung

So ist es richtig:



Schauen wir uns die Fahrt des Roboters an:



An den Positionen A, C, E und G berührt der Roboter Hindernisse. Dort ändert er jeweils seine Richtung zufällig (Regel 1).

Auch an den Positionen B, D und F ändert der Roboter seine Richtung. Dort sind aber keine Hindernisse. An diesen Positionen müssen also die Ereignisse passiert sein.

An Position D fällt auf, dass der Roboter von dort aus eine Kurve fährt, also immer wieder seine Richtung ändert. Das lässt sich nur durch dieses Ereignis erklären: «Eine Maus läuft durch die Eingangshalle.» Die Maus läuft von links nach rechts. Dadurch erkennt der Roboter immer wieder neu eine Bewegung und ändert seine Richtung immer wieder nach rechts, zur Bewegung hin (Regel 2). So versucht er, der Maus zu folgen, bis er ihre Bewegung nicht mehr erkennen kann.

An Position B ändert der Roboter seine Richtung zur Tür hin. Weil er danach geradeaus fährt, verfolgt er in diesem Fall keine Bewegung. Hier hat der Roboter also ein Geräusch an der Tür gehört (Regel 3): «Jemand klopft an die Tür».

An Position F ändert der Roboter seine Richtung zu einem Fenster hin und fährt danach geradeaus weiter. Hier hat er wieder ein Geräusch gehört (Regel 3), und zwar am Fenster: «Ein Ball trifft ein Fenster».

Die drei Ereignisse sind also in der Reihenfolge passiert wie oben angegeben.

 © Informatik-Biber 2026, SVIA

Dies ist Informatik!

Sicherheitsroboter werden zum Beispiel in öffentlichen Gebäuden, Lagerhallen und Bürokomplexen eingesetzt. Diese Roboter fahren üblicherweise in vorgegebenen, geschlossenen Bereichen, erkennen besondere Ereignisse und handeln nach programmierten Regeln. Auch wenn moderne Sicherheitsroboter maschinelles Lernen oder fortgeschrittene Wahrnehmungssysteme nutzen können, wird ihr grundlegendes Verhalten weiterhin von Algorithmen bestimmt, die Sensoreingaben auf Handlungen abbilden.

Die Informatik unterscheidet bei *autonomen Systemen* wie Sicherheitsrobotern oder Fahrzeugen verschiedene Autonomiestufen. Einfachere autonome Systeme wie der Roboter in dieser Biberaufgabe folgen Regeln, etwa «Wenn du ein Hindernis erkennst, wende» oder «Wenn du ein Geräusch erkennst, bewege dich darauf zu.» Das Verhalten von Systemen auf dieser Stufe ist vorhersehbar und lässt sich direkt auf die programmierten Regeln zurückführen.

Fortgeschrittenere autonome Systeme sollen sich frei in der Welt bewegen können. Die reale Welt ist aber komplex, und es können Ereignisse passieren, die kaum vorhersehbar sind. Ein autonomes Fahrzeug zum Beispiel muss in beliebigen Situationen selbstständig sinnvoll entscheiden können, wie es fahren soll. Es ist aber unmöglich, für jede Situation, der ein Fahrzeug auf der Strasse begegnen könnte, eine eigene Regel festzulegen. Deshalb benutzen Systeme auf höheren Autonomiestufen keine festen Regeln. Stattdessen haben sie Ziele und entscheiden dynamisch, wie sie handeln sollen, um Ziele zu erreichen. Dazu können sie Sensordaten, Planungsmethoden und maschinelles Lernen nutzen. Man nennt solche Systeme auch *agentisch*, da sie zum Erreichen ihrer Ziele ihr Handeln eigenständig planen und festlegen.

Stichwörter und Webseiten

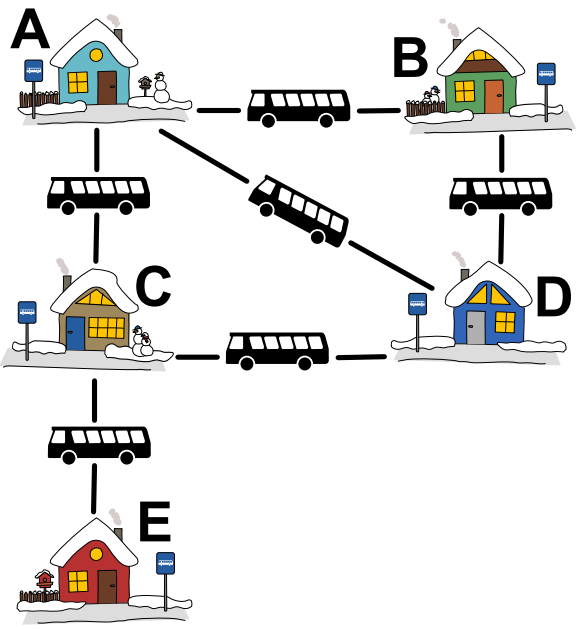
- Robotik: `https://de.wikipedia.org/wiki/Robotik`
- Sensor: `https://de.wikipedia.org/wiki/Sensor`
- Aktor: `https://de.wikipedia.org/wiki/Aktor`
- Sicherheitsroboter: `https://wirtschaftslexikon.gabler.de/definition/sicherheitsroboter-122267`
- Autonome Fahrzeuge: `https://de.wikipedia.org/wiki/Selbstfahrendes_Kraftfahrzeug`
- Open Roberta Lab: `https://lab.open-roberta.org`
- Kiddle - Robot Facts for Kids: `https://kids.kiddle.co/Robot`

© Informatik-Biber 2026, SVIA

4. Schneemobil-Taxi

Es ist Winter in Biberheim!

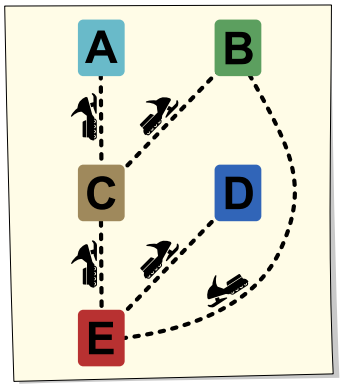
Im Ort gibt es fünf Haltestellen, A bis E. Der Plan rechts zeigt die direkten Busverbindungen zwischen je zwei Haltestellen. Auf jeder Verbindung fahren Busse in beiden Richtungen.



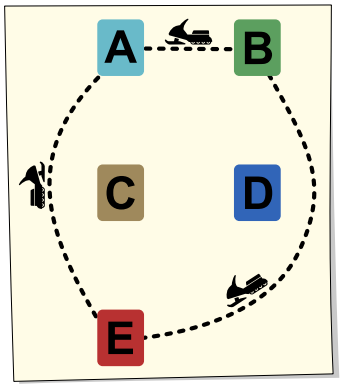
Marcel hat Schneemobil-Taxis. Er bietet genau dann eine direkte Taxi-Verbindung zwischen zwei Haltestellen an, wenn es *keine* direkte Busverbindung zwischen diesen beiden Haltestellen gibt.

Welcher Plan zeigt Marcels direkte Taxi-Verbindungen?

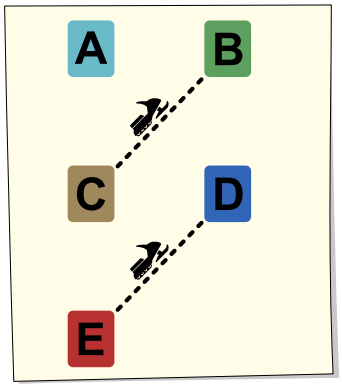
A)



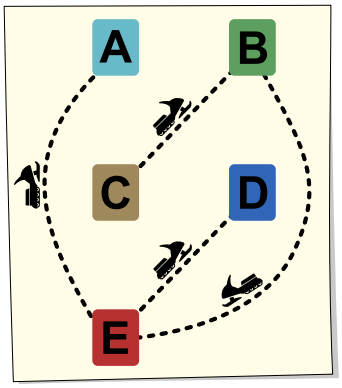
B)



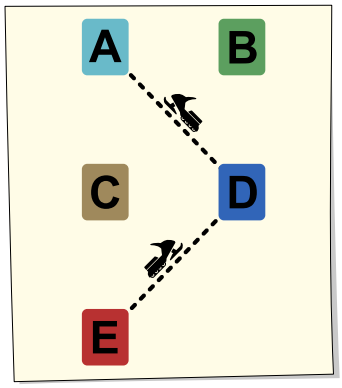
C)



D)



E)



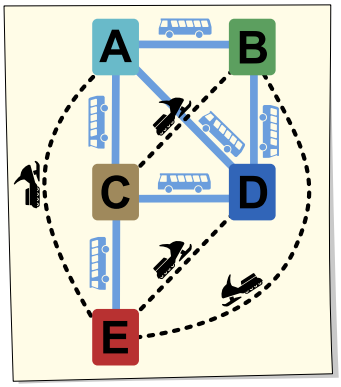
Lösung

Antwort D ist richtig.

So findest du die richtige Antwort: Für jedes Paar von Haltestellen, zwischen denen es keine direkte Busverbindung gibt, zeichnest du eine gestrichelte Linie in den Bus-Plan. Dazu wählst du nach und nach alle Haltestellen von A bis E aus. Für eine ausgewählte Haltestelle prüfst du für jede der anderen Haltestellen, ob es eine direkte Busverbindung zwischen der ausgewählten und der anderen Haltestelle gibt. Wenn nicht, zeichnest du eine Linie zwischen den beiden Haltestellen.

Ein Beispiel: Von Haltestelle **B** aus gibt es direkte Busverbindungen zu **A** und zu **D**, aber nicht zu **C** und zu **E**. Also zeichnest du zwischen **A** und **C** und zwischen **A** und **E** je eine neue Linie ein.

Insgesamt ergibt sich dieser Plan, in dem es zwischen je zwei Haltestellen immer eine Verbindung gibt, entweder per Bus oder per Schneemobil-Taxi (die neuen Linien sind gestrichelt):



Die neuen Linien sind die fehlenden direkten Verbindungen. Weil Marcel genau dort eine direkte Taxi-Verbindung anbietet, ergeben die neuen Linien (zusammen mit den Haltestellen) genau Marcels Schneemobil-Taxi-Plan – wie in Antwort **D**.

Dies ist Informatik!

Die fünf Haltestellen bilden gemeinsam mit den direkten Busverbindungen ein Netzwerk: das Busliniennetz von Biberheim. Netzwerke spielen in der Informatik eine wichtige Rolle, zum Beispiel Verkehrsnetze in Routenplanern. Wenn eine Computer-App ihre Benutzer als Kontakte miteinander verbindet, spricht man auch von «sozialen Netzwerken».

Für die Darstellung von Netzwerken verwendet die Informatik eine Struktur, die *Graph* genannt wird. Ein Graph besteht aus *Knoten* und *Kanten*. Die Knoten stehen für die Dinge, die miteinander verbunden sind. Die Kanten sind Knotenpaare und stehen für die Verbindungen. Auch das Busliniennetz von Biberheim kann man als Graph darstellen: Die Haltestellen sind die Knoten, und die direkten Busverbindungen sind die Kanten.

Marcel sucht im Busliniennetz-Graphen genau die Knotenpaare (also: Haltestellenpaare), die keine Kanten sind (also: keine direkten Busverbindungen haben). In der Informatik würde man sagen: Marcel sucht den *Komplementgraph* des Busliniennetz-Graphen. Ein Komplementgraph hat genau

 © Informatik-Biber 2026, SVIA

dieselben Knoten wie der ursprüngliche Graph, aber kehrt die Verbindungen um: Ein Knotenpaar ist genau dann eine Kante im Komplementgraph, wenn es keine Kante im ursprünglichen Graph ist.

Marcels Schneemobil-Taxi-Plan zeigt also den Komplementgraph des Busliniennetz-Graphen. Interessanterweise können manche Probleme, die auf einem Graphen schwer zu lösen sind, auf dessen Komplementgraphen einfacher werden.

Stichwörter und Webseiten

- Graph: `https://de.wikipedia.org/wiki/Graph_(Graphentheorie)`
- Komplementgraph: `https://de.wikipedia.org/wiki/Komplementgraph`

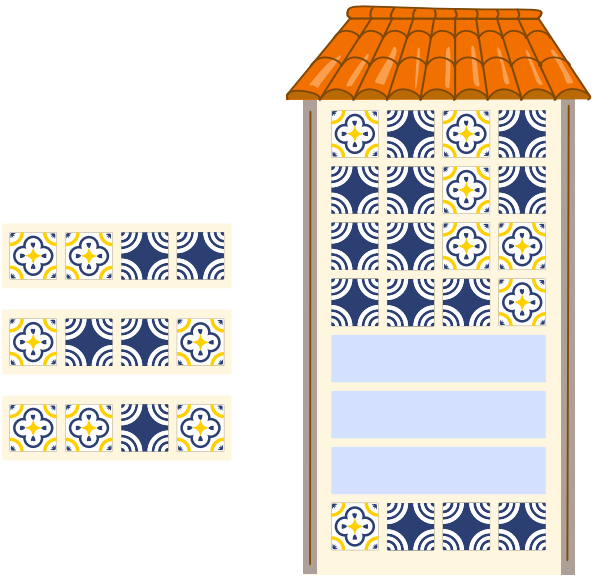
© Informatik-Biber 2026, SVIA

5. Azulejos

Ana wohnt in Portugal. Auf ihrer Hauswand verlegt sie Reihen mit je 4 Plättli (sogenannte Azulejos).

Jede Plättlireihe soll sich von der Reihe direkt darüber an genau einer Stelle unterscheiden. Dann liegen die Plättli richtig.

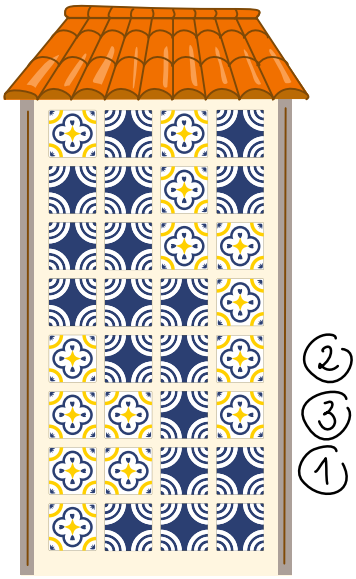
Ana ist noch nicht fertig: 3 Reihen fehlen.



Wie liegen die 3 Plättlireihen richtig auf der Wand?

Lösung

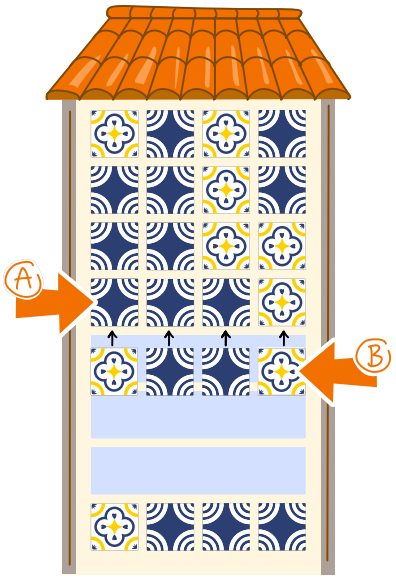
So ist es richtig:



Tatsächlich ändert sich in dieser Anordnung von Reihe zu Reihe immer genau ein Plättli. Die geänderten Stellen sind durch grüne Rahmen markiert.



So findest du die richtige Antwort: Für eine Reihe (A), die schon auf der Wand liegt und unter der noch eine Reihe fehlt, suchst du die einzige Reihe (B) heraus, die an genau einer Stelle anders ist. Jetzt fehlt unter dieser neuen Reihe eine Reihe, und du suchst wieder die einzige Reihe heraus, die dazu passt. Auf diese Weise legst du Reihe für Reihe, bis die Wand fertig ist.



 © Informatik-Biber 2026, SVIA

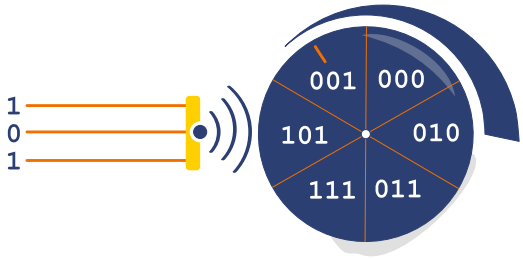
Dies ist Informatik!

Die Plättli in dieser Biberaufgabe haben nur zwei verschiedene Muster. Deshalb kann man die Reihen auch als Folgen aus Nullen und Einsen, also als Bitreihe vorstellen: 0 für das eine Muster, 1 für das andere. Für Anas Wand gilt: Ein Reihe muss sich an genau einer Stelle von einer Nachbarreihe unterscheiden. Eine Folge von Bitreihen, für die das Gleiche gilt (eine Bitreihe unterscheidet sich von der vorigen in genau einem Bit), nennt man *Gray-Code*. Anas Wand entspricht also diesem Gray-Code:

0101, 1101, 1100, 1110, 0110, 0010, 0011, 0111

Beachte: In einem Gray-Code muss man sich die Bitreihen im Kreis angeordnet vorstellen. Die erste Reihe muss sich deshalb von der letzten Reihe in genau einem Bit unterscheiden.

