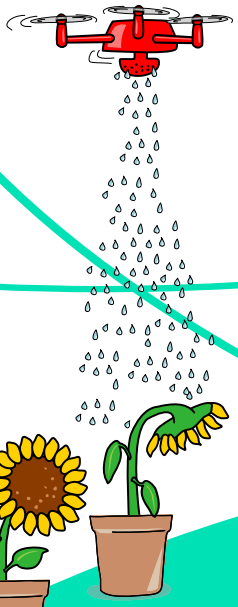


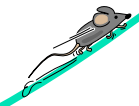


INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

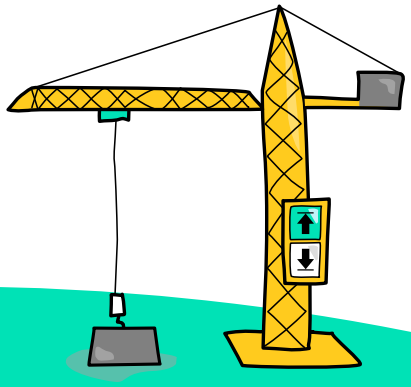
Aufgaben und Lösungen 2026

Schuljahre 5/6



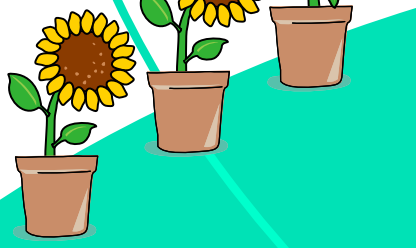


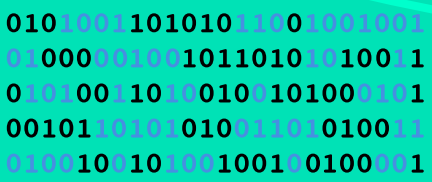
https://www.informatik-biber.ch/



Herausgeber:

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

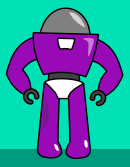




SV!A

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. 69 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. Batterien 2026-BR-02 1

2. Chläberli 2026-EC-01 5

3. Videos 2026-HU-01a 9

4. Laptops 2026-CH-03a 13

5. Spaziergang im Park 2026-DE-01 17

6. Eingangshalle 2026-DE-07 21

7. Sirup-Fabrik 2026-KR-04 25

8. Hausaufgaben 2026-SI-01a 29

9. Fisch-App 2026-DE-08 33

10. Kopf oder Zahl 2026-IN-05 37

11. Quadrat einfärben 2026-DE-09 39

12. Auto-Fabrik 2026-JP-03 43

13. Feuergeist 2026-CZ-02 47

14. Muddys Methode 2026-DE-10 51

15. Tacos für lau! 2026-MX-02b 55

16. Versteckspiel 2026-US-03b 59

17. Quartierhilfe 2026-DE-90 63

A. Aufgabenautoren 69

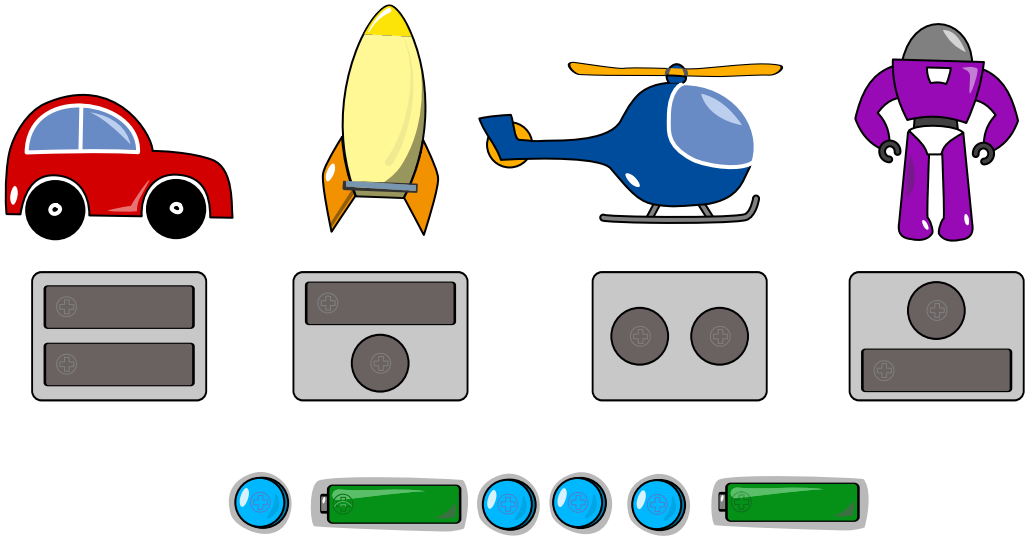
B. Akademische Partner 71

C. Sponsoring 72

© Informatik-Biber 2026, SVIA

1. Batterien

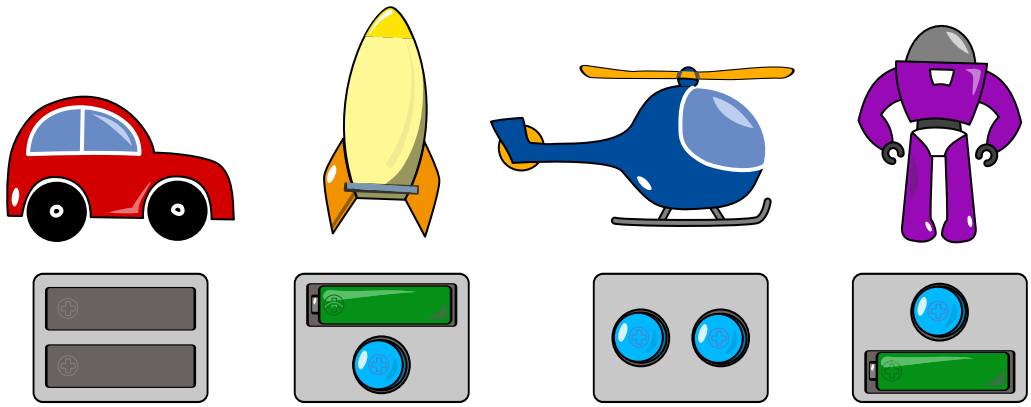
Bonas 4 Spielzeuge brauchen Batterien.
Die Batterie-Box unter einem Spielzeug muss voll sein. Dann funktioniert das Spielzeug.
Bona möchte, dass möglichst viele Spielzeuge funktionieren.



Wie müssen die Batterien in die Boxen gelegt werden?

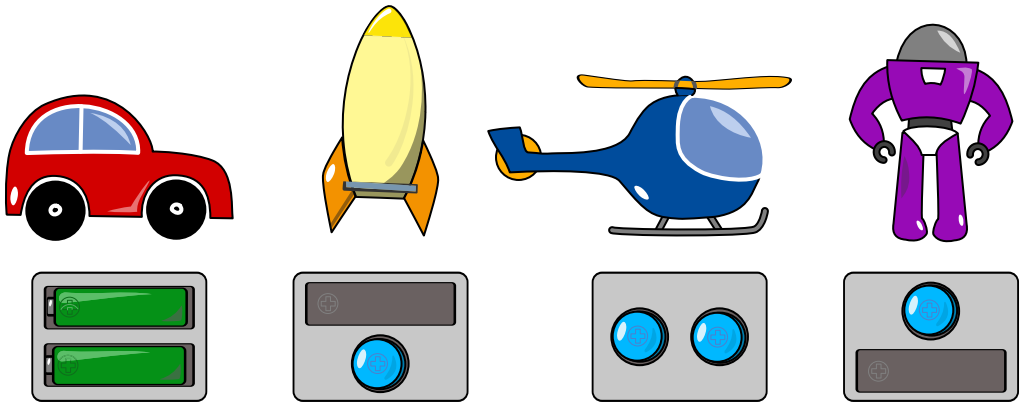
Lösung

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

2. 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:

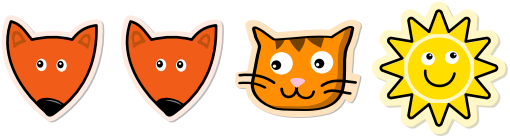
- Bei zwei Tierstickern kommt als Nächstes ein Stern ,

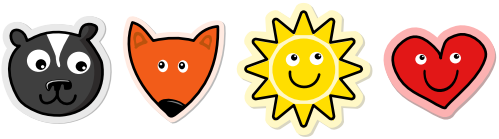
  bei drei Tierstickern eine Sonne .

