INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

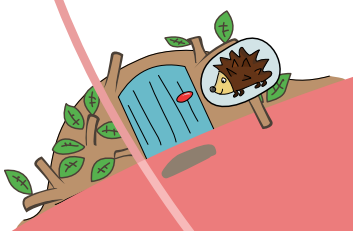
Aufgaben und Lösungen 2025

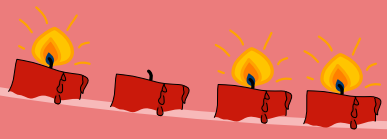
Alle Stufen

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

Herausgeber:

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





0101001101010110010010010100000100101101010100110101001101001010010100010100101010100110101001100100101001001010000101

SV!A

www.svia-ssie-ssii.ch
schweizerischerverein für informatikinderausbildung//société suisse pour l'informatiquedans l'enseignement//società svizzera 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 2025

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

Projektleitung: Nora A. Escherle

Herzlichen Dank für die Aufgabenentwicklung für den Schweizer Wettbewerb an:
Patricia Heckendorn, Gymnasium Kirschgarten
Juraj Hromkovič, Regula Lacher: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Jens Hartmann, Stephan Koch, Dirk Schmerenbeck und Jacqueline Staub: Universität Tier, Deutschland

Die Aufgabenauswahl wurde erstellt in Zusammenarbeit mit den Organisatoren von Bebras in Deutschland, Österreich und Ungarn. Besonders danken wir:
Philip Whittington, Silvan Horvath: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Wolfgang Pohl, Karsten Schulz, Franziska Kaltenberger, Margaretha Schlüter, Kirsten Schlüter, Michael Weigend: Bundesweite Informatikwettbewerbe (BWINF), Deutschland
Wilfried 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 Tier, Deutschland
Dirk Schmerenbeck: Universität Trier, Deutschland
Dave Oostendorp: cuttle.org, Niederlande

Für den Support während der Wettbewerbswochen danken wir:
Eveline Moor: Schweizer Verein für Informatik im Unterricht

Für die Organisation und Durchführung des Finales 2024 an der ETH danken wir:
Dennis Komm, Hans-Joachim Böckenhauer, Angélica Herrera Loyo, Andre Macejko, Moritz Stocker, Philip Whittington, Silvan Horvath: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Für die Korrektur der Finalaufgaben:
Clemens Bachmann, Morel Blaise, Tobias Boschung, Davud Evren, Jay Forrer, Sven Grübel, Urs Hauser, Fabian Heller, Jolanda Hofer, Alessandra Iacopino, Saskia Koller, Richard Královič, Jan

© Informatik-Biber 2025, SVIA

Mantsch, Adeline Pittet, Alexander Skodinis, Emanuel Skodinis, Jasmin Sudar, Valerie Verdan, Chris Wernke
Für die Übersetzung der Finalaufgaben ins Französische:
Jean-Philippe Pellet: Haute école pédagogique du canton de Vaud

Christoph Frei: Chragokyberneticks (Logo Informatik-Biber Schweiz)
Andrea Leu, Sarah Beyeler, 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 und Österreich verwendet.

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

© Informatik-Biber 2025, SVIA

INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Der Informatik-Biber 2025 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 die Abraxas Informatik AG, das Amt für Kindergarten, Volksschule und Beratung (AKVB) des Kantons Bern, Amt für Wirtschaft AWI des Kantons Zürich, die CYON AG 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 10. Dezember 2025 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 1. Dezember 2025 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. 172 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 2025 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 2025, 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
Nora A. Escherle

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

© Informatik-Biber 2025, SVIA

Inhaltsverzeichnis

Mitarbeit Informatik-Biber 2025 i

Vorwort iv

Inhaltsverzeichnis vi

1. Früchtekörbe 1

2. Fingerfarben 5

3. Flugzeuge 9

4. Baue mit Bauklötzen 13

5. Lefty I 17

6. Nach Hause 21

7. Hivobu 25

8. Holz für den Damm 29

9. Verrückte Lampe 33

10. Bibimbap 37

11. Zahlenmaschine 41

12. Blätter im Wind 45

13. Lichterstern 51

14. Dein Bauwerk 55

15. Blumentöpfe 59

16. Biberholz 63

17. Bebrasien 67

18. Lefty II 71

19. Momos Spiel 75

20. Ein Tag im Nebel 79

21. Stammbaum 83

22. Kurierdienst 85

23. Der Drachen ist weg! 89

© Informatik-Biber 2025, SVIA

24. Brennende Kerzen . . . 93
25. Helligkeitskarte . . . 97
26. Mehltransport . . . 101
27. Schwarz - Weiss . . . 105
28. Biber-Mail-Adressen . . . 109
29. Biberone . . . 113
30. ÖV . . . 117
31. Seoul entdecken! . . . 121
32. Bergseen . . . 125
33. Parkplätze . . . 129
34. Biber Jones . . . 133
35. Prüfungsplan . . . 137
36. Prüf-Biber . . . 141
37. Schere, Stein, Papier . . . 145
38. Nur ein Weg . . . 151
39. Verrückte Sandbank . . . 153
40. Wertvoller Baumstamm . . . 157
41. Noch mehr Holz . . . 161
42. Um die Felsen . . . 165
43. Hin und Her . . . 169
A. Aufgabenautoren . . . 172
B. Akademische Partner . . . 174
C. Sponsoring . . . 175

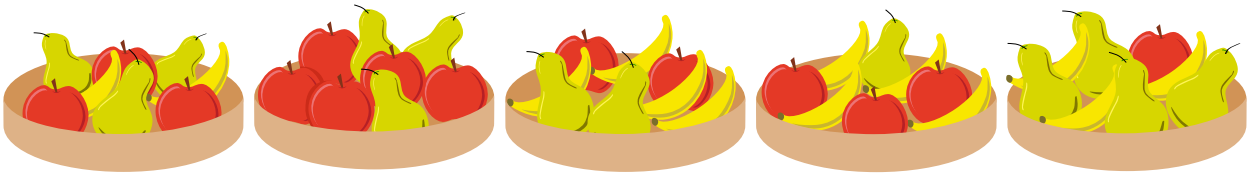
© Informatik-Biber 2025, SVIA

1. Früchtekörbe

Bella hat gern Früchte. Am liebsten hat sie Äpfel  , dann Bananen  und am wenigsten Birnen  .

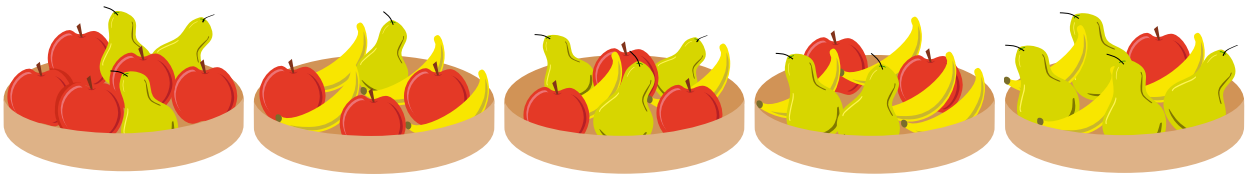
Bella hat fünf Körbe mit Früchten. Sie möchte diese Körbe nach ihrer Vorliebe anordnen. Je mehr Äpfel ein Korb hat, desto weiter links soll er stehen. Wenn zwei Körbe gleich viele Äpfel haben, soll der Korb mit mehr Bananen weiter links stehen.

Ordne die Körbe so an, wie Bella das möchte.

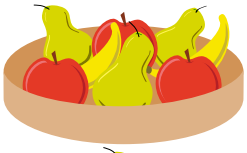
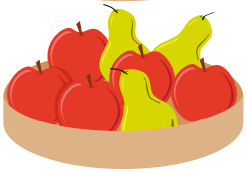
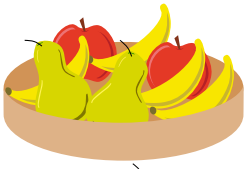
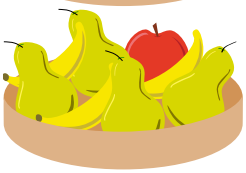


Lösung

So ist es richtig:



Wir zählen die Fruchtsorten in jedem Korb:

Korb	Bild	Anzahl Äpfel	Anzahl Bananen	Anzahl Birnen
A		**3**	**2**	3
B		**5**	0	3
C		**2**	4	2
D		**3**	**3**	1
E		**1**	3	4

In Korb B sind die meisten Äpfel, nämlich 5; er muss ganz links stehen. Danach kommen die Körbe A und D mit jeweils 3 Äpfeln. Weil Korb D mehr Bananen hat, kommt er als nächster in die Reihe, danach Korb A. Von den beiden übrigen Körben kommt Korb C mit 2 Äpfeln vor Korb E mit nur 1 Apfel. Insgesamt lautet die Reihenfolge also: B, D, A, C, E.

Dies ist Informatik!

Bella möchte in dieser Biberaufgabe die Früchtekörbe nach ihren Vorstellungen *sortieren*, also in eine ganz bestimmte Ordnung bringen. Da sie Äpfel am liebsten mag, ist es für sie sinnvoll, den Korb mit den meisten Äpfeln ganz vorne in der Reihe zu haben: Wenn die Körbe so sortiert sind, weiss sie genau, wie sie am schnellsten an viele Äpfel kommt.

 © Informatik-Biber 2025, SVIA

Man kann Dinge sortieren, wenn man für jeweils zwei Dinge immer entscheiden kann, in welcher Ordnung sie zueinander stehen. Dazu braucht es mindestens ein Merkmal, das alle zu sortierenden Dinge haben, und nach dem man sie unterscheiden kann. Für die Werte, die das Merkmal haben kann, muss es eine klare Ordnung geben. Zum Beispiel hat jedes Buch mindestens einen Autor oder eine Autorin, und wir können den Namen des ersten Autors bzw. der ersten Autorin als Merkmal nehmen. Namen kann man alphabetisch ordnen, wenn man alle Buchstaben und auch die Länge der Namen berücksichtigt. Dann kommt Meier vor Meyer, und Schach kommt vor Schacht. So können wir die Bücher nach dem Namen des ersten Autors bzw. der ersten Autorin sortieren.

Für die Informatik ist Sortieren eine ganz wichtige Angelegenheit. Computer sortieren ständig Daten, insbesondere weil sie in sortierten Daten schneller suchen können. Die Informatik kennt sehr viele Verfahren (in der Fachsprache *Algorithmen* genannt), mit denen Daten sortiert werden können. Es ist selbst für Informatikerinnen und Informatiker nicht immer einfach zu entscheiden, welches davon für eine gegebene Sortieraufgabe am besten geeignet ist.

Stichwörter und Webseiten

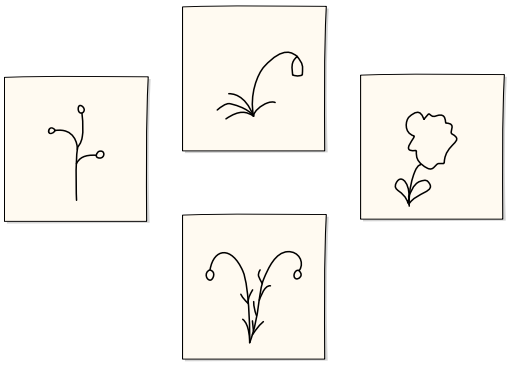
- Sortieren nach verschiedenen Eigenschaften: `https://pikas-mi.dzlm.de/inhalte/formen-erkunden/unterricht/aufgabenstellung-kompakt-„formen-sortieren“`

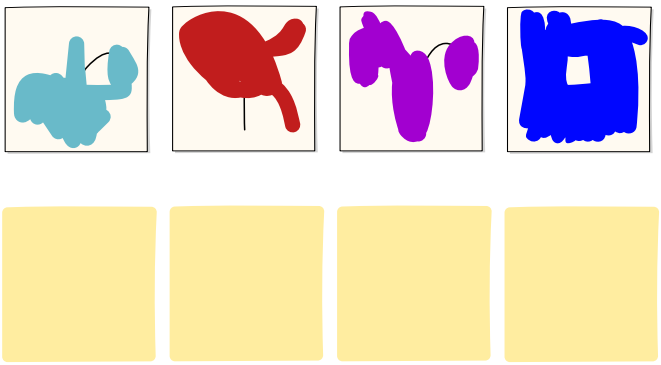
© Informatik-Biber 2025, SVIA

2. Fingerfarben

Lars hat Blumen gezeichnet. Seine kleine Schwester Carlotta findet die Zeichnungen und bemalt sie mit Fingerfarben.

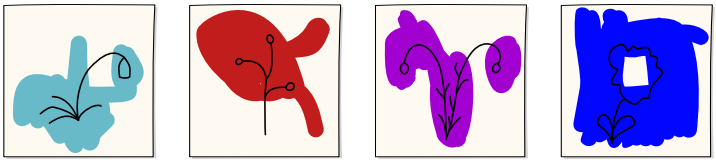
Wie sahen die bemalten Zeichnungen vorher aus?





Lösung

So ist es richtig: Hier siehst Du, was sich auf den bemalten Zeichnungen (Bilder A bis D) hinter den Fingerfarben verbirgt.



A B C D

Um die richtige Antwort zu finden, vergleichen wir

1. die schwarzen Linien jeder bemalten Zeichnung mit den Linien aller unbemalten Zeichnungen und
2. die weissen Bereiche jeder bemalten Zeichnung mit den weissen Bereichen aller unbemalten Zeichnungen.

Dabei stellen wir fest:

- Es gibt nur eine unbemalte Zeichnung mit einem weissen Bereich in der oberen linken Ecke wie in Bild A.
- Es gibt nur eine unbemalte Zeichnung mit einem Stiel in der unteren Hälfte wie in Bild B.
- Es gibt nur eine unbemalte Zeichnung mit einem weissen Bereich in der Mitte wie in Bild D.
- Damit bleibt genau eine unbemalte Zeichnung für Bild C übrig.

Dies ist Informatik!

Die kleine Carlotta hat die Zeichnungen ihres Bruders mit Fingerfarben bemalt. Danach konnte man kaum noch erkennen, wie die unbemalten Zeichnungen aussahen. Bei der Restaurierung von Kunstwerken in Museen bemühen sich Expertinnen und Experten, übermalte oder beschädigte Kunstwerke wieder in ihren ursprünglichen Zustand zu versetzen. Dabei müssen sie oft passende Referenzen aus anderen Quellen finden, um das Originalwerk genau wiederherzustellen. Dieser Prozess ähnelt dem, was du gerade in dieser Biberaufgabe gemacht hast.

In der Informatik werden Restaurierungstechniken in Systemen verwendet, die völlig neue Bilder erzeugen. Diese fortschrittlichen Systeme werden als *Diffusionsmodelle* bezeichnet. Ein Diffusionsmodell wird anhand zahlreicher Bilder trainiert, um zu lernen, wie beschädigte digitale Bilder restauriert werden können. Mit diesem Wissen kann das System dann aus Bildern, die aus vielen kleinen, verstreuten Punkten bestehen, neue Bilder erstellen. Es gibt jedoch einen wesentlichen Unterschied zu deinem Vorgehen bei dieser Aufgabe. Du hast die Aufgabe mit logischem Denken gelöst, während das Diffusionsmodell nicht mit Logik arbeitet. Stattdessen erlangt es seine Restaurierungsfähigkeiten, indem es mithilfe von *maschinellem Lernen* viele Beispiele analysiert.