Gray-Codes werden unter anderem bei technischen Geräten verwendet. Ein Beispiel: An einem Lautsprecher gibt es einen Drehknopf, mit dem man die Lautstärke einstellen kann. Die Stellung des Knopfes reguliert also die Lautstärke. Während der Drehknopf verstellt wird, gibt ein Sensor die aktuelle Position über drei Leitungen an einen Computer weiter. Jede einzelne Leitung überträgt dabei entweder den Wert 0 oder den Wert 1. So werden acht aufeinanderfolgende Positionen des Knopfes unterschieden. Wird dabei ein Gray-Code verwendet, ändert sich beim Drehen des Knopfes immer nur der Wert einer einzigen Leitung. Es werden sprunghafte Änderungen der Lautstärke verhindert, die mit einem anderen Code auftreten könnten.



Stichwörter und Webseiten

- *Gray-Code*: `https://de.wikipedia.org/wiki/Gray-Code`

© Informatik-Biber 2026, SVIA

6. Kopf oder Zahl

Jede Biber-Münze hat zwei Seiten: «Kopf»  und «Zahl» .

Alex hat 7 Münzen vor sich liegen:



Nun dreht Alex mehrere Male genau zwei Münzen gleichzeitig um.

Wie werden Alex' Münzen NIE liegen?

A) 

B) 

C) 

D) 

Lösung

Antwort B ist richtig.

Immer, nachdem Alex genau zwei Münzen umgedreht hat, zeigen die beiden Münzen andere Seiten als vorher. Es gibt drei Möglichkeiten:

1. Beide Münzen zeigen nun Zahl statt vorher Kopf,
2. beide zeigen nun Kopf statt vorher Zahl oder
3. eine zeigt nun Kopf statt vorher Zahl und die andere zeigt nun Zahl statt vorher Kopf.

Die *Kopf-Anzahl* der Münzen, also die Anzahl der Münzen, die Kopf zeigen, ändert sich entweder um 2 (Möglichkeiten 1 und 2) oder um 0 (Möglichkeit 3). Es gilt also: Ist die Kopf-Anzahl zu Beginn gerade, bleibt sie immer gerade; und ist sie zu Beginn ungerade, bleibt sie immer ungerade.

Zu Beginn ist die Kopf-Anzahl von Alex' Münzen 2, also gerade. Durch das Drehen können Alex' Münzen also nur so liegen, dass die Kopf-Anzahl gerade ist – so wie in den Antworten A (Kopf-Anzahl = 4), C (Kopf-Anzahl = 4) und D (Kopf-Anzahl = 2). Sie können aber nie so liegen, dass die Kopf-Anzahl ungerade ist. Nur in Antwort B liegen die Münzen mit ungerader Kopf-Anzahl 3.

Dies ist Informatik!

(Ganze) Zahlen können gerade oder ungerade sein; das nennt man auch ihre *Parität*. In der Antworterklärung oben haben wir festgestellt, dass sich die Parität der Kopf-Anzahl von Alex' Münzen-Reihe nicht ändern kann, wenn Alex immer genau zwei Münzen gleichzeitig dreht.

In der Informatik nennen wir eine Eigenschaft, die sich durch die Anwendung bestimmter Operationen nicht ändert, eine *Invariante*. Wer die Invariante dieser Biberaufgabe entdeckt, nämlich die Parität der Kopf-Anzahl, kann sehr einfach die richtige Antwort bestimmen und muss sie nicht durch Ausprobieren herausfinden.

Invarianten werden in der Informatik häufig verwendet, zum Beispiel:

- um zu überprüfen, ob Algorithmen korrekt funktionieren;
- um zu beweisen, dass bestimmte Zustände nicht erreicht werden können; oder
- um Systeme mit vielen möglichen Zuständen zu analysieren.

Stichwörter und Webseiten

- Invariante: `https://de.wikipedia.org/wiki/Invariante_(Informatik)`
- Parität: `https://de.wikipedia.org/wiki/Paritätsbit`

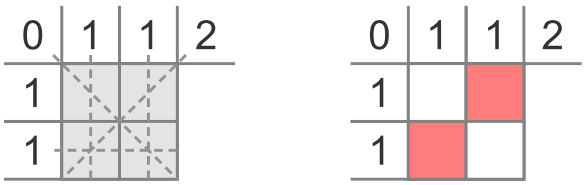
 © Informatik-Biber 2026, SVIA

7. Quadrat einfärben

In einem quadratischen Raster sollen die Felder rot ■ oder weiss □ eingefärbt werden. Am Anfang sind alle Felder grau.

Die Zahlen an den Rändern geben vor, wie viele Felder in jeder Zeile, jeder Spalte und in jeder der beiden Diagonalen **rot** sein müssen. Alle übrigen Felder sollen weiss werden.

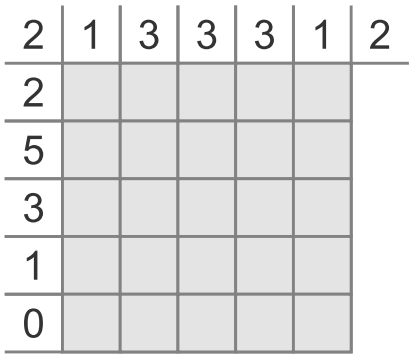
Nun ein kleines Beispiel mit einem 2×2-Quadrat:



In jeder Zeile und jeder Spalte muss genau ein Feld rot sein. Auf der Diagonale von links oben nach rechts unten sind keine roten Felder erlaubt, auf der anderen Diagonalen müssen es genau 2 rote Felder sein.

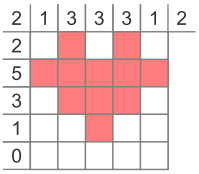
Die Färbung ist die einzige Lösung zu diesem Quadrat.

Wie sieht die Lösung zu diesem 5×5-Quadrat aus?

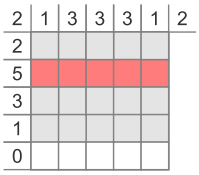


Lösung

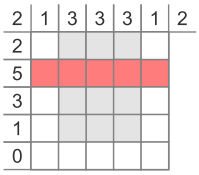
So ist es richtig:



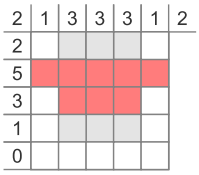
Es gibt keine andere Möglichkeit, die Felder gemäss der Vorgaben einzufärben. Das zeigt der nun beschriebene Weg, diese Lösung zu bestimmen:



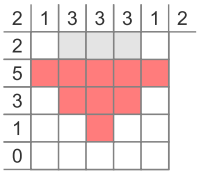
Schritt 1: Die Zeilen mit den Vorgaben **5** und **0** müssen vollständig rot bzw. weiss eingefärbt werden.



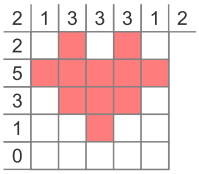
Schritt 2: Die noch grauen Felder in der ersten und der fünften Spalte müssen weiss sein; ein Feld ist ja bereits rot.



Schritt 3: Die übrigen grauen Felder der mittleren Zeile müssen rot gefärbt werden; nur so können noch 3 Felder in dieser Zeile rot werden.



Schritt 4: In der vierten Zeile müssen Feld 2 und 4 wegen der Vorgaben für die Diagonalen weiss sein; und dann muss Feld 3 rot sein.



Schritt 5: Schliesslich ergibt sich in der ersten Zeile zwingend die Färbung **rot–weiss–rot** für die Felder 2–3–4.

Ohne die Vorgaben für die Diagonalen gäbe es mehrere Lösungen.

Dies ist Informatik!

Alle möglichen Rot-Weiss-Färbungen des 5×5-Quadrats sind Lösungskandidaten für das Rätsel in dieser Biberaufgabe. Zusammen bilden sie den sogenannten *Lösungsraum* des Rätsels. Es gibt einen sehr einfachen Algorithmus, eine Lösung in diesem Raum zu finden: Er probiert alle Kandidaten aus und prüft jeweils, ob alle Vorgaben erfüllt sind. Bei einem 5×5-Quadrat gibt es aber $2^{25} = 33.554.432$ Kandidaten, und es würde ziemlich lange dauern, sie alle zu berechnen und zu prüfen. Von einem

 © Informatik-Biber 2026, SVIA

6×6-Quadrat mit über 68 Milliarden oder einem 7×7-Quadrat mit über 562 Billionen Kandidaten wollen wir erst gar nicht reden.

In der Informatik verwendet man deshalb effizientere Suchverfahren. Wenn wie in dieser Biberaufgabe Bedingungen an eine Lösung gestellt werden, nutzt man die vorhandenen Bedingungen, um Schritt für Schritt eine Lösung zu konstruieren. Häufig kommt dabei *Backtracking* zum Einsatz: Führt eine Entscheidung bei der Konstruktion der Lösung (etwa die Wahl einer Farbe) später in eine Sackgasse (keine Lösung mehr möglich), geht man zurück und entscheidet sich für eine andere Möglichkeit (eine andere Farbe).

Ein Vorteil dieser Methode ist, dass sie für alle möglichen Probleme funktioniert, bei denen ein Lösungsraum durchsucht werden soll – egal wie gross die Probleme sind und egal wieviele Lösungen sie besitzen. Am besten funktioniert die Methode, wenn die an eine Lösung gestellten Bedingungen bei möglichst vielen Entscheidungen keine Wahl lassen und z.B. nur eine einzige Farbe in Frage kommt. Bei dem oben beschriebenen Weg zur Lösung war das in allen Schritten der Fall, und so konnte die Lösung ohne Rückschritt gefunden werden.

Stichwörter und Webseiten

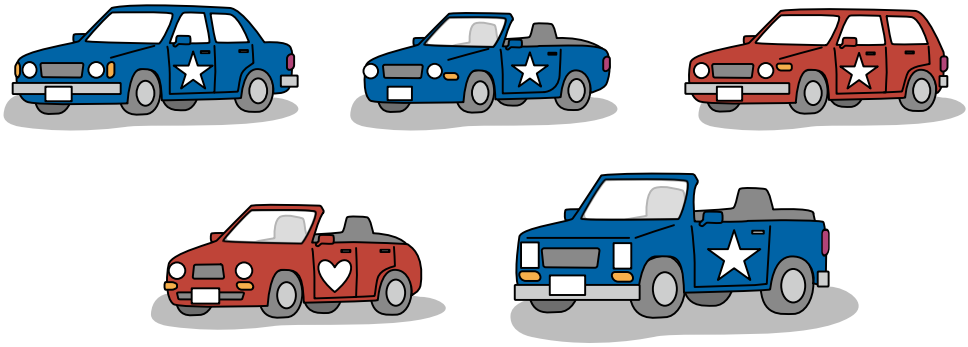
- Algorithmen: `https://de.wikipedia.org/wiki/Algorithmus`
- Suche im Lösungsraum: `https://de.wikipedia.org/wiki/Lösungsraum`
- Backtracking: `https://de.wikipedia.org/wiki/Backtracking`



© Informatik-Biber 2026, SVIA

8. Auto-Fabrik

In einer Fabrik werden diese fünf Automodelle gebaut:



Ein Roboter erkennt bei neu gebauten Autos diese Merkmale:

- Autofarbe: blau oder rot
- Logo: Stern oder Herz
- Türen: zwei (nur vorne) oder vier (vorne und hinten)
- Dach: offen oder geschlossen
- Scheinwerfer: rund oder eckig

Der Roboter soll für jedes neue Auto eindeutig bestimmen, zu welchem Modell es gehört. Informatikerin Binia sagt: Dazu braucht der Roboter nur drei Merkmale zu betrachten.

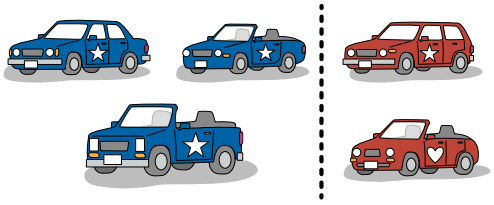
Welche drei Merkmale sind das? Kreise sie ein.

Autofarbe, Logo, Türen, Dach, Scheinwerfer

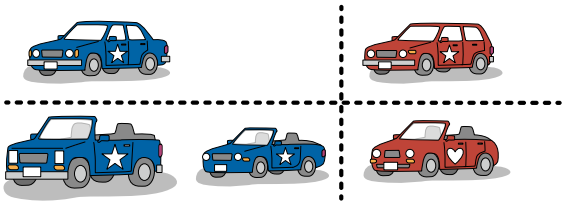
Lösung

Der Roboter kann für jedes neue Auto eindeutig dessen Modell bestimmen, wenn er die Autofarbe, das Dach und die Scheinwerfer betrachtet.

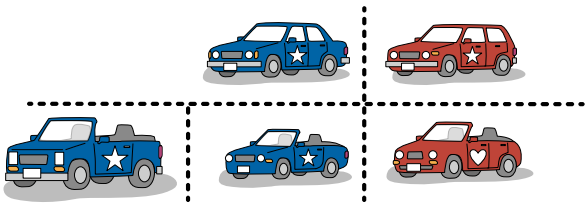
Mit jedem Merkmal kann der Roboter die Modelle in zwei Gruppen einteilen. Mit der richtigen Kombination aus drei Merkmalen kann er die Modelle in Gruppen einteilen, in denen jeweils nur noch ein Modell enthalten ist; dann kann er ein neues Auto diesem Modell zuordnen. Um die richtigen drei Merkmale zu finden, kann man so vorgehen:



Man fängt mit einem Merkmal an, zum Beispiel mit der Autofarbe, und teilt die Automodelle in zwei Gruppen: blaue und rote.



Als nächstes sucht man ein Merkmal, mit dem man möglichst beide Gruppen weiter aufteilen kann. Das funktioniert mit dem Dach, denn mit Autofarbe und Dach erhält man vier Gruppen: zwei blaue Modelle mit offenem Dach, ein blaues Modell mit geschlossenem Dach, ein rotes Modell mit offenem Dach, ein rotes Modell mit geschlossenem Dach.



Jetzt muss nur noch die Gruppe mit zwei Modellen aufgeteilt werden. Dazu eignet sich das Merkmal Scheinwerfer, denn eines der blauen Modelle mit offenem Dach hat runde Scheinwerfer, das andere eckige.

Es gibt nur eine richtige Antwort. Die folgende Tabelle zeigt, dass nur die Merkmalskombination Autofarbe, Dach und Scheinwerfer für jedes Modell anders ist. Über diese Merkmalskombination können alle Automodelle voneinander unterschieden - und neue Autos eindeutig einem Modell zugeordnet werden.

 © Informatik-Biber 2026, SVIA

Nummer	Modell	Autofarbe	Logo	Türen	Dach	Scheinwerfer
1		blau	Stern	vorne und hinten	geschlossen	rund
2		blau	Stern	nur vorne	offen	rund
3		rot	Stern	vorne und hinten	geschlossen	rund
4		rot	Herz	vorne und hinten	offen	rund
5		blau	Stern	nur vorne	offen	eckig

Bei jeder anderen Merkmalskombination können mindestens zwei Automodelle nicht voneinander unterschieden werden:

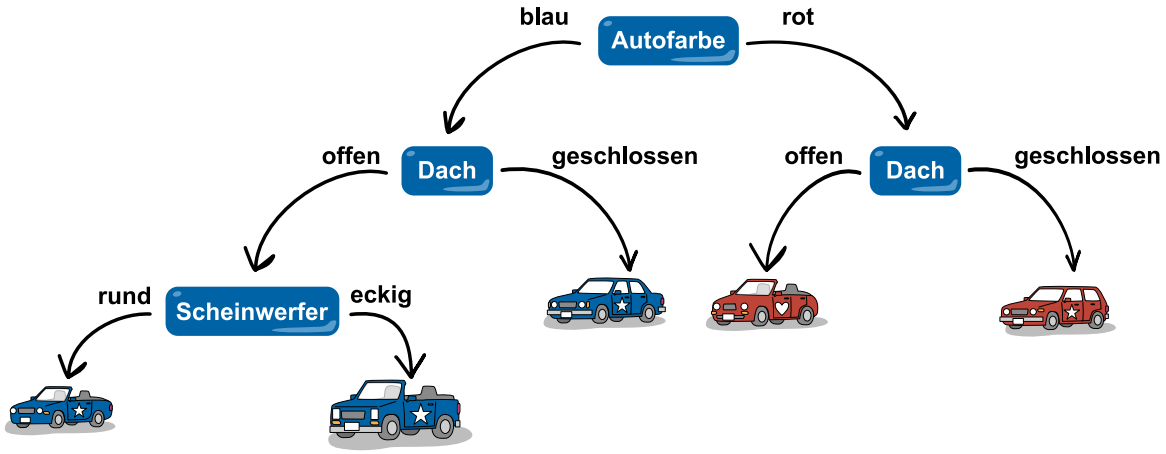
Merkmalskombination	Automodelle mit gleichen Merkmalen
Autofarbe, Logo, Türen	2 und 5
Autofarbe, Logo, Dach	2 und 5
Autofarbe, Logo, Scheinwerfer	1 und 2
Autofarbe, Türen, Dach	2 und 5
Autofarbe, Türen, Scheinwerfer	3 und 4
Logo, Türen, Dach	1 und 3 sowie 2 und 5
Logo, Türen, Scheinwerfer	1 und 3
Logo, Dach, Scheinwerfer	1 und 3
Türen, Dach, Scheinwerfer	1 und 3

Dies ist Informatik!