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

Welche Sticker-Reihe hat Amelie gelegt?

A) 

B) 

C) 

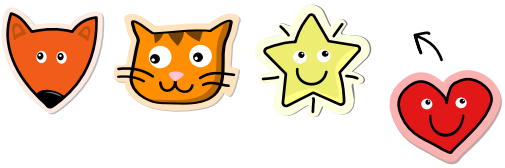
D) 

Lösung

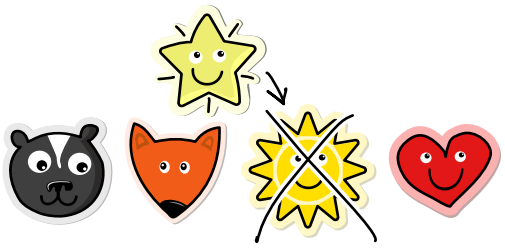
Antwort B ist richtig.

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

3. Videos

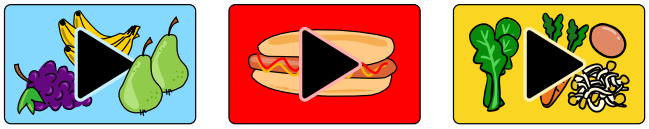
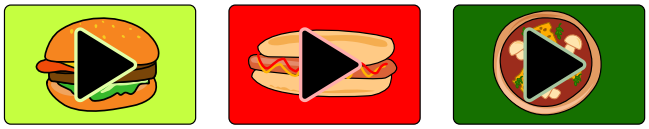
Zara schaut gerne Videos bei ZickZock.
Die schlagen ihr immer neue Videos vor. Das geht so:

- 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.

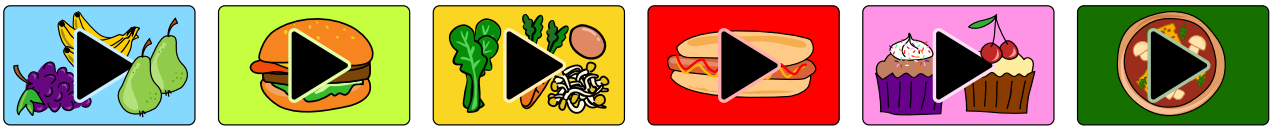
Zara hat diese Videos angesehen:



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:



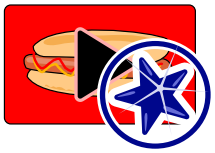
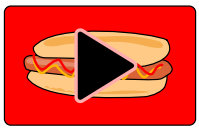
Du suchst in der Tabelle nach Personen, die wenigstens 2 gleiche Videos angesehen haben wie Zara.



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

4. 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.

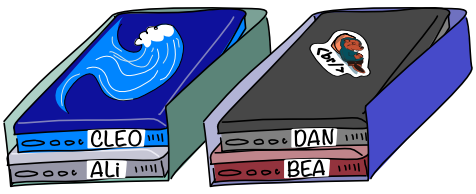
Heute verlassen sie den Raum nach dem Unterricht in dieser Reihenfolge:

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

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

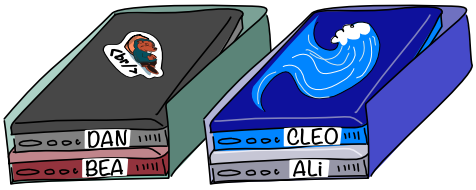
1. Bea
2. Ali
3. Dan
4. Cleo

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.

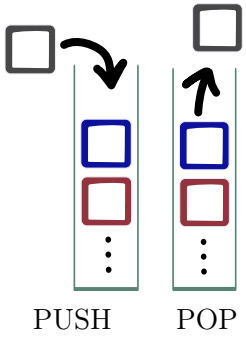
Dies ist Informatik!

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.

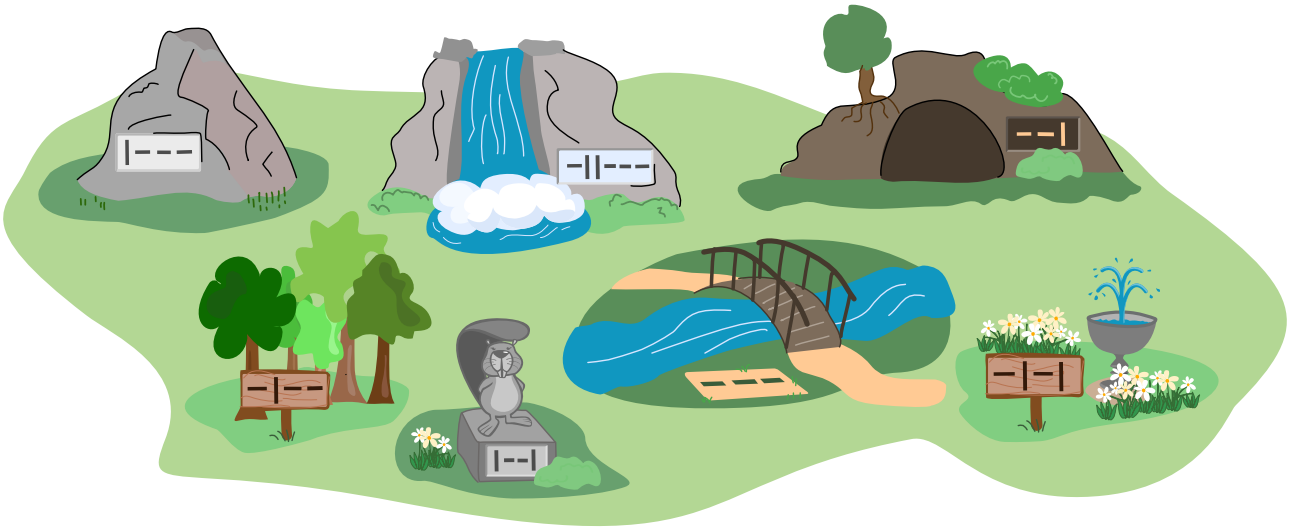
Stichwörter und Webseiten

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

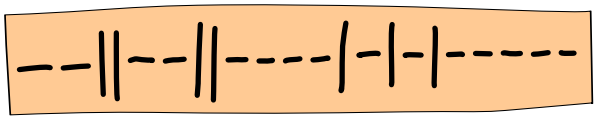
© Informatik-Biber 2026, SVIA

5. Spaziergang im Park

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



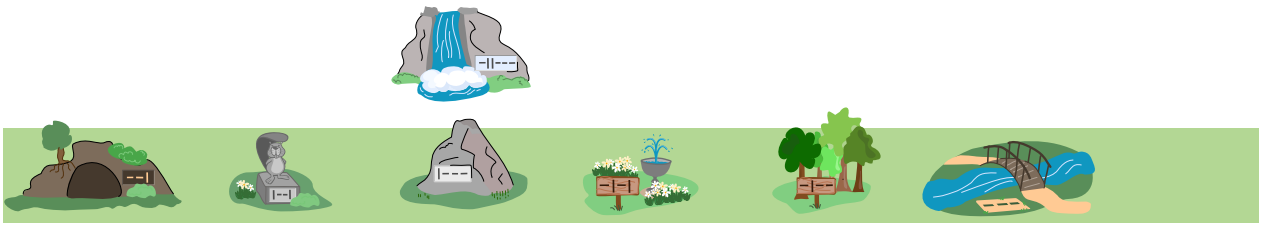
Simon geht im Biberpark spazieren. Immer wenn er an einem besonderen Ort vorbeikommt, schreibt er seine Markierung auf ein Blatt, eine nach der anderen. Am Ende hat er diese Zeichen aufgeschrieben:



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

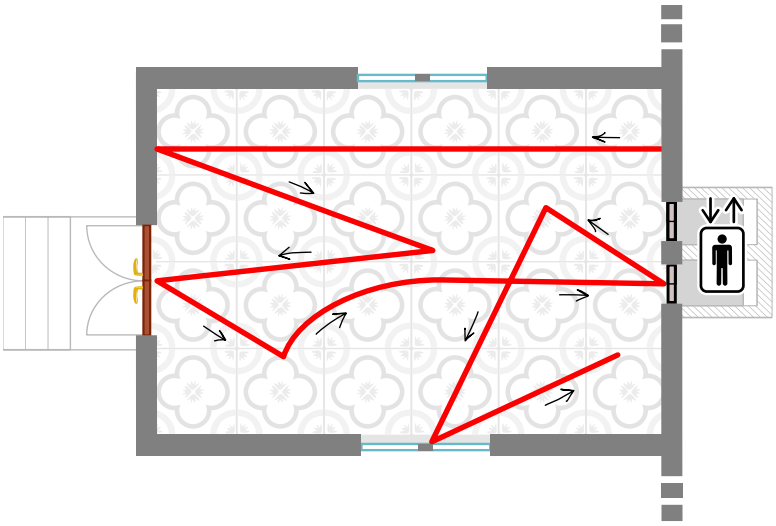
- *präfixfreie Codes*: `https://de.wikipedia.org/wiki/Präfixcode`
- *Datenkompression*: `https://de.wikipedia.org/wiki/Datenkompression`
- *Huffman-Kodierung*: `https://de.wikipedia.org/wiki/Huffman-Kodierung`

© Informatik-Biber 2026, SVIA

6. 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.

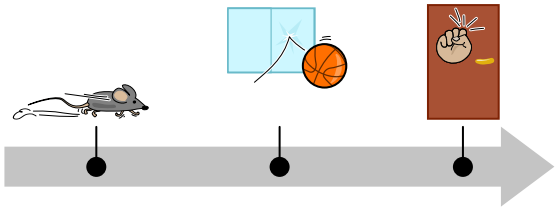


Jemand klopft an die Tür.



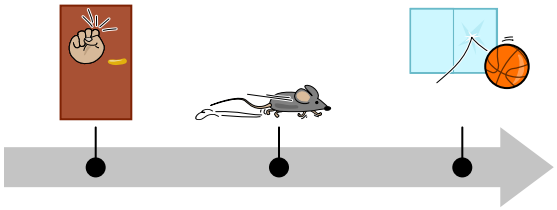
Eine Maus rennt durch die Eingangshalle.

In welcher Reihenfolge sind die Ereignisse passiert?

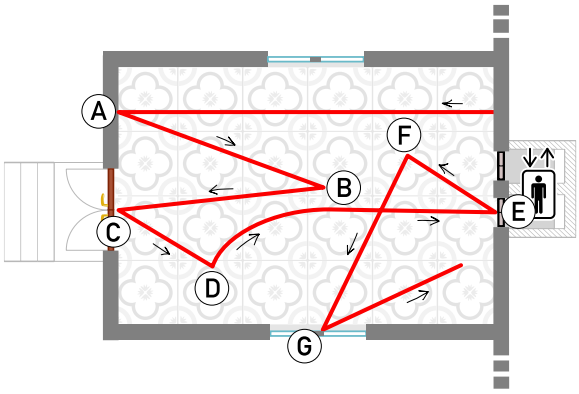


Lösung

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

7. Sirup-Fabrik

Eine Fabrik stellt Sirup her. Einige Aufgaben werden von Robotern erledigt.

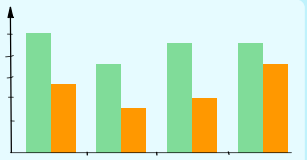
Ein Bildschirm zeigt 2 Tabellen mit Informationen zu den 6 Robotern:

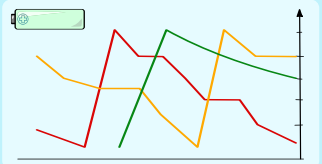
- Links stehen jeweils Hersteller und aktueller Standort,
- rechts die Funktion und der Akkustand.

Beide Tabellen enthalten die Nummern der Roboter.

Hersteller	Roboter-Nr.	Standort
Robras	R-01	Lager
Robras	R-03	Versand
Robras	R-05	Flaschenreinigung
Bibobot	R-02	Flaschenabfüllung
Bibobot	R-04	Sirupküche
Bibobot	R-06	Sirupküche

Roboter-Nr.	Funktion	Akku
R-01	Transport	85 %
R-02	Verschliessen	25 %
R-03	Transport	48 %
R-04	Mischen	90 %
R-05	Spülen	75 %
R-06	Rühren	55 %









Wenn der Akkustand eines Roboters unter 30 % liegt, muss er an die Ladestation.

Ein Roboter muss an die Ladestation. Was ist sein Standort?

A) Lager

B) Flaschenabfüllung

C) Versand

D) Sirupküche

Lösung

Antwort B ist richtig.

Hersteller	Roboter-Nr.	Standort
Robras	R-01	Lager
Robras	R-03	Versand
Robras	R-05	Flaschenreinigung
Bibobot	R-02	Flaschenabfüllung
Bibobot	R-04	Sirupküche
Bibobot	R-06	Sirupküche

Roboter-Nr.	Funktion	Akku
R-01	Transport	85 %
R-02	Verschliessen	25 %
R-03	Transport	48 %
R-04	Mischen	90 %
R-05	Spülen	75 %
R-06	Rühren	55 %

Rechts ist zu sehen, dass Roboter R-02 einen Akkustand von nur 25 % hat. Weil das unter 30 % liegt, muss Roboter R-02 an die Ladestation. Sein Standort ist die Flaschenabfüllung, wie man links sehen kann.

	Hersteller	Roboter-Nr.	Standort
a)	Robras	R-01	Lager
b)	Robras	R-03	Versand
	Robras	R-05	Flaschenreinigung
	Bibobot	R-02	Flaschenabfüllung
c)	Bibobot	R-04	Sirupküche
	Bibobot	R-06	Sirupküche

Roboter-Nr.	Funktion	Akku
R-01	Transport	85 %
R-02	Verschliessen	25 %
R-03	Transport	48 %
R-04	Mischen	90 %
R-05	Spülen	75 %
R-06	Rühren	55 %

Antwort A ist nicht richtig: Im Lager ist nur Roboter R-01. Er muss nicht zur Ladestation, da sein Akkustand mit 85 % über 30 % liegt.

Antwort C ist nicht richtig: Im Versand ist nur Roboter R-03. Er muss nicht zur Ladestation, da sein Akkustand mit 48 % über 30 % liegt.