 © Informatik-Biber 2025, SVIA

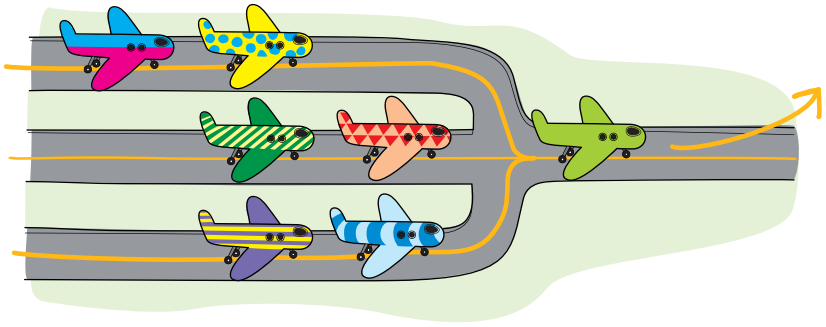
Stichwörter und Webseiten

- *Diffusionsmodell*: `https://www.ibm.com/de-de/think/topics/diffusion-models`
- *maschinelles Lernen*: `https://de.wikipedia.org/wiki/Maschinelles_Lernen`

© Informatik-Biber 2025, SVIA

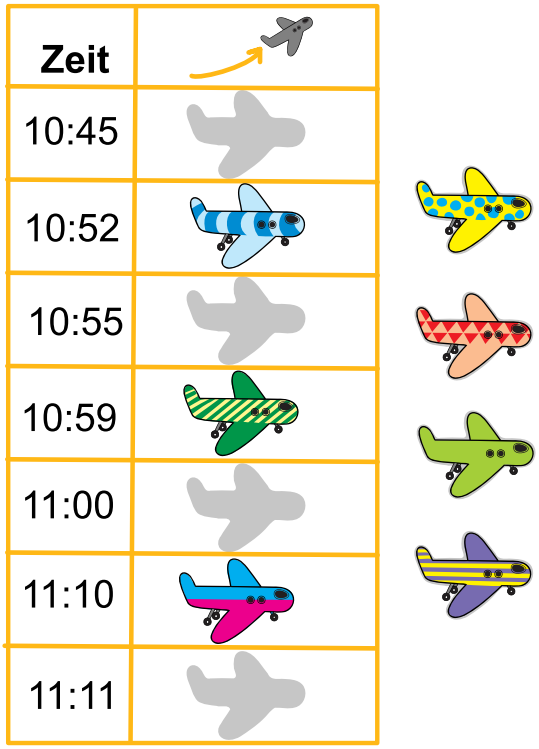
3. Flugzeuge

Heute Morgen wollen sieben Flugzeuge starten. Alle starten auf derselben Startbahn rechts. Die Flugzeuge fahren auf den Linien vorwärts und können einander nicht überholen.



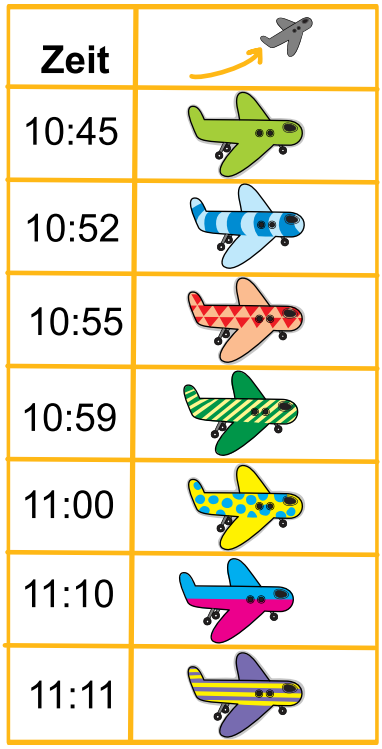
Der Startplan zeigt, in welcher Reihenfolge die sieben Flugzeuge starten. Einige Flugzeuge fehlen aber noch.

Ergänze die fehlenden Flugzeuge im Startplan.



Lösung

So ist es richtig:



1. Als erstes (um 10:45 Uhr) startet das Flugzeug  ; es steht schon auf der Startbahn.
2. Als zweites (um 10:52 Uhr) startet das Flugzeug  .
3. Als drittes (um 10:55 Uhr) startet das Flugzeug  : Es steht vor dem Flugzeug  . Damit das als nächstes starten kann, wie im Plan angegeben, muss vorher das Flugzeug gestartet sein.
4. Als viertes (um 10:59 Uhr) startet das Flugzeug  .
5. Als fünftes (um 11:00 Uhr) startet das Flugzeug  : Es steht vor dem Flugzeug  . Damit das als nächstes starten kann, wie im Plan angegeben, muss vorher das Flugzeug gestartet sein.
6. Als sechstes (um 11:10 Uhr) startet das Flugzeug  .
7. Als siebtes und letztes (um 11:11 Uhr) startet das Flugzeug  . Alle anderen Flugzeuge sind schon gestartet.

Dies ist Informatik!

Die Startreihenfolge der Flugzeuge hängt auch von der Reihenfolge beim Warten ab. Einige Flugzeuge können erst dann starten, wenn andere, die beim Warten vor ihnen stehen, bereits gestartet sind. An einem echten Flughafen müssen bei der Planung der Startreihenfolge noch viele andere Bedingungen berücksichtigt werden, zum Beispiel Verspätungen, technische Probleme oder Wetteränderungen. Für

 © Informatik-Biber 2025, SVIA

die Planung werden meist Computerprogramme verwendet, die auf Änderungen bei den Bedingungen schnell reagieren und dennoch gute Pläne erstellen können.

Die Erstellung von Zeitplänen, wie dem Startplan in dieser Biberaufgabe, fällt in der Informatik in den Bereich *Scheduling*. Scheduling kann kompliziert werden, zum Beispiel wenn nur eine Ressource (z. B. eine Startbahn) zur Verfügung steht, wenn bestimmte Ereignisse voneinander abhängig sind oder wenn eine Reihenfolge eingehalten werden muss, um Probleme zu vermeiden.

In der Informatik gilt Scheduling als besonders schwieriges Problem. Die Herausforderung besteht darin, für jede denkbare Situation einen optimalen Plan zu berechnen. Wenn viele Parameter und Bedingungen berücksichtigt werden müssen, kann selbst ein sehr schneller Computer dafür enorm viel Zeit benötigen. Informatik-Fachleute sagen: Scheduling ist *NP-schwer*. Das bedeutet: Es ist zwar einfach zu überprüfen, ob ein gegebener Plan korrekt ist, aber extrem schwierig, den besten Plan überhaupt zu finden. Das ist wie bei einem besonders schwierigen Rätsel: Die Lösung zu prüfen ist leicht, sie zu finden kann dagegen sehr aufwändig sein.

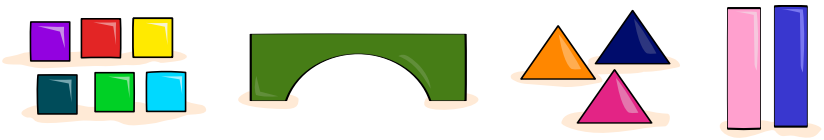
Stichwörter und Webseiten

- Scheduling: `https://de.wikipedia.org/wiki/Scheduling`
- NP-hart, NP-Vollständigkeit: `https://de.wikipedia.org/wiki/NP-Vollständigkeit`

© Informatik-Biber 2025, SVIA

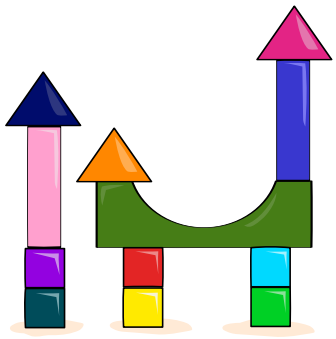
4. Baue mit Bauklötzen

Ali hat diese Bauklötze:



6 Würfel 1 Brücke 3 Pyramiden 2 Quader

Alis Schwester Zaha gibt ihm nach und nach Anweisungen, was er mit den Bauklötzen tun soll. Ali erledigt jede Anweisung sofort. Am Ende entsteht dieses Bauwerk:



In welcher Reihenfolge hat Zaha die Anweisungen gegeben?

Die einzelnen Anweisungen müssen in beliebiger Reihenfolge angeordnet werden können. Es gibt diese fünf Anweisungen:

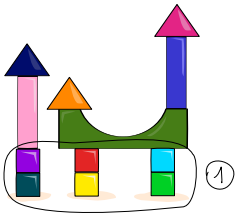
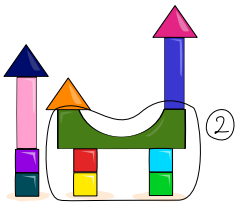
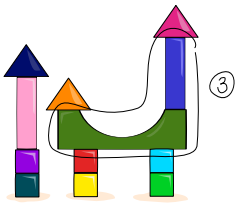
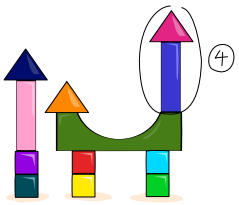
- Stelle beide Quader auf dein Bauwerk.
- Stelle die Würfel-Türme in eine Reihe.
- Lege die Pyramiden auf dein Bauwerk.
- Baue 3 Türme mit jeweils 2 Würfeln.
- Lege die Brücke auf dein Bauwerk.

Lösung

Zaha hat die Anweisungen in dieser Reihenfolge gegeben:

1. Baue 3 Türme mit jeweils 2 Würfeln.
2. Stelle die Würfel-Türme in eine Reihe.
3. Lege die Brücke auf dein Bauwerk.
4. Stelle beide Quader auf dein Bauwerk.
5. Lege die Pyramiden auf dein Bauwerk.

Wir erklären, warum genau diese Reihenfolge Alis Bauwerk entstehen lässt.

Bild	Schritt
	Weil die Würfel-Türme die einzigen Teile sind, die direkt auf dem Boden stehen, müssen sie als erste gebaut (1. Anweisung) und danach in eine Reihe gestellt werden (2. Anweisung).
	Danach muss die Brücke auf das bisherige Bauwerk gelegt werden (3. Anweisung). Sie muss auf die Würfel-Türme gelegt werden, aber unter den Quadern und Pyramiden sein.
	Danach müssen die Quader aufgestellt werden (4. Anweisung). Sie stehen nämlich direkt auf der Brücke, sind aber unter den Pyramiden.
	Zuletzt werden die Pyramiden gelegt (5. Anweisung). Sie liegen auf den Quadern, und es gibt keinen anderen Klotz, der auf die Pyramiden gebaut werden soll.

Da die meisten Anweisungen sagen, dass Klötze auf das (bisherige) Bauwerk gesetzt werden sollen, hängt die Reihenfolge der Anweisungen direkt damit zusammen, welche Klötze auf bzw. unter welchen anderen stehen.

Es ist nicht möglich, das Bauwerk mit einer anderen Reihenfolge der Anweisungen zu bauen.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Wenn eine Menge von Bauklötzen einzeln nebeneinander auf dem Boden stehen, kann man im Nachhinein nicht sagen, in welcher Reihenfolge die Klötze auf den Boden gestellt wurden. Die Handlungen, also das Legen oder Stellen eines Klotzes auf den Boden, sind voneinander unabhängig. Wenn man Bauklötze aufeinander stellt, geht das nur der Reihe nach, und der unterste Klotz wurde zuerst, der oberste Klotz zuletzt benutzt. Das Ablegen des untersten Klotzes ist Voraussetzung dafür, dass der nächste Klotz darauf abgelegt werden kann.

Computerprogramme bestehen aus einer Folge einzelner Anweisungen, so wie die von Zaha in dieser Biberaufgabe. Dabei können die einzelnen Anweisungen einfach sein und entsprechend einfache Auswirkungen haben, sie können aber auch sehr kompliziert sein. Auf jeden Fall ist es wichtig, dass Informatikerinnen und Informatiker sich beim Programmieren sehr gut überlegen, welche Wirkungen die einzelnen Anweisungen haben - und welche Wirkungen anderer Anweisungen eine Anweisung voraussetzt. Wenn man sich das vor dem Programmieren genau überlegt, hat man keine Schwierigkeiten, die Anweisungen im Programm in die richtige Reihenfolge zu bringen - so wie du das in dieser Biberaufgabe gemacht hast.

Stichwörter und Webseiten

- *Anweisung*: `https://de.wikipedia.org/wiki/Anweisung_(Programmierung)`

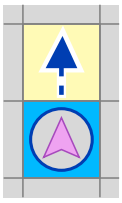
© Informatik-Biber 2025, SVIA

5. Lefty I

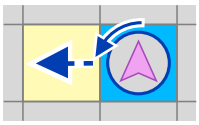
Roboter *Lefty* bewegt sich über ein Raster mit quadratischen Feldern. Zwischen Feldern kann es rote Mauern geben. Lefty soll das grüne Ziel erreichen.

Lefty kann sich auf genau zwei Arten bewegen:

Ein Feld vorwärts fahren

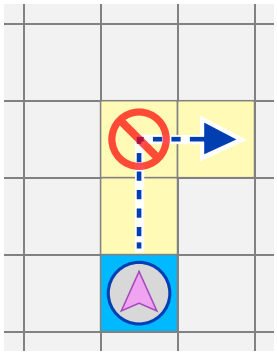


Nach links drehen und dann sofort ein Feld vorwärts fahren

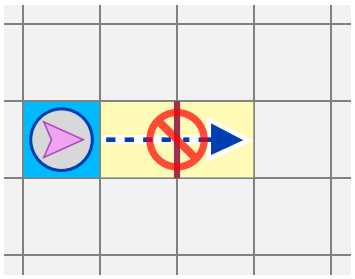


Lefty kann aber nicht alles. Zum Beispiel kann er

... **nicht** einfach rechts abbiegen und

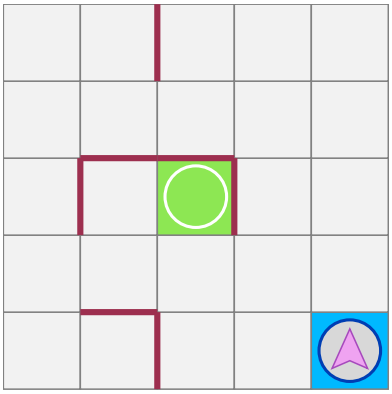


... **nicht** durch Mauern fahren.



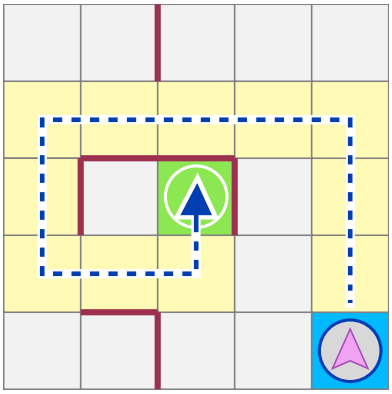
*Über welche Felder **muss** Lefty fahren, um das Ziel zu erreichen?*

*Wähle **so wenige Felder wie möglich** aus.*



Lösung

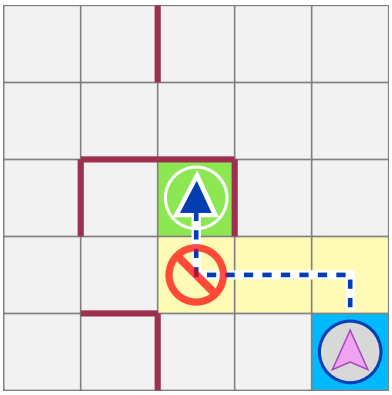
So ist es richtig:



Wenn Lefty über diese Felder fährt, erreicht er das Ziel. Dabei bewegt er sich nur auf die beiden Arten, die er kann.

Es gibt keinen anderen Weg mit weniger oder gleich vielen Feldern, auf dem Lefty das Ziel erreicht.

Den direkten Weg kann Lefty nicht nehmen, weil er dafür rechts abbiegen müsste.



Dies ist Informatik!