In dieser Biberaufgabe soll ein Roboter Objekte (Autos) nach bestimmten Merkmalen (wie der Farbe) verschiedenen Gruppen (nämlich Automodellen) eindeutig zuordnen. Für eine solche Zuordnung – Fachleute sprechen auch von *Klassifikation* – kann das Programm des Roboters mit einem *Entscheidungsbaum* arbeiten. Ein Entscheidungsbaum gibt vor, welche Merkmale der Roboter in welcher Reihenfolge prüfen soll und zu welchen Zuordnungen die Ergebnisse der Prüfungen führen.

Ein Programm zu schreiben, das Zuordnungen anhand eines Entscheidungsbaums trifft, ist nicht allzu schwierig. Schwieriger ist es, den Entscheidungsbaum aufzubauen, also die Entscheidungsmerkmale und ihre Reihenfolge geschickt zu wählen. Man könnte alle möglichen Entscheidungsbäume ausprobieren, um den richtigen zu finden. Weil es meist sehr viele mögliche Entscheidungsbäume gibt, ist dieses Vorgehen aber zu aufwändig. Stattdessen kann man *heuristisch* vorgehen und Beobachtungen über die zuzuordnenden Objekte nutzen. Man wählt z. B. «Farbe» als erstes Entscheidungsmerkmal, weil sich abhängig von der Farbe zwei etwa gleich grosse Gruppen von Autos bzw. Automodellen bilden lassen. Als zweites Entscheidungsmerkmal wählt man nicht das «Logo», denn alle blauen Autos haben den Stern als Logo, sodass die Prüfung dieses Merkmals bei der Zuordnung nicht weiterhilft.

So sieht der Entscheidungsbaum für den Roboter in dieser Biberaufgabe aus:



Der Ansatz, einen Entscheidungsbaum anhand von Beispielen zu generieren, lässt sich sogar automatisieren. Bereits in den 1980er-Jahren haben sich Informatiker im Rahmen der Forschung zu *Wissensbasierten Systemen* und *Maschinellem Lernen* damit auseinandergesetzt, wie man Entscheidungsbäume aus Beispielfällen und beobachteten Merkmalshäufigkeiten ableiten kann.

Stichwörter und Webseiten

- Entscheidungsbaum: `https://de.wikipedia.org/wiki/Entscheidungsbaum`
- Wissensbasierte Systeme: `https://inf-schule.de/ki/menueansicht/basiswissen_-ki/lernarten/erkundung_WissensbasierteSysteme`
- Maschinelles Lernen: `https://de.wikipedia.org/wiki/Maschinelles_Lernen`

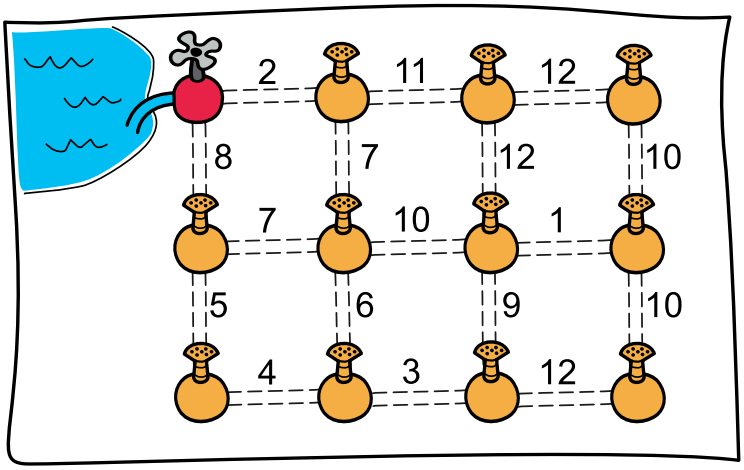
 © Informatik-Biber 2026, SVIA

9. Geralds Garten

Gärtner Gerald hat in seinem Blumengarten ein neues Sprinkler-System installiert.

Dazu hat Gerald die Skizze unten benutzt. Auf der Skizze sieht man:

- die Pumpe 
- alle Sprinkler  und
- alle Stellen _____, wo Gerald Schläuche verlegen konnte.



In der Skizze steht auch, wie viele Meter Schlauch an den einzelnen Stellen benötigt werden.

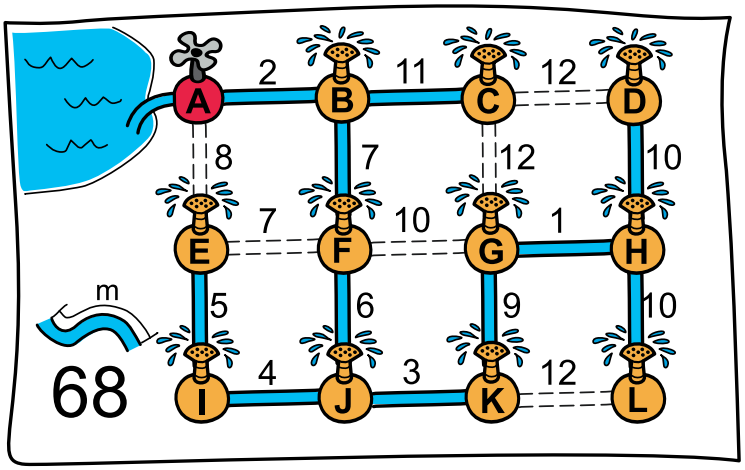
Gerald hat Schläuche nicht an allen Stellen verlegt, sondern nur so, dass

- alle Sprinkler von der Pumpe her Wasser bekommen **und**
- insgesamt so wenige Meter Schlauch wie möglich benötigt werden.

Wo hat Gerald Schläuche verlegt?

Lösung

So ist es richtig:



Zur leichteren Erklärung haben wir die Sprinkler mit Buchstaben bezeichnet. Ausserdem unterscheiden wir die Schlauch-Verlege-Stellen nach der dort jeweils benötigten Schlauch-Länge. Schliesslich sprechen wir von einem *Schläuche-Zyklus*, wenn man von einem Sprinkler (oder der Pumpe) aus starten und so entlang verlegter Schläuche gehen kann, dass man wieder zum Start zurück kommt; allerdings darf man dabei an keinem Schlauch mehr als einmal entlang gehen.

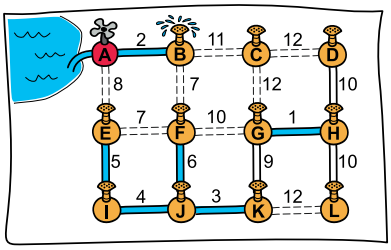
Um die richtige Antwort zu finden, kann man nun folgende Strategie anwenden:

- Gehe die Stellen der Länge nach durch; starte mit der kürzesten.
- Verlege dabei an einer Stelle genau dann einen Schlauch, wenn dadurch *kein* Zyklus entsteht.

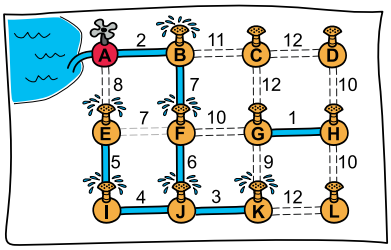
Warum ist es sinnvoll, keine Zyklen entstehen zu lassen? Ein Schläuche-Zyklus benötigt mehr Schlauchmeter als nötig. Aus einem Zyklus kann man nämlich immer einen Schlauch entfernen, und trotzdem bekommen alle Sprinkler im Zyklus weiterhin Wasser. Wenn wir beispielsweise die Schläuche in Geralds Garten mit den Längen 4, 5 und 6 schon verlegt haben und nun ausserdem den 7-Meter-Schlauch zwischen E und F verlegen, entsteht ein Zyklus E, F, J, I, E, ohne dass ein zusätzlicher Sprinkler Wasser bekommt. Einen Schlauch zwischen E und F kann man sich also sparen. Weil die Strategie mit den kürzesten Stellen bzw. Schläuchen anfängt, wird aus möglichen Zyklen der längste Schlauch eingespart. So werden insgesamt so wenige Meter Schlauch benötigt wie möglich.

Die obige Strategie ergibt immer eine Verlegung, die so wenige Meter Schlauch wie möglich benötigt und alle Sprinkler mit der Pumpe verbindet. Für Geralds Garten funktioniert die Strategie so:

 © Informatik-Biber 2026, SVIA

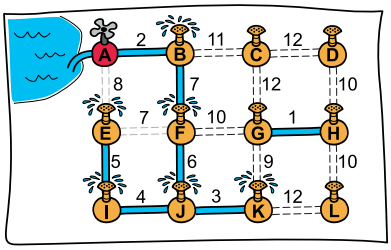


An allen Stellen der Längen 1 bis 6 wird ein Schlauch verlegt, weil dadurch kein Zyklus entsteht.

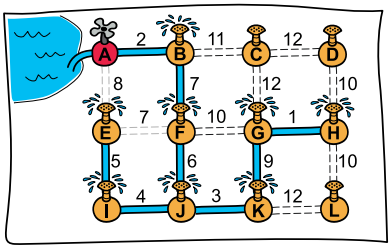


An den Stellen der Länge 7 wird

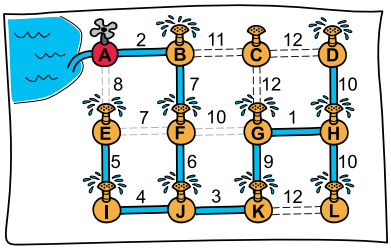
- zwischen B und F ein Schlauch verlegt, aber
- zwischen E und F kein Schlauch verlegt, weil dadurch der Zyklus E, F, J, I, E entstünde.



Zwischen A und E (Länge 8) wird kein Schlauch verlegt, weil dadurch der Zyklus A, B, F, J, I, E, A entstünde.

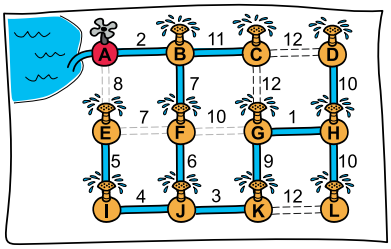


Zwischen G und K (Länge 9) wird ein Schlauch verlegt.



An den Stellen der Länge 10 wird

- zwischen D und H sowie H und L je ein Schlauch verlegt, aber
- zwischen F und G kein Schlauch verlegt, weil dadurch der Zyklus F, G, K, J, F entstünde.



Zwischen B und C (Länge 11) wird ein Schlauch verlegt.

An allen Stellen der Länge 12 wird kein Schlauch verlegt, weil dadurch jeweils ein Zyklus entstünde; du findest sicher jeweils selbst heraus, welcher.

Dies ist Informatik!

Pumpe, Sprinkler und Schlauch-Stellen bilden aus Sicht der Informatik zusammen einen *Graphen*. Das ist eine Struktur aus Objekten, den *Knoten*, und Objekt-Paaren, den *Kanten*. In dieser Biberaufgabe sind Pumpe und Sprinkler die Knoten, und die Schlauch-Stellen zwischen zwei Sprinklern (bzw. einem Sprinkler und der Pumpe) sind die Kanten. Ausserdem haben die Kanten Werte, nämlich die Schlauch-Meter; diese Werte nennt man auch die *Gewichte* der Kanten.

Graphen werden häufig eingesetzt, um Netzwerke zu modellieren, zum Beispiel aus Verkehrswegen, Stromleitungen oder Glasfaserkabeln. Dabei bringt das Verlegen von Leitungen, Schienen oder Ähnlichem Kosten mit sich. Um Kosten zu sparen, kann man den Teilgraph suchen, bei dem jeweils zwei Knoten immer über eine oder mehrere Kanten miteinander verbunden sind, der aber nur einen möglichst kleinen und kostengünstigen Teil der Kanten enthält (also den Teil mit dem geringstem Gesamtgewicht). In der Informatik wird ein solcher Teilgraph als *minimaler Spannbaum* bezeichnet. Die oben beschriebene Strategie, einen minimalen Spannbaum zu finden, ist als *Algorithmus von Kruskal* bekannt.

Dieser Algorithmus gehört zur Klasse der Greedy-Algorithmen (wörtlich: gierige Algorithmen). Diese Algorithmen treffen bei der Konstruktion einer Lösung in jedem Schritt die aktuell beste, «gierigste» Entscheidung (hier: wähle immer die kürzeste noch verfügbare Schlauchstelle), ohne zu prüfen, wie sich das auf spätere Schritte auswirkt. Bei der Suche nach einem minimalen Spannbaums führt diese lokale Strategie garantiert zur global besten Lösung. In der Praxis sparen «gierige» Algorithmen viel Geld und Ressourcen, weil sie schnell Lösungen finden und leicht zu implementieren sind. Allerdings finden sie nur für bestimmte Probleme optimale Lösungen. Dass ein gegebenes Problem dazugehört, sollte man erst beweisen, bevor man es «gierig» lösen will.

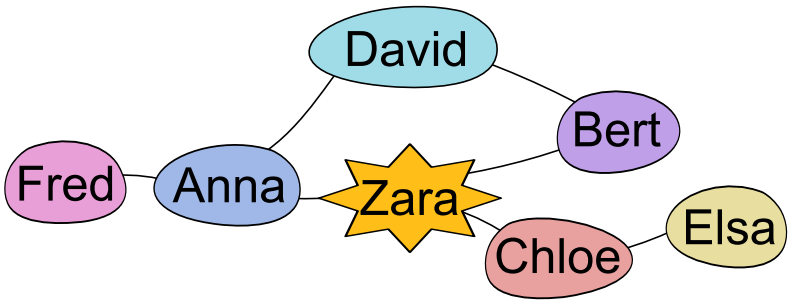
Stichwörter und Webseiten

- Algorithmus von Kruskal: `https://de.wikipedia.org/wiki/Algorithmus_von_Kruskal`

 © Informatik-Biber 2026, SVIA

10. Video-Kontakte

Zara schaut gerne Videos bei ZickZock. Die nutzen die Kontakte der User, um neue Videos vorzuschlagen. Das Bild zeigt Zaras Kontakte: Anna, Bert und Chloe sind *direkte* Kontakte; David, Elsa und Fred sind *indirekte* Kontakte.



ZickZock schlägt alle Videos vor, die die Kontakte gesehen haben, man selbst aber nicht, und zwar genau dann, wenn:

- ein direkter Kontakt mindestens 2 gleiche Videos gesehen hat wie man selbst.
- ein indirekter Kontakt mindestens 3 gleiche Videos gesehen hat wie man selbst.

Die Tabelle zeigt, welche Videos Zara und ihre Kontakte gesehen haben (✔).

								
Zara	✔	✔	✔	✔				
Anna	✔				✔	✔	✔	
Bert			✔	✔			✔	
Chloe	✔					✔	✔	✔
David		✔			✔	✔		
Elsa	✔	✔	✔			✔		
Fred			✔			✔	✔	✔

Welche neuen Videos schlägt ZickZock Zara vor?

Lösung

ZickZock schlägt Zara die Videos  und  vor.

Die richtige Antwort findet man, wenn man die Tabelle mit den angesehenen Videos genau so analysiert, wie ZickZock das macht:

Person	**Kontakt**								
Zara		✔	✔	✔	✔				
Anna	direkt	✔				✔	✔	✔	
Bert	**direkt**			✔	✔			★	
Chloe	direkt	✔					✔	✔	✔
David	indirekt		✔			✔	✔		
Elsa	**indirekt**	✔	✔	✔			★		
Fred	indirekt			✔			✔	✔	✔

In der ersten Zeile sehen wir, welche Videos Zara gesehen hat. Nun vergleichen wir diese Videos mit den Videos der Kontakte.

- Zuerst betrachten wir die direkten Kontakte Anna, Bert und Chloe. Bert ist der einzige direkte Kontakt, der mindestens 2 gleiche Videos gesehen hat wie Zara (✔). ZickZock schlägt deshalb das einzige Video vor, das Bert gesehen hat, Zara aber noch nicht (★). Videos von Anna und Chloe schlägt ZickZock nicht vor, weil die nicht mindestens 2 gleiche Videos gesehen haben wie Zara.
- Danach betrachten wir die indirekten Kontakte David, Elsa und Fred. Elsa ist der einzige indirekte Kontakt, der mindestens 3 gleiche Videos gesehen hat wie Zara. ZickZock schlägt deshalb das einzige Video vor, das Elsa gesehen hat, Zara aber noch nicht. Videos von David und Fred schlägt ZickZock nicht vor, weil die nicht mindestens 3 gleiche Videos gesehen haben wie Zara.

Dies ist Informatik!

ZickZock ist ein *Empfehlungsdienst*. So etwas kennst du vielleicht von Apps wie Netflix, YouTube oder Spotify: Sie schlagen dir Videos, Lieder oder andere Inhalte vor, die dir gefallen könnten.

Die App ZickZock aus dieser Biberaufgabe verwendet eine einfache Form einer Methode, die in der Informatik *kollaboratives Filtern* heisst. Dabei entstehen Vorschläge aus dem Vergleich mit anderen Personen. Der Vorschlags-Algorithmus der App sucht nach anderen Usern, die gleiche oder ähnliche Dinge mögen wie du. Wenn diese User ein Video gesehen oder ein Lied gehört haben, das du noch nicht kennst, kann dir die App genau dieses Video oder Lied vorschlagen.

In dieser Aufgabe wurden nicht alle anderen User berücksichtigt, sondern nur die Kontakte. Auch soziale Netzwerke wie Facebook oder LinkedIn nutzen Kontakte oder andere «soziale» Informationen

 © Informatik-Biber 2026, SVIA

für Vorschläge. Sie können zum Beispiel berücksichtigen, wer mit wem verbunden ist, welche Gruppen oder Themen jemanden interessieren oder welche Beiträge ähnliche Personen ansehen. So können sie neue Kontakte, Gruppen, Beiträge oder Inhalte vorschlagen.