Antwort D ist nicht richtig: In der Sirupküche sind die Roboter R-04 und R-06. Beide müssen nicht zur Ladestation, da ihr Akkustand mit 90 % bzw. 55 % über 30 % liegt.

Dies ist Informatik!

Die Tabellen in dieser Biberaufgabe zeigen Daten zu den Robotern in der Limonadenfabrik. Die Datensätze in den Zeilen der linken Tabelle enthalten den Hersteller und den Standort des Roboters, die Datensätze in der rechten Tabelle enthalten die Funktion und den Akkustand. Weil beide Tabellen ausserdem die Roboter-Nummer enthalten, kann man die Daten aus beiden Tabellen miteinander verknüpfen und so den Standort des Roboters bestimmen, der an die Ladestation muss.

In der Informatik spricht man von einer *Datenbank*, wenn Daten zu Objekten (z. B. Roboter in einer Produktion, Kunden eines Unternehmens oder Artikel in einem Online-Shop) in mehreren Tabellen gespeichert sind und die Daten aus den Tabellen über *Schlüssel* zusammengeführt werden können. In jeder Tabelle einer Datenbank muss ein Schlüssel eindeutig sein, darf also nur in einer Zeile

 © Informatik-Biber 2026, SVIA

vorkommen. Bei einem Schlüssel handelt es sich deshalb meistens um eine Identifikationsnummer, zum Beispiel eine Kundennummer oder Artikelnummer; in dieser Biberaufgabe ist die Roboter-Nummer ein Schlüssel. Mit Hilfe der Schlüssel ist es in Datenbanken auch möglich, mit der Operation *Join* aus mehreren Tabellen eine neue Tabelle zu erzeugen. Deren Zeilen enthalten dann die Daten aus allen Tabellen zusammen.

Das Zusammenführen von Information aus verschiedenen Datenquellen wird als *Informationsintegration* bezeichnet. Informationsintegration passiert sehr häufig, z. B. im Online-Shop, um den Lagerbestand eines Artikels zu überprüfen, oder in der Forschung, bei der Zusammenführung von Daten verschiedener Messinstrumente. In dieser Biberaufgabe musstest du Information aus zwei Tabellen zusammenführen.

Stichwörter und Webseiten

- Datenbanken: `https://de.wikipedia.org/wiki/Datenbank`
- Informationsintegration: `https://de.wikipedia.org/wiki/Informationsintegration`



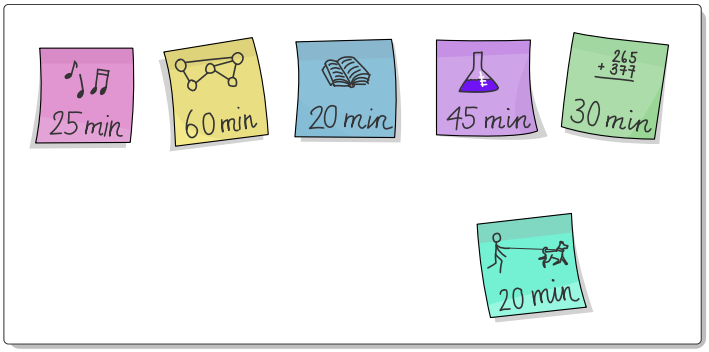
© Informatik-Biber 2026, SVIA

8. Hausaufgaben

Sarah erledigt ihre Hausaufgaben systematisch, schön nacheinander. Sie wählt immer diejenige Aufgabe als nächste aus, für die sie am wenigsten Zeit braucht. Das tut sie zu Beginn und immer dann, wenn sie mit einer Aufgabe fertig ist.

Heute hat Sarah fünf Hausaufgaben. Zuerst notiert sie die einzelnen Aufgaben und wieviel Zeit sie dafür braucht. Dann legt sie los.

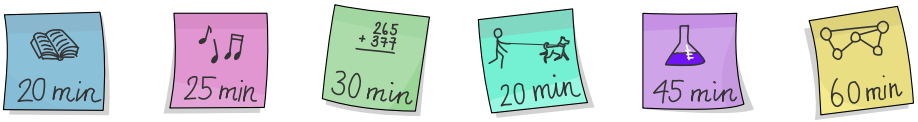
Nachdem sie schon 50 Minuten gearbeitet hat, fällt ihr ein, dass sie auch noch 20 Minuten mit dem Hund spazieren gehen muss. Auch das schreibt sie sich als Aufgabe auf:



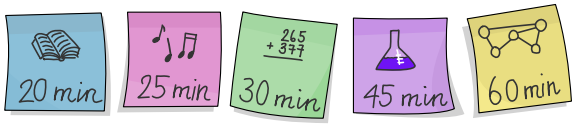
In welcher Reihenfolge bearbeitet Sarah die Aufgaben?

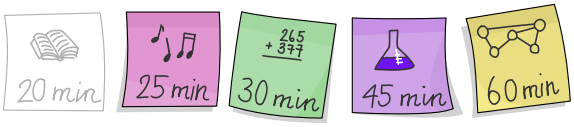
Lösung

So ist es richtig:



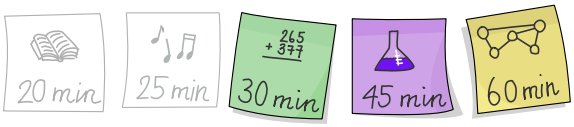
Aus ihrer aktuellen Aufgabenmenge wählt Sarah immer die Aufgabe aus, für die sie am wenigsten Zeit braucht. Um immer einfach zu wissen, welche Aufgabe das ist, sortieren wir die Aufgaben nach dieser Zeit in eine Liste, so dass immer die nächste Aufgabe vorne steht.





Zuerst wählt sie Englisch 

, die am schnellsten zu erledigende Aufgabe von allen.



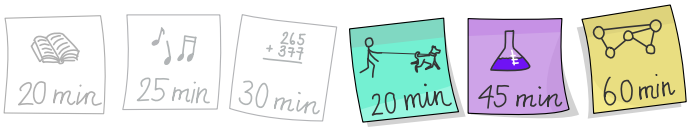
Als nächste wählt sie die Musik-Aufgabe 

. Nachdem sie die erledigt hat, hat sie insgesamt 45 Minuten gearbeitet.

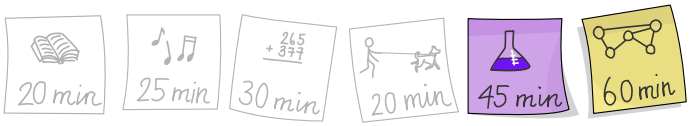


Sie wählt die nächste Aufgabe: Mathematik 

.



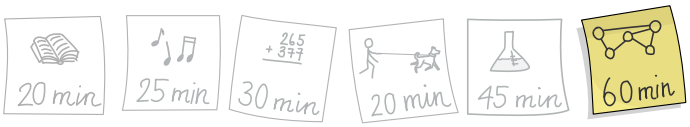
Nach 5 Minuten konzentriertem Rechnen fällt ihr der 20-minütige Spaziergang mit dem Hund ein. Diese Aufgabe wird zur verbleibenden Liste hinzugefügt und gemäss ihrer Bearbeitungszeit vorne einsortiert.



Nun geht Sarah als Nächstes mit dem Hund spazieren 

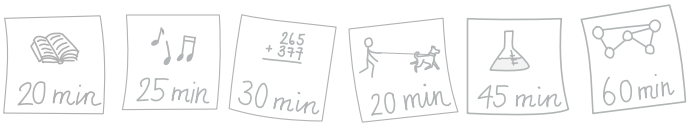
.