Armer Lefty! Seine Funktionsweise ist stark eingeschränkt. Wenn er doch nur andere Bewegungen machen könnte! Wenn er sich nach rechts drehen und vielleicht sogar über Mauern klettern könnte, wäre sein Leben im Raster viel einfacher. Lefty würde sich viel selbstbewusster fühlen, wenn er einen *komplexeren Befehlssatz* hätte. (Bei einem Roboter werden die grundlegenden Dinge, die er tun kann, durch entsprechende Befehle in der Software des Roboters gesteuert).

Aber ist das wirklich notwendig? Lefty könnte sich zum Beispiel nach rechts drehen, indem er sich dreimal hintereinander nach links dreht. Wir müssen nur die Regel abschaffen, dass sich Lefty nach der Linksdrehung sofort nach vorne bewegen muss. Dann könnte er sich in alle Richtungen drehen und fahren. Und anstatt über eine Mauer zu klettern, könnte er um sie herumfahren, wenn dafür genug Platz ist. Hey Lefty, du kannst mit deinem *reduzierten Befehlssatz* glücklich und zufrieden leben - solange die, die dich programmieren, die Befehle klug einsetzen.

 © Informatik-Biber 2025, SVIA

In der Informatik sind genau diese beiden Ansätze, den Befehlssatz eines *Prozessors* zu gestalten, am weitesten verbreitet: Manche Prozessoren sind ein CISC (Complex Instruction Set Computer / deutsch: Computer mit komplexem Befehlssatz), andere sind ein RISC (Reduced Instruction Set Computer / Computer mit reduziertem Befehlssatz) - wie jener von Lefty in dieser Biberaufgabe. Ein CISC hat in der Regel viele verschiedene Befehle, die sehr mächtig sein können (wie das Klettern über eine Mauer), aber dafür seltener verwendet werden. Ein RISC hingegen hat nur wirklich nötige Befehle mit eher einfacher Wirkung, die dann häufig verwendet werden.

Beide Arten von *Architekturen* haben ihre Vor- und Nachteile. Die Prozessoren bekannter Marken sind entweder CISC- oder RISC-Prozessoren, wobei RISC-Prozessoren in letzter Zeit etwas beliebter geworden sind.

Stichwörter und Webseiten

- Prozessor: `https://de.wikipedia.org/wiki/Prozessor`
- Befehlssatz: `https://de.wikipedia.org/wiki/Befehlssatz`
- CISC: `https://de.wikipedia.org/wiki/Complex_Instruction_Set_Computer`
- RISC: `https://de.wikipedia.org/wiki/Reduced_Instruction_Set_Computer`

© Informatik-Biber 2025, SVIA

6. Nach Hause

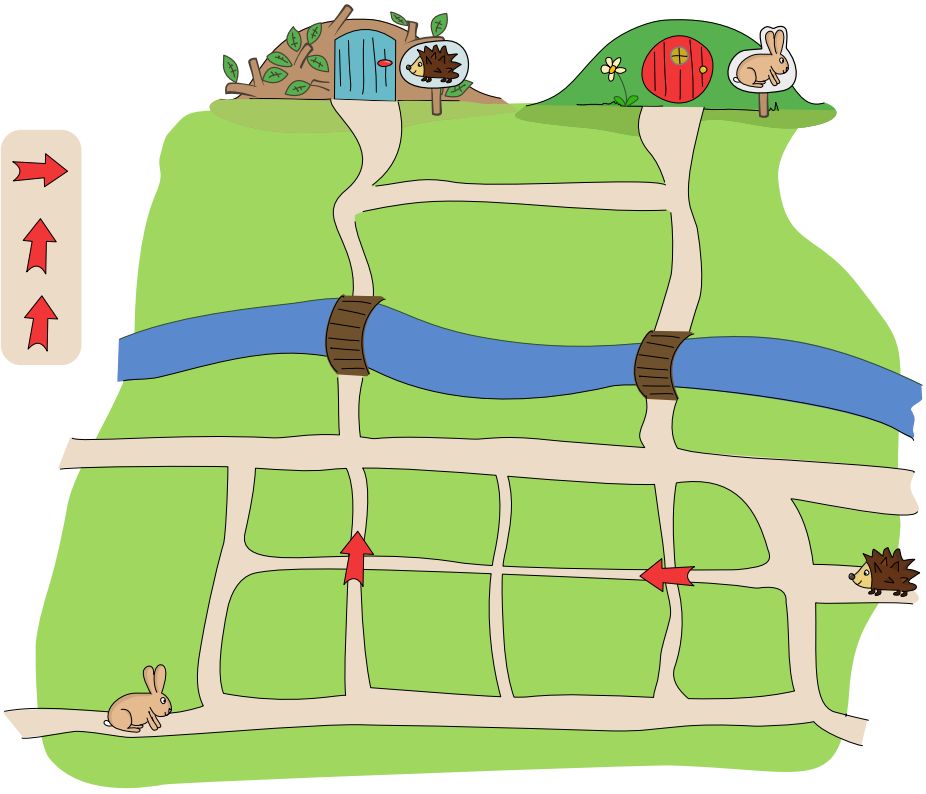
Ein Hase  und ein Igel  wollen nach Hause.

Jeder hat ein eigenes Haus:  und  .

Hase und Igel laufen auf den Wegen, und zwar geradeaus. Nur wenn sie zu einer Kreuzung mit Pfeil kommen, folgen sie der Richtung des Pfeils.

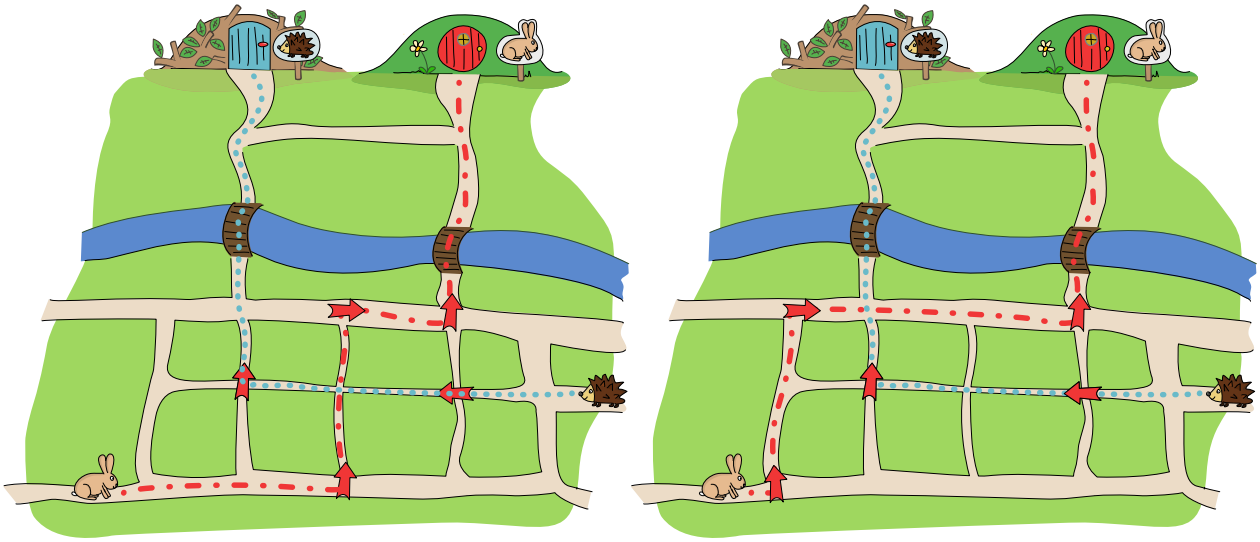
An einigen Kreuzungen liegen schon Pfeile.
Entlang der Pfeile findet der Igel nach Hause, der Hase aber nicht. Zum Glück sind noch drei Pfeile übrig.

*Lege die drei übrigen Pfeile auf Kreuzungen, so dass Hase **und** Igel **beide** nach Hause finden.*



Lösung

Es gibt zwei richtige Antworten:



Um nach Hause zu kommen, muss der Hase irgendwann nach «oben» (in Richtung der Häuser) gehen. Das kann er an der ersten oder der dritten Kreuzung machen, an die er kommt.

Wichtig ist, dass Hase und Igel auf ihren Wegen nie zu demselben Pfeil kommen, weil sie sonst beide von dort aus denselben Weg laufen und somit beide am selben Haus ankommen würden. Das würde zum Beispiel passieren, wenn der Hase an der zweiten oder vierten Kreuzung nach oben geht und zu Pfeilen kommt, die dem Igel den Weg nach Hause weisen. Auch an der fünften Kreuzung kann der Hause nicht nach oben gehen; er müsste dann später nach links, aber es ist kein Pfeil übrig, der nach links zeigt.

Der Hase kann also nur an der ersten oder dritten Kreuzung nach oben gehen. Die anderen Pfeile müssen dann jeweils so gelegt werden wie oben abgebildet. Nur so kommt der Hase nach Hause, und der Weg des Igels nach Hause wird nicht gestört.

Dies ist Informatik!

Auf ihrem Weg folgen Hase und Igel an den Kreuzungen dieser Vorschrift: «Wenn an der Kreuzung ein Pfeil liegt, folge der Richtung des Pfeils. Sonst gehe geradeaus.»

In der Informatik nennt man solch eine Vorschrift *Algorithmus*. Der Algorithmus bleibt immer gleich: Er sagt, ob und wie man die Richtung ändert, wenn man an eine Kreuzung kommt. Die Richtungsänderungen werden von den Pfeilen bestimmt. Sie sind Daten, die der Algorithmus *verarbeitet*; in der Informatik spricht man auch von *Eingaben* für den Algorithmus. Im Hase-Igel-Wege-Algorithmus dieser Biberaufgabe bestimmt die Eingabe, also Lage und Art der Pfeile, ob die *Ausgabe* des Algorithmus richtig ist, also ob die Wege, welche die Tiere gehen, sie beide nach Hause führen.

Eine ähnliche Vorschrift könnte es beim Waschen eines Kleidungsstücks geben: «Wähle die Waschtemperatur so wie auf dem Waschzettel angegeben.» Nur wenn die Eingabe, also die Temperatur-Angabe

 © Informatik-Biber 2025, SVIA

auf dem Waschzettel, stimmt, ist auch die Ausgabe richtig, also ein sauberes Kleidungsstück, das noch genau so gross ist wie vor dem Waschen.

Diese Beispiele zeigen, dass es kritisch ist, wenn die Qualität der Ausgaben eines Algorithmus von der Qualität der Eingaben abhängt. Das ist zum Beispiel bei vielen KI-Systemen der Fall. KI-Systeme arbeiten mit Modellen der Wirklichkeit, und diese Modelle werden aus Daten gebildet. Sind diese Daten nicht gut ausgewählt, wird auch das Modell nicht gut funktionieren. Ein KI-Modell von Krankheiten etwa wird für Frauen nicht gut funktionieren, wenn die Eingabedaten nur von Männern stammen.

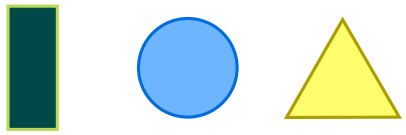
Stichwörter und Webseiten

- Pathfinding: `https://de.wikipedia.org/wiki/Pathfinding`

© Informatik-Biber 2025, SVIA

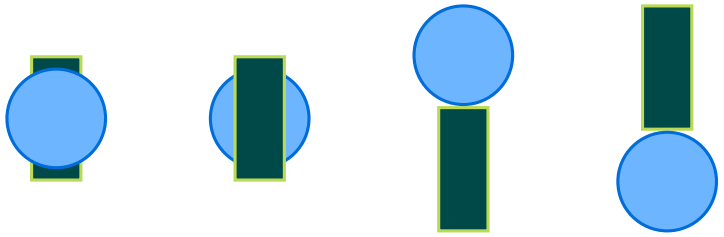
7. Hivobu

Im Land Hivobu heissen diese drei Formen:



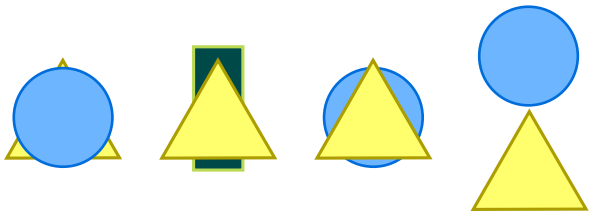
RAH OH TEH

Wenn man in Hivobu zwei Formen hintereinander oder untereinander legt, heisst das so:



OH RAH CO RAH OH CO OH RAH DU RAH OH DU

Was heisst in Hivobu: TEH OH CO?



A) B) C) D)

Lösung

Antwort C ist richtig.

Wir wissen, wie die einzelnen Formen heissen:

- Das Rechteck  heisst: RAH
- der Kreis  heisst: OH
- das Dreieck  heisst: TEH

Wir wissen auch, wie es heisst, wenn man zwei Formen hintereinander oder untereinander legt: Zuerst kommt der Name der ersten Form, dann der Name der zweiten Form, und dann heisst es,

- wenn die zweite Form hinter der ersten Form liegt: CO;
- wenn die zweite Form unter der ersten liegt: DU.

Wenn wir also, wie in Antwort C, ein Dreieck (TEH) haben und einen Kreis (OH), der hinter (CO) dem Dreieck liegt, heisst das: TEH OH CO

Aber wie heissen die anderen Antworten in Hivobu?

- Antwort A heisst: OH TEH CO – ein Kreis (OH) und ein Dreieck (TEH), das hinter (CO) dem Kreis liegt.
- Antwort B heisst: TEH RAH CO- ein Dreieck (TEH) und ein Rechteck (RAH), das hinter (CO) dem Dreieck liegt.
- Antwort D heisst: OH TEH DU - ein Kreis (OH) und ein Dreieck (TEH), das unter (DU) dem Kreis liegt.

Dies ist Informatik!

Computer sind schlau? Stimmt, sie können sehr schnell komplizierte Dinge ausrechnen, Informationen im Internet finden und vieles mehr. Stimmt aber auch nicht, denn damit sie das können, muss man ihnen sehr genau sagen, wie sie das tun sollen. Auch den KI-Systemen, mit denen wir scheinbar ganz normal reden können, muss man erst einmal mühsam sagen, wie das geht.

Im Grunde verstehen Computer nämlich nur genaue Anweisungen, die man in Sprachen mit klaren Strukturen formulieren muss. In der Informatik heissen solche Sprachen auch *formale Sprachen*, und Computer verstehen nur diejenigen formalen Sprachen gut, die eine eher einfache Struktur haben.

Das ist so wie in dieser Biberaufgabe: Das, was wir von der Sprache in Hivobu kennengelernt haben, wenn es um den Umgang mit Formen geht, hat eine sehr einfache Struktur. Zuerst werden zwei

Formen genannt, und dann sagt ein Kommando, was mit den beiden Formen passiert. Bringt man diese *Syntax*, also die Struktur der Sprache durcheinander – wenn man zum Beispiel das Kommando in der Mitte sagt statt am Schluss –, weiss niemand in Hivobu, was gemeint ist. In der Informatik kennt man diese Art der Syntax (erst die Dinge sagen, dann das Kommando) als *Postfix-Notation*. Auch ein Computer, der eine Anweisung in Postfix-Notation erwartet, käme bei einem Fehler durcheinander.

