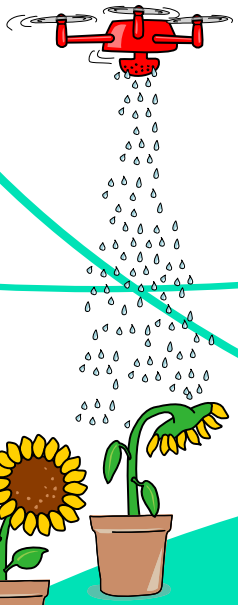


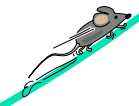


INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

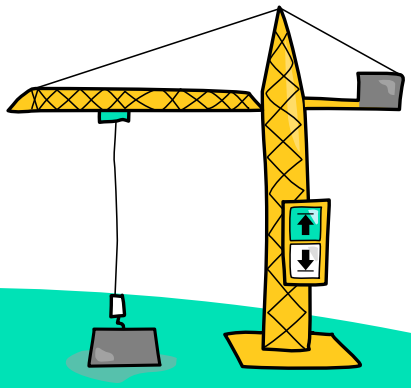
Aufgaben und Lösungen 2026

Schuljahre 3/4



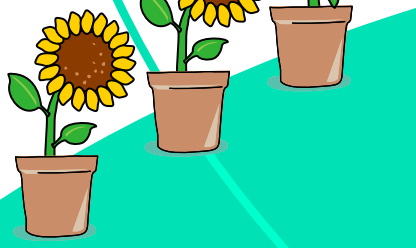


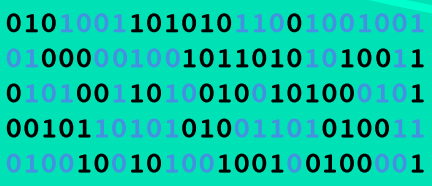
https://www.informatik-biber.ch/



Herausgeber:

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

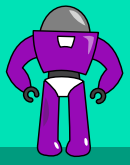




SVIA

www.svia-ssie-ssii.ch
schweizerischer verein für informatik in
der 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. 55 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. Die scheue Ameise 2026-MX-03 1

2. Apfel essen 2026-US-02 5

3. Batterien 2026-BR-02 7

4. Chläberli 2026-EC-01 11

5. Videos 2026-HU-01a 15

6. Kran 2026-CZ-05 19

7. Schneemobil-Taxi 2026-AT-01 23

8. Azulejos 2026-CH-06 27

9. Fisch-App 2026-DE-08 31

10. Kopf oder Zahl 2026-IN-05 35

11. Geralds Garten 2026-LT-01a 37

12. Tacos für lau! 2026-MX-02b 41

13. Versteckspiel 2026-US-03b 45

14. Quartierhilfe 2026-DE-90 49

A. Aufgabenautoren 55

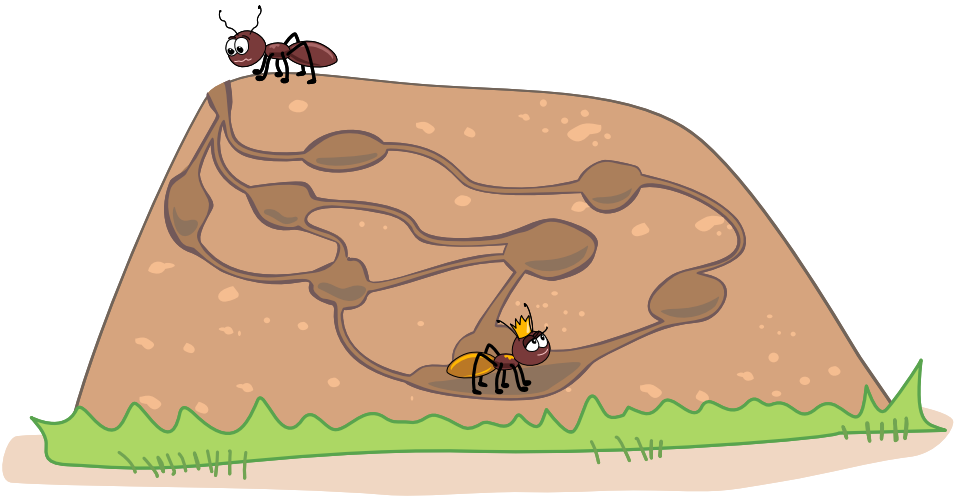
B. Akademische Partner 56

C. Sponsoring 57

© Informatik-Biber 2026, SVIA

1. Die scheue Ameise

Ein Ameisenbau hat mehrere Räume. Die Räume sind mit Gängen verbunden.
So hat jeder Raum 2 oder mehr Zugänge.



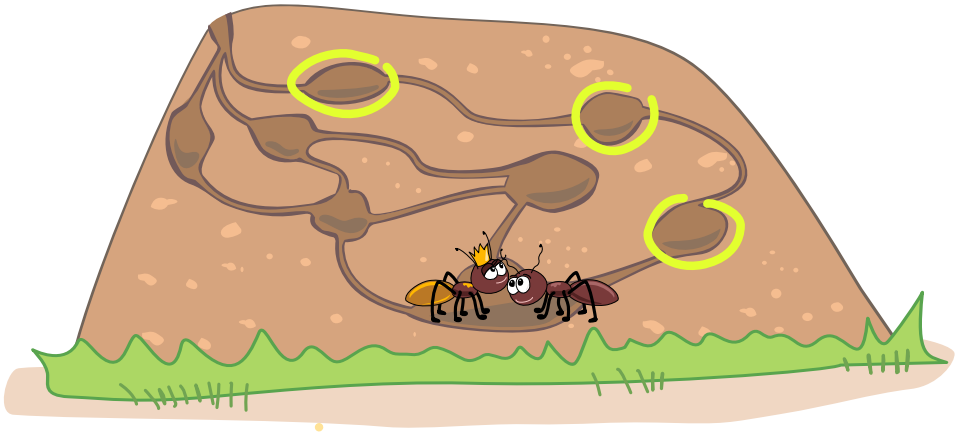
Die scheue Ameise Ani  geht zur Königin  .

Ani geht aber nur durch Räume mit genau 2 Zugängen.

Durch welche Räume geht Ani zur Königin?

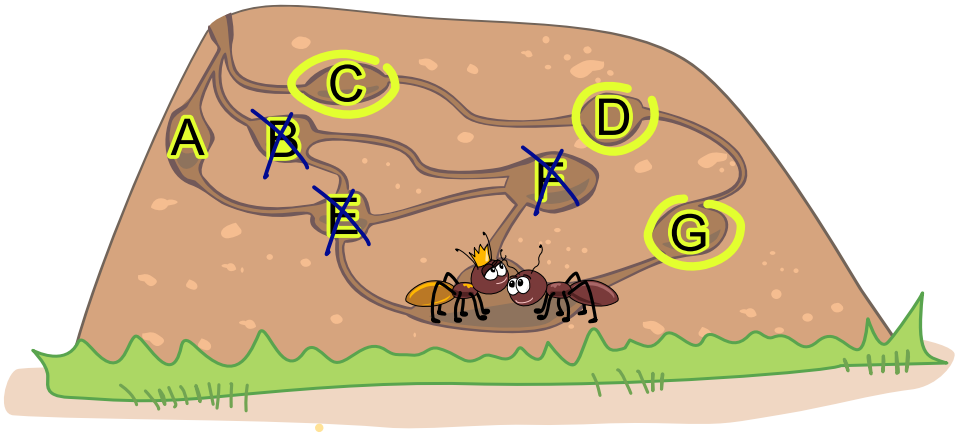
Lösung

So ist es richtig:



Zähle zunächst für jeden Raum die Anzahl der Zugänge. Ani geht nicht durch diese Räume mit mehr als 2 Zugängen:

- Raum **B** hat 3 Zugänge
- Raum **E** hat 4 Zugänge
- Raum **F** hat 3 Zugänge



Raum **A** hat 2 Zugänge. Ani könnte also durch Raum A gehen. Aber von dort aus kommt sie nicht weiter, weil der einzige nächste Raum **E** 4 Zugänge hat.

Die Räume **C**, **D** und **G** haben jeweils 2 Zugänge. Der Weg durch die Räume **C**, **D** und **G** ist der einzige Weg, den Ani zur Königin gehen kann. Alle anderen Wege zur Königin führen durch mindestens einen Raum mit mehr als 2 Zugängen.

Dies ist Informatik!

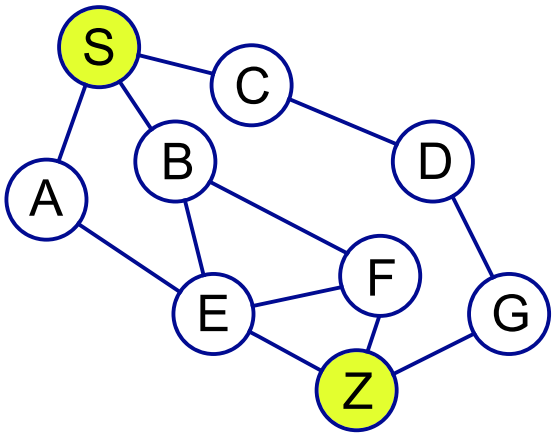
Die Räume im Ameisenbau sind durch die Gänge miteinander verbunden. So bilden Räume und Gänge zusammen ein Netzwerk. Es gibt viele Beispiele für Netzwerke: Computer, die miteinander

 © Informatik-Biber 2026, SVIA

über das Internet verbunden sind (mit oder ohne Kabel); Orte und Strassen, die die Orte verbinden; Menschen, die durch Freundschaften miteinander verbunden sind; usw.