 © Informatik-Biber 2026, SVIA



Danach wählt sie die nächste Aufgabe in der Liste, nämlich Chemie 

.



Zuletzt wählt Sarah die insgesamt längste Aufgabe: Informatik 

. Dann ist ihre Aufgabenliste endlich leer.

Dies ist Informatik!

Sarah bearbeitet von ihren unerledigten Aufgaben immer diejenige, für die sie am wenigsten Zeit braucht. Auch die Informatik kennt diese Strategie, nämlich als *Shortest Job Next*.

Dies ist eine Planungsregel, die z.B. Betriebssysteme von Computern verwenden, um zu entscheiden, welche Aufgabe die CPU des Computers als Nächstes ausführen soll. Dabei wählt das Betriebssystem aus der Liste der noch nicht abgeschlossenen Aufgaben diejenige mit der voraussichtlich kürzesten Ausführungszeit aus. Damit diese Auswahl nicht unnötig Zeit kostet, wird die Liste in der Regel als *Vorrangwarteschlange* geführt, aus der effizient die nächste Aufgabe gewählt werden kann. Aufgaben können zu unterschiedlichen Zeitpunkten hinzukommen und in die Warteschlange einsortiert werden, so wie der Spaziergang mit dem Hund in dieser Biberaufgabe. In jedem Fall wird die neue Aufgabe frühestens dann ausgewählt, wenn die aktuelle Aufgabe beendet ist.

Der Vorteil von Shortest Job Next ist, dass kurze Aufgaben schnell abgearbeitet werden können, ohne dass lange Wartezeiten entstehen. Längeren Aufgaben kann es aber passieren, dass sie lange warten müssen (in der Informatik sagt man dazu auch «verhungern»), wenn immer wieder neue kurze Aufgaben hinzukommen. In dieser Biberaufgabe verhungert aber niemand, weder Sarah noch ihr Hund: Sarah hat zum Glück nicht allzu viele Aufgaben, und es kommt nur eine neue Aufgabe hinzu.

Stichwörter und Webseiten

- Shortest Job Next: `https://de.wikipedia.org/wiki/Shortest-Job-Next`

© Informatik-Biber 2026, SVIA

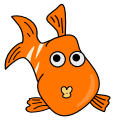
9. Fisch-App

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:

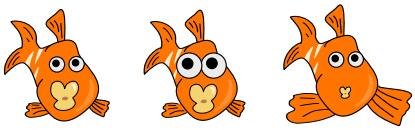


Jetzt nimmst du in der App einen Smiley, bei dem etwas gross ist. Je nachdem, was gross ist, wird beim Fisch eine bestimmte Sache auch gross. Zum Beispiel:

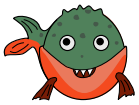
Smiley



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?

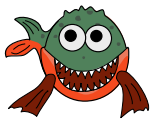
A)



B)



C)



D)



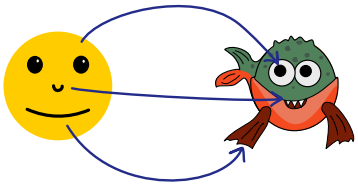
Lösung

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`

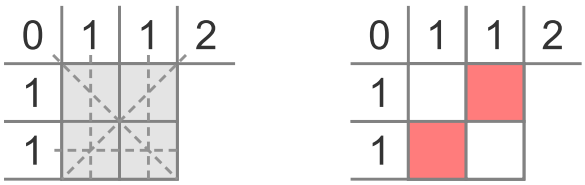
 © Informatik-Biber 2026, SVIA

11. 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.

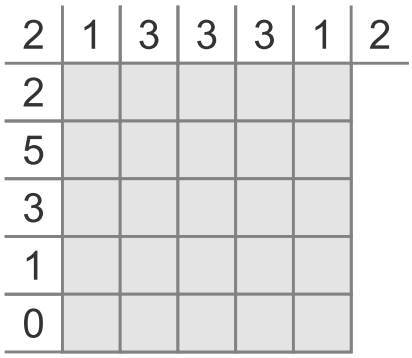
Nun ein kleines Beispiel mit einem 2×2-Quadrat:



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.

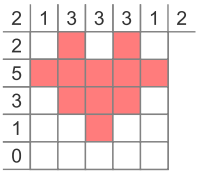
Die Färbung ist die einzige Lösung zu diesem Quadrat.

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

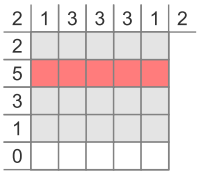


Lösung

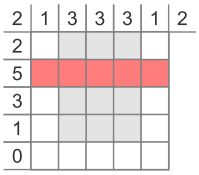
So ist es richtig:



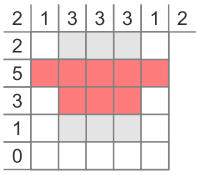
Es gibt keine andere Möglichkeit, die Felder gemäss der Vorgaben einzufärben. Das zeigt der nun beschriebene Weg, diese Lösung zu bestimmen:



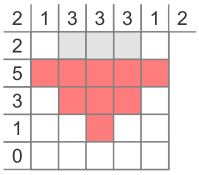
Schritt 1: Die Zeilen mit den Vorgaben **5** und **0** müssen vollständig rot bzw. weiss eingefärbt werden.



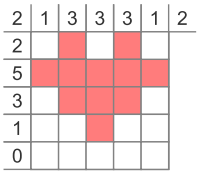
Schritt 2: Die noch grauen Felder in der ersten und der fünften Spalte müssen weiss sein; ein Feld ist ja bereits rot.



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

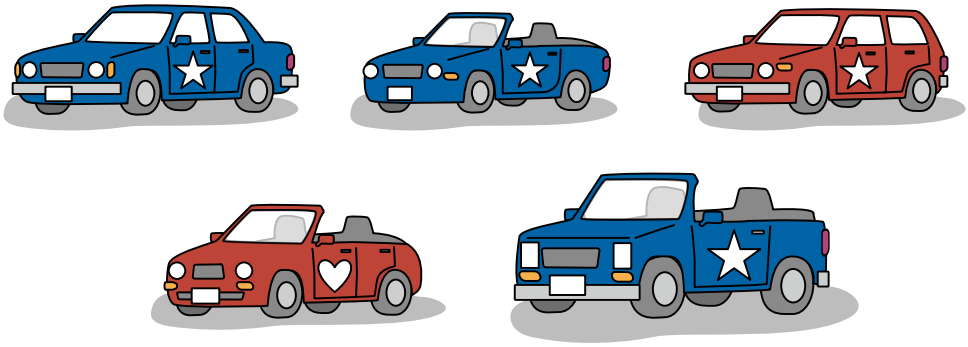
- Algorithmen: `https://de.wikipedia.org/wiki/Algorithmus`
- Suche im Lösungsraum: `https://de.wikipedia.org/wiki/Lösungsraum`
- Backtracking: `https://de.wikipedia.org/wiki/Backtracking`



© Informatik-Biber 2026, SVIA

12. 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.

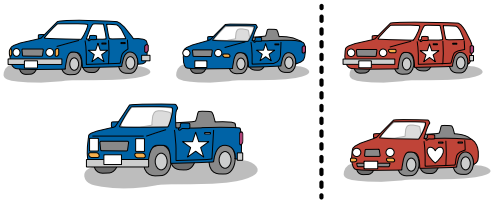
Welche drei Merkmale sind das? Kreise sie ein.