Empfehlungsdienste können aber auch andere Verfahren nutzen. Bei *inhaltsbasierten Empfehlungen* achtet der Vorschlags-Algorithmus der App stärker darauf, welche Inhalte dich bisher interessiert haben. Wenn du in einer Nachrichten-App oft Artikel über Raumfahrt liest, schlägt dir die App vielleicht weitere Artikel über Raketen, Planeten oder neue Weltraummissionen vor.

Solche Vorschläge sind nützlich, weil du dadurch leichter Dinge findest, die zu deinen Interessen passen. Sie können aber auch zu einer *Filterblase* beitragen. Das bedeutet: Die App zeigt dir vor allem Dinge, die dem ähneln, was du ohnehin schon magst. Dadurch verpasst du vielleicht wichtige oder überraschende Dinge aus anderen Gebieten.

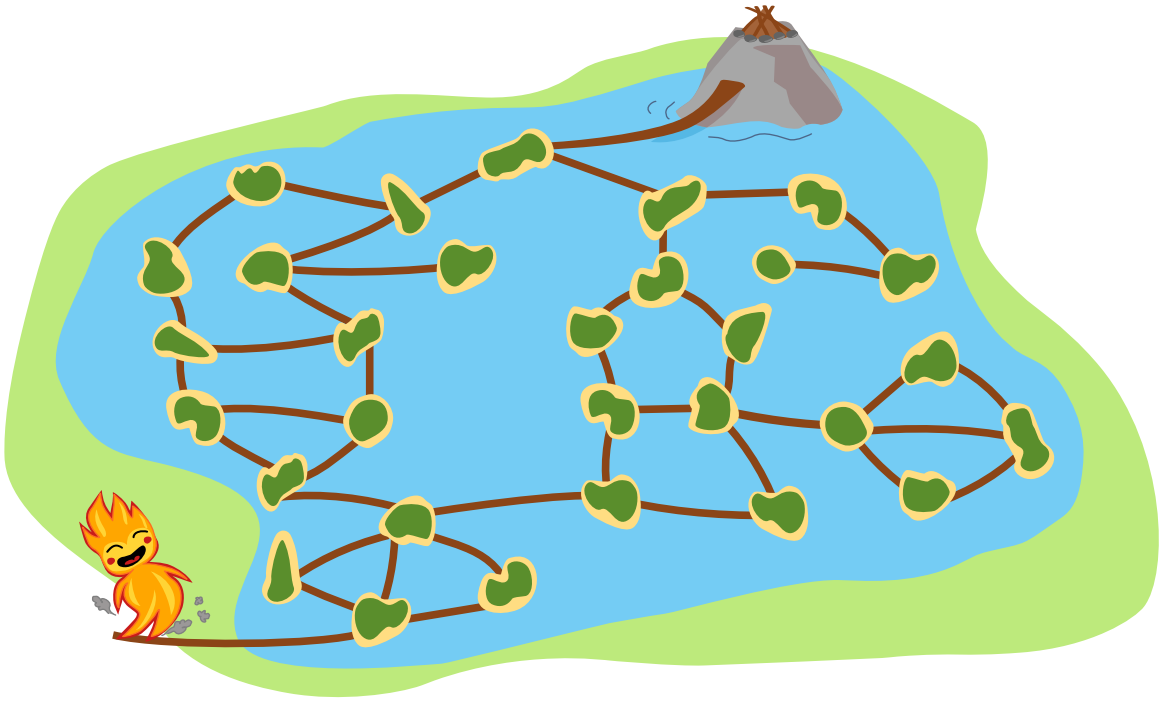
Stichwörter und Webseiten

- Empfehlungsdienst: `https://de.wikipedia.org/wiki/Empfehlungsdienst`
- Kollaboratives Filtern: `https://de.wikipedia.org/wiki/Kollaboratives_Filtern`
- Filterblase: `https://de.wikipedia.org/wiki/Filterblase`

© Informatik-Biber 2026, SVIA

11. Feuergeist

Auf dem See gibt es viele Inseln, die durch Stege miteinander verbunden sind.



Der Feuergeist  geht zur Feuerstelle  auf dem Fels. Auf dem Weg dahin will er so viele Inseln wie möglich besuchen.

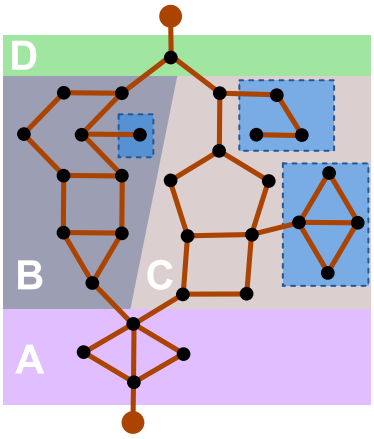
Der Feuergeist muss über die Stege gehen. Das Wasser im See würde ihn löschen. Die Stege aber sind aus Holz und brennen ab, sobald der Feuergeist darüber geht. Daher kann er über keinen Steg zweimal gehen. Die Inseln kann er aber mehrmals besuchen.

Wie viele Inseln kann der Feuergeist höchstens besuchen?

Lösung

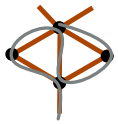
Die richtige Antwort ist 13 Inseln.

Für die Erklärung vereinfachen wir die Darstellung: Wir zeichnen Punkte für die Inseln und Linien für die Stege.

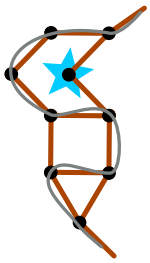


Wir können die Inseln im See in vier Abschnitte unterteilen: vier Inseln unten (A), die letzte Insel vor dem Felsen ganz oben (D) und dazwischen ein Abschnitt links (B) und einer rechts (C). Diese Abschnitte können wir unabhängig voneinander untersuchen, denn: Von jedem Abschnitt gibt es höchstens einen Steg zu einem anderen Abschnitt. Der Feuergeist kann also nicht zwischen den Abschnitten hin und her gehen. Insbesondere kann er von den mittleren Abschnitten B und C nur einen besuchen, da es keinen Steg zwischen diesen Abschnitten gibt. Schliesslich stellen wir noch fest, dass es Gruppen von Inseln gibt, die jeweils nur über einen einzigen Steg mit den anderen Inseln verbunden sind (siehe blaue Rechtecke). Würde der Feuergeist sie betreten, käme er nicht zurück und könnte nicht zur Feuerstelle gehen. Wir brauchen diese Gruppen also nicht berücksichtigen.

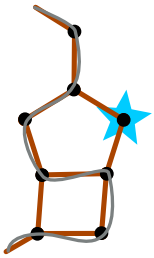
Nun schauen wir uns die einzelnen Abschnitte an.



In Abschnitt A gibt es 4 Inseln und genügend Stege, um alle 4 Inseln zu besuchen und dabei jeden Steg höchstens einmal zu benutzen.



In Abschnitt B gibt es (ohne die Insel im blauen Rechteck) 9 Inseln. Der Feuergeist kann nicht alle besuchen; mindestens 1 Insel bleibt unbesucht. Würde der Geist genau diese Insel (mit einem Stern markiert) besuchen, müsste er mindestens 2 andere Inseln auslassen. In Abschnitt B kann der Feuergeist also höchstens 8 Inseln besuchen.



Abschnitt C hat (ohne die Inseln in den blauen Rechtecken) 8 Inseln. Auch hier bleibt mindestens 1 Insel unbesucht. Auch hier gilt: Würde der Geist genau diese Insel (mit einem Stern markiert) besuchen, müsste er mindestens 2 andere Inseln auslassen. In Abschnitt C kann der Feuergeist also höchstens 7 Inseln besuchen.

In Abschnitt A kann der Feuergeist also höchstens 4 Inseln besuchen, in den Abschnitten B und C höchstens 8 Inseln (indem er durch Abschnitt B geht) und in Abschnitt D 1 Insel; insgesamt also 13 Inseln.

 © Informatik-Biber 2026, SVIA

Dies ist Informatik!

Die Inseln im See und die Stege dazwischen haben wir für die Erklärung als *Graph* modelliert. Diese mathematische Struktur besteht aus einer Menge von *Knoten* und einer weiteren Menge von Knotenpaaren, die als *Kanten* bezeichnet werden. Die Inselwelt des Sees lässt sich recht unmittelbar als Graph darstellen, indem man die Inseln als Knoten und die Stege als Kanten wählt.

Eine Modellierung mit Hilfe von Graphen ist nützlich, weil die Informatik für viele Fragen oder Probleme, die sich für Graphen stellen, Algorithmen zu deren Lösung kennt. Häufig spielen *Wege* durch Graphen eine wichtige Rolle: Ein Weg ist eine Abfolge von Knoten, so dass es zwischen zwei aufeinanderfolgenden Knoten eine Kante gibt.

In zahlreichen Graphenproblemen geht es darum, einen Weg durch einen Graphen zu finden, der bestimmte Bedingungen erfüllt. In dieser Biberaufgabe zum Beispiel ist der Weg mit der grössten Anzahl von Knoten gesucht, der bei der «Start-Insel» des Feuergeistes unten links beginnt, mit dem Felsen endet und in dem jede Kante höchstens einmal vorkommen darf. In anderen Fällen müssen ähnliche Bedingungen erfüllt werden, zum Beispiel bei der Routenplanung für eine Inspektion von Rohrleitungen durch einen Roboter: Hier soll der Roboter alle Rohre (= Kanten) besuchen, aber insgesamt möglichst wenig Strecke zurücklegen müssen.

Stichwörter und Webseiten

- Graph: `https://de.wikipedia.org/wiki/Graph_(Graphentheorie)`
- Graphentheorie: `https://de.wikipedia.org/wiki/Graphentheorie`

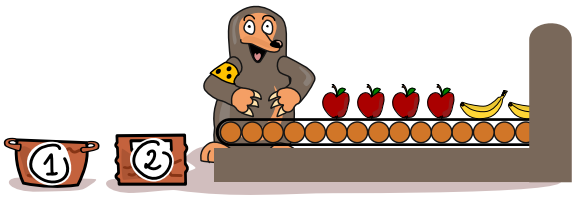


© Informatik-Biber 2026, SVIA

12. Muddys Methode

Auf einem Förderband kommen Früchte in folgender Reihenfolge an: zuerst Äpfel, dann Bananen und schliesslich Birnen.

Muddy der Maulwurf möchte wissen, ob von jeder Sorte gleich viele Früchte über das Förderband kommen. Muddy kann Äpfel, Bananen und Birnen voneinander unterscheiden. Leider kann Muddy nicht zählen, hat aber zwei Körbe zur Hand, die er auch voneinander unterscheiden kann.



Mit den Früchten und den Körben kann er Folgendes machen:



eine Frucht in einen der Körbe legen



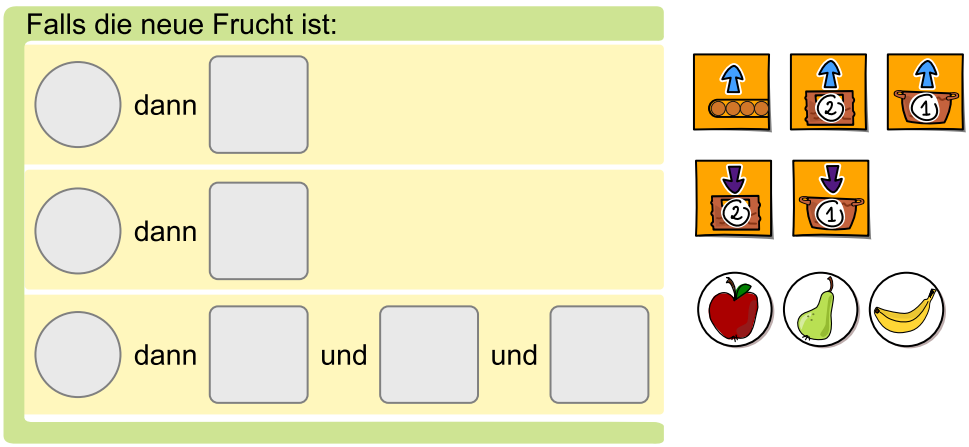
eine beliebige Frucht (Muddy kann nicht sehen, welche) aus einem der Körbe nehmen und beiseite legen



eine Frucht vom Band nehmen und beiseite legen

Damit hat Muddy sich eine Methode überlegt, wie er mit den Früchten auf dem Förderband umgeht. Er wendet diese Methode auf jede neue Frucht an, die bei ihm ankommt. Das macht er solange, bis das Förderband leer ist oder er versucht, eine Frucht aus einem leeren Korb zu nehmen. Wenn dann das Förderband leer ist und beide Körbe ebenfalls leer sind, sind gleich viele Äpfel, Bananen und Birnen über das Band gekommen.

Wie sieht Muddys Methode aus?



Lösung

Muddy hat sich diese Methode überlegt: Er legt alle Äpfel in Korb 1 und alle Bananen in Korb 2 - oder umgekehrt. Für jede Birne auf dem Förderband nimmt er anschliessend genau einen Apfel und eine Banane aus den Körben und legt dann auch die Birne beiseite.



Nur wenn gleich viele Äpfel, Bananen und Birnen über das Band gekommen sind, werden gleichzeitig beide Körbe und das Förderband leer sein. Genauer betrachtet gibt es mehrere Fälle, in denen Muddy aufhört, seine Methode anzuwenden:

Fall 1: Mindestens ein Korb ist leer, aber es ist noch mindestens eine Birne auf dem Band. Dann sind, ...

- falls der Apfelkorb leer ist, mehr Birnen als Äpfel über das Band gekommen;
- falls der Bananenkorb leer ist, mehr Birnen als Bananen über das Band gekommen.

Fall 2: Das Förderband ist leer, aber in mindestens einem Korb liegen noch Früchte. Dann sind, ...

- falls noch Äpfel im Korb liegen, weniger Birnen als Äpfel über das Band gekommen;
- falls noch Bananen im Korb liegen, weniger Birnen als Bananen über das Band gekommen.

Fall 3: Das Förderband und beide Körbe sind leer. Dann sind von allen Sorten gleich viele Früchte über das Band gekommen.

Dies ist Informatik!

Muddy nutzt zwei Körbe, um die Anzahl der Früchte zu vergleichen. Genauer betrachtet verwendet er die Körbe, um die Äpfel und Bananen solange zu «speichern», bis die Birnen kommen. Weil er Äpfel und Bananen getrennt voneinander speichert, kann er, ohne hinzusehen, für jede Birne auf dem Band aus jedem Korb eine Frucht nehmen und sicher sein, dass er insgesamt einen Apfel und eine Banane genommen hat.

In der Informatik gibt es eine ähnliche Struktur wie in dieser Biberaufgabe: Sie besteht aus einem Eingabeband (wie Muddys Förderband) und zwei *Kellern* (das sind Speicher, wie Muddys Körbe) und wird als *Zweikellerautomat* bezeichnet. Der Unterschied ist, dass im Gegensatz zu Muddys

 © Informatik-Biber 2026, SVIA

Körben in den Kellern Ordnung herrscht: Was zuletzt in den Keller gelegt wurde, wird als erstes wieder herausgenommen.

Generell spielen *Automaten* in der theoretischen Informatik eine wichtige Rolle. Ein Informatik-Automat liest Eingabesymbole, führt abhängig vom Symbol und seinem aktuellen Zustand Aktionen aus und geht in einen Folgezustand über. Die Eingabe wird akzeptiert, wenn das Wort komplett gelesen und ein Zielzustand erreicht wurde. Die Menge aller akzeptierten Wörter ist die Sprache, die der Automat erkennt. Ein Automat definiert also eine *formale Sprache*, die alternativ auch durch eine *formale Grammatik* beschrieben werden kann.

Generell besteht in der theoretischen Informatik eine fundamentale Verbindung zwischen *formalen Sprachen* (Mengen von Wörtern), *formalen Grammatiken* (den Regelwerken, die diese Sprachen erzeugen) und *Automaten* (den Maschinen-Modellen, die diese Sprachen erkennen bzw. akzeptieren). Bestimmte Arten von Automaten entsprechen dabei bestimmten Arten von Grammatiken und damit Sprachen. Zum Beispiel wird eine Sprache, die von einem Kellerautomaten mit nur einem Keller akzeptiert wird, in der *Chomsky-Hierarchie* formaler Sprachen als kontextfreie Sprache bezeichnet. Typische Computersprachen wie Programmier- oder Auszeichnungssprachen (z. B. HTML) sind kontextfrei und lassen sich durch effiziente Algorithmen schnell verarbeiten.

Mit einem Keller bzw. Korb könnte Muddy zwei Fruchtsorten vergleichen, mit zwei Kellern drei. Und so weiter? Nein, tatsächlich macht der zweite Keller schon den entscheidenden Unterschied, und weitere Keller bringen keinen zusätzlichen Vorteil. Es ist nämlich eine bekannte Tatsache in der Informatik, dass ein Kellerautomat mit zwei Kellern bereits eine Turingmaschine simulieren kann. Die Turingmaschine steht an der Spitze der Chomsky-Hierarchie und kann jede formale Sprache erkennen, die überhaupt berechenbar ist. Sie - und damit auch der Zweikellerautomat - ist genauso mächtig wie jeder moderne Computer.