Viele Computerprogramme arbeiten mit Netzwerken. Ein gutes Beispiel sind Routenplaner, die mit grossen Netzen aus Strassen und anderen Verkehrswegen arbeiten. Für die Darstellung von Netzwerken verwendet die Informatik meist *Graphen*, die aus *Kanten* (für die Verbindungen) und *Knoten* (für die Dinge, die miteinander verbunden werden sollen) bestehen.

Die Räume und Gänge im Ameisenbau kann man auch als Graph darstellen. Die Knoten (als Kreise gezeichnet) sind die Räume, plus Eingang S, und die Kanten (Linien) sind die Gänge:



Die Informatik kennt viele unterschiedliche Algorithmen, um Graphen und ihre Eigenschaften zu untersuchen. Im Ameisenbau-Graph haben wir den *Knotengrad* untersucht, das ist die Anzahl der mit einem Knoten verbundenen Kanten.

In dieser Biberaufgabe sucht die scheue Ameise Ani einen Weg von S nach Z, der nur über Knoten mit Grad 2 geht. Auch in Computernetzwerken suchen «scheue», also sensible Daten, Wege über Knoten mit geringem Vernetzungsgrad. Denn Daten-Angreifer bevorzugen stark vernetzte und deshalb viel besuchte Knoten (sogenannte *Hubs*), um dort möglichst viele Daten abfangen zu können.

Stichwörter und Webseiten

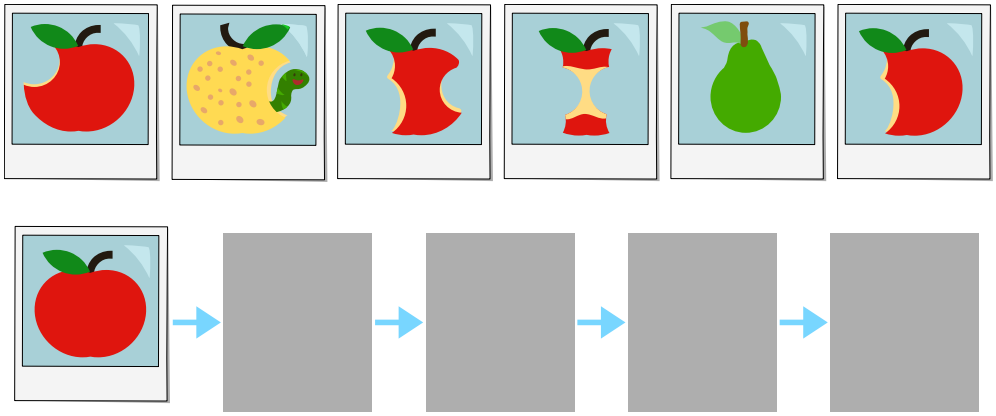
- Graphentheorie: `https://de.wikipedia.org/wiki/Graphentheorie`
- Grad von Knoten: `https://de.wikipedia.org/wiki/Grad_(Graphentheorie)`
- Routing: `https://ausbildung-in-der-it.de/lexikon/routing`

© Informatik-Biber 2026, SVIA

2. Apfel essen

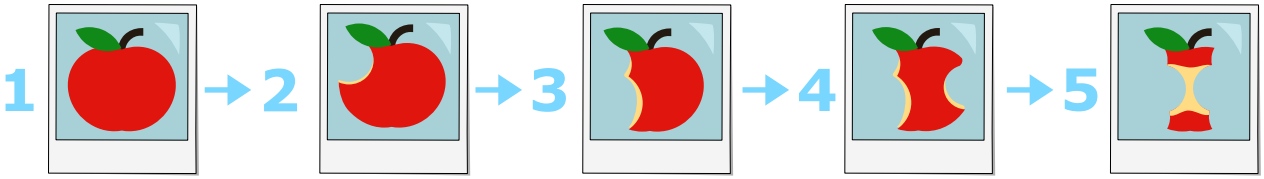
Dani hat einen roten Apfel gegessen.
Dabei hat Dani Fotos vom Apfel gemacht.
Die Fotos sind durcheinandergeraten, auch mit anderen Fotos.

In welcher Reihenfolge hat Dani die Fotos vom roten Apfel gemacht?



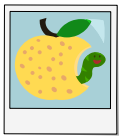
Lösung

So ist es richtig:



Wenn man einen Apfel isst, bleibt immer weniger vom Apfel übrig. In der richtigen Reihenfolge muss also auf jedem Foto weniger Apfel zu sehen sein als auf dem Foto davor – so wie oben:

1. Der rote Apfel ist noch ganz.
2. Der rote Apfel ist einmal angebissen.
3. Der rote Apfel ist fast zur Hälfte gegessen.
4. Der rote Apfel ist auf der anderen Seite angebissen.
5. Vom roten Apfel ist nur noch das Gehäuse übrig.

Die Fotos  und  sind keine Fotos vom roten Apfel.

Dies ist Informatik!

Auf Danis Fotos vom roten Apfel ist unterschiedlich viel vom Apfel übrig. Deshalb kann man zwei Fotos miteinander vergleichen und genau feststellen: Das Foto, auf dem mehr vom Apfel zu sehen ist, hat Dani vor dem anderen Foto gemacht. Das ist so wie bei zwei unterschiedlichen Zahlen, für die man genau feststellen kann, welche von beiden kleiner ist als die andere. Für die Fotos kann man so wie für Zahlen sagen, dass sie eine Ordnung haben.

Wenn Dinge eine Ordnung haben, kann man sie in einer Reihenfolge anordnen: Zahlen nach ihrer Grösse und die Fotos vom roten Apfel danach, wann Dani sie gemacht hat. Dieses Anordnen nennt man auch Sortieren. In dieser Biberaufgabe haben wir nach umgekehrter Grösse sortiert, und zwar so: Zuerst haben wir das Foto genommen, auf dem der Apfel noch am grössten war. Aus den restlichen Fotos haben wir dann wieder das genommen, auf dem der Apfel noch am grössten war. Und so weiter. Diese Sortier-Methode ist in der Informatik bekannt und hat sogar einen englischen Namen: Selection Sort. Diese Methode ist ganz praktisch, zum Beispiel wenn man Bücher im Regal nach Autoren sortieren will. Aber sie ist nicht besonders schnell. Und weil Computer meistens viele Daten sortieren müssen, kennt die Informatik schnellere Sortier-Methoden, die für Computerprogramme besser geeignet sind.

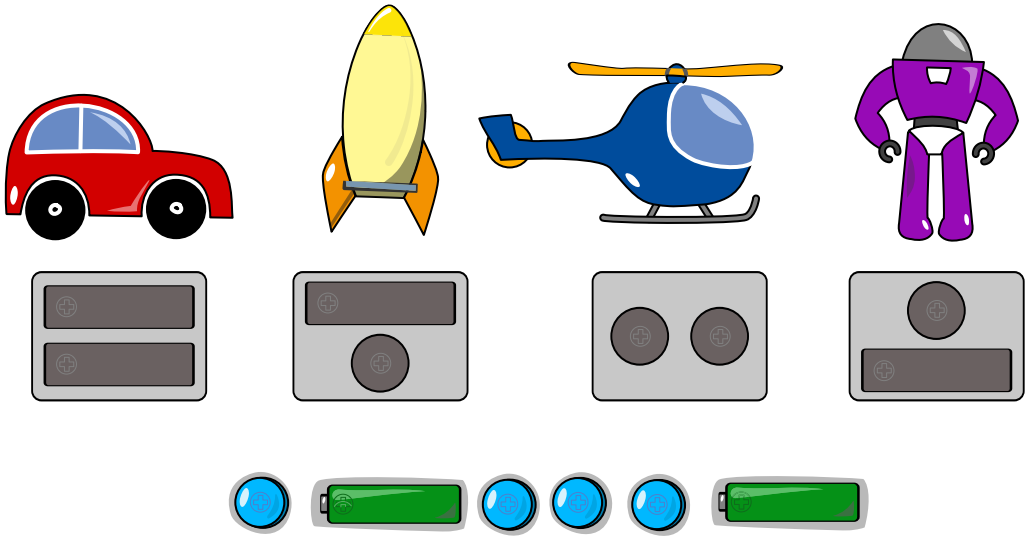
Stichwörter und Webseiten

- Selection Sort Algorithmus: `https://de.wikipedia.org/wiki/Selectionsort`
- Algorithmus: `https://scratch.mit.edu`, `https://de.wikipedia.org/wiki/Algorithmus`