Autofarbe, Logo, Türen, Dach, Scheinwerfer

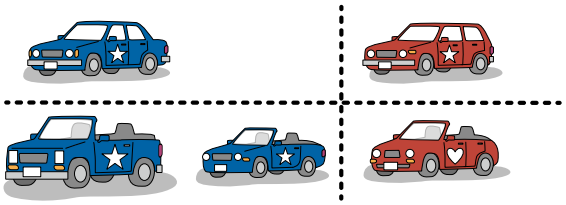
Lösung

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

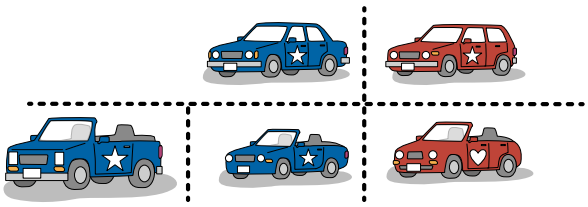
Mit jedem Merkmal kann der Roboter die Modelle in zwei Gruppen einteilen. Mit der richtigen Kombination aus drei Merkmalen kann er die Modelle in Gruppen einteilen, in denen jeweils nur noch ein Modell enthalten ist; dann kann er ein neues Auto diesem Modell zuordnen. Um die richtigen drei Merkmale zu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

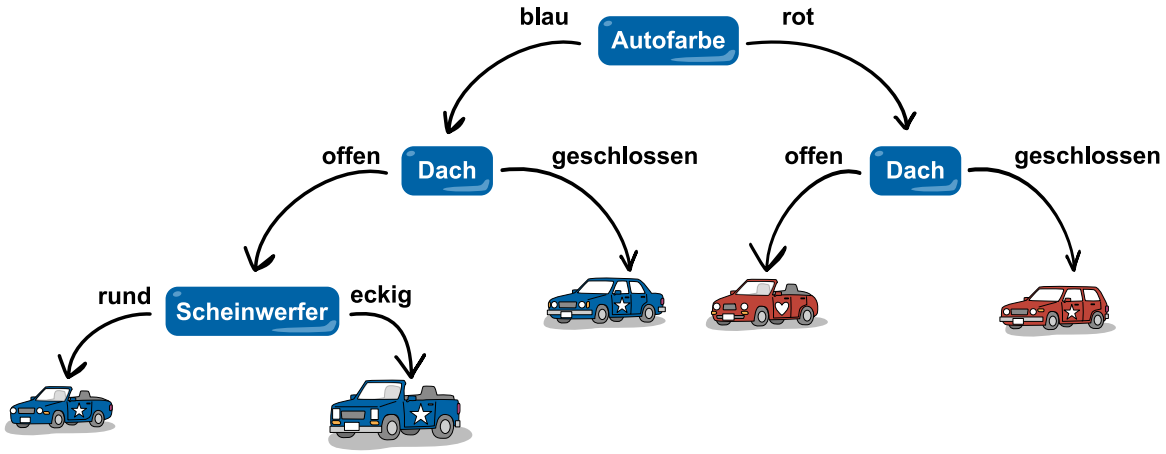
Merkmalskombination	Automodelle mit gleichen Merkmalen
Autofarbe, Logo, Türen	2 und 5
Autofarbe, Logo, Dach	2 und 5
Autofarbe, Logo, Scheinwerfer	1 und 2
Autofarbe, Türen, Dach	2 und 5
Autofarbe, Türen, Scheinwerfer	3 und 4
Logo, Türen, Dach	1 und 3 sowie 2 und 5
Logo, Türen, Scheinwerfer	1 und 3
Logo,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

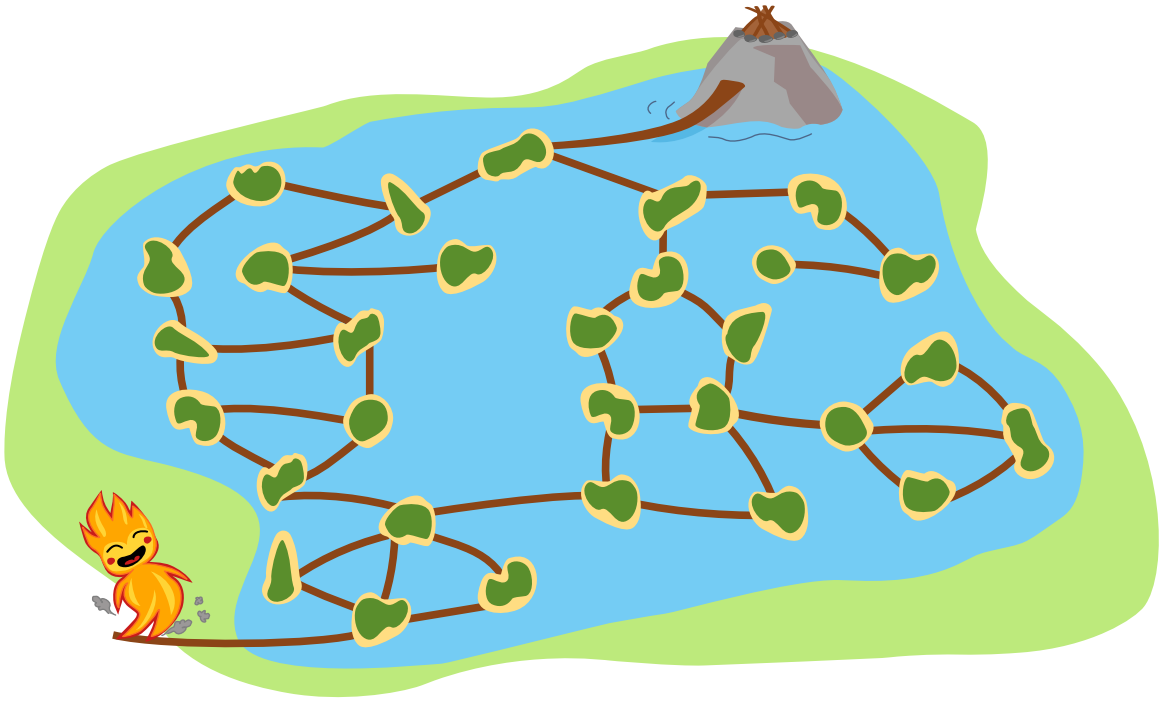
Stichwörter und Webseiten

- 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

13. 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.

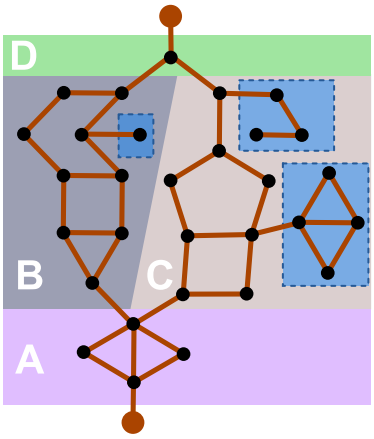
Der Feuergeist muss über die Stege gehen. Das Wasser im See würde ihn löschen. Die Stege aber sind aus Holz und brennen ab, sobald der Feuergeist darüber geht. Daher kann er über keinen Steg zweimal gehen. Die Inseln kann er aber mehrmals besuchen.

Wie viele Inseln kann der Feuergeist höchstens besuchen?

Lösung

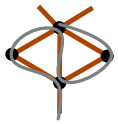
Die richtige Antwort ist 13 Inseln.

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

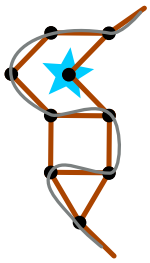


Wir können die Inseln im See in vier Abschnitte unterteilen: vier Inseln unten (A), die letzte Insel vor dem Felsen ganz oben (D) und dazwischen ein Abschnitt links (B) und einer rechts (C). Diese Abschnitte können wir unabhängig voneinander untersuchen, denn: Von jedem Abschnitt gibt es höchstens einen 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.