Stichwörter und Webseiten

- Postfix Notation: `https://de.wikipedia.org/wiki/Umgekehrte_polnische_Notation`

© Informatik-Biber 2025, SVIA

8. Holz für den Damm

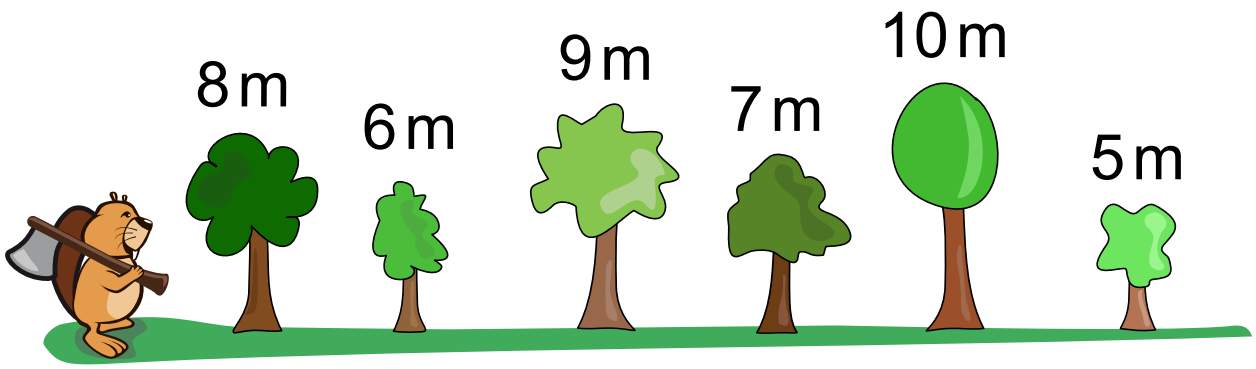
Für ihren nächsten Dammbau müssen die Biber einige Bäume fällen.

6 Bäume kommen in Frage. Die Biber wissen, wieviele Meter Holz jeder Baum hat. Sie wollen insgesamt möglichst viele Meter Holz haben. Den ersten Baum können Sie frei wählen. Immer wenn sie danach einen nächsten Baum fällen wollen, müssen sie 2 Regeln befolgen:

- Regel 1: Der nächste Baum muss weiter rechts stehen als der vorherige.
- Regel 2: Der nächste Baum muss kleiner sein, also weniger Meter Holz haben als der vorherige.

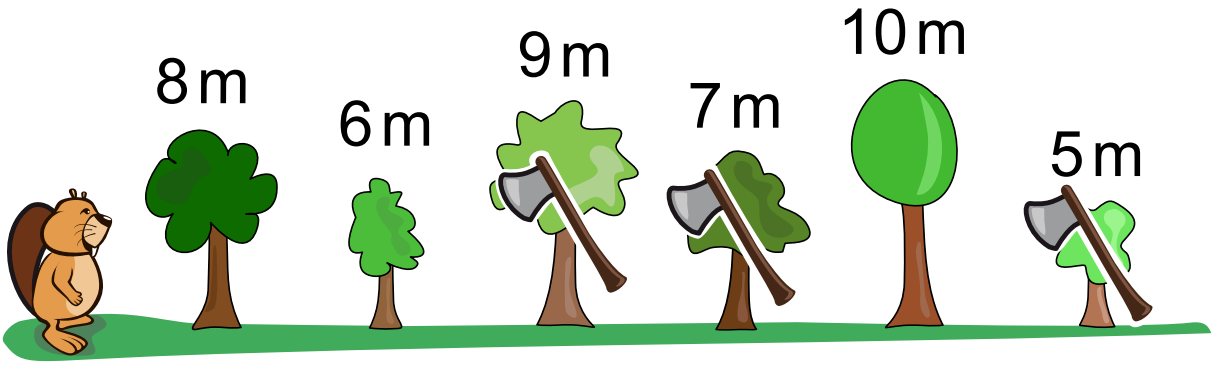
Wenn sie beispielsweise den 6m-Baum fällen, dürfen sie danach nur noch den 5m-Baum fällen. Dann haben sie am Ende insgesamt 11 Meter Holz.

Welche Bäume können die Biber nach ihren Regeln fällen, damit sie am Ende möglichst viele Meter Holz haben?



Lösung

So ist es richtig:



Die Aufgabe besteht darin, von links nach rechts gehend eine nach Holzmetern absteigende Teilfolge der sechs Bäume zu finden, sodass die Summe der Holzmeter in dieser Teilfolge maximal ist. Da wir die maximale Gesamthöhe suchen, müssen wir nur Teilfolgen berücksichtigen, die nicht mehr erweitert werden können, weil das Hinzunehmen egal welchen Baumes zur Teilfolge die Regeln verletzen würde. Wir nennen solche Teilfolgen *vollständig*.

Wenn wir zum Beispiel mit dem 8m-Baum (kurz: 8) beginnen, gibt es zwei vollständige Teilfolgen: 8, 6, 5 und 8, 7, 5. Die Teilfolgen 8, 7 (kann um 5 erweitert werden) und 8, 5 (kann um 6 oder 7 erweitert werden) sind nicht vollständig.

Die Tabelle zeigt alle vollständigen Teilfolgen. Ausserdem gibt sie für jede Teilfolge an, wieviele Meter Holz die Biber am Ende haben, wenn sie die Bäume dieser Teilfolge fällen.

vollständige Teilfolge	**Meter Holz**
8, 6, 5	19
8, 7, 5	20
6, 5	11
9, 7, 5	21
10, 5	15

Die Biber können also die Bäume 9, 7 und 5 nach ihren Regeln fällen, damit sie am Ende möglichst viel Holz haben, nämlich 21 Meter.

Dies ist Informatik!

Die Biber versuchen, im Rahmen ihrer Regeln eine Menge von Bäumen zu finden, so dass sie die grösstmögliche Anzahl an Metern Holz bekommen. In dieser Biberaufgabe sind 21 Meter Holz das *optimale*, das heisst beste Ergebnis, das sie erzielen können. Bei der Beantwortung dieser Biberaufgabe hast du also ein *Optimierungsproblem* gelöst.

Allgemein geht es bei einem Optimierungsproblem darum, einen bestmöglichen Wert zu finden. Das kann ein grösster Wert sein, wie in dieser Biberaufgabe, oder auch ein kleinster Wert, etwa wenn du

 © Informatik-Biber 2025, SVIA

den schnellsten Weg suchst, den du zur Schule nehmen kannst. Es liegt in der Natur der meisten Menschen, für ein Vorhaben nach der schnellsten, kürzesten, günstigsten oder unterhaltsamsten Variante zu suchen: Das ist Optimierung. Es ist aber auch Optimierung, wenn ein Unternehmen versucht, möglichst wenig Gehalt an seine Mitarbeiterinnen und Mitarbeiter auszuzahlen, oder bei der Vermietung von Wohnungen versucht wird, möglichst hohe Mieteinnahmen zu erzielen.

Viele Optimierungsprobleme sind so komplex, dass sie mit Hilfe von Informatiksystemen, also Computerprogrammen bzw. «Apps» gelöst werden. Wenn ein Paketauslieferer nach Möglichkeiten sucht, die Routen der Auslieferungsfahrzeuge so zu planen, dass der Energieverbrauch möglichst gering ist, oder ein Energieversorger seine Anlagen zur Produktion erneuerbarer Energie möglichst wirtschaftlich einsetzen will, müssen so viele Daten und Bedingungen berücksichtigt werden bzw. die Steuerung so flexibel sein, dass mit Hilfe von Computern oft sehr viel bessere Ergebnisse erzielt werden als ohne. Da ist es gut, dass die Informatik viele Methoden zur Lösung von Optimierungsproblemen kennt und sagen kann, welche Methoden für welche Arten von Problemen gut eingesetzt werden können.

Für besonders Interessierte: Das Problem in dieser Aufgabe kennt die Informatik auch als «Maximum Sum Decreasing Subsequence». Schau dir einmal `diese Webseite` an. Dort kannst du nachverfolgen, wie man von einfachen, aber schlechten Ansätzen zur Lösung eines Optimierungsproblems nach und nach immer bessere Ansätze finden und diese in Programme umsetzen kann.

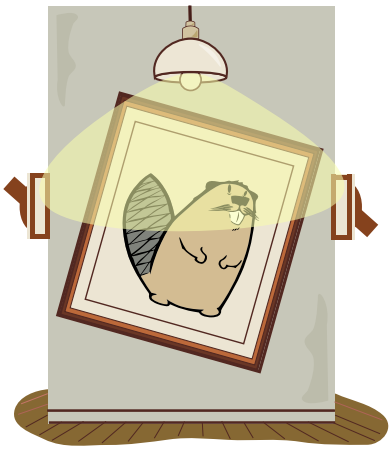
Stichwörter und Webseiten

- *Optimisierungsproblem*: `https://u-helmich.de/inf/TSP/TSPIndex.html`
- *erschöpfende Suche*: `https://de.wikipedia.org/wiki/Brute-Force-Methode`

© Informatik-Biber 2025, SVIA

9. Verrückte Lampe

Viktoria Volt hat eine verrückte Lampe gebaut. Die Lampe hat zwei Lichtschalter: einer links und einer rechts. Jeder Lichtschalter kann entweder **an** (,) oder **aus** (,) sein.



Die verrückte Lampe hat aber noch einen dritten, geheimen Schalter: ein Bild!

Je nachdem, wie das Bild hängt ( ,  oder ), funktionieren die Schalter anders.

Diese Tabelle sagt, wie die Schalter jeweils das Licht **an** oder **aus** machen:

Bild	Schalter		Licht
	genau einer ist **an**: oder		**an**
		sonst	**aus**
	beide sind **aus**:		**aus**
		sonst	**an**
	beide sind **an**		**an**
		sonst	**aus**

Der linke Lichtschalter ist aus , der rechte Lichtschalter ist an . Wie muss das Bild hängen, damit das Licht aus ist?

Lösung

So ist es richtig:

Wenn das Bild so hängt  (also nach links gekippt), ist das Licht aus.

Wir schauen alle drei Möglichkeiten, wie das Bild hängen kann, genauer an:

Weil genau ein Schalter **an** ist (nämlich der rechte), ist das Licht **an**. Diese Antwort ist falsch.

Weil *nicht* beide Lichtschalter **aus** sind, ist das Licht **an**. Auch diese Antwort ist falsch.

Weil *nicht* beide Lichtschalter **an** sind, ist das Licht **aus**. Diese Antwort ist richtig.

Dies ist Informatik!

Viktorias Lichtschalter können jeweils zwei Werte haben: **an** oder **aus**. Und auch das Licht hat nur diese beiden Werte. In jeder Position des Bildes lässt sich der Wert des Lichts aus den Werten der Schalter ausrechnen, und zwar so, wie in der Tabelle angegeben.

Auch die kleinste Speichereinheit in Computern, nämlich das *Bit*, kann nur zwei Werte annehmen. In der Informatik heissen diese Werte häufig auch **an** und **aus**, manchmal aber auch **wahr** und **falsch** oder auch **1** und **0**. So wie man Zahlen mit *Operatoren* ($+$, $-$, $\times$, $:$) kombinieren und daraus neue Werte berechnen kann, kann man Bits mit *logischen Operatoren* kombinieren. In Computern sind diese Operatoren als *logische Gatter* eingebaut. Mit den Gattern kann man Bits auf nur alle erdenklichen Arten kombinieren und verrechnen. Deshalb können Computer Daten auf fast beliebige Weise verarbeiten und verändern.

Das Bild in dieser Biberaufgabe entscheidet, welche logische Operation die beiden Lichtschalter umsetzen. Hängt das Bild gerade, funktionieren die Schalter wie ein XOR («entweder oder»): Wenn entweder der linke oder der rechte Schalter an ist, ist das Licht an. Hängt das Bild nach rechts gekippt, funktionieren sie wie ein OR («oder»): Wenn der linke Schalter oder der rechte (also beide oder einer von beiden) an ist, ist das Licht an. Hängt das Bild nach links gekippt, funktionieren die Schalter wie ein AND («und»): Wenn der linke und der rechte Schalter (beide gleichzeitig) an ist, ist das Licht an.

 © Informatik-Biber 2025, SVIA

Stichwörter und Webseiten

- Logik: `https://de.wikipedia.org/wiki/Logik`
- Logikgatter: `https://de.wikipedia.org/wiki/Logikgatter`
- Boolesche Algebra: `https://de.wikipedia.org/wiki/Boolesche_Algebra`

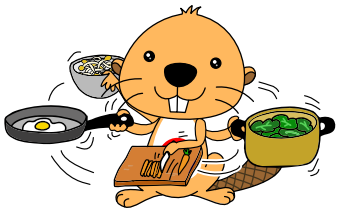
© Informatik-Biber 2025, SVIA

10. Bibimbap

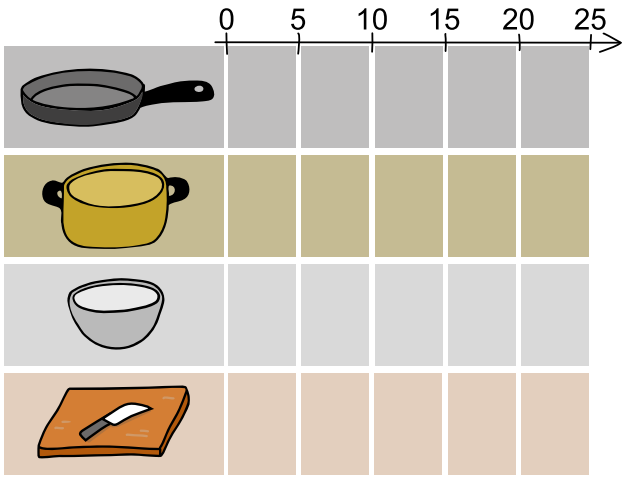
Ein Koch möchte das traditionelle koreanische Gericht Bibimbap (비빔밥) zubereiten. Er benutzt dazu u.a. vier Geräte, nämlich Kochtopf, Bratpfanne, Schneidebrett und Schüssel. Damit bereitet er die vier Zutaten für Bibimbap so vor:

	Spinat:	zuerst kochen (dauert 10 Minuten), danach schneiden (5 Minuten)
	Sprossen:	zuerst wässern (5 Minuten), danach kochen (10 Minuten)
	Rüebli:	zuerst schneiden (5 Minuten), danach braten (10 Minuten)
	Ei:	braten (5 Minuten)

Der Koch kann mit unterschiedlichen Geräten gleichzeitig arbeiten. Aber er kann ein Gerät immer nur für eine Zutat verwenden. Zum Beispiel kann der Koch gleichzeitig Spinat im Topf kochen und ein Ei in der Bratpfanne braten, aber er kann in der Bratpfanne nicht gleichzeitig ein Ei und Rüebli braten.

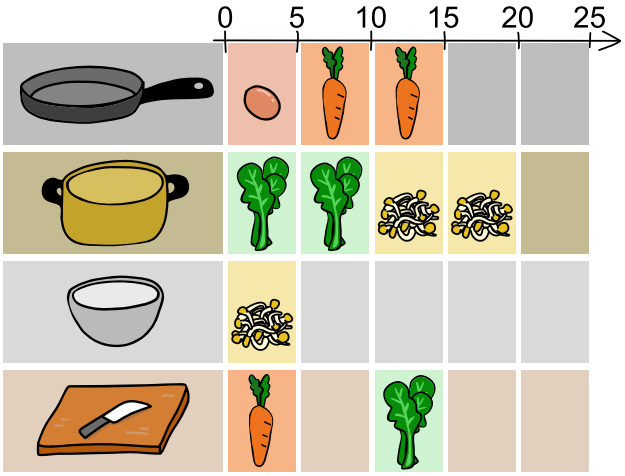


Erstelle einen Plan, mit dem der Koch die Zutaten für Bibimbap in kürzester Zeit vorbereiten kann.



Lösung

So ist es richtig:



Der Koch kann mehrere Geräte gleichzeitig nutzen - aber ein Gerät kann zu jeder Zeit nur für eine Zutat verwendet werden. Es muss also für jedes Gerät überlegt werden, zu welchen Zeiten welche Zutaten damit verarbeitet werden können und der Plan für die Vorbereitung der Zutaten insgesamt am kürzesten ist.