 © Informatik-Biber 2026, SVIA

3. Batterien

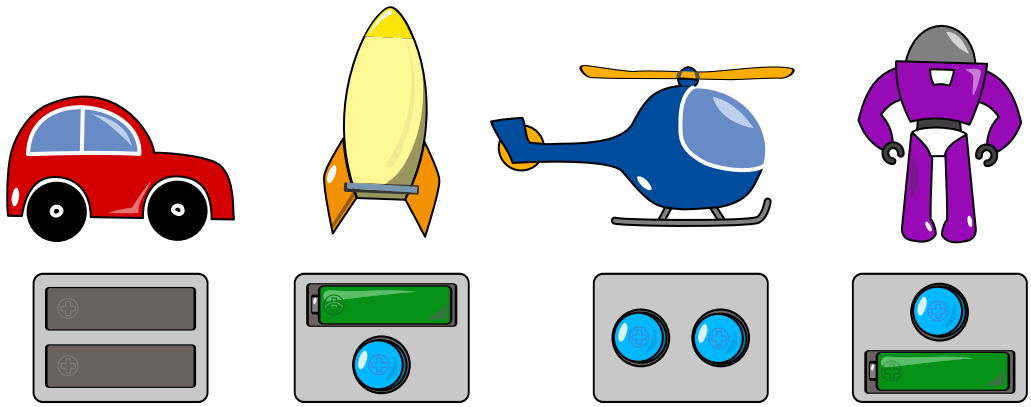
Bonas 4 Spielzeuge brauchen Batterien.
Die Batterie-Box unter einem Spielzeug muss voll sein. Dann funktioniert das Spielzeug.
Bona möchte, dass möglichst viele Spielzeuge funktionieren.



Wie müssen die Batterien in die Boxen gelegt werden?

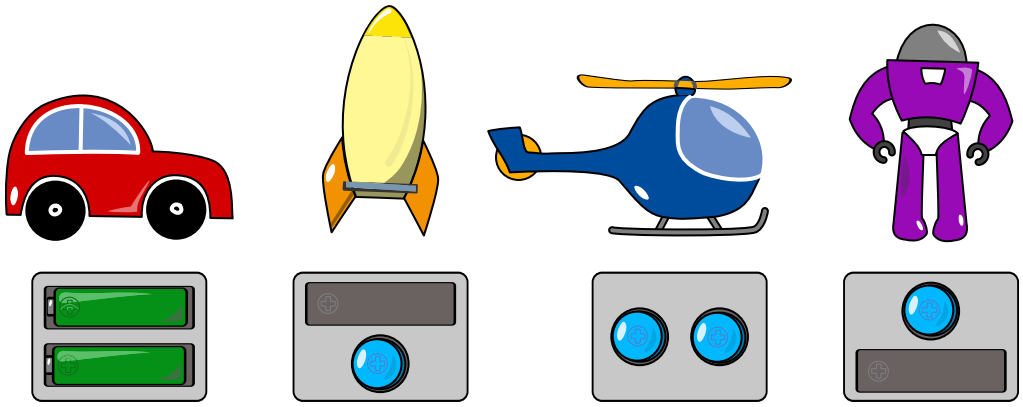
Lösung

Wenn man die Batterien so in die Boxen legt, funktionieren drei Spielzeuge:



Es ist nicht möglich, dass alle vier Spielzeuge funktionieren. Dazu bräuchten wir vier lange Batterien, es gibt aber nur zwei. Mehr als drei Spielzeuge können also nicht funktionieren.

Gibt es vielleicht noch eine andere Möglichkeit, die Batterien so zu legen, dass drei Spielzeuge funktionieren? Man könnte die langen Batterien anstatt für Rakete und Roboter für das Auto gebrauchen. Dann funktioniert das Auto, aber ansonsten kann nur noch der Hubschrauber funktionieren:



Damit drei Spielzeuge funktionieren, müssen also die Batterien genau so in die Boxen gelegt werden wie oben.

Dies ist Informatik!

Um eine Antwort zu finden, könnte man die Spielzeuge der Reihe nach zum Funktionieren bringen. Was man hat, das hat man! Dieses «gierige» Vorgehen, bei dem es möglichst schnell ein funktionierendes Spielzeug geben soll, liefert aber nicht die beste Lösung: Wenn wir links beim Auto anfangen und bei jedem Spielzeug möglichst viele passende Batterien in die Box legen, funktionieren am Ende nur zwei Spielzeuge - siehe oben. Wer systematisch und ganz sicher die beste Lösung finden will, muss alle Möglichkeiten ausprobieren, die Batterien in die Boxen zu legen.

Probleme, bei denen eine beste (oder auch: optimale) Lösung gesucht ist, kennt die Informatik als *Optimierungsprobleme*. Wenn Computer für die Suche nach optimalen Lösungen verwendet

 © Informatik-Biber 2026, SVIA

werden, müssen sie meistens viele Daten verarbeiten, mit vielen möglichen Antworten. Dann würde die Methode «alles ausprobieren» viel zu lange dauern. Die Informatik kennt deshalb zahlreiche Methoden zur Suche nach optimalen Lösungen. Eine davon heisst «greedy», auf Deutsch: gierig. Die Greedy-Methode führt immer denjenigen nächsten Schritt aus, der am meisten Zugewinn bei der Lösung des Problems verspricht. Aber die optimale Lösung wird «greedy» nicht immer gefunden. Wie wir oben gesehen haben, ist das auch bei Bonas Spielzeugen so.

Stichwörter und Webseiten

- Greedy: `https://wiki.neuro.tu-berlin.de/home/teaching/2025ss-AlgoDat-SE/algorithms/greedy-algorithm` oder `https://de.wikipedia.org/wiki/Greedy-Algorithmus`

© Informatik-Biber 2026, SVIA

4. Chläberli

Amelie legt gerne Sticker in eine Reihe, und zwar genau so:

Zuerst kommen entweder zwei oder drei Sticker mit Tieren. Dahinter kommen andere Sticker:

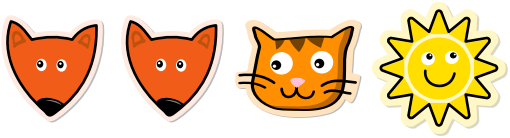
- Bei zwei Tierstickern kommt als Nächstes ein Stern ,

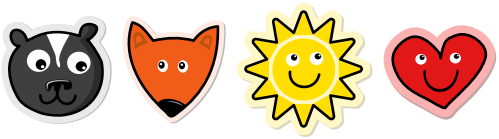
  bei drei Tierstickern eine Sonne .

- Wenn die Tiersticker alle verschieden sind, kommt dahinter noch ein Herz .

Welche Sticker-Reihe hat Amelie gelegt?

A) 

B) 

C) 

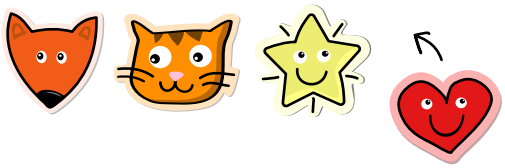
D) 

Lösung

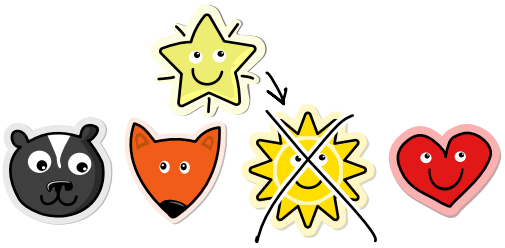
Antwort B ist richtig.

Die Reihe aus Antwort B hat drei Tiersticker. Dahinter kommt eine Sonne. Das ist so, wie Amelie die Sticker legt. Die Tiersticker sind *nicht* alle verschieden, und es kommt kein Herz mehr dahinter. Auch das ist so, wie Amelie Sticker legt.

Die anderen Reihen hat Amelie nicht gelegt:



Die Reihe aus Antwort A hat zwei Tiersticker, und es kommt ein Stern dahinter - wie bei Amelie. Aber weil die beiden Tiersticker verschieden sind, käme bei Amelie dahinter ein Herz; das fehlt.



Die Reihe aus Antwort C hat zwei Tiersticker. Bei Amelie käme dahinter ein Stern; hier kommt aber eine Sonne. Wenigstens ist das Herz so wie bei Amelie, weil die beiden Tiersticker verschieden sind.



Die Reihe aus Antwort D hat drei Tiersticker. Bei Amelie käme dahinter eine Sonne, aber die fehlt hier. Auch bei dieser Reihe ist das Herz so wie bei Amelie, weil die drei Tiersticker alle verschieden sind.

Dies ist Informatik!

Weil Amelie die Tiersticker nach einer festen Vorschrift um weitere Sticker ergänzt, können wir feststellen, welche Reihen sie gelegt hat und welche nicht. So ähnlich wie Amelies Sticker-Reihen in dieser Biberaufgabe funktionieren zum Beispiel Strichcodes. Du findest Strichcodes auf Produktverpackungen, so wie auf diesem Foto:



Der Strichcode, der im Geschäft an der Kasse gescannt wird, stellt die unter den Strichen stehende Nummer aus 13 Ziffern dar. Eigentlich besteht die Nummer eines Produkts zunächst nur aus 12

 © Informatik-Biber 2026, SVIA

Ziffern. Die letzte, 13. Ziffer wird nach einer festen Vorschrift aus den vorherigen 12 Ziffern berechnet. Beim Lesen des Strichcodes rechnet die Kasse noch einmal nach. So kann die Kasse prüfen, ob ein Fehler passiert ist, zum Beispiel weil die Striche durch Verschmutzung nicht richtig gelesen werden konnten. Deshalb nennt man die 13. Ziffer *Prüfziffer*. Prüfziffern gibt es auch bei vielen anderen Nummern, etwa den ISBN auf Büchern oder den internationalen Kontonummern IBAN. Insbesondere für die automatische Verarbeitung mit Computern ist es unerlässlich, Fehler in solchen Nummern bzw. in den sie darstellenden Strich- oder anderen Codes feststellen zu können, um Probleme zu vermeiden: Wenn man im Supermarkt zum Beispiel eine kleine Flasche Wasser kauft, möchte man nicht den Preis für eine grosse Packung Pralinen bezahlen müssen.