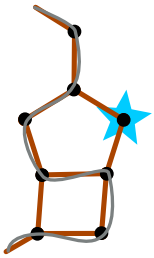
Nun schauen wir uns die einzelnen Abschnitte an.



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

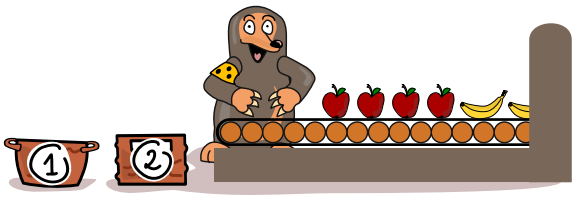
- Graph: `https://de.wikipedia.org/wiki/Graph_(Graphentheorie)`
- Graphentheorie: `https://de.wikipedia.org/wiki/Graphentheorie`

© Informatik-Biber 2026, SVIA

14. 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



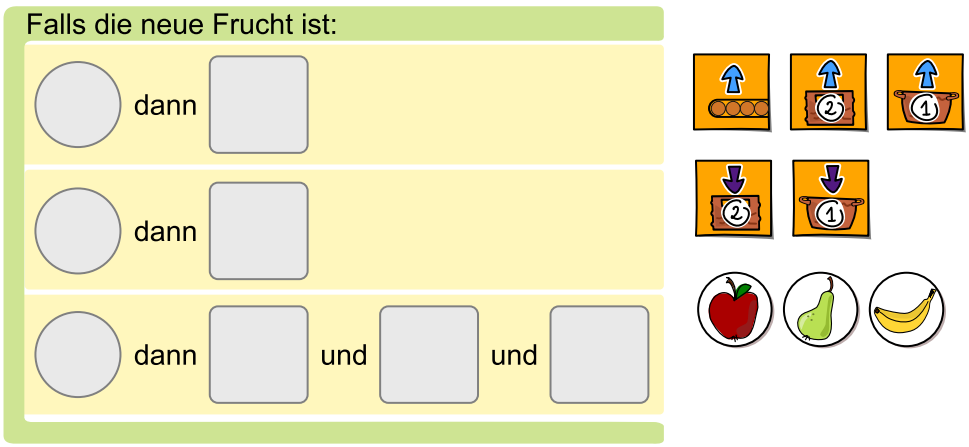
eine beliebige Frucht (Muddy kann nicht sehen, welche) aus einem der Körbe nehmen und beiseite legen



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

15. 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

16. Versteckspiel

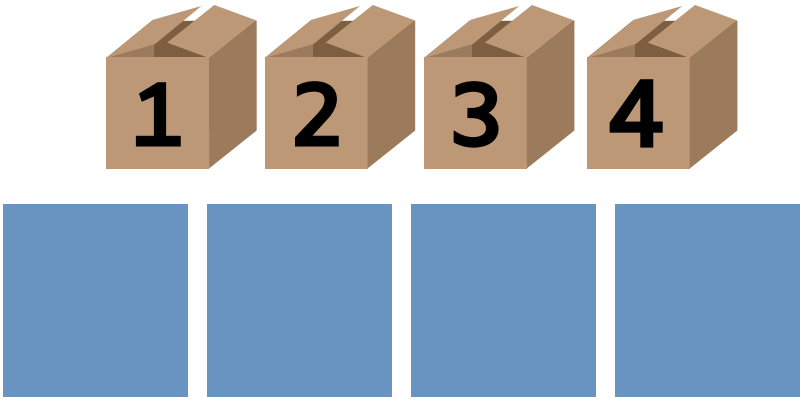
Ben versteckt sich in einer von vier nummerierten Kisten, z.B. in Kiste 2.



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.

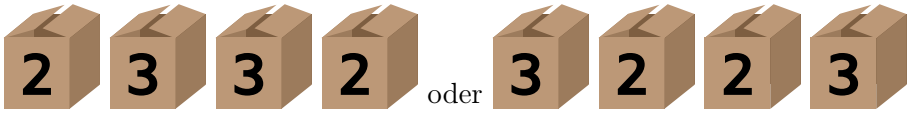
Gib eine solche Reihenfolge an!



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:

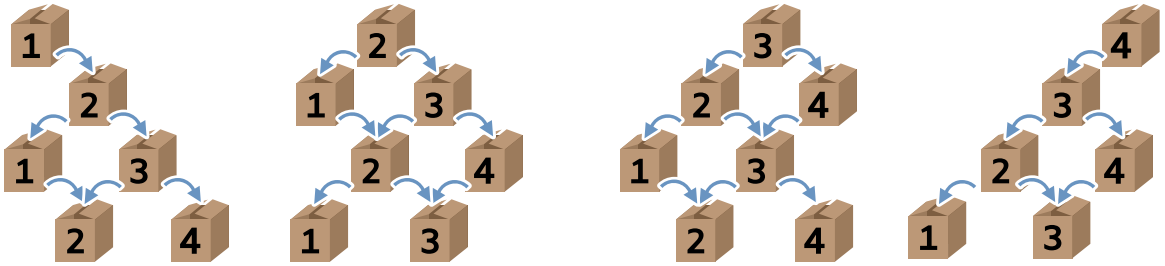


Wir zeigen, dass diese Antworten richtig sind. Dabei bezeichnen wir Bens Bewegungen als *Zug* in diesem Versteckspiel.

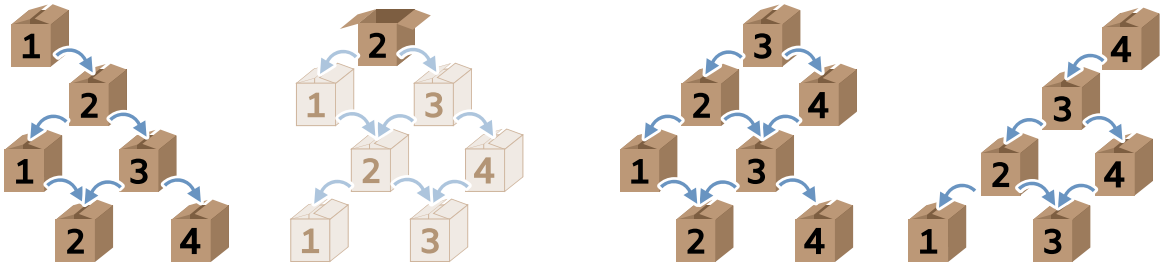
Am Anfang kann Ben seine «Start-Kiste» frei wählen. Danach hat Ben, wenn er

- 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:

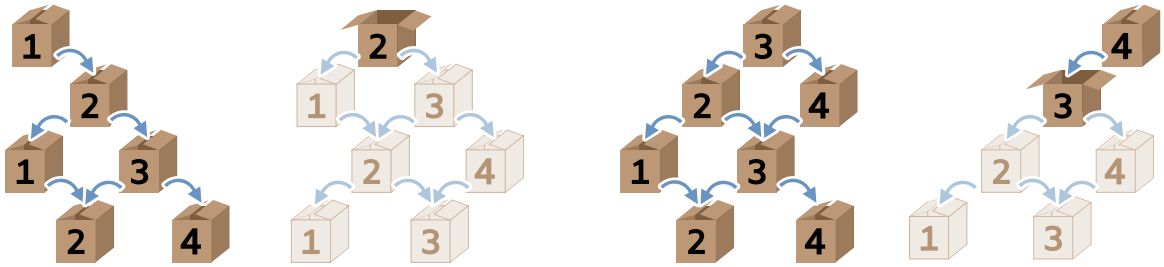


Was passiert, wenn Anja in der Reihenfolge **2-3-3-2** in die Kisten schaut? Zuerst schaut Anja in Kiste 2. Wenn sie Ben dort findet, ist Kiste 2 seine Start-Kiste, und er kann die von dort ausgehenden Züge nicht mehr machen.