Die Vorbereitung wird mindestens 20 Minuten dauern, denn sowohl Spinat als auch Sprossen müssen nacheinander jeweils 10 Minuten im Topf gekocht werden. Wenn zuerst der Spinat gekocht wird, können zur gleichen Zeit die Sprossen gewässert werden. In der Zeit, in der später dann die Sprossen kochen, kann der Spinat geschnitten werden. In der Bratpfanne wird am besten zuerst das Ei gebraten, denn dann kann der Koch in dieser Zeit schon die Rüebli schneiden und sie nach dem Ei braten – so wie im Bild oben.

Der obige Plan zeigt, dass alle Zutaten nicht nur in mindestens 20 Minuten, sondern auch in höchstens 20 Minuten vorbereitet werden können. Alle Pläne, die insgesamt nicht länger als 20 Minuten benötigen und in denen (a) die Verarbeitungsschritte die richtigen Dauern haben und (b) die Verarbeitungsschritte der einzelnen Zutaten nacheinander und in der richtigen Reihenfolge stattfinden, sind richtige Antworten.

Dies ist Informatik!

In dieser Biberaufgabe geht es darum, einen Ablauf von Aktivitäten zu planen. Dies ist in der Informatik unter dem Begriff *Scheduling* bekannt. Wie bei vielen Scheduling-Problemen sind die zur Verfügung stehenden Ressourcen begrenzt: Topf, Bratpfanne, Brett und Schüssel gibt es jeweils nur einmal. Ausserdem können einige Aktivitäten gleichzeitig passieren (der Koch ist wirklich toll!), während es andere gibt, die nacheinander passieren müssen - entweder, weil sie die gleiche Zutat oder das gleiche Gerät verwenden.

Scheduling ist ein recht altes Problem, und Gedanken dazu wurden sich schon gemacht, bevor es Computer gab. Die auch heute noch in Werkzeugen zur Projektplanung verwendeten Techniken

stammen aber aus der Zeit der ersten Computer, nämlich aus den 1950er Jahren. Dazu gehört unter anderem die Methode des «kritischen Pfades» (engl.: Critical Path Method, kurz: CPM), die in etwa dem entspricht, was wir oben in der Antworterklärung gemacht haben: nämlich eine Folge von voneinander abhängigen Aktivitäten zu bestimmen, die möglichst viel Zeit benötigt und deshalb ohne Pause zwischen den Aktivitäten durchgeführt werden sollte.

Kurz nach Veröffentlichung von CPM war John Fondahl, Professor an der Stanford University, unzufrieden mit den Computer-Implementierungen der Methode. Er hat dann eigene Ansätze präsentiert, CPM ohne Computer umzusetzen. Interessant ist, dass nun genau diese Ansätze bis heute in den meisten Softwarelösungen für Projektplanung umgesetzt werden - was John Fondahl auch vorhergesagt hat. Algorithmen gibt es schon seit weit über tausend Jahren, und auch heute noch lohnt es sich, über Algorithmen nachzudenken, ohne gleich zu überlegen, wie sie von Computern ausgeführt werden können. Am Ende gilt auch dann meist: Je besser der Algorithmus, desto besser ist er für die Umsetzung auf Computern geeignet.

Stichwörter und Webseiten

- Scheduling: `https://www.swisseduc.ch/informatik/theoretische_-informatik/scheduling/algorithmus1.html`
- Critical Path Method: `https://de.wikipedia.org/wiki/Methode_des_kritischen_Pfades`
- Stanford Technical Report John Fondahl (siehe Vorwort): `https://catalog.hathitrust.org/Record/005766951`

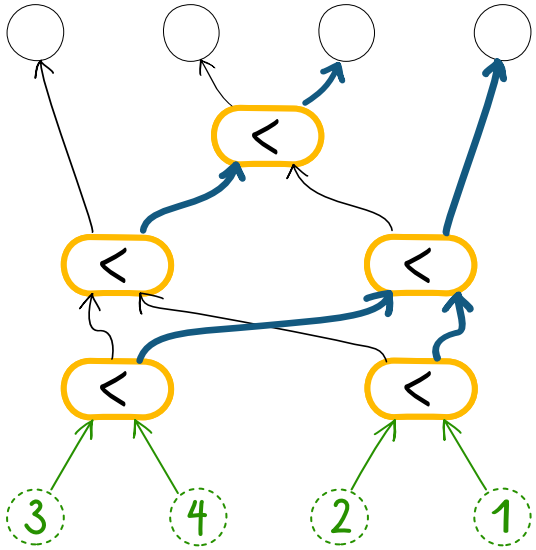
© Informatik-Biber 2025, SVIA

11. Zahlenmaschine

Die Biber haben eine Zahlenmaschine.

Vier Zahlen werden unten in die Eingabefelder eingegeben, zum Beispiel 3, 4, 2 und 1.

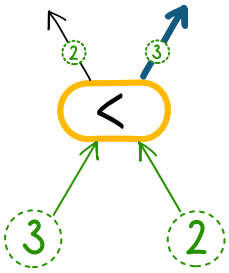
Entlang von Pfeilen und Schaltern wandern die Zahlen durch die Maschine nach oben bis zu den Ausgabefeldern.



Jeder der fünf Schalter vergleicht die beiden eingehenden Zahlen und leitet ...

- ... die kleinere Zahl nach links und
- ... die grössere Zahl nach rechts weiter.

Beispiel:



Welche Aufgabe führt die Maschine aus?

A) Sie sortiert die Zahlen in absteigender Reihenfolge. Ergebnis: 4, 3, 2, 1

B) Sie sortiert die Zahlen in aufsteigender Reihenfolge. Ergebnis: 1, 2, 3, 4

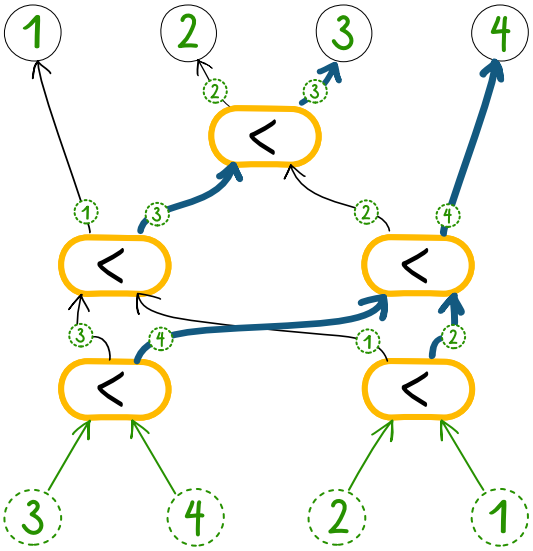
C) Sie gibt die Zahlen in derselben Reihenfolge aus. Ergebnis: 3, 4, 2, 1

D) Sie gibt die Zahlen in umgekehrter Reihenfolge aus. Ergebnis: 1, 2, 4, 3

Lösung

Antwort B ist richtig: Die Zahlenmaschine sortiert die Zahlen in aufsteigender Reihenfolge. Ergebnis: 1, 2, 3, 4

Durch Ausprobieren der Maschine kann man sowohl die richtige Antwort feststellen als auch alle anderen Antworten ausschliessen.



Die Zahlenmaschine macht in einem ersten Schritt zwei Vergleiche von je zwei Zahlen. Danach vergleicht sie zum einen die beiden grösseren und zum anderen die beiden kleineren Zahlen aus den ersten beiden Vergleichen miteinander, um den ingesamt grössten Wert (Maximum) bzw. insgesamt kleinsten Wert (Minimum) der vier Zahlen zu ermitteln. Durch einen Vergleich der übrigen beiden Zahlen ist die Sortierung dann vollständig.

Dies ist Informatik!

In der Informatik ist die Zahlenmaschine aus dieser Biberaufgabe als *Sortiernetzwerk* bekannt. Ein Sortiernetzwerk besteht aus einer Reihe identischer, sehr einfacher Komponenten, den *Komparatoren*. Jeder Komparator empfängt zwei numerische Werte auf zwei Eingangsleitungen und vergleicht sie. Dann leitet er die Werte auf zwei Ausgangsleitungen weiter: den kleineren Wert auf der linken Leitung und den grösseren Wert auf der rechten Leitung weiter. (Häufig werden Sortiernetzwerke auch quer dargestellt, mit den Eingängen links und Ausgängen rechts und den Komparatoren als Brücken zwischen zwei Leitungen; dann wird die kleinere Zahl in der Regel nach oben und die grössere Zahl nach unten geleitet.)

Durch die Kombination einer ausreichenden Anzahl von Komparatoren kann jede Zahlenfolge sortiert werden. Die Zahlenmascheine in dieser Biberaufgabe zeigt, dass vier Zahlen mit fünf Komparatoren sortiert werden können. Für fünf Zahlen sind mindestens neun und für sechs Zahlen mindestens zwölf Komparatoren erforderlich.

 © Informatik-Biber 2025, SVIA

Sortiernetzwerke sind in der Informatik von praktischer Bedeutung, weil Komparatoren als sehr einfache und daher kostengünstige elektronische Bauteile und damit Sortiernetzwerke auch insgesamt gut in Hardware realisiert werden können. Ausserdem können die Komparatoren zum Teil gleichzeitig arbeiten, was die Sortierung beschleunigt: Im Allgemeinen ist die Anzahl der nicht-parallelen Schritte eines Sortiernetzwerks kleiner als die Anzahl der zu sortierenden Zahlen. Ein Nachteil von Sortiernetzwerken ist, dass sie jeweils nur für Eingaben von fester Länge ausgelegt sind.

Stichwörter und Webseiten

- *Sortiernetzwerk*: `https://www.csunplugged.org/de/topics/sorting-networks/`
- *Komparatoren*: `https://de.wikipedia.org/wiki/Komparator_(Analogtechnik)`
- *Parallelrechner*: `https://de.wikipedia.org/wiki/Parallelrechner`

© Informatik-Biber 2025, SVIA

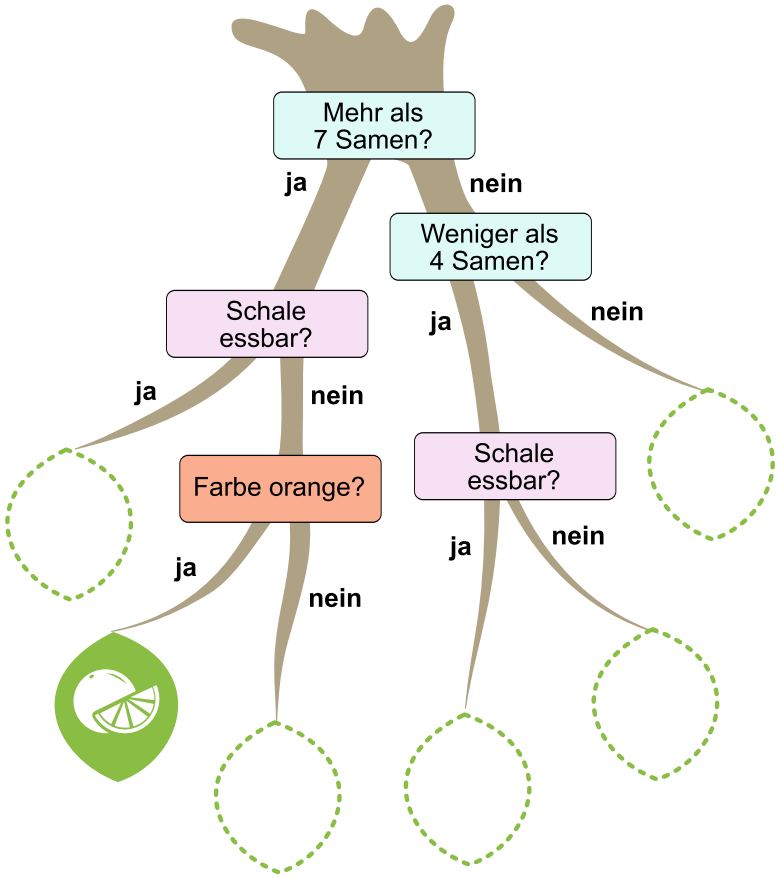
12. Blätter im Wind

In einer Schulklasse werden Früchte analysiert. Für jede Frucht werden drei Eigenschaften betrachtet und aus den Werten die Fruchtsorte bestimmt. Die Eigenschaften sind: Äussere Farbe, Anzahl der Samen und Essbarkeit der Schale. Für zehn Früchte hat die Klasse deren Werte und die daraus bestimmten Fruchtsorten in einer Tabelle notiert:

Farbe	**Anzahl Samen**	**Schale essbar?**	**Fruchtsorte**
grün	391	nein	Wassermelone
gelb	5	ja	Apfel
orange	9	nein	Orange
gelb	0	nein	Banane
rot	5	ja	Apfel
grün	0	ja	Weintraube
rot	206	ja	Erdbeere
grün	6	ja	Apfel
orange	10	nein	Orange
rot	173	ja	Erdbeere

Die Fruchtsorten werden mit dem Entscheidungsbaum bestimmt. Ein Entscheidungsbaum sieht aus wie ein Baum, der auf dem Kopf steht: Oben ist die Wurzel und unten sind die Blätter. Ausserdem sind die Wurzel und die Astgabeln mit Fragen beschriftet, die man mit ja oder nein beantworten kann.

Zur Bestimmung der Fruchtsorte werden die Fragen im Baum anhand der Werte beantwortet. Das geht so: Beginne oben an der Wurzel. Beantworte die Frage dort und gehe auf den passenden Ast mit der richtigen Antwort (ja oder nein). Beantworte die nächste Frage und gehe auf den nächsten passenden Ast. Mache so weiter, bis du ein Blatt erreicht hast. Das Blatt zeigt die Fruchtsorte.



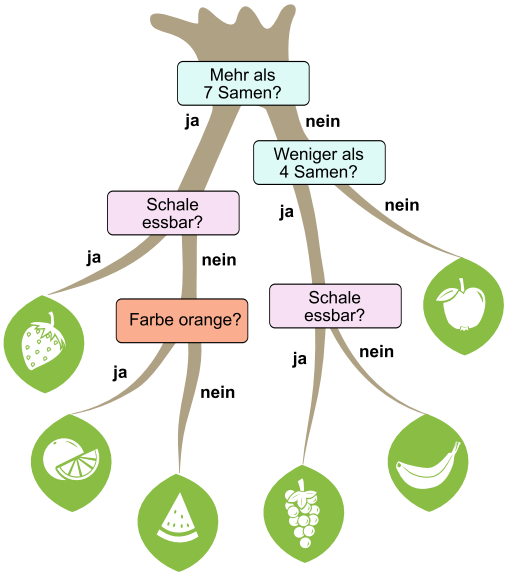
Nach den zehn Früchten ist der Entscheidungsbaum leider kaputt gegangen: Fast alle Blätter sind abgefallen.

An welchen Stellen waren die Blätter?

 © Informatik-Biber 2025, SVIA

Lösung

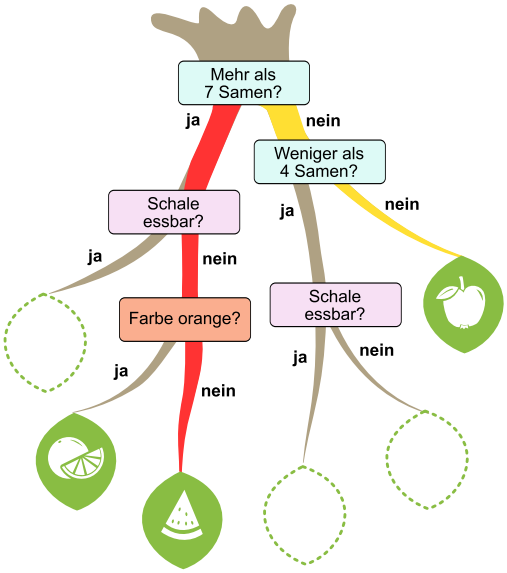
So ist es richtig:



Die Früchte auf den Blättern sind (von links nach rechts): Erdbeere, Orange, Wassermelone, Trauben, Banane, Apfel.