Stichwörter und Webseiten

- Prüfziffer: `https://de.wikipedia.org/wiki/Prüfziffer`
- EAN Code: `https://de.wikipedia.org/wiki/European_Article_Number`



© Informatik-Biber 2026, SVIA

5. Videos

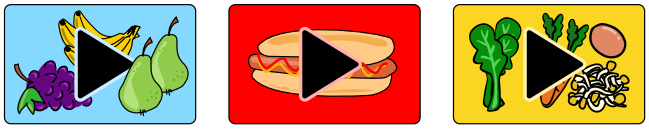
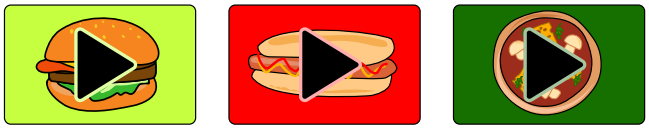
Zara schaut gerne Videos bei ZickZock.
Die schlagen ihr immer neue Videos vor. Das geht so:

- ZickZock sucht Personen, die mindestens zwei gleiche Videos angesehen haben wie Zara.
- Dann schlagen sie alle Videos vor, die die Personen angesehen haben, Zara aber noch nicht.

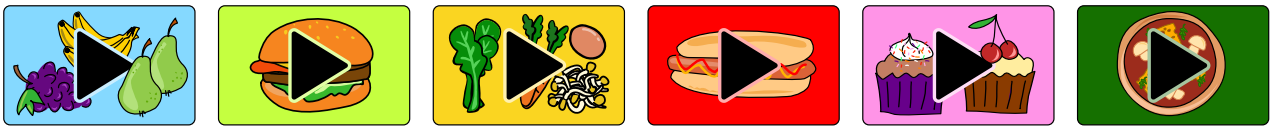
Zara hat diese Videos angesehen:



Zaras Freunde Anna, Bert und Chloe haben diese Videos angesehen:

Person	**Videos**
Anna	
Bert	
Chloe	

Welche dieser Videos schlägt ZickZock Zara vor?



Lösung

ZickZock schlägt Zara diese Videos vor:



Die richtige Antwort findest du, wenn du die Tabelle mit den Videos von Zaras Freunden genau so durchsuchst, wie ZickZock das macht:



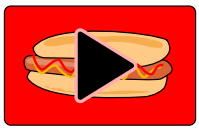
Du suchst in der Tabelle nach Personen, die wenigstens 2 gleiche Videos angesehen haben wie Zara.



In den Zeilen der gefundenen Personen suchst du nach Videos, die Zara noch nicht angesehen hat. Diese Videos empfiehlt ZickZock.

Also los:

- **Anna** hat genau zwei gleiche Videos angesehen wie Zara, nämlich die ersten beiden. ZickZock schlägt das dritte Video vor, weil Zara das noch nicht angesehen hat.
- **Bert** hat genau zwei gleiche Videos angesehen wie Zara, nämlich die Videos links und rechts. ZickZock schlägt das Video in der Mitte vor, weil Zara das noch nicht angesehen hat.
- **Chloe** hat nur ein gleiches Video angesehen wie Zara, nämlich das erste. ZickZock schlägt deshalb keine Videos von Chloe vor.

Person	**Angesehene Videos**		
Anna			
Bert			
Chloe			

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.

 © Informatik-Biber 2026, SVIA

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 Personen, die gleiche oder ähnliche Dinge mögen wie du. Wenn diese Personen ein Video gesehen oder ein Lied gehört haben, das du noch nicht kennst, kann dir die App genau dieses Video oder Lied 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`

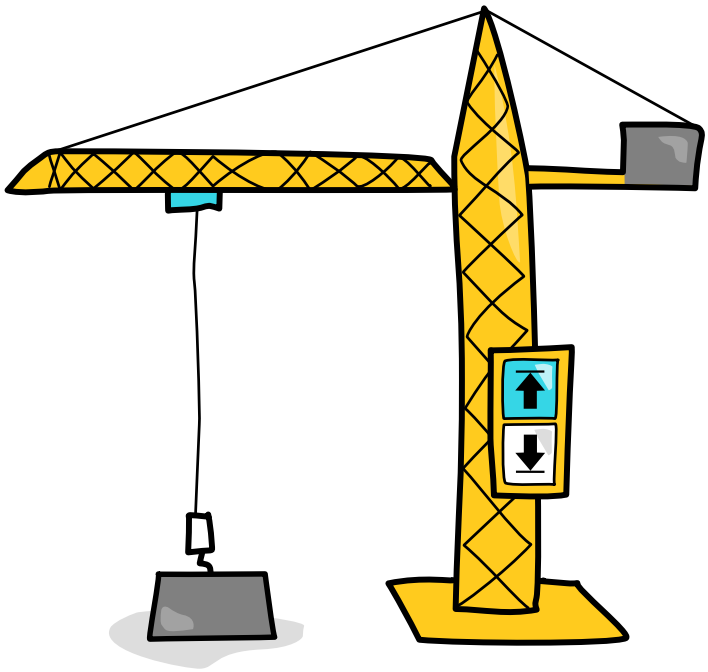
© Informatik-Biber 2026, SVIA

6. Kran

Ein Lastenkran hat zwei Tasten, eine für «auf»  und eine für «ab» .

Wenn man  drückt, fährt die Last ganz nach oben.

Wenn man  drückt, fährt sie ganz nach unten.



Am Anfang ist die Last ganz unten. Dann werden die Tasten in dieser Reihenfolge gedrückt:



Wie oft fährt die Last ganz nach oben?

A) 4-mal

B) 5-mal

C) 8-mal

D) 10-mal

Lösung

Antwort B ist richtig.

Die Last fährt nur dann ganz nach oben, wenn sie ganz unten ist und dann  gedrückt wird.

Das ist 1-mal ganz am Anfang so.

Ausserdem ist das noch 4-mal so, nämlich wenn die Last mit  nach ganz unten fährt und direkt danach mit  wieder ganz nach oben:



Wenn die Last durch  schon oben ist und dann wieder  gedrückt wird, passiert hingegen nichts. Insgesamt fährt die Last also 5-mal ganz nach oben, wie in Antwort B.

Antwort A: Wenn du «4-mal» ausgewählt hast, dann hast du vielleicht die Aufwärtsfahrt am Anfang nicht mitgezählt.

Antwort C: Wenn du «8-mal» ausgewählt hast, dann hast du vielleicht angenommen, dass die Last bei  immer ganz noch oben fährt.

Antwort D: Wenn du «10-mal» ausgewählt hast, dann hast du vielleicht die Fahrten ganz nach unten mitgezählt.

Dies ist Informatik!

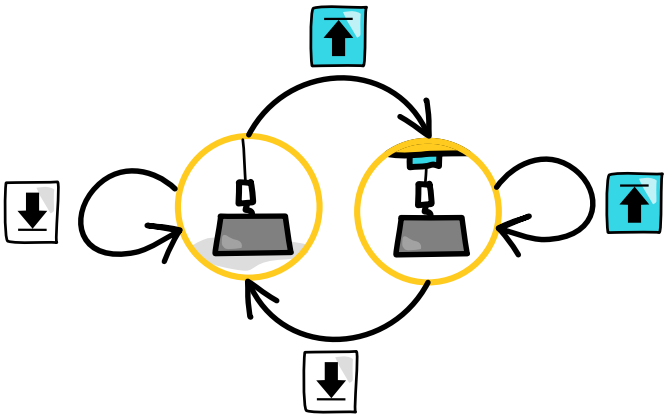
Mit den zwei Tasten kannst du dem Kran zwei verschiedene Befehle geben, nämlich «Fahre die Last nach ganz oben!» und «Fahre die Last ganz nach unten!». Wir haben aber oben schon gesehen, dass nicht jeder Befehl eine Wirkung hat. Die Wirkung der Befehle hängt nämlich davon ab, in welchem *Zustand* die Last ist: unten oder oben. Diese Tabelle zeigt die Wirkung der Befehle abhängig vom Zustand.

Befehl	**Zustand: Last unten** 	**Zustand: Last oben** 
	Last fährt nach oben.	Es passiert nichts.
	Es passiert nichts.	Last fährt nach unten.

In der Informatik wird die Funktion von Maschinen wie dem Kran in dieser Biberaufgabe häufig mit Hilfe von Zuständen beschrieben. Dabei werden nicht nur Tabellen, sondern gerne auch Diagramme, also Zeichnungen benutzt. Ein Zustand wird dann als Kreis dargestellt, und Pfeile zeigen die Wirkung der Befehle in den verschiedenen Zuständen.