Stichwörter und Webseiten

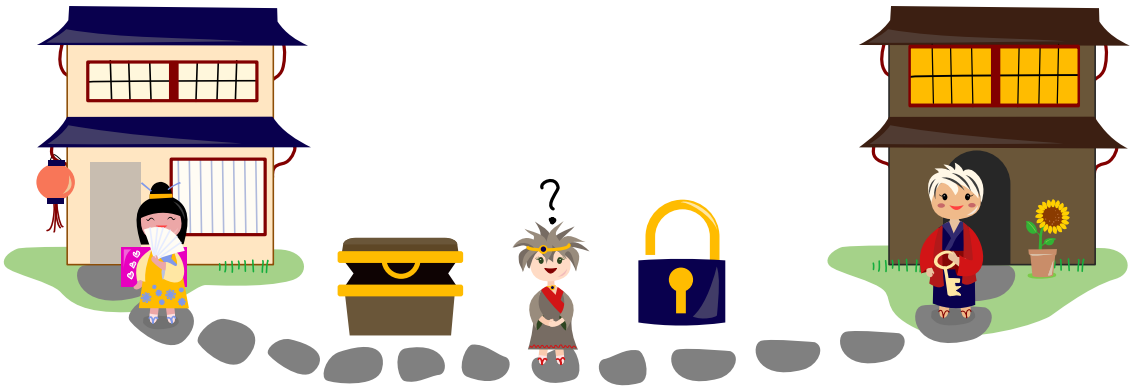
- kontextfreie Sprache (CFL – context-free language): `https://de.wikipedia.org/wiki/Kontextfreie_Sprache`
- Zweikellerautomat: `https://de.wikipedia.org/wiki/Zweikellerautomat` (`https://de.wikipedia.org/wiki/Kellerautomat`)



© Informatik-Biber 2026, SVIA

13. Aika und Benji


Aika möchte Benji 
etwas geben, ohne dass jemand anders weiss, was es ist. Dafür tut sie es in eine Kiste. Aika und Benji bleiben am liebsten zu Hause. Aber Mana besucht beide regelmässig und kann die Kiste für sie transportieren.



Damit auch Mana nicht weiss, was Aika Benji geben möchte, verschliesst Aika die Kiste mithilfe eines besonderen Vorhängeschlosses von Benji:

- Aika kann das Schloss ohne Schlüssel verschliessen; es schnappt ein.
- Nur Benji kann das Schloss öffnen, denn er hat den einzigen Schlüssel.
- Aika weiss sicher, dass das Schloss von Benji ist, denn er kann ein Schloss auf eine Weise markieren, die niemand anderes nachmachen kann.

Aika und Benji haben sich eine Reihenfolge von Schritten überlegt, mit der Mana die Kiste zwischen Benji und Aika hin und her transportieren kann, ohne dass Mana irgendwie herausfinden kann, was Aika Benji geben möchte. Leider sind die Schritte durcheinander geraten.

Das sind die Schritte. Wie ist die richtige Reihenfolge?

- *Mana bringt die Kiste zu Aika.*
- *Aika legt etwas in die Kiste und verschliesst sie mit dem Vorhängeschloss aus der Kiste.*
- *Benji legt das von ihm markierte Vorhängeschloss in die Kiste und behält den Schlüssel.*
- *Benji öffnet das Vorhängeschloss mit dem Schlüssel.*
- *Benji nimmt das offene Vorhängeschloss und markiert es.*
- *Aika gibt Mana die Kiste.*
- *Benji gibt Mana die Kiste.*
- *Mana bringt die Kiste zu Benji.*

Lösung

Aika möchte Benji in der Kiste etwas schicken, ohne dass Mana irgendwie herausfinden kann, was es ist. Dazu muss Benji zunächst sein besonderes und von ihm markiertes (oder: *signiertes*) Vorhängeschloss ohne Schlüssel an Aika geben:

1. Benji nimmt das offene Vorhängeschloss und markiert es.
2. Benji legt das von ihm markierte Vorhängeschloss in die Kiste und behält den Schlüssel.
3. Benji gibt Mana die Kiste.
4. Mana bringt die Kiste zu Aika.

Nun hat Aika eine Kiste und ein Vorhängeschloss. An der Markierung erkennt sie, dass es Benjis Schloss ist, also nur er das Schloss öffnen kann. Deshalb kann Aika nun beruhigt etwas in die Kiste tun, sie mit Benjis Vorhängeschloss verschliessen und an Benji zurückgeben. Er kann sie mit seinem Schlüssel wieder öffnen:

5. Aika legt etwas in die Kiste und verschliesst sie mit dem Vorhängeschloss aus der Kiste.
6. Aika gibt Mana die Kiste.
7. Mana bringt die Kiste zu Benji.
8. Benji öffnet das Vorhängeschloss mit dem Schlüssel.

Nur Benji kann das Schloss und damit die Kiste öffnen, weil nur er den Schlüssel hat.

Entscheidend ist die Markierung des Schlosses und dass niemand anderes als Benji sie nachmachen kann. Ohne das kann Mana ein anderes Schloss nehmen, für das sie einen Schlüssel hat, und mit Benjis Markierung versehen. Auf dem Weg von Benji zu Aika tauscht Mana dann Benjis Schloss gegen ihres. Aika erkennt Benjis Markierung und kann nicht entdecken, was Mana gemacht hat. Also verschliesst sie die Kiste mit diesem Schloss. Später, auf dem Weg von Aika zu Benji kann Mana das Schloss mit ihrem Schlüssel öffnen und so herausfinden, was Aika Benji geben möchte. Hätte Mana sogar ein Schloss, dass auch Benji mit seinem Schlüssel öffnen könnte, könnte sie die Kiste wieder verschliessen, und auch Benji könnte nicht entdecken, was Mana gemacht hat.

Dies ist Informatik!

Das Vorhängeschloss aus dieser Biberaufgabe, das jede Person ohne Schlüssel schliessen, aber nur eine Person mit Schlüssel öffnen kann, funktioniert so ähnlich wie *asymmetrische Kryptographie*. Bei asymmetrischer Kryptographie gibt es zwei Schlüssel, einen *öffentlichen Schlüssel* zum *Verschlüsseln* einer Nachricht und einen *privaten Schlüssel*, ohne den eine Nachricht nicht oder nur mit unvertretbar hohem Aufwand *entschlüsselt* werden kann. Der öffentliche Schlüssel darf jedem in die Hände gelangen und ist oftmals sogar frei im Internet lesbar. Den privaten Schlüssel darf niemand anders als der Empfänger haben, denn nur er soll die Nachricht wieder entschlüsseln können.

Asymmetrische Kryptographie ist sicher – wenn der *Austausch der Schlüssel* abgesichert werden kann. Vor allem, wenn sich zwei Menschen nicht persönlich treffen (können), muss klar sein, dass ein öffentlicher Schlüssel von einer bestimmten Person ist. Dies kann beispielsweise passieren, indem man

 © Informatik-Biber 2026, SVIA

vertrauenswürdigen Dritten (sogenannten *Zertifizierungsstellen*, die in der Regel im Betriebssystem oder im Webbrowser voreingestellt sind) gegenüber seine Identität nachweist und diese dann bestätigen, dass ein öffentlicher Schlüssel von einer bestimmten Person ist. Wenn das nicht funktionieren würde, könnte ein bösartiger Akteur den öffentlichen Schlüssel des Empfängers durch seinen eigenen ersetzen, die Nachricht abfangen und lesen oder verändern.

Zertifizierungen bzw. Zertifikate werden in Informatiksystemen sehr häufig verwendet, ohne dass man sich dessen bewusst ist. Wenn man beispielsweise auf `https://wettbewerb.informatik-biber.ch/` geht, ist die Übertragung der Webseite verschlüsselt (durch das Protokoll «https» erkennbar). Wenn man genauer hinschaut, stellt man fest, dass die Domäne ein Zertifikat hat, das von einer Zertifizierungsstelle (in diesem Fall «Let's Encrypt») als für diese Domäne gültig bestätigt wurde.

Was ebenfalls nicht zu unterschätzen ist, ist der sichere Umgang mit privaten Schlüsseln. Regelmässig gibt es Berichte, dass beispielsweise durch Social Engineering, Einbruch in Computersysteme oder schlichtweg Verarbeitungsfehler private Schlüssel öffentlich bekannt werden. Da kann es schon genügen, einen Browser zu verwenden, der Passwörter in eine Cloud synchronisiert, oder dass ein Akteur seine bösartige Zertifizierungsstelle im Betriebssystem oder Browser hinterlegt. In einigen Ländern nehmen sich staatliche Stellen das Recht heraus, ansonsten sichere Kommunikation auf diese Weise zu knacken. Wenn dann die vermeintlich sichere Kommunikation nicht mehr sicher ist, kann das höchst unangenehme Folgen für die Betroffenen haben.

Stichwörter und Webseiten

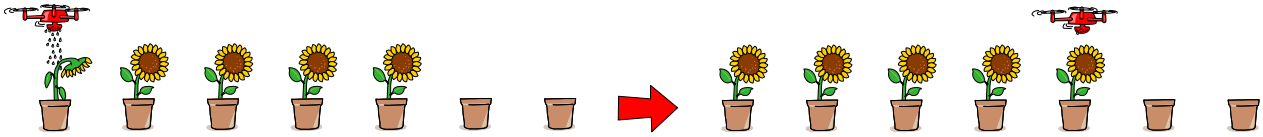
- Asymmetrische Kryptographie:
  `https://de.wikipedia.org/wiki/Asymmetrisches_Kryptosystem`
- Ende-zu-Ende-Verschlüsselung:
  `https://de.wikipedia.org/wiki/Ende-zu-Ende-Verschlüsselung`
- `https://de.wikipedia.org/wiki/Transport_Layer_Security`
- `https://de.wikipedia.org/wiki/Chatkontrolle`

© Informatik-Biber 2026, SVIA

14. Operation Giesskanne

Archie hat eine Drohne .

Er will, dass die Drohne über eine Reihe von Blumentöpfen fliegt und alle durstigen Blumen  darin giesst.



Archie hat das folgende Programm für die Drohne geschrieben.

Schritt 1:

Starte und fliege zum Topf ganz links.

Schritt 2:

Falls im Topf eine durstige Blume ist, giesse sie.
Sonst fliege einen Topf weiter nach rechts.

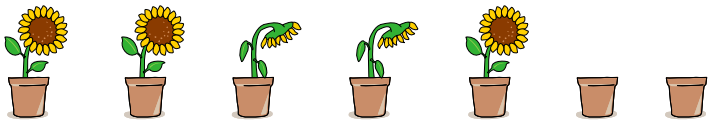
Schritt 3:

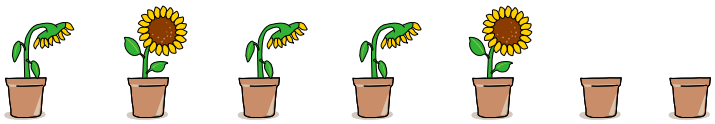
Falls im Topf keine Blume ist, lande.
Sonst fliege einen Topf weiter nach rechts und gehe zurück zu Schritt 2.

Die Drohne führt die Schritte des Programms nacheinander aus, bis sie landet. Dabei macht die Drohne nicht immer das, was Archie will. In einer dieser 4 Topfreihen wird nämlich eine durstige Blume NICHT gegossen.

In welcher?

A) 

B) 

C) 

D) 

Lösung

Antwort C ist richtig.

Archies Programm hat ein Problem. Sowohl Schritt 2 als auch Schritt 3 führen eine von zwei möglichen Anweisungen aus, abhängig davon, was der Blumentopf unter der Drohne enthält:

Optionen in Schritt 2:

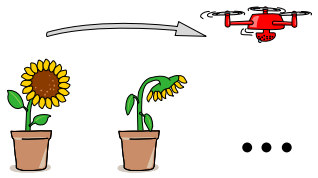
- Wenn der Topf eine durstige Blume enthält → *Blume giessen*
- Wenn der Topf keine durstige Blume enthält → *einen Topf weiter nach rechts fliegen*

Optionen in Schritt 3:

- Wenn der Topf keine Blume enthält → *landen*
- Wenn der Topf eine Blume enthält → *einen Topf weiter nach rechts fliegen*

Das bedeutet: Wenn ein Topf eine durstige Blume enthält, macht die Drohne den «Gießtest» von Schritt 2 (und giesst), fliegt in Schritt 3 zum nächsten Topf nach rechts, wo sie dann wieder Schritt 2 ausführen kann. Insgesamt «springt» die Drohne bezüglich der Ausführung von Schritt 2 um *einen* Topf nach rechts. Wenn ein Topf aber eine nicht-durstige Blume enthält (oder leer ist) und der Topf rechts daneben irgendeine Blume enthält, fliegt die Drohne sowohl in Schritt 2 als auch in Schritt 3 nach rechts. Für den so überflogenen Topf wird Schritt 2 also nicht ausgeführt (und eine durstige Blume darin nicht gegossen), sondern erst wieder für den Topf danach. In diesem Fall springt die Drohne also um *zwei* Töpfe nach rechts.

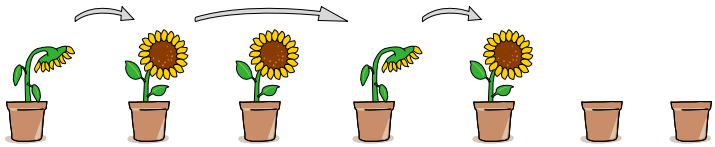
Weil in den Topfreihen leere Töpfe nur am Ende vorkommen, lassen wir leere Töpfe im Weiteren aussen vor und stellen fest: Eine durstige Blume wird übersprungen und *nicht* gegossen, wenn links neben ihr eine Blume steht, die nicht durstig ist und selbst nicht übersprungen wird.



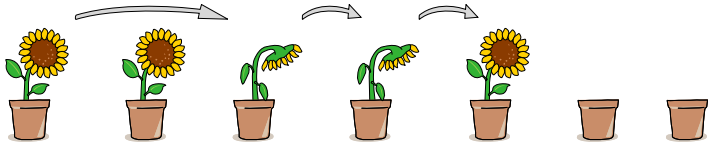
Diese Bedingung ist nur für die dritte Blume von links in Antwort C erfüllt. Auch wenn wir uns die Sprünge der Drohne für die vier Antworten ansehen, erkennen wir, dass nur in Antwort C eine durstige Blume übersprungen und nicht gegossen wird:

 © Informatik-Biber 2026, SVIA

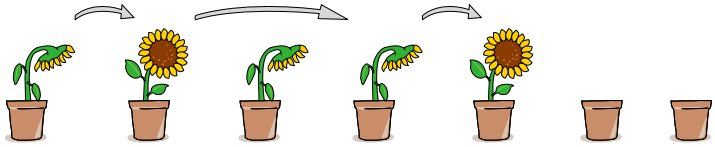
Antwort A:



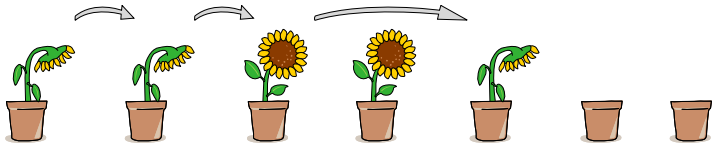
Antwort B:



Antwort C:



Antwort D:



Dies ist Informatik!

Die Drohne in dieser Biberaufgabe arbeitet einen *Algorithmus* ab. Ein Algorithmus ist eine Folge von Anweisungen, die eine konkrete Aufgabe löst. Informatiksysteme wie Computer, Smartphones oder Drohnen werden durch Algorithmen gesteuert. Diese werden typischerweise von Menschen erfunden und in für Informatiksysteme verständlichen Programmiersprachen beschrieben. Ein so enstandenes Programm nennt man in der Informatik auch die *Implementierung* des zugrundeliegenden Algorithmus.

Algorithmen oder auch ihre Implementierungen können Fehler enthalten, wenn etwas im Ablauf nicht den Absichten der Entwicklerin oder des Entwicklers entspricht. Solche Fehler nennt man in der Informatik oft *Bugs*; sie zu finden und zu beheben heisst deshalb *Debugging*. In dieser Biberaufgabe enthält der Algorithmus selbst einen Fehler, der darin besteht, dass in bestimmten Fällen ein Blumentopf mit einer durstigen Blume nicht gegossen wird.

Entwicklerinnen und Entwickler von Programmen testen diese mit verschiedenen Eingaben, um zu prüfen, ob ein Programm bzw. sein Algorithmus einen Fehler enthält. Eine solche Eingabe wird dann *Testfall* genannt. Die vier Topfreihen der Antworten A bis D sind also vier Testfälle. Aber nur einer von ihnen deckt Archies Fehler auf.