Wie findet man die richtigen Stellen für die Blätter?

Die Fruchtsorten in der Tabelle wurden mit dem Entscheidungsbaum bestimmt. Gehen wir also die Zeilen der Tabelle durch und schauen wir, zu welcher Blatt-Stelle der Weg durch den Entscheidungsbaum anhand der Fragen führt. An dieser Stelle muss dann das Blatt mit der Fruchtsorte sein, die in der letzten Spalte der Tabelle angegeben ist. Das Bild zeigt für die ersten beiden Zeilen der Tabelle die Wege durch den Entscheidungsbaum:



Zeile 1 (roter Weg): Die Frucht hat 391 Samen (Mehr als 7 Samen? - Ja), ihre Schale ist nicht essbar (Schale essbar? - Nein), und die Frucht ist grün (Farbe orange? - Nein). Dieser Weg führt zur dritten Blatt-Stelle von links. Dorthin muss das Blatt mit der Fruchtsorte, die in dieser Zeile der Tabelle eingetragen ist: Wassermelone.

Zeile 2 (gelber Weg): Die Frucht hat 5 Samen (Mehr als 7 Samen? - Nein. Weniger als 4 Samen? - Nein). Der Weg führt zur Blatt-Stelle ganz rechts, dorthin muss also das Blatt mit dem Apfel.

In ähnlicher Weise führen

- Zeile 3 zur zweiten Blatt-Stelle von links, dort ist bereits die Orange;
- Zeile 4 zur zweiten Blatt-Stelle von rechts, dorthin muss die Banane;
- Zeile 5 zum Apfel;
- Zeile 6 zur dritten Blatt-Stelle von rechts, dorthin muss die Weintraube;
- Zeile 7 zur Blatt-Stelle ganz links, dorthin muss die Erdbeere.

Dies ist Informatik!

Ein *Entscheidungsbaum* ist ein einfacher, aber cleverer Weg, um Dinge in Sorten oder Gruppen einzuordnen (bzw. in Fachsprache: zu *klassifizieren*) und anhand dieser Einordnung Entscheidungen zu treffen. Entscheidungsbäume werden in vielen Informatiksystemen verwendet: zum Beispiel in medizinischen Diagnoseprogrammen, bei Online-Assistenten, die dir helfen, Produkte zu finden, und auch in Videospielen, wenn die Software entscheiden muss, wie eine Spielfigur reagieren soll.

Der Entscheidungsbaum in dieser Biberaufgabe wurde vorgegeben, zum Beispiel durch die Biologie-Lehrkraft. Sie hat den Baum aufgebaut und sich überlegt, welche Fragen wichtig sind, in welcher Reihenfolge sie gestellt werden müssen und wie der Baum sich verzweigen soll, damit die Früchte korrekt klassifiziert werden können.

In einem in den letzten Jahren immer wichtiger werdenden Bereich der Informatik, nämlich *Künstliche Intelligenz* (KI), geht es meist auch darum, beobachtete Daten zu klassifizieren und dann Entscheidungen zu treffen. Die nötigen Klassifikationswerkzeuge – wie etwa ein Entscheidungsbaum – sollen aber automatisch aus (den gleichen oder anderen) beobachteten Daten abgeleitet werden. Informatikmethoden zum automatischen Aufbau von Klassifikations- und Entscheidungsstrukturen aus Daten fallen unter Begriffe wie *Machine Learning* oder *Data Science*. Auch Entscheidungsbäume können «gelernt», also automatisch aufgebaut werden, und zwar durch Algorithmen wie CART oder C4.5. Diese Methoden finden aus vielen Beispielen selbst heraus, welche Fragen wann gestellt werden müssen, um gute Entscheidungen zu treffen. Automatisch erzeugte Entscheidungsbäume kommen z. B. bei der Erkennung von Spam-E-Mails, medizinischen Diagnosen oder der Bestimmung von Pflanzenarten zum Einsatz.

Allerdings verwenden die meisten bekannten KI-Systeme **keine** Entscheidungsbäume. Stattdessen verwenden viele Systeme grosse *Modelle* aus *neuronalen Netzen*. Diese Strukturen enthalten keine nachvollziehbaren Entscheidungsfragen, sondern komplexe statistische Muster, die für Menschen meist schwer verständlich sind.

 © Informatik-Biber 2025, SVIA

Stichwörter und Webseiten

- Entscheidungsbaum, `https://de.wikipedia.org/wiki/Entscheidungsbaum`
- Klassifikation, `https://de.wikipedia.org/wiki/Klassifikation`
- Künstliche Intelligenz, AI Unplugged `https://www.aiunplugged.org/german.pdf`

© Informatik-Biber 2025, SVIA

13. Lichterstern

In ihrem Elektronikkasten hat Sophie drei Sorten Lichter: runde rote, quadratische blaue und fünfeckige gelbe. Sie hat einige Lichter zu einem Lichterstern verkabelt. Die Pfeile zeigen, wie die Kabel liegen.

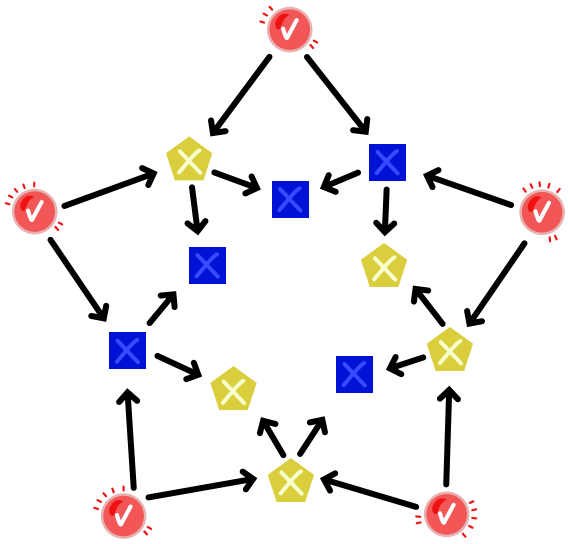
Die blauen und gelben Lichter werden über die Kabel gesteuert, in Pfeilrichtung. Jedes blaue oder gelbe Licht hat also genau zwei «Steuer-Lichter».

So funktionieren die Lichter:

- Die roten Lichter kann Sophie selbst an- und ausschalten.
- Ein blaues Licht ist an, wenn beide Steuer-Lichter an sind; sonst ist es aus.
- Ein gelbes Licht ist an, wenn genau eines der beiden Steuer-Lichter an ist; sonst ist es aus.

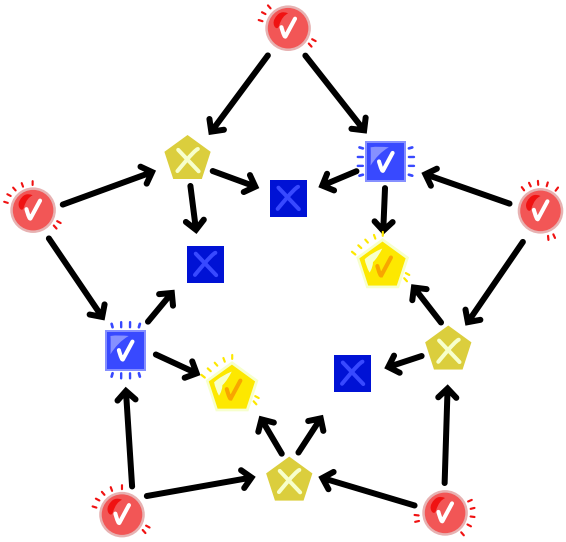
Sophie schaltet alle roten Lichter an.

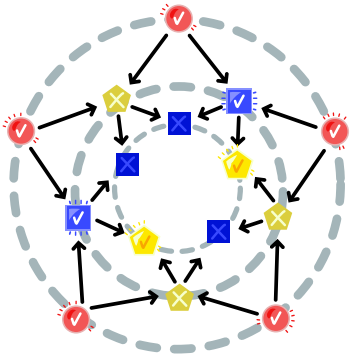
Welche anderen Lichter sind dann auch an?



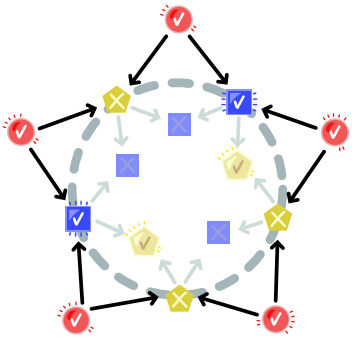
Lösung

So ist es richtig:

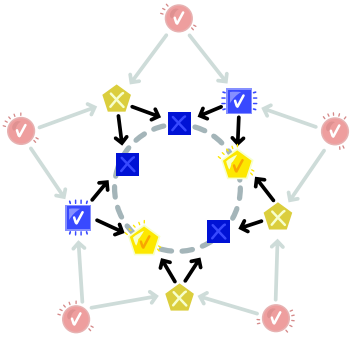




Die Lichter bilden drei Ringe: einen äusseren mit allen roten Lichtern sowie einen mittleren und einen inneren mit blauen und gelben Lichtern.



Alle roten Lichter und damit alle Steuer-Lichter für die mittleren Lichter sind an. Damit sind im mittleren Ring genau die blauen Lichter an (weil beide Steuer-Lichter an sind), und die gelben Lichter sind aus (weil beide Steuer-Lichter an sind, also *nicht* genau ein Steuer-Licht an ist).



Im inneren Ring ist es genau anders herum: Jedes Licht im inneren Ring hat je ein gelbes und blaues Steuer-Licht aus dem mittleren Ring. Wir wissen schon, dass von diesen beiden Lichtern immer genau eines an ist. Deswegen sind im inneren Ring genau die gelben Lichter an, und die blauen Lichter sind aus.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Die Lichter in Sophies Stern können nur an- oder ausgeschaltet sein. Wir können dazu auch sagen, dass jedes Licht zwei Werte haben kann, nämlich **an** und **aus** – so wie eine Ampel im Strassenverkehr drei Werte haben kann, nämlich «rot», «gelb» und «grün», oder eine Uhr mit Digitalanzeige 1.440 Werte haben kann, nämlich alle Uhrzeiten von 00:00 bis 23:59, oder die Anzeige des Gesamtbetrags an einer Supermarktkasse viele Geldbeträge als Werte haben kann.

Mit Zahlen kann man rechnen, nämlich mit *Operatoren* für Zahlen wie plus und minus. Auch für die zwei Werte der Lichter gibt es Operatoren, und dabei ist es egal, ob die Werte **an** und **aus**, **wahr** und **falsch** oder **1** und **0** heissen. Diese Operatoren rechnen aus zwei *Eingabewerten* ein Ergebnis aus. In dieser Biberaufgabe werden zwei solche Operatoren verwendet:

- Der Wert der blauen Lichter wird mit dem Operator AND (englisch für «und») ausgerechnet: Das Ergebnis von UND ist 1, wenn beide Eingabewerte 1 sind, sonst ist es 0.
- Der Wert der gelben Lichter wird mit dem Operator XOR (für «exklusives oder») ausgerechnet: Das Ergebnis von XOR ist 1, wenn die Eingabewerte verschieden sind, sonst ist es 0.

XOR ist übrigens nicht exklusiv, weil es besonders teuer ist. «Exklusiv» bedeutet, es wird ausgeschlossen, dass das Ergebnis auch dann 1 ist, wenn beide Eingabewerte 1 sind. Es gibt auch einen Operator, der wie XOR funktioniert, aber ohne diesen Ausschluss; er heisst OR («oder»).

In der Informatik sind diese und einige weitere Operatoren auch als *logische Operatoren* bekannt. Logische Operatoren kann man gut als elektronische Schaltungen bauen. Die wiederum sind Grundbausteine von Computerprozessoren, denn auch die kleinste Einheit im Computer, das Bit, hat zwei Werte. Geschickt zusammengebaut kann man mit ihnen komplizierte Berechnungen machen, Programmabläufe steuern oder sogar jedes beliebige Programm schreiben.

Stichwörter und Webseiten

- Logik: `https://de.wikipedia.org/wiki/Logik`
- Boolesche Logik: `https://de.wikipedia.org/wiki/Boolesche_Algebra`
- Logischer Operator: `https://de.wikipedia.org/wiki/Logischer_Operator`
- AND: `https://de.wikipedia.org/wiki/Konjunktion_(Logik)`
- XOR: `https://de.wikipedia.org/wiki/Kontravalenz`
- OR: `https://de.wikipedia.org/wiki/Disjunktion`
- Flipflop: `https://de.wikipedia.org/wiki/Flipflop`

© Informatik-Biber 2025, SVIA

14. Dein Bauwerk

Du hast diese Bauklötze:

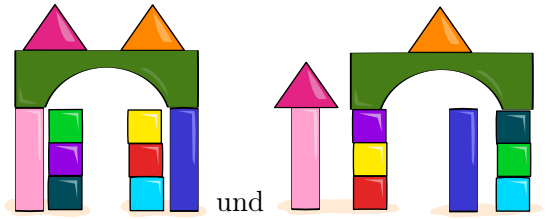
6 Würfel 2 Quader 1 Brücke 2 Pyramiden

Dein Freund gibt dir diese Anleitung, um aus den Klötzen Bauwerke zu bauen:

1. Nimm 3 Würfel.
2. Staple die Würfel übereinander, um einen Turm zu bauen.
3. Baue einen weiteren Turm mit den 3 restlichen Würfeln.
4. Stelle die Quader neben die Türme.
5. Lege die Brücke auf dein Bauwerk.
6. Nimm die beiden Pyramiden und lege sie auf dein Bauwerk.

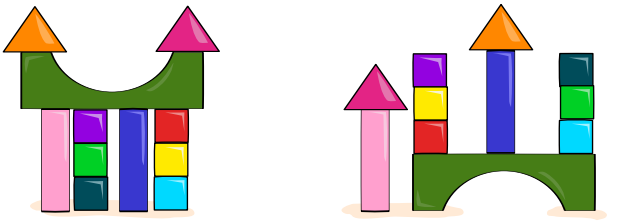
Beim Bauen musst du dich an die Reihenfolge der 6 Anweisungen halten. Mit der Bauanleitung kannst du trotzdem viele verschiedene Bauwerke bauen.

Beispiele:

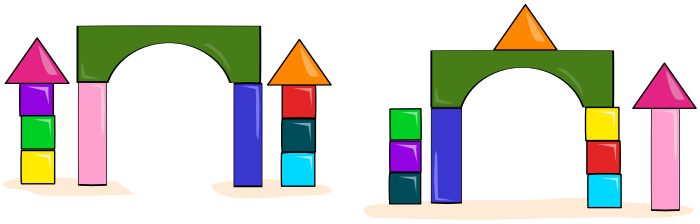


und

*Hier sind 4 weitere Bauwerke. Eines davon kannst du **NICHT** mit der Bauanleitung bauen. Welches?*



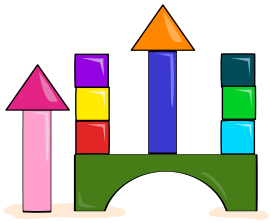
A) B)



C) D)

Lösung

Antwort B ist richtig.