 © Informatik-Biber 2026, SVIA

Das Zustandsdiagramm des Krans sieht so aus:



Stichwörter und Webseiten

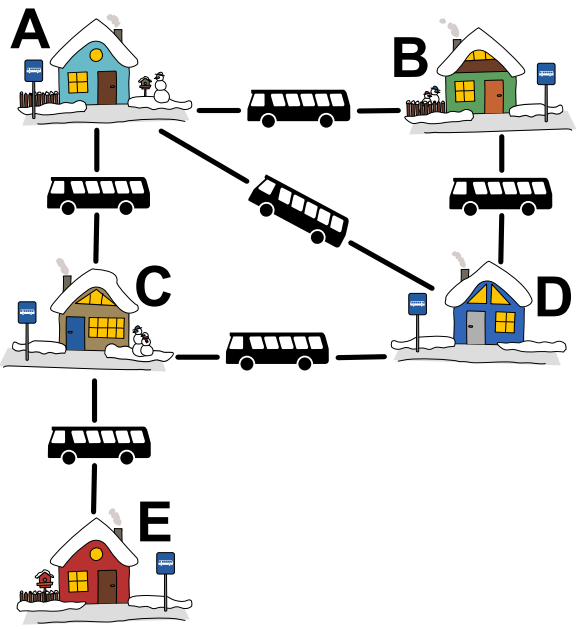
- *Befehlssequenz:* `https://de.wikipedia.org/wiki/Kontrollstruktur`
- *Endlicher Automat, Zustandsautomat:*
  `https://de.wikipedia.org/wiki/Endlicher_Automat`
- *Endlicher Automat:*
  `https://informatik.mygymer.ch/ef2023/02-automaten/01-automaten.html`

© Informatik-Biber 2026, SVIA

7. Schneemobil-Taxi

Es ist Winter in Biberheim!

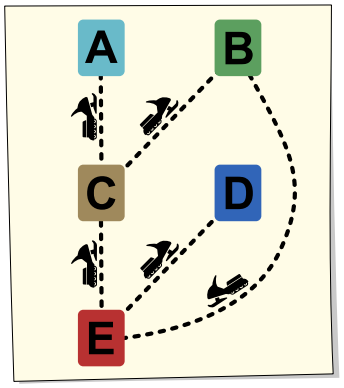
Im Ort gibt es fünf Haltestellen, A bis E. Der Plan rechts zeigt die direkten Busverbindungen zwischen je zwei Haltestellen. Auf jeder Verbindung fahren Busse in beiden Richtungen.



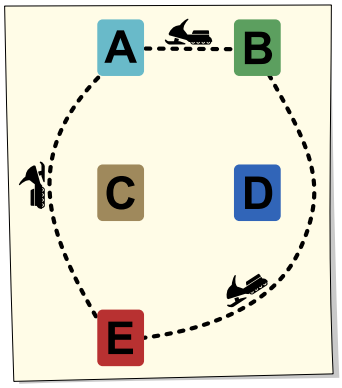
Marcel hat Schneemobil-Taxis. Er bietet genau dann eine direkte Taxi-Verbindung zwischen zwei Haltestellen an, wenn es *keine* direkte Busverbindung zwischen diesen beiden Haltestellen gibt.

Welcher Plan zeigt Marcels direkte Taxi-Verbindungen?

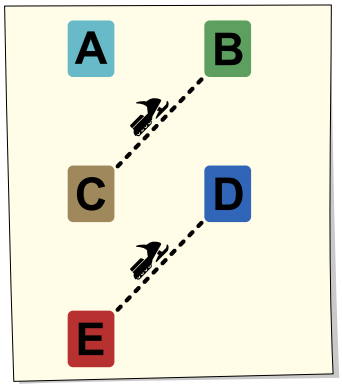
A)



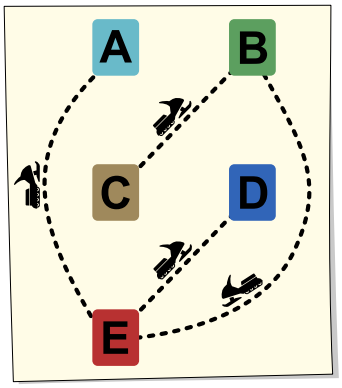
B)



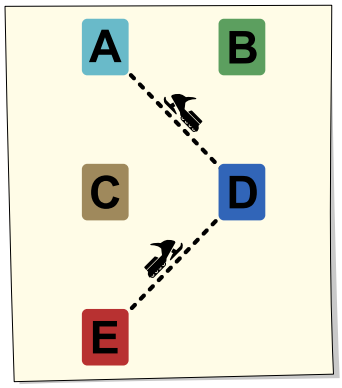
C)



D)



E)



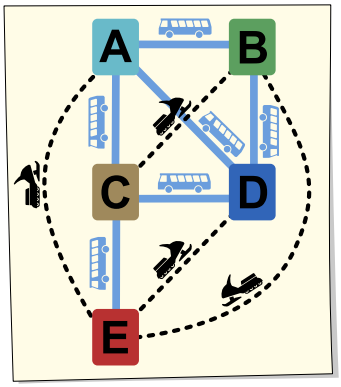
Lösung

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`



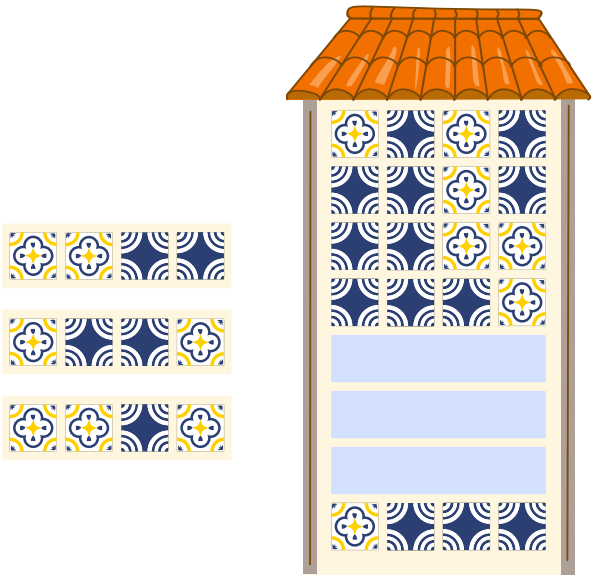
© Informatik-Biber 2026, SVIA

8. 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.

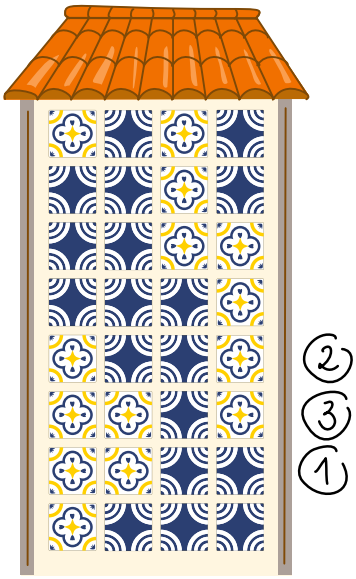
Ana ist noch nicht fertig: 3 Reihen fehlen.



Wie liegen die 3 Plättlireihen richtig auf der Wand?

Lösung

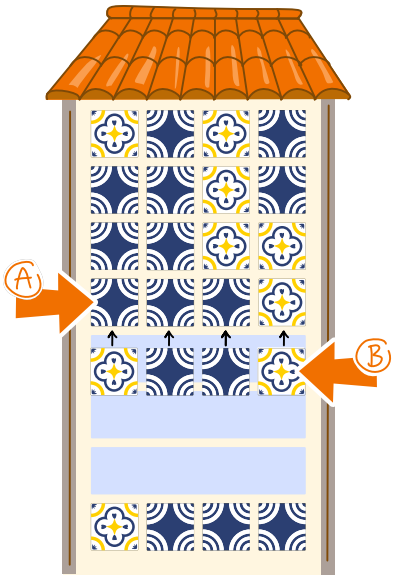
So ist es richtig:



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

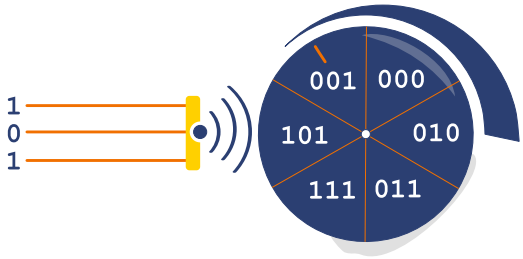
Dies ist Informatik!

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

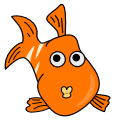
9. Fisch-App

Die Fisch-App kann Fisch-Bilder verändern. Mit einem Smiley zeigst du ihr, wie.