Stichwörter und Webseiten

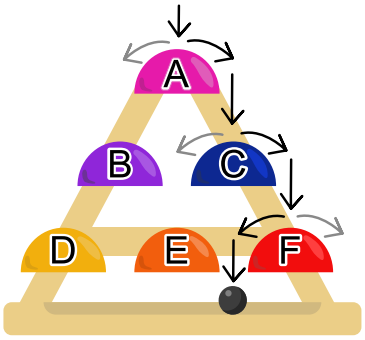
Algorithm, Debugging, Test Case

- Algorithmus: `https://de.wikipedia.org/wiki/Algorithmus`
- Debugging: `https://de.wikipedia.org/wiki/Debuggen`
- Softwaretest: `https://de.wikipedia.org/wiki/Softwaretest`

 © Informatik-Biber 2026, SVIA

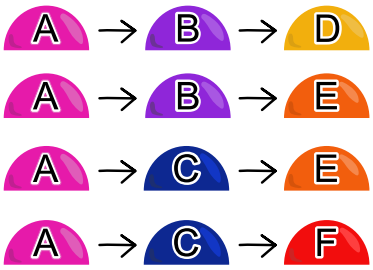
15. Murmelmusik

Marta hat einen Baukasten für musikalische Murmelbahnen. Er hat sechs Glocken, A bis F, die alle unterschiedlich klingen, wenn eine Murmel darauf fällt. Die Glocken können an einem Gestell an sechs Stellen befestigt werden. Rechts ist eine fertige Murmelbahn.



Eine Murmel läuft von oben nach unten durch eine Bahn und fällt auf ihrem Weg auf die oberste Glocke, eine Glocke in der zweiten und eine in der dritten Reihe. So erklingt eine Melodie aus drei Tönen. Auf jeder Glocke rollt die Murmel zufällig nach links oder rechts, so dass es verschiedene Murmelwege entlang den Glocken gibt.

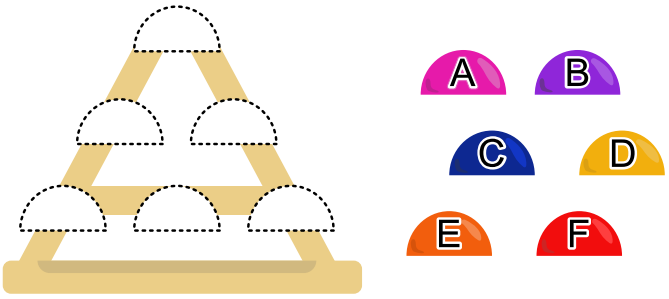
Bei der Murmelbahn oben gibt es diese Murmelwege:



Marta mag bestimmte Melodien sehr und andere gar nicht. Deshalb will sie eine Murmelbahn bauen, so dass es …

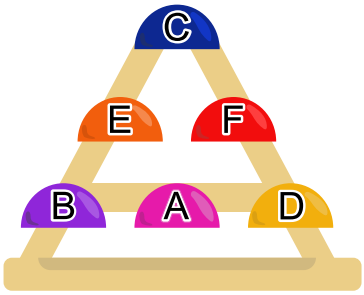
… mindestens diese Murmelwege, …	**… aber keinesfalls diese Murmelwege gibt:**
irgendein Weg mit den drei Glocken A, C, E	alle Wege mit den beiden Glocken B, F
irgendein Weg mit den beiden Glocken B, E	der Weg B → A → D
der Weg C → F → D	

Hilf Marta, die Glocken an den richtigen Stellen zu befestigen.

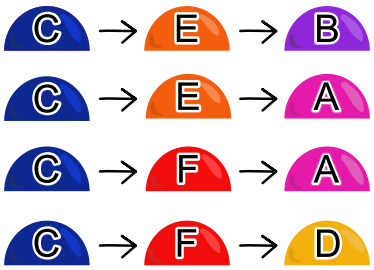


Lösung

So sind die Glocken richtig befestigt:



Es gibt diese «erlaubten» Murmelwege:



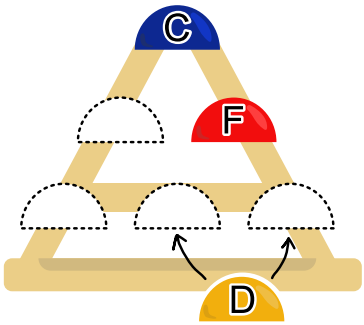
Darunter sind alle gewünschten, aber keine «verbotenen» Wege:

- Der Weg C → E → A ist ein Weg mit den drei Glocken A, C, E; der Weg C → E → B ist ein Weg mit den beiden Glocken B und E; und der Weg C → F → D ist auch enthalten.
- Die Murmel kann weder auf einem Weg mit den Tönen B und F noch auf dem Weg B → A → D laufen.

Wie lässt sich diese Befestigung der Glocken bestimmen? Wir müssen dafür sorgen, dass die Murmel auf allen gewünschten Wegen, aber nicht auf einem verbotenen Weg laufen kann. Dazu fügen wir nach und nach die gewünschten Wege in die Murmelbahn ein und vermeiden dann die verbotenen Wege.

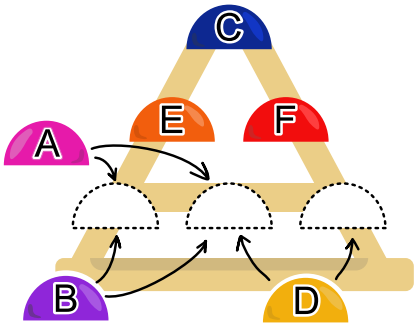
Schritt 1: Weg C → F → D einfügen

C muss oben, F in der zweiten Reihe und D in der dritten Reihe befestigt werden. Wir entscheiden uns für F für die rechte Stelle; dann muss D an die mittlere oder rechte Stelle der dritten Reihe.



Schritt 2: Wege mit A, C, E bzw. B, E einfügen

C ist bereits oben befestigt. Darunter muss E sowohl in einem Weg mit A als auch in einem mit B vorkommen. Dazu muss E oberhalb von A oder B befestigt werden. Also müssen E in der zweiten Reihe links sowie A und B in der dritten Reihe links oder mittig befestigt werden.



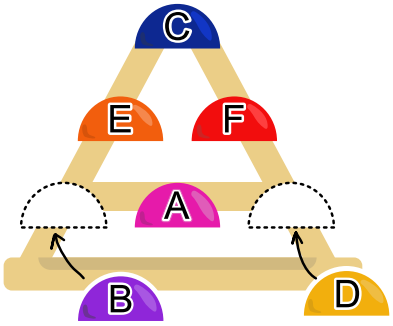
 © Informatik-Biber 2026, SVIA

Schritt 3: Weg B → A → D vermeiden

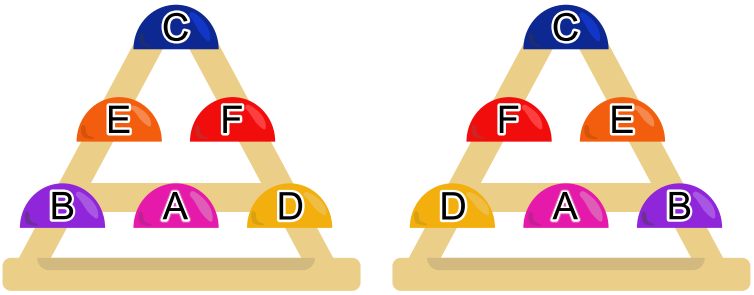
Weil C bereits oben befestigt ist, kann es keinen Weg geben, der mit B beginnt.

Schritt 4: alle Wege mit B und F vermeiden

Weil C bereits ganz oben und F in der zweiten Reihe befestigt ist, muss nur vermieden werden, dass die Murmel von F auf B rollen kann. Also müssen in der dritten Reihe B links, A in der Mitte und D rechts befestigt werden.



Insgesamt gibt es 2 richtige Lösungen, da die Anordnung der Glocken auch gespiegelt die selben 4 Murmelwege ergeben.



Dies ist Informatik!

Du hast die Glocken so befestigt und zu einer Murmelbahn kombiniert, dass es für die Murmel nur gewünschte Wege und keine verbotenen Wege gibt. Deine Kombination der Stellen für die verschiedenen Glocken erfüllt (engl.: satisfies) also Martas Bedingungen (engl.: constraints). In der Informatik kommen solche *constraint satisfaction problems* (CSPs) sehr häufig vor, zum Beispiel bei der Erstellung von Stundenplänen in einer Schule.

Dabei geht es üblicherweise darum, einzelne Elemente so auszuwählen, dass deren Kombination eine Menge von vorgegebenen Bedingungen erfüllt und damit eine Lösung darstellt. Ein einfacher Algorithmus für CSPs wäre also, solange verschiedene Kombinationen von Elementen auszuprobieren, bis eine Lösung gefunden ist. Wenn es aber viele Kombinationen gibt, kann das so viel Arbeit sein, dass es selbst für einen Computer sehr aufwändig wäre, eine Lösung so zu berechnen.

Eine bessere Strategie ist *Backtracking*: Dabei baut man mögliche Lösungen Schritt für Schritt auf und nimmt den letzten Schritt zurück, wenn ein Widerspruch auftaucht. So werden gleich viele ungeeignete Möglichkeiten ausgeschlossen. Auch in dieser Biberaufgabe lässt sich diese Strategie anwenden: Man wählt nach und nach für eine Stelle eine Glocke aus, so dass alle Bedingungen erfüllt sind. Wenn das an einer Stelle nicht möglich ist, geht man zur vorigen Stelle zurück und tauscht die dafür gewählte Glocke gegen eine andere aus.

Stichwörter und Webseiten

- Constraint satisfaction problems (CSP):
  `https://de.wikipedia.org/wiki/Constraint-Satisfaction-Problem`
- Backtracking: `https://de.wikipedia.org/wiki/Backtracking`

 © Informatik-Biber 2026, SVIA

16. Münzspiel

In einem Spiel gibt es

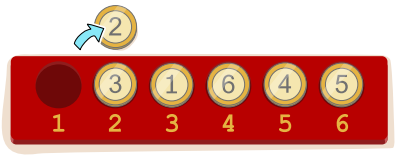
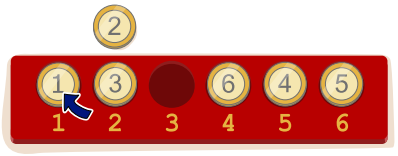
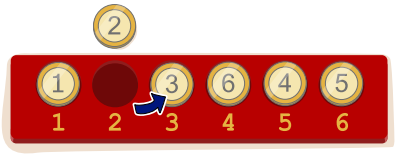
- 6 Münzen mit den Nummern 1 bis 6 und
- ein Spielbrett mit Plätzen 1 bis 6.

Zu Beginn des Spiels werden die Münzen beliebig auf die Plätze gelegt, zum Beispiel so:



Ziel des Spiels ist es, in möglichst wenigen Runden die Münzen auf die richtigen Plätze zu legen: Münze 1 auf Platz 1, Münze 2 auf Platz 2 und so weiter.

Eine Runde besteht aus den folgenden Schritten:

	Schritt	**Beispiel**
1	Nimm die erste Münze von links, die am falschen Platz liegt. Lege sie nach oben, über ihren richtigen Platz. Ein Platz wird frei.	
2	Lege so lange die richtige Münze auf den aktuell freien Platz (und mache dadurch einen anderen Platz frei), bis der Platz unter der nach oben gelegten Münze frei wird.	 
3	Lege die Münze von oben nun auf den freien, richtigen Platz. Damit ist die Runde zu Ende, und alle Plätze sind wieder belegt.	

Die Runde wird so lange wiederholt, bis alle Münzen am richtigen Platz liegen.

Nun beginnt das Spiel von vorne. Die Münzen werden wieder beliebig auf die Plätze gelegt, aber so, dass keine Münze auf dem richtigen Platz liegt.

*Wie viele Runden sind dann **mindestens** nötig, um jede Münze auf den richtigen Platz zu legen? Wie viele Runden sind dann **höchstens** dazu nötig?*

A) mindestens 1 Runde, höchstens 6 Runden

B) mindestens 2 Runden, höchstens 3 Runden

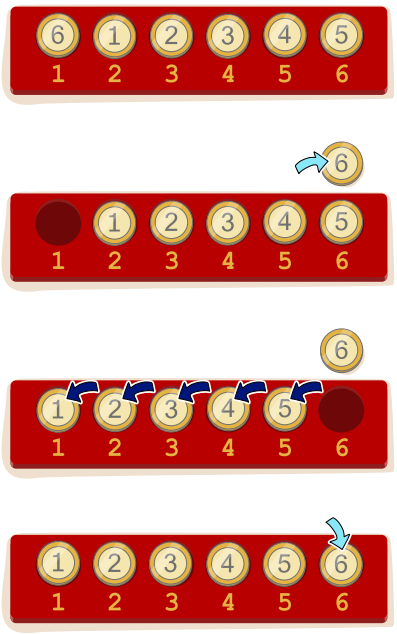
C) mindestens 1 Runde, höchstens 3 Runden

D) mindestens 2 Runden, höchstens 6 Runden

Lösung

Antwort C ist richtig.

Zuerst überlegen wir, wie viele Runden *mindestens* nötig sind, um jede Münze auf den richtigen Platz zu legen.



Da am Anfang keine Münze auf dem richtigen Platz liegt, ist sicher mindestens eine Runde nötig. Aber kann man mit einer Runde auskommen?

Links sieht man, dass das tatsächlich möglich ist. Alle Münzen liegen genau einen Platz rechts neben dem richtigen Platz; mit Ausnahme der Münze 6, die auf Platz 1 ganz links liegt.

- Im Runden-Schritt 1 wird Münze 6 nach oben gelegt, über Platz 6.
- In Schritt 2 können alle anderen Münzen um einen Platz nach links, also auf den richtigen Platz gelegt werden. Dann ist Platz 6 frei, so dass
- in Schritt 3 auch die Münze 6 auf den richtigen Platz gelegt werden kann.

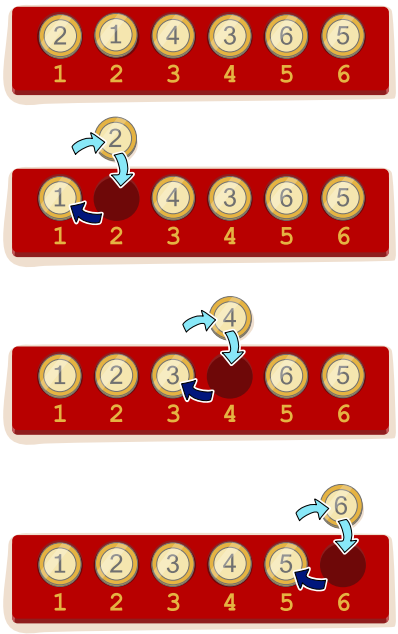
Und wie viele Runden sind *höchstens* nötig?

Im obigen Beispiel für eine Runde bilden alle sechs Münzen eine besondere Gruppe: Jede Münze liegt auf einem Platz, der für eine andere Münze aus dieser Gruppe richtig ist.

Die Münzen einer solchen Gruppe bilden einen *Zyklus*; im obigen Beispiel: $6 \to 1$ (bedeutet: Münze 6 liegt auf dem Platz von Münze 1) $\to 2 \to 3 \to 4 \to 5 \to 6$.

Im Beispiel aus der Aufgabenstellung gibt es zwei Zyklen: $2 \to 1 \to 3 \to 2$ und $6 \to 4 \to 5 \to 6$.

 © Informatik-Biber 2026, SVIA



In jeder Runde des Spiels wird genau ein Zyklus aufgelöst, werden also alle seine Münzen auf die richtigen Plätze gelegt.

Da ein Zyklus aus mindestens zwei Münzen besteht, können sechs Münzen höchstens drei Zyklen enthalten. In diesem für das Spiel schlechtesten Fall sind also höchstens drei Runden nötig, um jede Münze auf den richtigen Platz zu legen.

Links sieht man, wie die Münz-Belegung mit den Zyklen $2 \to 1 \to 2$, $4 \to 3 \to 4$ und $6 \to 5 \to 6$ in drei Runden sortiert wird.

Dies ist Informatik!

Beim Münzspiel in dieser Biberaufgabe werden die Münzen *sortiert*. Sortieren bedeutet, Daten nach einer bestimmten Vorschrift anzuordnen. Computerprogramme sortieren sehr häufig und ordnen zum Beispiel Zahlen nach dem Wert, Dateien nach dem Namen oder Daten im Arbeitsspeicher nach dem Zeitpunkt des letzten Zugriffs an. Sie tun das, weil der Zugriff auf sortierte Daten in der Regel sehr viel schneller ist als der auf unsortierte Daten. In allen obigen Beispielen ist die Grundlage für das Sortieren eine *Ordnung*, die Vergleiche zwischen Objekten erlaubt. Die Programme können also die Werte zweier Objekte miteinander vergleichen, um die Position der Objekte in der sortierten Anordnung festzulegen. Diese Art des Sortierens heisst deswegen *vergleichsbasiert*.

Im Münzspiel gibt jede Münze selbst an, wo sie hingehört, nämlich auf den Platz mit der gleichen Nummer wie die Münze. So lassen sich mit dem Münzspiel Münzen bzw. Zahlen sortieren, ohne dass Vergleiche der Werte nötig sind. Diese Art des Sortierens heisst deswegen *nicht-vergleichsbasiert*.

Weil Sortieren in Informatiksystemen so häufig passiert, ist es wichtig, dass Sortieralgorithmen möglichst wenig Ressourcen verbrauchen. Unser Münzspiel-Sortieralgorithmus zum Beispiel benötigt zur Ablage der Münzen fast nur die ohnehin nötigen Plätze auf dem Spielbrett. Abgesehen von dem zusätzlichen Platz für die Münze, die in Schritt 1 vorübergehend über ihren richtigen Platz gelegt wird, werden für alle anderen Münz-Bewegungen nur die vorhandenen Plätze benötigt. In der Informatik werden Sortieralgorithmen, die im Wesentlichen mit dem ohnehin für die zu sortierenden Objekte benötigten Speicherplatz auskommen, als «In-place-Algorithmen» bezeichnet, was bedeutet, dass sie die Objekte «an ihrem Platz» umordnen.