Wir wissen schon, dass man mit der Bauanleitung verschiedene Bauwerke bauen kann. Die Anleitung ist also nicht *eindeutig*. Aber eines ist klar: Die Reihenfolge der einzelnen Anweisungen der Bauanleitung muss eingehalten werden. Schauen wir uns genauer an, was passieren kann, wenn man die Bauanleitung Schritt für Schritt befolgt. Wir beschreiben in einer Tabelle, wie das Bauwerk nach Ausführung der einzelnen Anweisungen aussieht, und prüfen, welches Bauwerk zu der Beschreibung passt.

Nach Schritt	**Beschreibung**	**A**	**B**	**C**	**D**
1 und 2	Ein erster Turm aus drei Würfeln steht auf dem Boden.	✓	✗	✓	✓
3	Zwei Türme stehen auf dem Boden.	✓	✗	✓	✓
4	Zwei Türme und zwei Quader stehen auf dem Boden.	✓	✗	✓	✓
5	Wie oben, und die Brücke liegt auf anderen, vorher gebauten Teilen des Bauwerks, also auf Türmen oder Quadern.	✓	✗	✓	✓
6	Wie oben, und die Pyramiden liegen auf anderen, vorher gebauten Teilen des Bauwerks (Türmen, Quadern oder Brücke).	✓	✗	✓	✓

Das Bauwerk von Antwort B passt also nicht zur Bauanleitung. Insbesondere wurde Anweisung 5 nicht richtig umgesetzt. Anweisung 5 gibt vor, dass die Brücke *auf* das bisherige Bauwerk gelegt werden muss. Im Bauwerk von Antwort B ist die Brücke aber nicht *auf* mindestens einem anderen Teil des Bauwerks, sondern nur *unter* Teilen, nämlich den zwei Türmen und einem Quader. Das Bauwerk von Antwort B kann nur entstehen, wenn die Brücke vor den beiden Türmen und einem Quader gebaut wird.

Die Tabelle zeigt, dass die Bauwerke der Antworten A, C und D mit der Anleitung gebaut werden können.

Dies ist Informatik!

Diese Aufgabe verdeutlicht, dass klare, eindeutige Anleitungen wichtig sind. Beispiele dafür sind Bauanleitungen oder auch Kochrezepte. Uneindeutig formulierte Anleitungen können zu unterschiedlichen und unvorhersehbaren Ergebnissen führen. In der Bauanleitung dieser Biberaufgabe ist beispielsweise nicht klar vorgeschrieben, wohin genau die einzelnen Bauklötze bzw. die aus den Würfeln gebauten Türme gestellt oder gelegt werden sollen.

 © Informatik-Biber 2025, SVIA

Computer brauchen klare und eindeutige Anleitungen, wenn sie so funktionieren sollen, wie Menschen sich das wünschen. Die Informatik nennt solche klaren Anleitungen *Algorithmen*: Schritt-für-Schritt-Anleitungen, um eine Aufgabe zu erledigen oder ein Problem zu lösen. Auch die einzelnen Anweisungen eines Algorithmus müssen eindeutig sein, denn anders als Menschen können Computer nicht interpretieren oder raten. Sie brauchen eindeutige Anweisungen, um Aufgaben vorhersehbar auszuführen. Uneindeutige Anweisungen in Computerprogrammen können dazu führen, dass Programme nicht wie gewollt ausgeführt werden, Software nicht gut funktioniert oder unerwartete Ergebnisse liefert.

Bevor ein Programm geschrieben wird, muss der Entwickler überlegen, welche *Einschränkungen* (engl. *constraints*) definiert werden müssen, um ein vorhersehbares Ergebnis zu erzielen. In dieser Biberaufgabe können solche Einschränkungen zum Beispiel Folgendes festlegen: die genaue vertikale und horizontale Anordnung der Klötze, die Platzierung von Teilen des Bauwerks und wie Bauklötze mit den bereits vorhandenen verbunden werden müssen.

Stichwörter und Webseiten

- *Algorithmen*: `https://de.wikipedia.org/wiki/Algorithmus`
- *Einschränkungen bzw. Constraintprogrammierung*: `https://de.wikipedia.org/wiki/Constraintprogrammierung`

 © Informatik-Biber 2025, SVIA

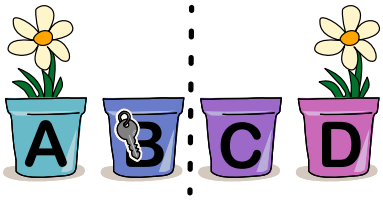
15. Blumentöpfe

Biber Florian dekoriert den Eingang seines Baus mit Blumentöpfen. In manchen Töpfen ist **genau eine Blume** gepflanzt, die anderen sind **leer**.

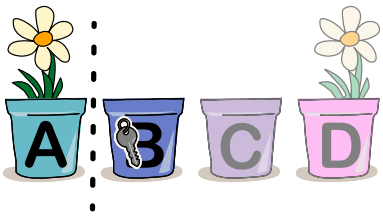
In einem Topf ist ein Schlüssel versteckt. Florian erklärt seine Methode, wie man den Schlüssel finden kann.

«Zuerst betrachtet man alle Töpfe an und zählt, wie viele Blumen insgesamt in den Töpfen gepflanzt sind. Wenn die Anzahl an Blumen gerade ist, ist der Schlüssel in der linken Hälfte der Töpfe, sonst ist er in der rechten Hälfte. Jetzt betrachtet man nur die Hälfte, in der der Schlüssel ist, und wiederholt das Verfahren, bis nur noch ein Topf übrig ist. Dort ist der Schlüssel versteckt.»

Florian zeigt ein Beispiel, wie man den Schlüssel in 4 Töpfen A, B, C, D finden kann.



Betrachte die Töpfe A, B, C und D. Es gibt insgesamt 2 Blumen, also eine **gerade** Anzahl. Das heisst, der Schlüssel ist in der **linken** Hälfte, also in Topf A oder B.



Betrachte die Töpfe A und B. Es gibt insgesamt 1 Blume, also eine **ungerade** Anzahl. Das heisst, der Schlüssel ist in der **rechten** Hälfte, also in Topf B.

Florian hat acht Blumentöpfe und versteckt den Schlüssel in Topf C. In welche Töpfe sollte er je eine Blume pflanzen, damit man den Schlüssel mit seiner Methode finden kann?

Es gibt mehrere richtige Antworten. Auch 0 ist eine gerade Zahl.

Lösung

Eine richtige Antwort ist:

Eine andere richtige Antwort ist:

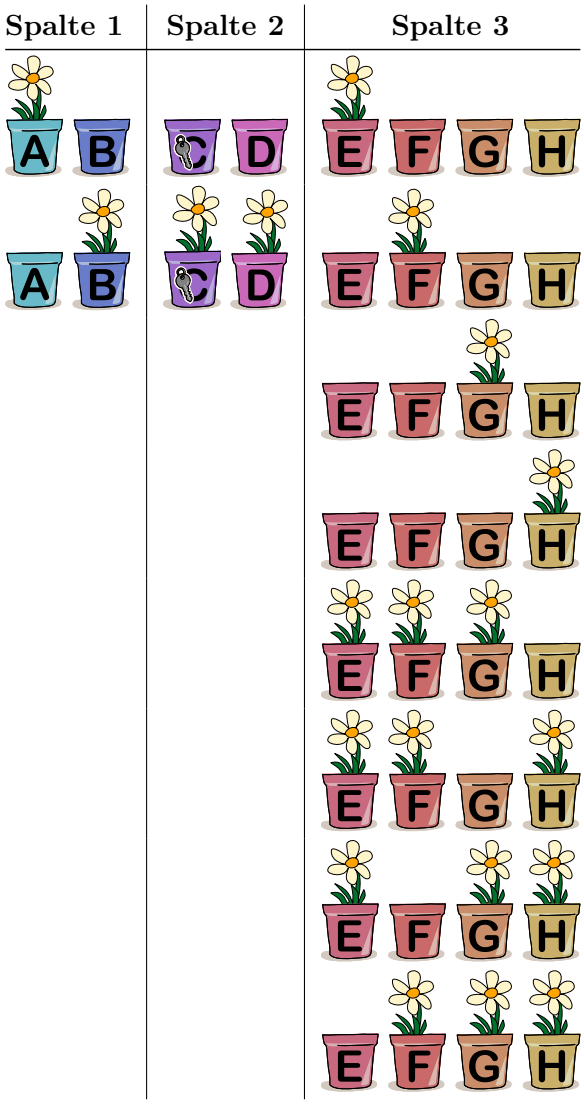
Es gibt noch mehr richtige Antworten, insgesamt 32 Stück. Man kann diese so bestimmen:

Es ist am einfachsten, die Antwort «von unten nach oben» zu konstruieren, sprich, bei kleinen Teilen der Antwort anzufangen und sich von da aus zur ganzen Antwort zu arbeiten.

- Wenn der Schlüssel in Topf C ist, werden beim Finden des Schlüssels zuletzt die Töpfe C und D betrachtet. Damit sich dabei für die linke Hälfte C entschieden wird, muss es in C und D eine gerade Anzahl Blumen geben. Also ist entweder in beiden Töpfen oder in keinem dieser Töpfe eine Blume.
- Im Schritt davor werden die Töpfe A bis D betrachtet. Weil sich dabei für die rechte Hälfte (C und D) entschieden werden soll, braucht esin A bis D eine ungerade Anzahl Blumen. Wie vorher beschrieben, hat die Hälfte C und D auf jeden Fall eine gerade Anzahl Blumen. Daher muss genau einer der beiden Töpfe A und B eine Blume enthalten.
- Im ersten Schlüssel-Finde-Schritt werden alle Töpfe betrachtet. Weil sich für die linke Hälfte (A bis D) entschieden werden soll, braucht man insgesamt eine gerade Anzahl Blumen. Da in der bereits betrachteten linken Hälfte eine ungerade Anzahl Blumen ist, braucht man auch in der rechten Hälfte (E bis H) eine ungerade Anzahl Blumen. Man kann sich also für E bis H zwischen entweder 1 oder 3 Blumen entscheiden und diese beliebig auf die Töpfe E, F, G und H verteilen.

Die folgenden Tabelle fasst alle richtigen Antworten zusammen: Indem man aus jeder Spalte eine Option wählt, kann man eine richtige Antwort zusammenstellen. Auf diese Weise kann man alle $2 \times 2 \times 8 = 32$ richtigen Antworten erhalten.

 © Informatik-Biber 2025, SVIA



Dies ist Informatik!

Florian *kodiert* die Position des Schlüssels mit Hilfe einer Folge von Blumen in seinen Töpfen. Seine Freunde können diese Darstellung dekodieren, weil sie Florians Kodierungsmethode kennen. Damit Computer Daten verarbeiten können, werden die Daten nicht in einer für den Menschen natürlichen und gut verständlichen Form, sondern in einer für Computer lesbaren Form kodiert abgespeichert. Die Informatik kennt sehr viele Methoden zur Kodierung. Manche Kodierungen sollen dafür sorgen, dass Speicherplatz gespart wird; man spricht dann von *Komprimierung*. Wenn eine Kodierung zur Dekodierung über die Kenntnis der Kodierungsmethode hinaus die Kenntnis eines sogenannten «Schlüssels» voraussetzt, spricht man auch von *Verschlüsselung*. In beiden Fällen ist wichtig, dass die Dekodierung genau die ursprünglichen Daten wiederherstellt, also eindeutig ist. Jede Kombination aus Töpfen mit und ohne Blumen legt eindeutig fest, in welchem Topf der Schlüssel versteckt ist, sodass der «Blumen-Code» in dieser Biberaufgabe diese Anforderung erfüllt.

Florians Methode, den Schlüssel in den Blumentöpfen zu finden, funktioniert ähnlich zur *binären Suche*, da in jedem Schritt der Suchraum halbiert wird. Damit kommt man recht schnell zum Ziel. Bei doppelt so vielen Blumentöpfen bräuchte man nur einen Schritt mehr.

Die richtige Antwort lässt sich bei dieser Biberaufgabe am besten mit einem sogenannten *Bottom-up Ansatz* finden. Dabei betrachtet man die kleinsten Teile der Antwort zuerst, da die grösseren Teile davon abhängen. Statt alle möglichen Bepflanzungen der Töpfe durchzuprobieren, was hier schon $2^8 = 256$ viele wären, kommt man so durch kluges Kombinieren schneller zu einer richtigen Antwort.

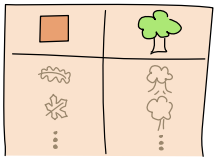
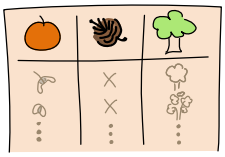
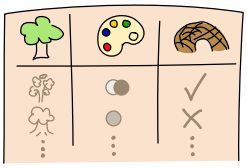
Stichwörter und Webseiten

- Kodierung: `https://de.wikipedia.org/wiki/Code`
- Binäre Suche: `https://de.wikipedia.org/wiki/Binäre_Suche`

 © Informatik-Biber 2025, SVIA

16. Biberholz

Reto und seine Freunde gehen gern wandern. Während ihrer Wanderungen sammeln sie Informationen über die Bäume, die sie sehen, und notieren diese in lange Tabellen.

Tabelle	Beschreibung
	**Severin** sammelt Information über Blattformen und die zugehörigen Baumarten.
	**Quirina** sammelt Informationen über Baumfrüchte, ob diese von Nadelbäumen stammen und über die zugehörigen Baumarten.
	**Ladina** sammelt Informationen über Baumarten, über deren Holzfarben, und darüber, ob sie Biberholz für Biberburgen liefern.

Reto hat im Wald ein Blatt gefunden und kennt dessen Form. Nun möchte er erfahren, ob die zugehörige Baumart Biberholz für Biberburgen liefert.

Welchen seiner Freunde muss Reto fragen, und in welcher Reihenfolge, um das zu erfahren?

A) Nur Ladina.

B) Erst Severin, dann Quirina.

C) Erst Severin, dann Ladina.

D) Erst Quirina, dann Severin, dann Ladina.

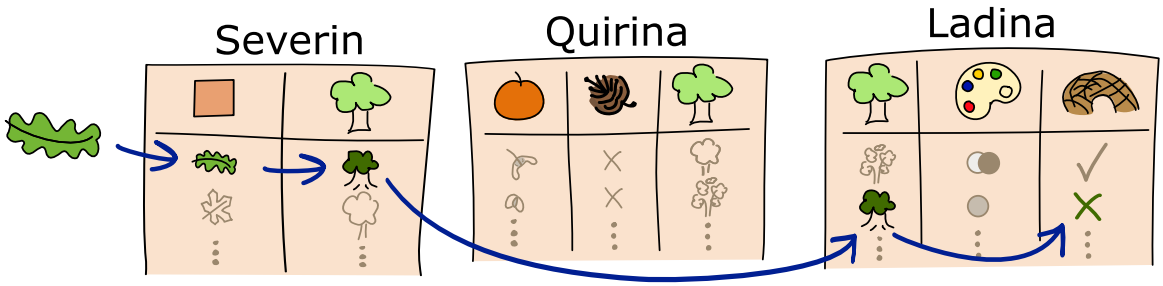
Lösung

Antwort C ist richtig: Erst Severin, dann Ladina.

Die Information darüber, ob eine Baumart Biberholz liefert, findet sich nur in Ladinas Tabelle. Wenn Reto jedoch nur Information über das Blatt hat, kann er keine Zeile aus Ladinas Tabelle auswählen. Er benötigt Information über die Baumart oder die Farbe des Holzes. Es genügt also nicht, nur Ladina zu fragen; Antwort A ist also falsch.

Quirinas Tabelle enthält weder Information über Blätter, noch über Biberholz. Ihre Tabelle nützt Reto nichts, also sind die Antworten B und D falsch.