Beim Smiley können Augen, Nase und Mund klein oder gross sein. So ist beim Smiley alles klein:



Nun kommt ein Fisch-Bild. Die App kann Augen, Mund und Flossen klein oder gross machen. Bei diesem Fisch ist alles klein:

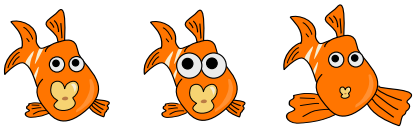


Jetzt nimmst du in der App einen Smiley, bei dem etwas gross ist. Je nachdem, was gross ist, wird beim Fisch eine bestimmte Sache auch gross. Zum Beispiel:

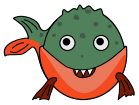
Smiley



Fisch



Hier ist ein anderes Fisch-Bild. Auch bei diesem Fisch ist alles klein:



Du nimmst in der App diesen Smiley:



Was wird dann aus dem Fisch-Bild?

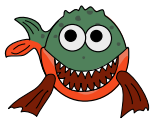
A)



B)



C)



D)



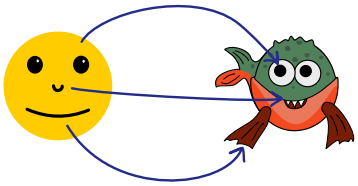
Lösung

Antwort B ist richtig.



Aus den Beispielen erkennt man diese Regeln:

- Hat das Smiley grosse Augen, werden beim Fisch die Augen gross.
- Hat das Smiley eine grosse Nase, wird beim Fisch der Mund gross.
- Hat das Smiley einen grossen Mund, werden beim Fisch die Flossen gross.



In der Aufgabe hat das Smiley grosse Augen, eine kleine Nase und einen grossen Mund. Wegen der oben genannten Regeln müssen also beim Fisch die Augen und die Flossen gross werden. Der Mund bleibt klein.

Die anderen Antworten sind falsch. Bei diesen Fisch-Bildern sind ...

- Antwort A: ... die Augen klein.
- Antwort C: ... der Mund gross.
- Antwort D: ... die Augen klein, der Mund gross und die Flossen klein.

Dies ist Informatik!

Die App erzeugt aus zwei Eingabebildern ein neues Bild. Eigenschaften des Smileys werden dabei auf bestimmte Eigenschaften des Fisches übertragen: Die Grösse der Augen des Smileys bestimmt die Grösse der Augen des Fisches, die Grösse der Nase bestimmt die Grösse des Munds und die Grösse des Munds bestimmt die Grösse der Flossen. So zeigt diese Biberaufgabe die Grundidee eines Filters: Eigenschaften eines Objekts werden systematisch auf Eigenschaften eines anderen Objekts übertragen. Das Fisch-Bild liefert dabei die Grundform, während das Smiley-Bild festlegt, welche Merkmale verändert werden.

Filter sind in der Informatik für viele Verfahren der Bildverarbeitung wichtig. Moderne Programme analysieren Merkmale in Bildern, beispielsweise Gesichter, Augen oder andere Objekte, und nutzen diese Informationen, um andere Bilder zu verändern oder auszuwerten. Ein bekanntes Beispiel ist der *Style Transfer*, bei dem der Stil eines Bildes – etwa der Malstil eines Gemäldes – auf ein anderes Bild übertragen wird.

 © Informatik-Biber 2026, SVIA

Die Fisch-App in dieser Biberaufgabe verarbeitet Bilder nach fest vorgegebenen Regeln. Beim Style Transfer und vielen anderen modernen Verfahren wird die Art der Bildverarbeitung dagegen nicht von Menschen vorgegeben, sondern von Modellen des maschinellen Lernens aus vielen Trainingsbeispielen ermittelt.

Stichwörter und Webseiten

- Mapping: `https://de.wikipedia.org/wiki/Funktion_höherer_Ordnung`
- Style Transfer: `https://en.wikipedia.org/wiki/Neural_style_transfer`

© Informatik-Biber 2026, SVIA

10. 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`

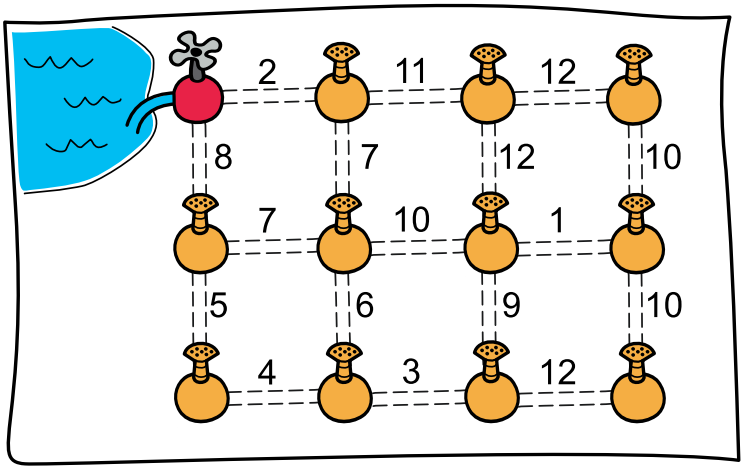
 © Informatik-Biber 2026, SVIA

11. 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.

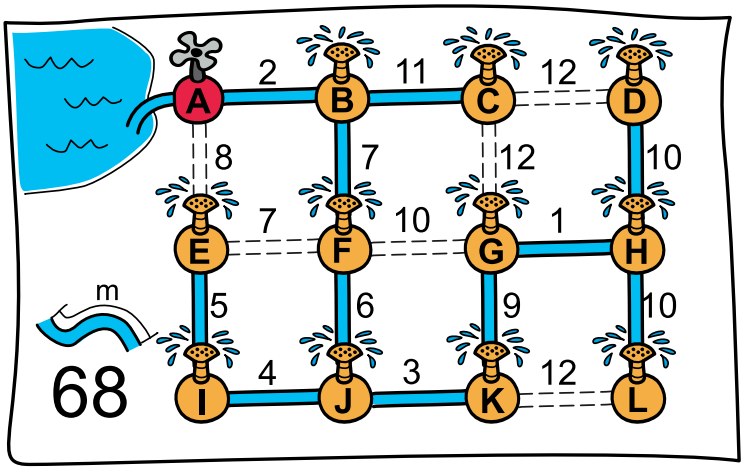
Gerald hat Schläuche nicht an allen Stellen verlegt, sondern nur so, dass

- 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.

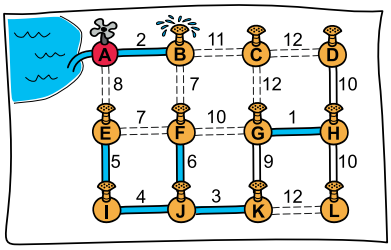
Um die richtige Antwort zu finden, kann man nun folgende Strategie anwenden:

- 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.

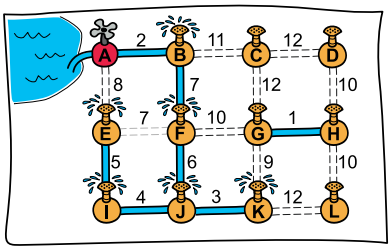
Warum ist es sinnvoll, keine Zyklen entstehen zu lassen? Ein Schläuche-Zyklus benötigt mehr Schlauchmeter als nötig. Aus einem Zyklus kann man nämlich immer einen Schlauch entfernen, und trotzdem bekommen alle Sprinkler im Zyklus weiterhin Wasser. Wenn wir beispielsweise die Schläuche in Geralds Garten 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

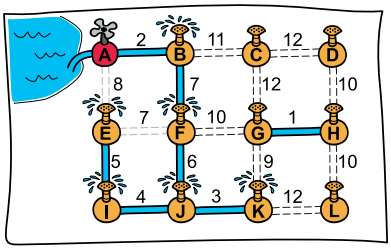


An allen Stellen der Längen 1 bis 6 wird ein Schlauch verlegt, weil dadurch kein Zyklus entsteht.

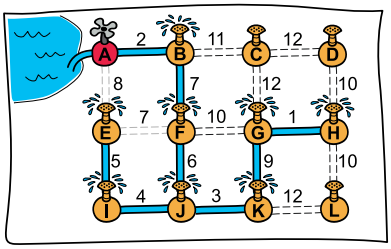


An den Stellen der Länge 7 wird

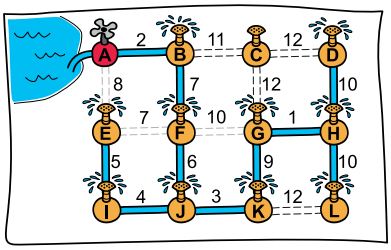
- zwischen B und F ein Schlauch verlegt, aber
- zwischen E und F kein Schlauch verlegt, weil dadurch der Zyklus E, F, J, I, E entstünde.



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

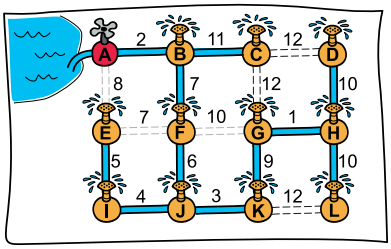


Zwischen G und K (Länge 9) wird ein Schlauch verlegt.



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

12. Tacos für lau!

Tacos sind ein schnell zubereitetes Gericht aus Mexiko. Eine Taquería verspricht: «Wenn wir für eine Bestellung von höchstens 7 Tacos länger als 15 Minuten brauchen, ist sie kostenlos.»