Ausser für den *Speicherbedarf* eines Algorithmus interessiert sich die Informatik insbesondere für seine *Laufzeit*: die Anzahl der Schritte, die ein Algorithmus abhängig von der Grösse der Eingabe machen muss. Die Laufzeit des Münzspiel-Algorithmus lässt sich anhand der Anzahl der Münzbewegungen messen. Jede Münze gehört zu genau einem Zyklus und wird deshalb nur in einer Runde bewegt,

genau einmal auf den richtigen Platz. Ausserdem wird in jeder Runde die erste Münze bewegt, über den richtigen Platz. Bei n Münzen und m Runden ist die Anzahl Münzbewegungen also $n + m$. Und weil die Anzahl der Runden höchstens $n/2$ sein kann, weil im schlechtesten Fall Zweier-Zyklen aufzulösen sind, ergibt sich eine maximale Anzahl von $n + n/2 = 3/2n$ Bewegungen. Eine solche Laufzeit, die im Wesentlichen der Eingabegrösse n entspricht, heisst in der Informatik *linear*. Nur nicht-vergleichsbasierte Sortieralgorithmen können mit linearer Laufzeit auskommen, sind aber, im Gegensatz zu vergleichsbasierten Algorithmen, nur unter bestimmten Bedingungen anwendbar.

Stichwörter und Webseiten

- Sortierverfahren: `https://de.wikipedia.org/wiki/Sortierverfahren`
- O-Notation: `https://de.wikipedia.org/wiki/Landau-Symbole`

 © Informatik-Biber 2026, SVIA

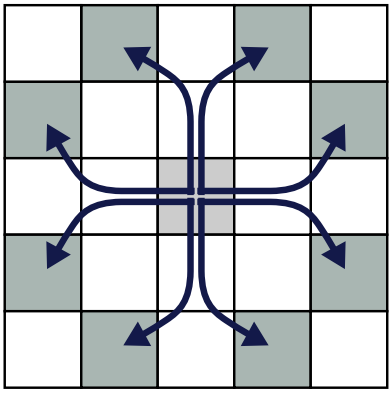
17. Sprung-Spiel



Ein Pferd springt über ein Spielbrett mit 25 Feldern.
Ein Sprung geht entweder über:

- 2 Felder horizontal und 1 vertikal oder
- 2 vertikal und 1 horizontal.

Das Bild zeigt alle möglichen Sprünge vom mittleren Feld aus.



Nun startet das Pferd auf Feld 21 und besucht mit mehreren Sprüngen ausschliesslich die nummerierten Felder. Dabei besucht es jedes nummerierte Feld genau einmal.

1		3		
	7			10
	12	13	14	
16		18		
21				25

In welcher Reihenfolge hat das Pferd die Felder besucht?

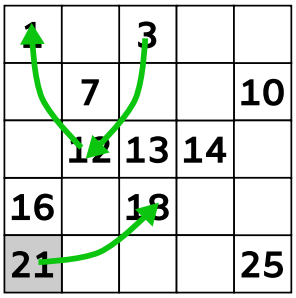
Lösung

So ist es richtig:

21	18	25	14	7	16	13	10	3	12	1

Aus den Regeln und dem Aufbau des Spielbretts lässt sich diese Sprungfolge ableiten:

- Das Pferd muss Feld 1 als letztes besuchen: Nur von Feld 12 aus kann es auf Feld 1 springen, und von dort aus kann es nur zurück zu Feld 12 - aber das ist nicht erlaubt. Somit sind die letzten zwei Felder 12, dann 1. (12→1)
- Auf 12 kann das Pferd von 3 oder 21 aus springen. Das Start-Feld 21 kann aber nicht das drittletzte Feld der Sprungfolge sein; das ist also Feld 3. (3→12→1)
- Vom Start-Feld 21 aus kann das Pferd auf die Felder 12 und 18 springen. Feld 12 ist aber schon das zweitletzte Feld in der Sprungfolge; das zweite Feld kann also nur Feld 18 sein (21→18).

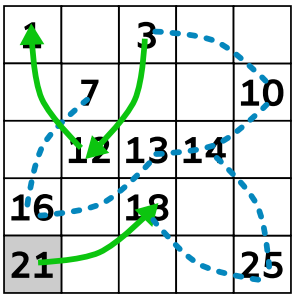


Wir kennen nun also den Beginn (21→18) und das Ende (3→12→1) der Sprungfolge (grün).

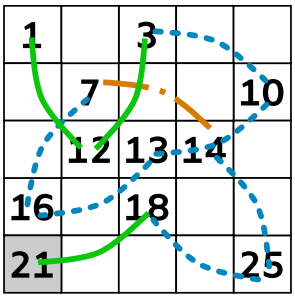
Einige weitere Überlegungen helfen die Route zu vervollständigen:

- Von den Feldern 25, 16 und 10 aus kann das Pferd jeweils zu genau zwei anderen Feldern springen.

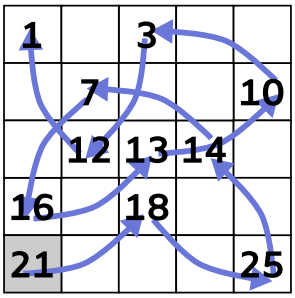
 © Informatik-Biber 2026, SVIA



Die Feldfolgen 7-16−13, 18-25−14, 13-10−3 müssen also Teile der Sprungfolge (blau) sein, nur die Richtung ist noch nicht klar.



Nur für die Felder 7 und 14 fehlt noch jeweils ein Sprung. Weil das Pferd von einem zum anderen Feld springen kann, ist auch 7−14 ein Teil der Sprungfolge (orange).

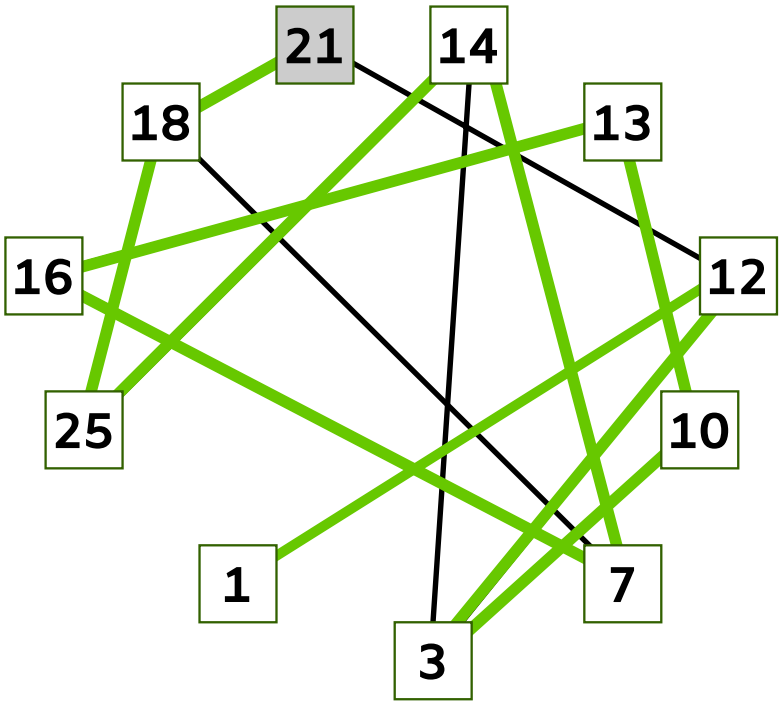


Nun sind wir fertig: Jedes nummerierte Feld wird genau ein Mal besucht. Die Sprungrichtungen innerhalb der Teilstücke ergeben sich, indem man von 21 startet und den Linien folgt.

Dies ist Informatik!

Viele Fragestellungen in der Informatik lassen sich mit Hilfe von *Graphen* modellieren. Ein Graph besteht aus Knoten, die Objekte repräsentieren, und Kanten, die Verbindungen zwischen den Objekten beschreiben. Die Objekte können alle möglichen Dinge sein, und so können beispielsweise Strassennetze, Flugverbindungen oder Kommunikationsnetze als Graphen dargestellt werden.

Auch in dieser Biberaufgabe ist ein Graph nützlich: Wenn wir die Felder als Knoten verstehen, stellen die Kanten die Möglichkeit dar, von einem Feld zu einem anderen zu springen. Hier ist der Pferd-Sprung-Graph; Kanten, die Teil der Sprungfolge sind, sind grün gezeichnet:



Eine wichtige Klasse von Graphenproblemen besteht darin, einen Weg zu finden, der bestimmte Bedingungen erfüllt. Ein bekanntes Beispiel ist die Suche nach einem *Hamilton-Pfad:* Das ist ein Weg entlang von Kanten des Graphen, der jeden Knoten genau einmal besucht. Die in dieser Biberaufgabe gesuchte Sprungfolge ist also ein Hamilton-Pfad.

Hamilton-Pfade haben besonders dann interessante Anwendungen, wenn die Kanten des Graphen Gewichte haben, etwa wenn sie für Enfernungen zwischen Orten stehen. Wer dann die kürzeste Tour durch alle Orte sucht, sucht im Orte-Entfernungen-Graph den Hamilton-Pfad (oder genauer: den Hamilton-Kreis, da eine Tour zum Ausgangsort zurückführt) mit geringstem Gesamtgewicht der im Pfad vorkommenden Kanten.

Obwohl die Frage «Gibt es in einem Graph einen Hamilton-Pfad?» einfach formuliert werden kann, gehört sie zu den klassischen schwierigen Problemen der Informatik. Für beliebige Graphen ist bis heute kein effizienter Algorithmus bekannt, der einen Hamilton-Pfad immer findet – oder nachweist, dass keiner existiert.

Stichwörter und Webseiten

- Hamilton-Pfad: `https://de.wikipedia.org/wiki/Hamiltonkreisproblem#Eigenschaften`

 © Informatik-Biber 2026, SVIA

18. Versteckspiel

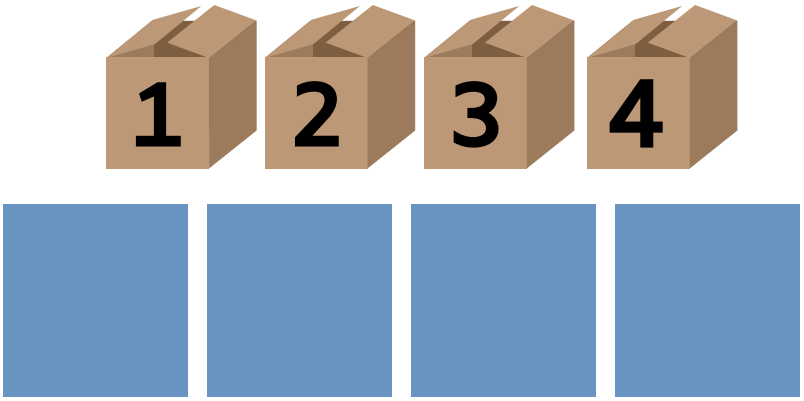
Ben versteckt sich in einer von vier nummerierten Kisten, z.B. in Kiste 2.



Seine Freundin Anja will ihn finden und schaut jeweils in genau einer Kiste nach. Findet sie Ben dort nicht, bewegt sich Ben unbemerkt in eine benachbarte Kiste (eine Kiste nach links oder rechts), bevor Anja erneut in einer Kiste nachschaut.

Anja überlegt: Wenn sie in der richtigen Reihenfolge in die Kisten schaut, muss sie höchstens 4-mal nachschauen, um Ben auf jeden Fall zu finden, egal wie er sich bewegt.

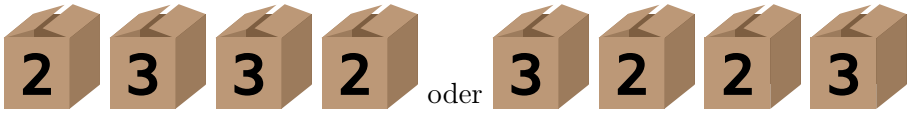
Gib eine solche Reihenfolge an!



Die Kisten können in die einzelnen Felder gezogen werden. Die Kisten können mehrfach verwendet werden. Man kann eine Kiste in ein anderes Feld schieben, dadurch werden die Kisten getauscht.

Lösung

Um Ben auf jeden Fall mit 4-mal Nachschauen zu finden, muss Anja in einer dieser beiden Reihenfolgen in die Kisten schauen:

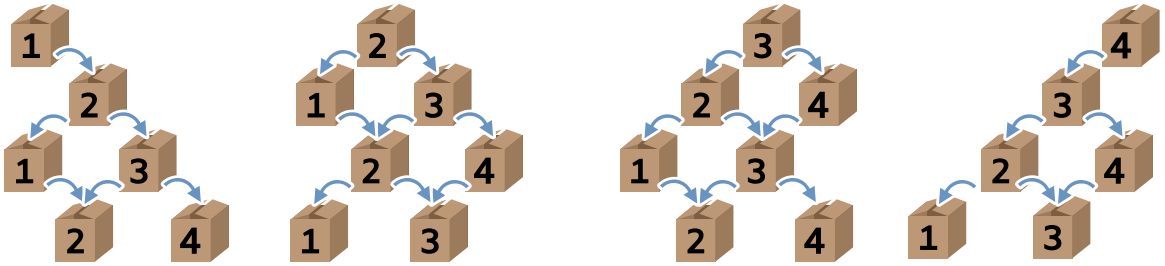


Wir zeigen, dass diese Antworten richtig sind. Dabei bezeichnen wir Bens Bewegungen als *Zug* in diesem Versteckspiel.

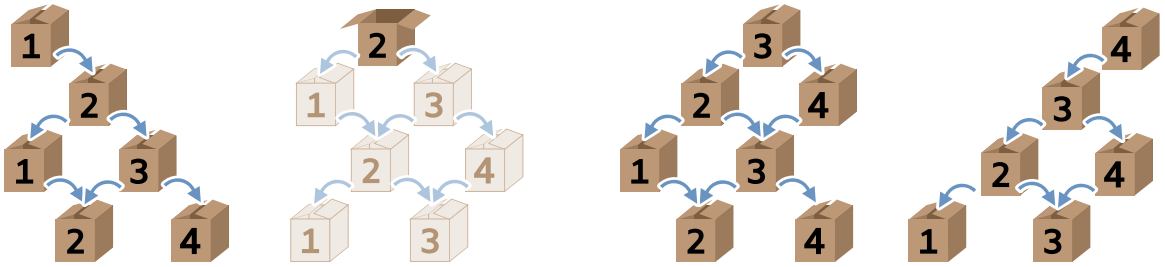
Am Anfang kann Ben seine «Start-Kiste» frei wählen. Danach hat Ben, wenn er

- in Kiste 2 oder 3 sitzt, jeweils zwei Möglichkeiten für den nächsten Zug: entweder geht er nach links oder nach rechts;
- in Kiste 1 oder 4 sitzt, jeweils eine Möglichkeit für den nächsten Zug: er geht nach rechts (von Kiste 1 aus) bzw. nach links (von Kiste 4 aus).

Ben macht seinen Zug, nachdem Anja in einer Kiste nachgeschaut und ihn nicht gefunden hat. Wenn Anja ihn mit (höchstens) 4-mal Nachschauen finden soll, kann er bis dahin höchstens drei Züge machen. Deshalb schauen wir uns alle möglichen Folgen von drei Zügen an, und zwar von jeder möglichen Start-Kiste (1–4) aus:

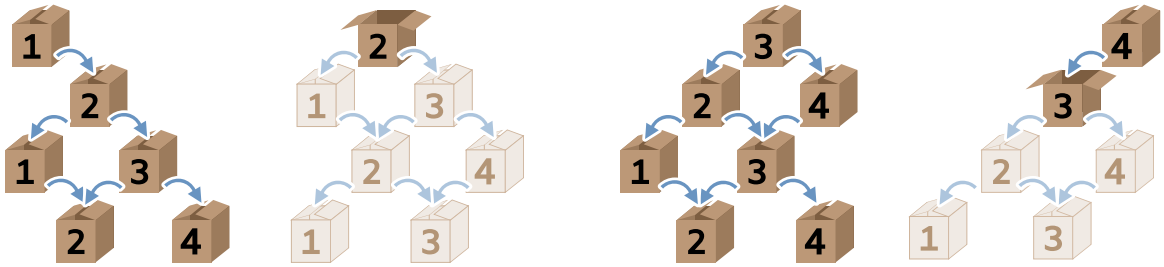


Was passiert, wenn Anja in der Reihenfolge **2-3-3-2** in die Kisten schaut? Zuerst schaut Anja in Kiste 2. Wenn sie Ben dort findet, ist Kiste 2 seine Start-Kiste, und er kann die von dort ausgehenden Züge nicht mehr machen.