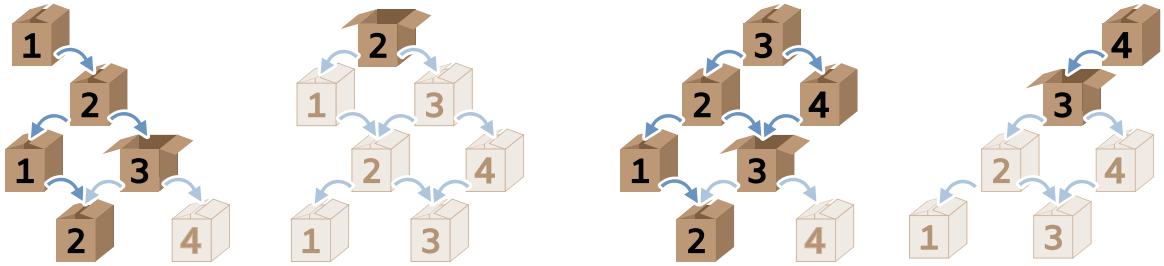


Dann schaut Anja in Kiste 3. Wenn sie Ben dort findet, ist das seine zweite Kiste. Auf der zweiten Ebene der Zugfolgen kommt Kiste 3 noch ganz rechts vor. Die von dort ausgehenden Züge kann Ben nun nicht mehr machen, und wir brauchen sie nicht weiter zu betrachten.

 © 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

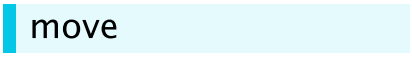
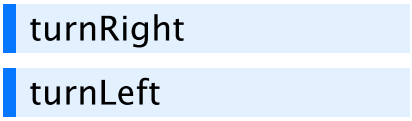
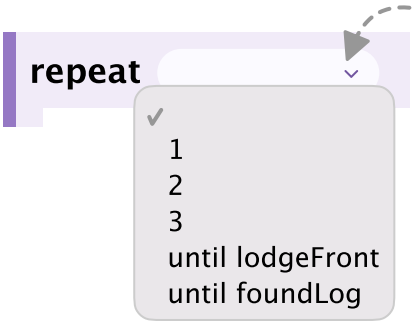
- Binärbaum: `https://de.wikipedia.org/wiki/Binärbaum`
- Spieltheorie: `https://de.wikipedia.org/wiki/Spieltheorie`

© Informatik-Biber 2026, SVIA

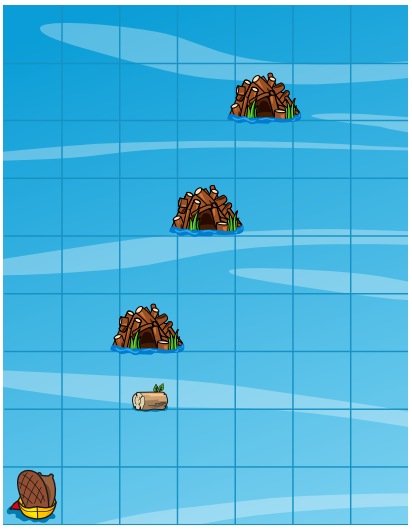
17. Quartierhilfe

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?

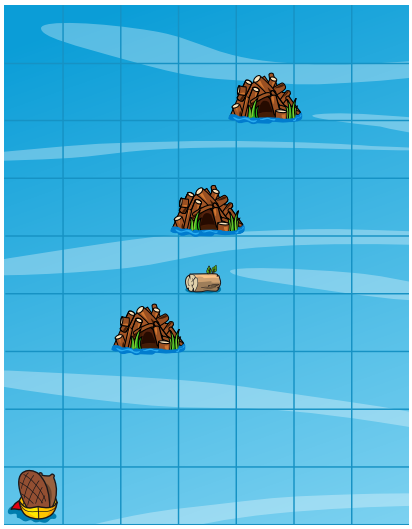
Du kannst folgende Anweisungen verwenden:

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.

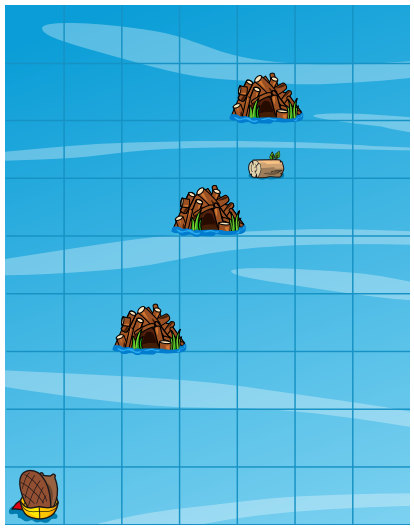
Notiere eine Wegbeschreibung, mit der Belinda an allen drei Tagen den Baumstamm einsammelt. Sobald sie ihn hat, soll sie bei keinem nachfolgenden Nachbarn vorbeifahren.



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

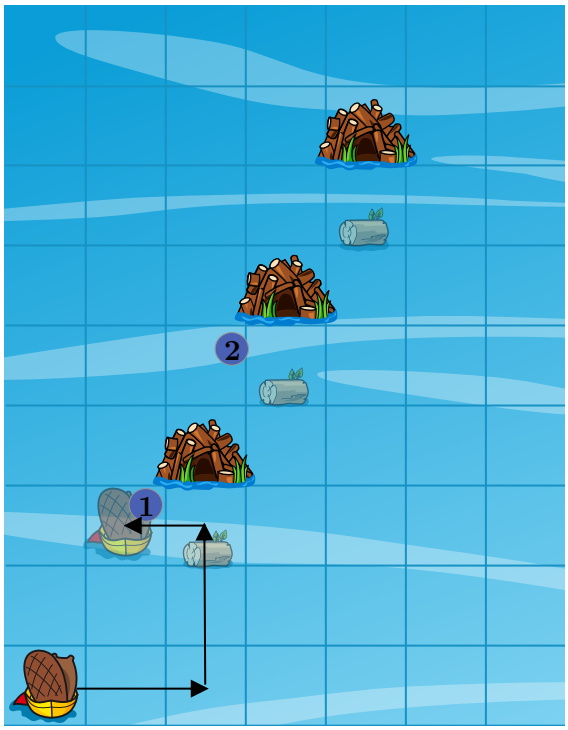
```
  turnRight()
  move()
```

```
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

Maxat Aubakirov
Masiar Babazadeh
Liam Baumann
Wilfried Baumann
Leonardo Cavalcante
Emily Cavalcanti
Cesar Cepeda
Maria Cepeda
Špela Cerar
Christian Datzko
Heather Duan
Nora A. Escherle
Susanne Falkenberg-Thut
Nadine Fröhlich
Gerald Futschek
Daniel Aagrén Geert
Arnheiður Guðmundsdóttir
Jens Hartmann
Yong-ju Jeon
Heidi Kaarto
Asterios Karagiannis
Geetanjali Khatri
Doyong Kim
Vaidotas Kinčius
Patrik Klofáč
Lukas Lehner
Yoshiaki Matsuzawa
Anna Morpurgo
Madhavan Mukund
Justina Oostendorp
Harriet Page
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
Maiko Shimabuku
Itsuki Shimada
Bronius Skūpas
Emma Sotomayor
Jacqueline Staub
Alieke Stijf
Pedro Suarez

Maciej M. Syslo

Yahya Tabesh

Ezra Templonuevo

Jiří Vaníček

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