Die Taquería hat drei Köche, die für die verschiedenen Taco-Sorten unterschiedlich lange brauchen (in Minuten):

Taco-Sorte	**Ana**	**Beto**	**Carlos**
Poulet	5	3	5
Fleisch	3	5	5
Vegetarisch	2	11	11

Bei einer neuen Bestellung werden die bestellten Tacos der Reihe nach den Köchen zugeteilt, und zwar nach diesen Regeln:

1. Wähle nur einen Koch, der bisher höchstens 9 Minuten braucht. Gibt es so einen Koch nicht, wähle nur nach Regel 2.
2. Wähle den Koch, der den Taco am schnellsten zubereiten kann. Bei Gleichstand den, der bisher am wenigsten Zeit braucht. Bei nochmal Gleichstand nach dem Alphabet.

Wenn alle Tacos zugeteilt sind, legen die Köche gleichzeitig los. Wenn dann ein Koch länger als 15 Minuten braucht, sind die Tacos für lau!

Finde eine Bestellung, für die die Taquería mehr als 15 Minuten braucht.

Scanne den QR-Code, um auf die interaktive Version dieser Übung zuzugreifen.

Lösung

Um eine kostenlose Bestellung zu erhalten, muss einer der drei Köche für «seine» Tacos mehr als 15 Minuten brauchen. Das kann man insbesondere mit dieser Strategie erreichen: Weise einem Koch, der bisher höchstens 9 Minuten braucht, einen Taco zu, für den er mehr als 6 Minuten braucht.

Man sieht, dass das nur bei Beto und Carlos geht, mit einem vegetarischen Taco (11 Minuten). Dazu muss Ana für ihre bisherige Zuteilung schon mehr als 9 Minuten brauchen, sonst würde dieser Taco nach Regel 2 ihr zugeteilt.

Also beschäftigen wir Ana mit z.B. 4 mal 🥩 (12 Minuten). Mit weiteren 2 mal 🥩 bekommen Beto und Carlos je 5 Minuten. Dann braucht man nur noch einmal 🥬 zu bestellen. Der wird dann nämlich Beto zugeteilt, so dass er für seine Zuteilung 16 Minuten braucht - die Tacos 🌮 sind für lau!

Es gibt 11 verschiedene Möglichkeiten, am Anfang Tacos zu bestellen, die alle Ana zugeteilt werden und für die sie mehr als 9 Minuten braucht (🍗 kommt darin nicht vor, weil Beto und Carlos bei dieser Sorte schneller sind als Ana):

- 🥩🥩🥩🥩
- 🥩🥩🥩🥬 (in 4 verschiedenen Reihenfolgen)
- 🥩🥩🥬🥬 (in 6 verschiedenen Reihenfolgen)

Danach müssen wir entweder Beto oder Carlos mit Vegi auf mindestens 16 Minuten bringen. Dafür gibt es 4 Möglichkeiten:

- 🥩🥩🥬
- 🥬🥩🥬
- 🥬🍗🥬
- 🥩🥬🥬

Es gibt also $11 \times 4 = 44$ Möglichkeiten, die Taquería zu überlasten und kostenlos zu essen. Bei insgesamt 2.187 möglichen Bestellungen ist die Chance mit ca. 2 % aber nicht allzu gross.

Dies ist Informatik!

Die Regeln der Taquería in dieser Biberaufgabe haben das Ziel, die Bestellungen schnell abzuarbeiten. Allerdings gibt es bestimmte Bestellungen, für die diese Regeln kein optimales Ergebnis liefern. Nun kann man versuchen, wie ein Angreifer diese Tatsache auszunutzen, um kostenlose Tacos zu bekommen. Die Taquería wiederum könnte dann die Regeln anpassen, um weitere Angriffe abzuwehren. Zum Beispiel könnte sie eine weitere Regel einbauen: Wenn ein Taco die Arbeitszeit des ausgewählten Kochs über 15 Minuten steigert, dann nimm einen anderen Koch. Aber hält das *alle* Angriffe ab? Leider nicht! Fallen dir Regeln ein, die absolut sicher sind?

In der Informatik ist die Strategie, für einen Algorithmus eine möglichst schwierig zu bearbeitende Eingabe zu finden, auch als *adversarial attack* bekannt. Man nimmt dabei die Rolle eines Angreifers ein und versucht einen Verteidiger zu etwas zu bringen, was er eigentlich nicht möchte. Eine besondere

 © Informatik-Biber 2026, SVIA

Rolle spielen adversarial attacks für KI-Systeme: Mit gezielt manipulierten Trainingsdaten können bei künstlichen neuronalen Netzen Fehlverhalten ausgelöst werden.

Bei Informatiksystemen bzw. den sie steuernden Programmen kann es also Regeln bzw. Algorithmen geben, die anfällig für gezielte Angriffe sind. Im harmlosesten Fall kann dies zu einem langsameren Ablauf führen, oder das Programm stürzt ab. Schlimmer wäre es, wenn Angreifer eine Anfälligkeit ausnutzen, um unberechtigt auf Daten zuzugreifen oder gar beliebige Programme auszuführen. Daher führen Sicherheitsexperten bewusst solche Angriffe selber durch, um mögliche Fehler zu identifizieren und rechtzeitig zu beheben. Das nennt man *Adversarial Testing*.

Stichwörter und Webseiten

- Lastverteilung: `https://de.wikipedia.org/wiki/Lastverteilung_(Informatik)`
- Adversarial Attack: `https://de.wikipedia.org/wiki/Adversarial_Attack`
- Adversarial Testing: `https://www.security-insider.de/was-ist-adversarial-testing-a-4d8f9ad661ac79548140fe547cfd5152/`
- Softwaretest / Sicherheitstest: `https://de.wikipedia.org/wiki/Softwaretest`, `https://de.wikipedia.org/wiki/Sicherheitstest_(Software)`

© Informatik-Biber 2026, SVIA

13. Versteckspiel

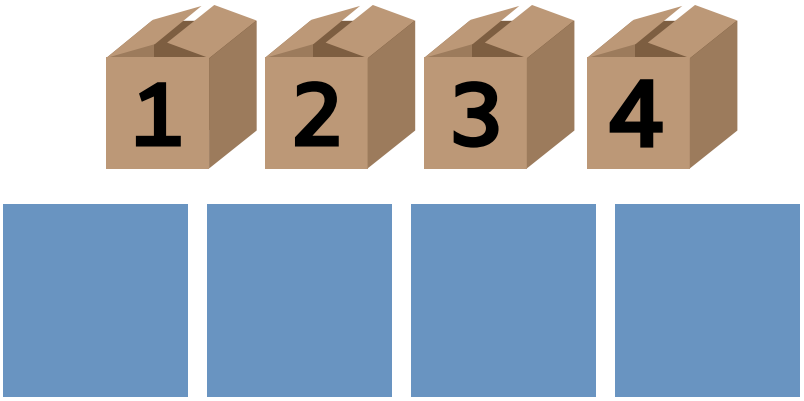
Ben versteckt sich in einer von vier nummerierten Kisten, z.B. in Kiste 2.



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.

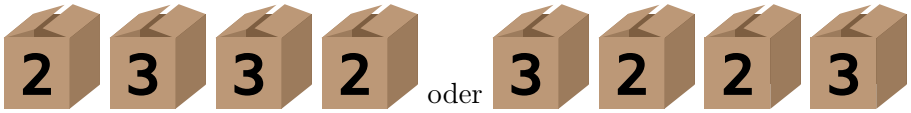
Gib eine solche Reihenfolge an!



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:

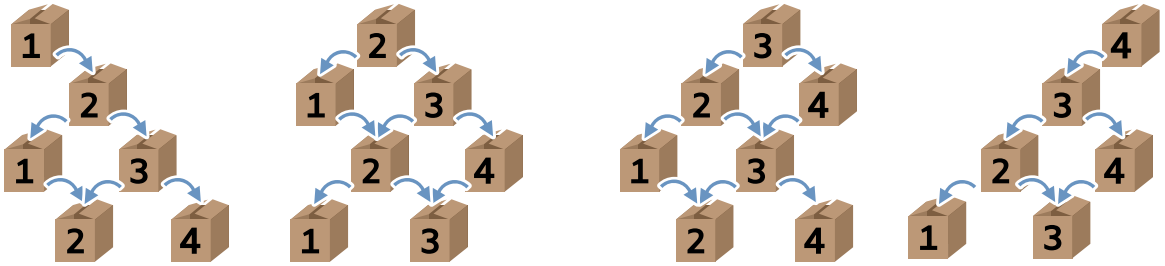


Wir zeigen, dass diese Antworten richtig sind. Dabei bezeichnen wir Bens Bewegungen als *Zug* in diesem Versteckspiel.