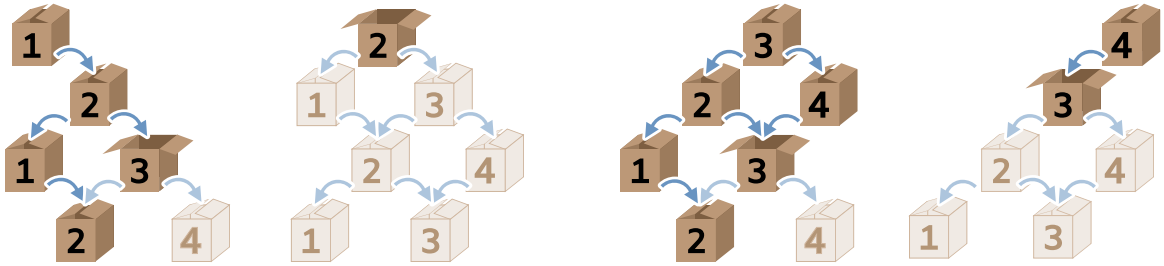


Dann schaut Anja in Kiste 3. Wenn sie Ben dort findet, ist das seine zweite Kiste. Auf der zweiten Ebene der Zugfolgen kommt Kiste 3 noch ganz rechts vor. Die von dort ausgehenden Züge kann Ben nun nicht mehr machen, und wir brauchen sie nicht weiter zu betrachten.

 © Informatik-Biber 2026, SVIA



Als Drittes schaut Anja wieder in Kiste 3. Diese kommt in den Zugfolgen noch an zwei Stellen auf der dritten Ebene vor. Die von dort ausgehenden Züge kann Ben nun nicht mehr machen.



Auf der vierten Ebene der Zugfolgen kommt jetzt nur noch Kiste 2 vor. Also kann nur Kiste 2 Bens vierte Kiste sein, falls Anja ihn bis dahin noch nicht gefunden hat. Wenn Anja jetzt in Kiste 2 schaut, muss Ben in dieser Kiste sein.

Für die Antwort **3–2–2–3** gilt die gleiche Begründung spiegelbildlich, denn sowohl die Zugfolgen von den Start-Kisten 1 und 4 aus als auch die von 2 und 3 aus sind spiegelbildlich zueinander.

Dies ist Informatik!

In der Informatik werden mögliche Abläufe, von einem oder mehreren Startpunkten aus, oft als Zustandsgraphen oder Bäume dargestellt. Das sind Datenstrukturen, die zum einen aus Knoten und zum anderen aus Kanten zwischen jeweils zwei Knoten bestehen. Jeder Knoten steht für einen möglichen Zustand. Jede Kante steht für einen möglichen nächsten Schritt von einem zu einem nächsten Zustand und hat deshalb eine Richtung.

Solche Darstellungen möglicher Abläufe kann man u.a. dann verwenden, wenn es zwei Akteure gibt, die gegensätzliche Ziele verfolgen, wie in dieser Biberaufgabe: Ben will sich vor Anja verstecken, Anja will Ben finden. Mit Hilfe der Graphen für die Zugfolgen konnten wir aus Sicht von Akteur Anja systematisch alle Möglichkeiten des Verhaltens ihres «Gegners» Ben untersuchen und so eine Strategie finden, die für Anja garantiert zum Erfolg führt. Solche Vorgehensweisen finden unter anderem in der *Spieltheorie*, bei der Planung von Algorithmen und in der Robotik Verwendung.

Stichwörter und Webseiten

- Binärbaum: `https://de.wikipedia.org/wiki/Binärbaum`
- Spieltheorie: `https://de.wikipedia.org/wiki/Spieltheorie`

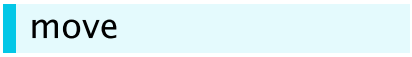
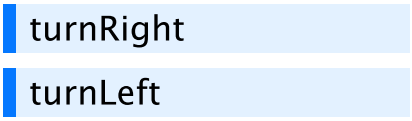
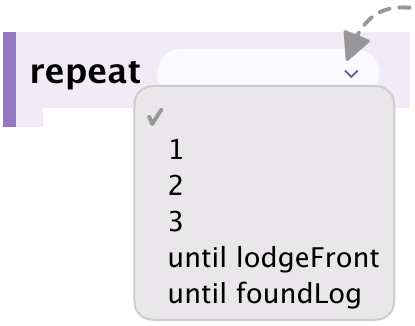


© Informatik-Biber 2026, SVIA

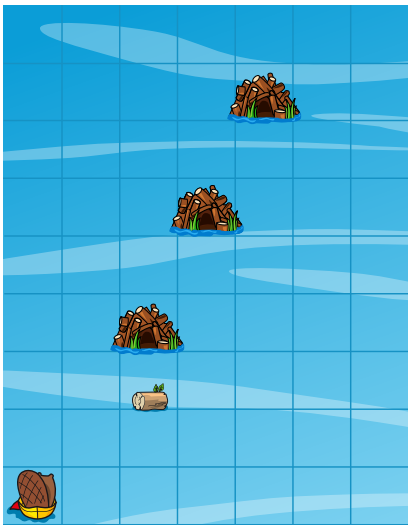
19. Quartierhilfe

Biber Belinda braucht dringend Baumstämme. Jeden Tag legt ein anderer Nachbar einen Baumstamm vor den Bau. Belinda möchte an jedem Tag mit der gleichen Wegbeschreibung den Baumstamm einsammeln. Hat Belinda den Baumstamm eingesammelt, will sie nicht mehr bei den nachfolgenden Nachbarn vorbeifahren. Kannst du Belinda helfen, diese Wegbeschreibung zu finden?

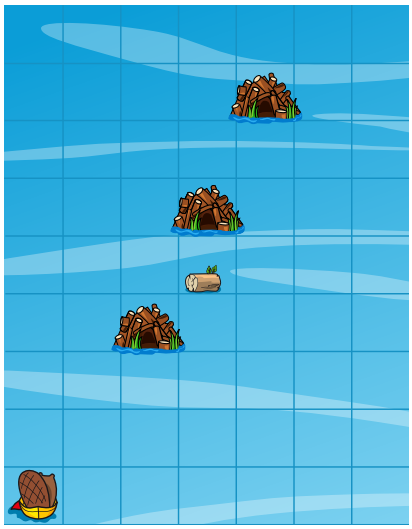
Du kannst folgende Anweisungen verwenden:

Anweisung	**Erklärung**
	Belinda bewegt sich genau ein Feld in Blickrichtung nach vorne. Landet Belinda auf einem Feld mit einem Baumstamm, sammelt sie diesen automatisch ein.
	Belinda dreht sich am Ort nach rechts / links.
	Wenn du Anweisungen in den Block ziehst, werden diese wiederholt (`repeat` = wiederhole).
	Klicke auf das helle Feld und wähle aus, wie oft die Anweisungen wiederholt werden. Bei `until` wird geprüft, ob: • Belinda direkt vor einem Biberbau steht (`lodgeFront`). • Belinda den Baumstamm eingesammelt hat (`foundLog`). Trifft dies zu, endet die Wiederholung nach der letzten Anweisung im `repeat`-Block. Wählst du nichts aus, werden die Anweisungen *dauerhaft* wiederholt.

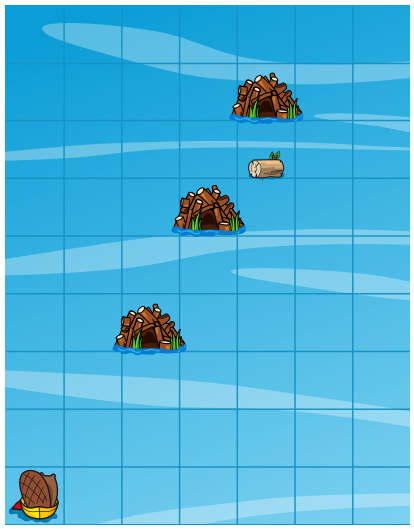
Notiere eine Wegbeschreibung, mit der Belinda an allen drei Tagen den Baumstamm einsammelt. Sobald sie ihn hat, soll sie bei keinem nachfolgenden Nachbarn vorbeifahren.



Tag 1



Tag 2



Tag 3

 © Informatik-Biber 2026, SVIA

Lösung

Eine richtige Lösung lautet wie folgt:

```
repeat until foundLog
  turnRight()
  move()
  move()
  turnLeft()
  move()
  move()
  turnLeft()
  move()
  turnRight()
```

Um an allen Tagen zum Baumstamm zu gelangen, muss man als Erstes erkennen, dass der Weg durch ein wiederkehrendes Muster beschrieben werden kann:

```
  turnRight()
  move()
  move()
  turnLeft()
  move()
  move()
  turnLeft()
  move()
  turnRight()
```

Um am dritten Tag bis zum Nachbarn mit dem Baumstamm zu gelangen, schreibt man die Anweisungen des Musters dreimal hintereinander auf. Damit sammelt sie auch am ersten und zweiten Tag den Baumstamm auf. Die Wegbeschreibung würde somit insgesamt 27 Anweisungen enthalten. Um die Beschreibung zu kürzen, kann man die Anweisungen des Musters auch wiederholt ausführen lassen. Dazu packt man sie in einen `repeat` - Block und wählt `3` Wiederholungen aus:

```
repeat 3
  turnRight()
  move()
  move()
  turnLeft()
  move()
  move()
  turnLeft()
  move()
  turnRight()
```

Damit enthält die Wegbeschreibung statt 27 nur noch zehn Befehle.

Wir haben jetzt eigentlich eine Wegbeschreibung für Belinda gefunden. Allerdings ist die Lösung mit `repeat` und einer festen Anzahl von Wiederholungen nicht ganz zufriedenstellend. Denn wenn das Biberdorf nicht aus drei, sondern z.B. aus fünf Bauten bestehen würde, müssten wir immer die Anzahl der Wiederholungen in der Wegbeschreibung anpassen.

Das Problem lösen wir, indem wir Belindas zweite Einschränkung beachten: Hat Sie einen Baumstamm eingesammelt, möchte sie nicht mehr bei den nachfolgenden Nachbarn vorbeifahren. Genau das wäre aber der Fall, wenn wir `repeat` immer mit einer festen Anzahl an Wiederholungen festlegen. Damit würde Belinda am ersten Tag, obwohl sie den Baumstamm bereits beim ersten Nachbarn eingesammelt hat, immer noch zu Nachbar zwei und drei fahren. An Tag 2 wäre es ähnlich.

Daher nutzen wir die Bedingung `until foundLog` im `repeat` - Block statt einer konkreten Zahl:

```
repeat until foundLog
  ...
```

Denn jetzt werden die Anweisungen des Musters nur so lange wiederholt, bis Belinda den Baumstamm eingesammelt hat. Bevor die Anweisungen im `repeat`- Block erneut ausgeführt werden, wird mithilfe der Bedingung `until foundLog` überprüft, ob bereits ein Baumstamm gefunden wurde. Wenn ja, werden die Anweisungen nicht nochmals ausgeführt und sie fährt nicht mehr zu den nachfolgenden Nachbarn weiter.

Beachte: Obwohl Belinda den Baumstamm bereits nach dem vierten `move` aufgesammelt hat und damit die Bedingung `until foundLog` eigentlich erfüllt ist, werden dennoch die drei nachfolgenden Befehle ausgeführt. Denn die (erneute) Überprüfung mithilfe der Bedingung `until foundLog` erfolgt immer erst dann, wenn alle Befehle im `repeat` - Block vollständig ausgeführt worden sind. Daher stoppt Belinda nicht direkt nach dem Einsammeln des Baumstamms, sondern fährt noch weiter zu ihrer nächsten Startposition, jedoch nicht mehr zum nächsten Nachbar.

```
repeat until foundLog
  turnRight()
  move()
  move()
  turnLeft()
  move()
  move() <-- hier sammelt Belinda den Baumstamm auf
  turnLeft()
  move()
  turnRight() <- hier erreicht sie ihre neue Startposition
```

Vertiefung: Wir haben den Weg über ein Muster beschrieben. Das Muster wird im `repeat` - Block in zwei Abschnitte unterteilt:

Abschnitt 1: Weg bis zum Baumstamm

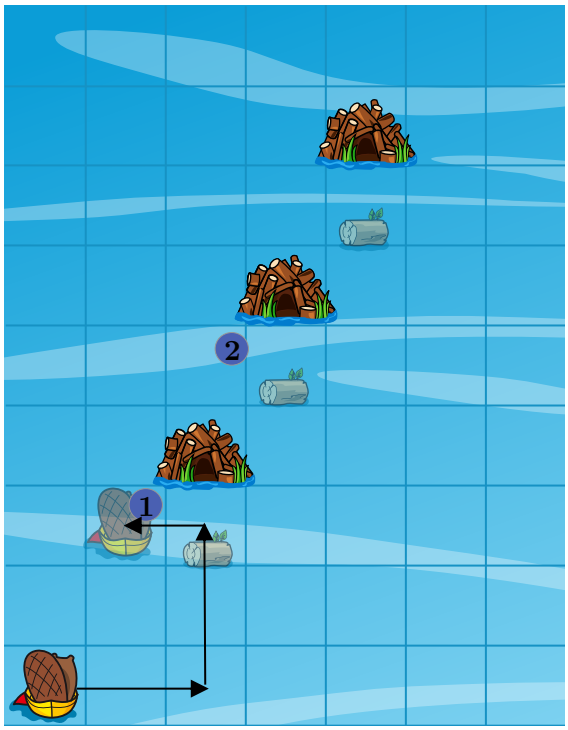
```
  turnRight()
  move()
```

```
move()
turnLeft()
move()
move()
```

Abschnitt 2: Weg weiter zur neuen Startposition

```
turnLeft()
move()
turnRight()
```

Unterteilst du dein Muster in diese beiden Abschnitte, kann Belinda einen beliebigen Weg zum Baumstamm **und** einen beliebigen Weg zur neuen Startposition (Position 1 für den zweiten Baumstamm, Position 2 für den dritten Baumstamm) fahren. Dennoch sammelt Belinda immer alle Baumstämme auf und fährt nie weiter zum nächsten Nachbarn.



Dies ist Informatik!

Eine *Invariante* (hier: *Schleifeninvariante*) ist eine Eigenschaft, die vor und nach jedem Durchlauf einer Schleife gültig ist. Sie hilft dabei, den Zustand eines Systems zu beschreiben und die Korrektheit eines Verfahrens zu überprüfen. Besonders häufig werden Invarianten bei Schleifen verwendet, um zu zeigen, dass bestimmte Bedingungen bei jedem Schleifendurchlauf erhalten bleiben. Bezogen auf unser Beispiel müssen wir unseren Biber so ausrichten, dass Startposition (vor dem Schleifendurchlauf) und Endposition (nach dem Schleifendurchlauf) identisch sind. Dabei definiert sich die jeweilige Start-/Endposition durch die Blickrichtung und durch den Standort des Bibers in Relation zum nächsten Biberbau.

Stichwörter und Webseiten

- Schleifeninvariante: `https://de.wikipedia.org/wiki/Schleifeninvariante`
- Wiederholung: `https://de.wikipedia.org/wiki/Schleife_(Programmierung)`

 © Informatik-Biber 2026, SVIA

A. Aufgabenautoren

Sakuya Araki

Masiar Babazadeh

Liam Baumann

Wilfried Baumann

Nils Böhnke

Leonardo Cavalcante

Maria Cepeda

Špela Cerar

Valentina Dagienė

Christian Datzko

Nora A. Escherle

Susanne Falkenberg-Thut

Nadine Fröhlich

Gerald Futschek

Daniel Aagrén Geert

Arnheiður Guðmundsdóttir

Jens Hartmann

Heikki Hyyro

Heidi Kaarto

Filiz Kalelioğlu

Asterios Karagiannis

Geetanjali Khatri

Vaidotas Kinčius

Patrik Klofáč

Lukas Lehner

Vaida Masiulionytė-Dagienė

Yoshiaki Matsuzawa

Anna Morpurgo

Thomas Naughton

Justina Oostendorp

Belén Palop

Praphan Pavarangkoon

Jean-Philippe Pellet

Emmanuel Plan

Zsuzsa Pluhár

Wolfgang Pohl

Liane Raji

Lorenzo Repetto

Kirsten Schlüter

Margareta Schlüter

Dirk Schmerenbeck

Eljakim Schrijvers

Darija Dasovic Sebalj

Giovanni Serafini

Maiko Shimabuku

Itsuki Shimada

Bronius Skūpas

Jacqueline Staub

Alieke Stijf

Maciej M. Syslo

Yahya Tabesh

Ezra Templonuevo

Jiří Vaníček

Marcel Vlastuin

Michael Weigend

Kyra Willekes

Yichan Yuan

© Informatik-Biber 2026, SVIA

B. Akademische Partner

Haute école pédagogique du canton de Vaud
https://www.hepl.ch/

Ausbildungs- und Beratungszentrum für Informatikunterricht der ETH Zürich
https://www.abz.inf.ethz.ch/

La Scuola universitaria professionale della Svizzera italiana (SUPSI)
https://www.supsi.ch/

Pädagogische Hochschule Luzern
https://www.phlu.ch/

Universität Trier
https://www.uni-trier.de/

C. Sponsoring

Hasler Stiftung
https://www.haslerstiftung.ch/

Amt für Kindergarten, Volksschule und Beratung, Bildungs- und Kulturdirektion, Kanton Bern
https://www.bkd.be.ch/de/start/ueber-uns/die-organisation/amt-fuer-kindergarten-volksschule-und-beratung.html

Amt für Wirtschaft, Kanton Zürich
https://www.zh.ch/de/volkswirtschaftsdirektion/amt-fuer-wirtschaft.html

Informatik Stiftung Schweiz
https://informatics-foundation.ch

cyon
https://www.cyon.ch

Senarclens Leu & Partner
https://senarclens.com/

Wealth Management IT and UBS Switzerland IT
https://www.ubs.com/

Basellandschaftliche Gebäudeversicherung
https://bgv.ch

© Informatik-Biber 2026, SVIA