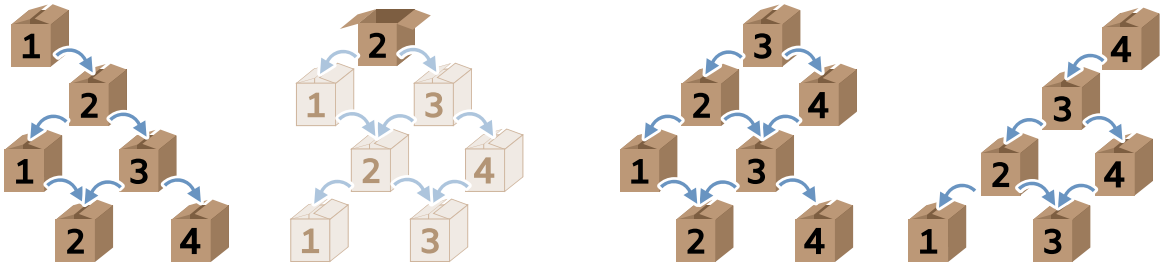
Am Anfang kann Ben seine «Start-Kiste» frei wählen. Danach hat Ben, wenn er

- 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:

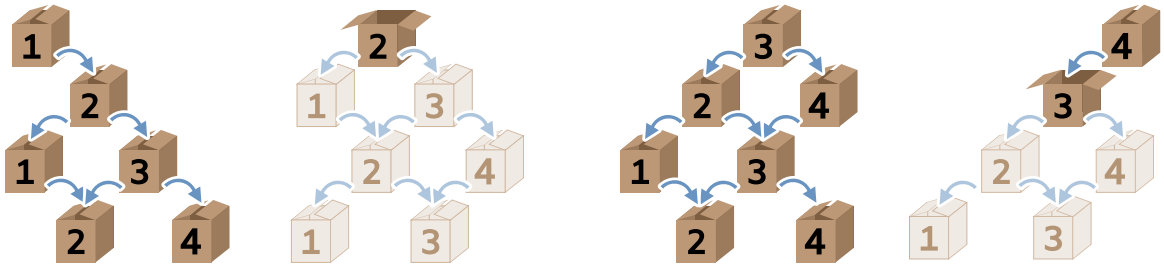


Was passiert, wenn Anja in der Reihenfolge **2-3-3-2** in die Kisten schaut? Zuerst schaut Anja in Kiste 2. Wenn sie Ben dort findet, ist Kiste 2 seine Start-Kiste, und er kann die von dort ausgehenden Züge nicht mehr machen.

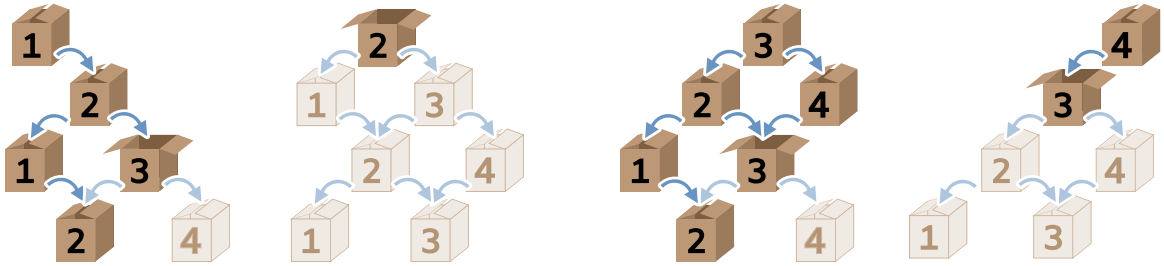


Dann schaut Anja in Kiste 3. Wenn sie Ben dort findet, ist das seine zweite Kiste. Auf der zweiten Ebene der Zugfolgen kommt Kiste 3 noch ganz rechts vor. Die von dort ausgehenden Züge kann Ben nun nicht mehr machen, und wir brauchen sie nicht weiter zu betrachten.

 © 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`

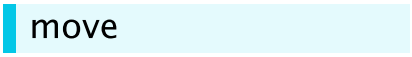
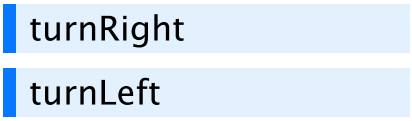
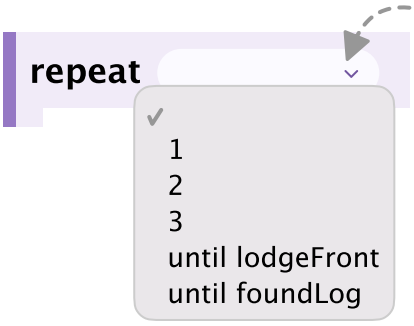


© Informatik-Biber 2026, SVIA

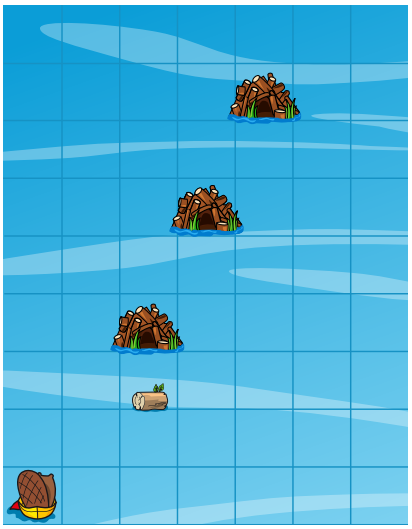
14. Quartierhilfe

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?

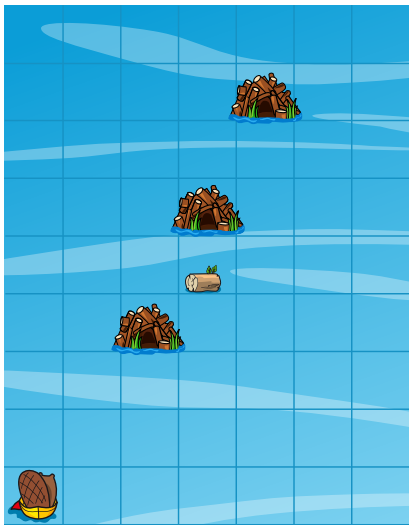
Du kannst folgende Anweisungen verwenden:

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.

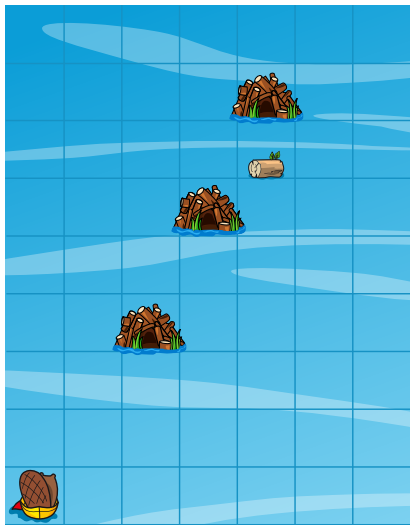
Notiere eine Wegbeschreibung, mit der Belinda an allen drei Tagen den Baumstamm einsammelt. Sobald sie ihn hat, soll sie bei keinem nachfolgenden Nachbarn vorbeifahren.



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

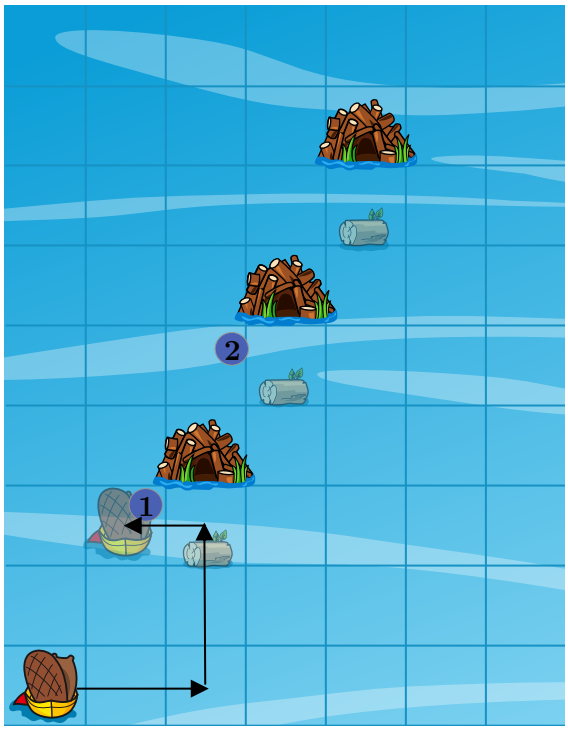
```
  turnRight()
  move()
```

```
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

Masiar Babazadeh

Liam Baumann

Wilfried Baumann

Nils Böhnke

Emily Cavalcanti

Cesar Cepeda

Maria Cepeda

Valentina Dagienė

Christian Datzko

Heather Duan

Susanne Falkenberg-Thut

Gerald Futschek

Jens Hartmann

Asterios Karagiannis

Geetanjali Khatri

Doyong Kim

Vaidotas Kinčius

Lukas Lehner

Justina Oostendorp

Özgür Özdemir

Omar Paladines

Belén Palop

Alex Parry

Jean-Philippe Pellet

Zsuzsa Pluhár

Wolfgang Pohl

Liane Raji

Lorenzo Repetto

Kirsten Schlüter

Margareta Schlüter

Dirk Schmerenbeck

Andrea Maria Schmid

Eljakim Schrijvers

Giovanni Serafini

Bronius Skūpas

Emma Sotomayor

Jacqueline Staub

Alieke Stijf

Pedro Suarez

Mario Urena

Jiří Vaníček

Michael Weigend

Kyra Willekes

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/

© Informatik-Biber 2026, SVIA

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