Aber Severins Tabelle enthält Information über Blätter. Da Reto die Form des Blattes kennt, kann er zuerst die passende Zeile in Severins Tabelle auswählen und die fehlende Information über die Baumart erhalten. Anschliessend kann er damit die passende Zeile in Ladinas Tabelle auswählen, um die gesuchte Information über das Holz zu erhalten. Wenn Reto zum Beispiel anhand der Blattform und Severins Tabelle herausfindet, dass es sich um ein Eichenblatt handelt, kann er die passende Zeile in Ladinas Tabelle auswählen und herausfinden, ob Eichen Biberholz liefern.



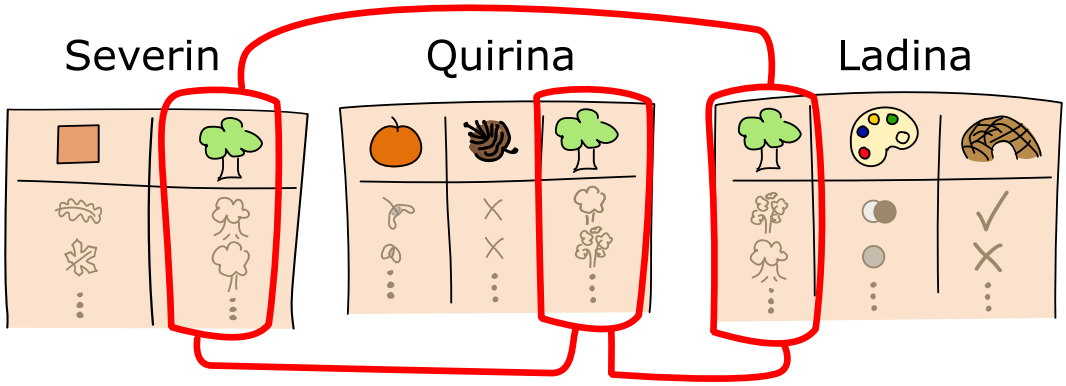
Dies ist Informatik!

Diese Biberaufgabe veranschaulicht grundlegende Konzepte *relationaler Datenbanken*. Diese werden in Informatiksystemen extrem häufig eingesetzt, nämlich zur Verwaltung grosser (und auch kleiner) Datenmengen. Relationale Datenbanken bestehen aus Tabellen mit Daten - so wie die von Retos Freunden erstellten Tabellen. Tabellen heissen auch *Relationen*, und in einer Tabelle ist jede Spalte ein *Attribut* und jede Zeile ein Datensatz bzw. ein Element der durch die Tabelle gegebenen Relation. Die Verbindung zwischen Tabellen über ein gemeinsames Attribut – hier die «Baumart» – entspricht dem Konzept der *Fremdschlüssel* in relationalen Datenbanken, die Beziehungen zwischen verschiedenen Tabellen herstellen.

Retos Frage nach Information über gutes Bauholz für eine Biberburg anhand der Blattform würde in einem System mit Datenbanken als *Abfrage* bezeichnet. Diese Abfrage erfordert die Verknüpfung mehrerer Tabellen, um die gewünschte Information zu erhalten. Die Verknüpfungsoperation kombiniert Zeilen aus verschiedenen Tabellen anhand des gemeinsamen Schlüssels kurzzeitig zu einer gemeinsamen, grösseren Tabelle. So können Daten, die über mehrere Tabellen verteilt sind (in diesem Fall Baumarten, Blattform und Biberholz), für die Beantwortung von Abfragen zusammengeführt werden.

 © Informatik-Biber 2025, SVIA

Die Daten in den Tabellen der Freunde können insbesondere über die Baumart miteinander verbunden werden:



Relationale Datenbanken sind so wichtig, dass es in der Informatik für Abfragen und andere Operationen auf Datenbanken eine eigene Sprache gibt, nämlich *SQL (Structured Query Language)*.

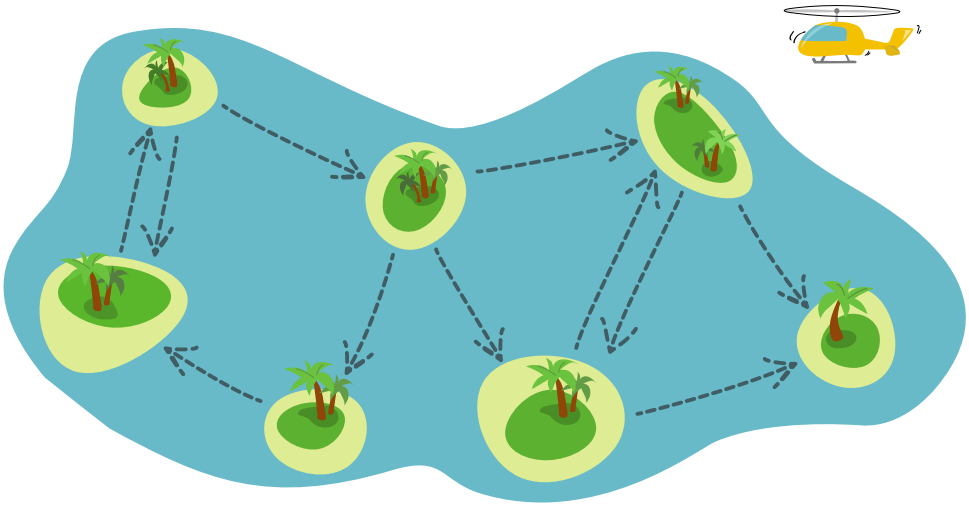
Stichwörter und Webseiten

- Relationale Datenbanken: `https://de.wikipedia.org/wiki/Relationale_Datenbank`
- Fremdschlüssel: `https://de.wikipedia.org/wiki/Schlüssel_(Datenbank)`
- SQL: `https://de.wikipedia.org/wiki/SQL`
- Attribut: `https://www.datenbanken-verstehen.de/lexikon/attribut/`
- Tupel: `https://de.wikipedia.org/wiki/Tupel`

© Informatik-Biber 2025, SVIA

17. Bebrasien

Vor der Küste von Bebrasien liegen sieben Inseln. Zwischen den Inseln kann man mit Fähren  fahren, aber nur in Richtung der Pfeile.



Ein Forschungsteam möchte die Tierwelt auf allen sieben Inseln erkunden. Ein einzelner Ausflug des Teams zu den Inseln läuft so ab:

Das Team ...

1. ... fliegt mit einem Hubschrauber  zu irgendeiner Insel,
2. benutzt die Fähren, um weitere Inseln zu besuchen, und
3. kehrt zum Schluss zur Insel mit dem Hubschrauber zurück, um zurückzufliegen.

Das Team stellt fest: Ein einziger Ausflug reicht nicht, um alle Inseln zu besuchen.

Wieviele Ausflüge muss das Team dazu mindestens machen?

A) 2 Ausflüge
B) 3 Ausflüge
C) 4 Ausflüge
D) 5 Ausflüge
E) 6 Ausflüge
F) 7 Ausflüge

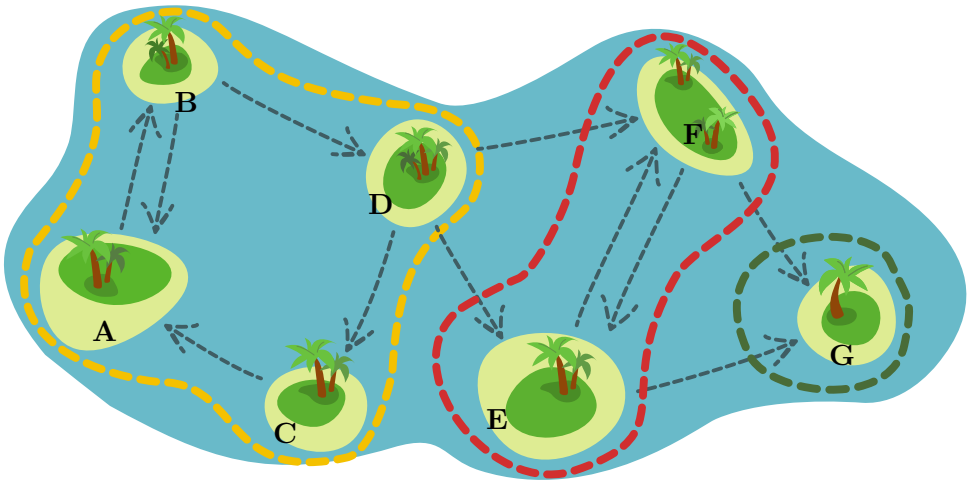
Lösung

3 ist die richtige Antwort.

Um auf die Mindestzahl an Ausflügen zu kommen, muss das Team bei jedem Ausflug möglichst viele Inseln besuchen. Das Forschungsteam muss aber auch am Ende eines Ausflugs zur «Startinsel» mit dem Hubschrauber zurück. Bei einem Ausflug kann das Team ausser der Startinsel also nur solche Inseln besuchen, von denen aus es wieder zur Startinsel zurückkehren kann.

Wir teilen deshalb die Inseln in Gruppen ein. In jeder Gruppe sind so viele Inseln wie möglich, so dass gilt: Von jeder Insel einer Gruppe aus kann man mit den Fähren jede andere Insel der Gruppe besuchen. Dann kann das Team bei einem Ausflug irgendeine Insel der Gruppe als Startinsel anfliegen und alle Inseln der Gruppe besuchen. Aber auch nicht mehr, denn: Wenn das Team eine Insel-Gruppe verlässt, kommt es nicht mehr zu der Gruppe und damit auch nicht zum Hubschrauber zurück. Das Team muss also so viele Ausflüge machen, wie es Gruppen gibt.

Eine Gruppe kann man so finden: Man wählt eine beliebige Insel, die noch keiner Gruppe zugeordnet ist, als erste Insel einer neuen Gruppe. Dann ordnet man der neuen Gruppe alle Inseln zu, zu denen man von der ersten Insel aus fahren und von denen man auch zu ihr zurückfahren kann. Das Bild zeigt, dass die sieben Inseln aus drei solchen Gruppen bestehen: {A,B,C,D}, {E,F} und {G}. Das Forschungsteam muss also drei Ausflüge machen, um alle Inseln zu besuchen.



Aber wieso genügt nicht ein Ausflug? Man kann doch von einigen Inseln (zum Beispiel: C) aus alle anderen Inseln besuchen! Aber dabei verlässt man die Gruppe der ersten Insel und kommt nicht zu ihr und damit nicht zum Hubschrauber zurück.

Dies ist Informatik!

Die Inseln sind durch die Fähren teilweise miteinander verbunden. Inseln und Fährverbindungen kann man als *Graph* modellieren. Ein Graph ist eine Struktur, die Beziehungen zwischen Objekten beschreibt – wie die Fährverbindungen zwischen den Inseln in dieser Biberaufgabe. Dabei sind die Inseln die *Knoten* und die Fährverbindungen die *gerichteten Kanten* des Graphen.

 © Informatik-Biber 2025, SVIA

Entsprechend lässt sich auch das Konzept der Gruppen auf Graphen übertragen: Eine Gruppe ist eine Menge von Knoten, so dass jedes Knoten-Paar aus dieser Menge direkt oder indirekt durch die Kanten des Graphen verbunden ist. Bei Graphen heissen solche Gruppen *starke Zusammenhangskomponenten* (SZK). Bei vielen Anwendungen von Informatiksystemen spielen Graphen und ihre starken Zusammenhangskomponenten eine wichtige Rolle. Einige Beispiele:

- Im World-Wide-Web sind SZKs Gruppen von Websites, die alle direkt oder indirekt miteinander verlinkt sind.
- In sozialen Netzwerken sind SZKs «Blasen» von Nutzern, die sich in einem Netzwerk alle direkt oder indirekt gegenseitig folgen.
- In Verkehrsnetzen sind SZKs Regionen, in denen zwischen allen Haltestellen Fahrten möglich sind.

Die Informatik kennt Algorithmen, mit denen man die SZKs eines Graphen effizient ermitteln kann. Am bekanntesten und besten ist der Algorithmus von Tarjan. Robert Tarjan ist ein US-amerikanischer Informatiker, der alleine oder zusammen mit anderen viele Algorithmen erfunden hat, von denen einige nach ihm benannt wurden. Er war erst 38 Jahre alt, als er mit dem wichtigsten Preis der Informatik ausgezeichnet wurde, dem «Turing Award».

Stichwörter und Webseiten

- *Gerichteter Graph*: `https://de.wikipedia.org/wiki/Gerichteter_Graph`
- *stark zusammenhängende Komponente*: `https://de.wikipedia.org/wiki/Zusammenhang_(Graphentheorie)`

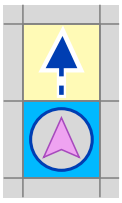
© Informatik-Biber 2025, SVIA

18. Lefty II

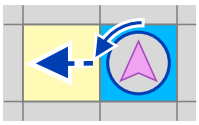
Roboter *Lefty* bewegt sich über ein Raster mit quadratischen Feldern. Zwischen Feldern kann es rote Mauern geben. Lefty soll das grüne Ziel erreichen.

Lefty kann sich auf genau zwei Arten bewegen:

Ein Feld vorwärts fahren

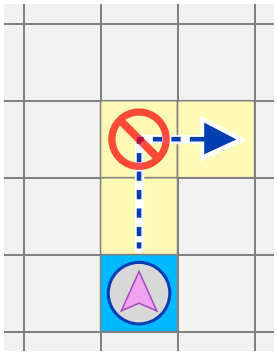


Nach links drehen und dann sofort ein Feld vorwärts fahren

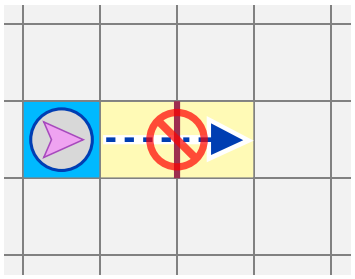


Lefty kann aber nicht alles. Zum Beispiel kann er

... **nicht** einfach rechts abbiegen und

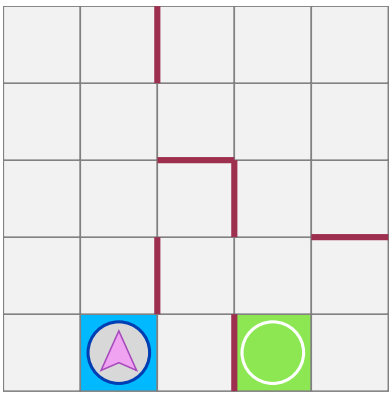


... **nicht** durch Mauern fahren.



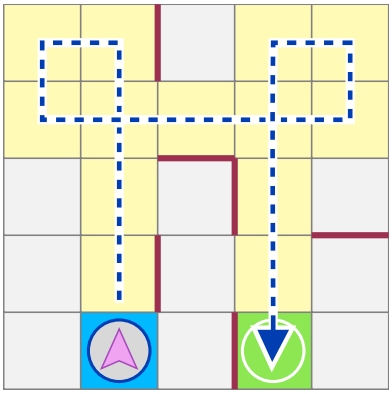
*Über welche Felder **muss** Lefty fahren, um das Ziel zu erreichen?*

*Wähle **so wenige Felder wie möglich** aus.*



Lösung

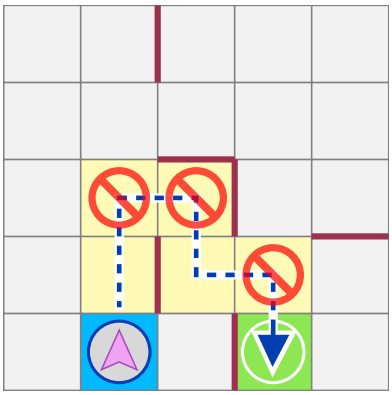
So ist es richtig:



Wenn Lefty über diese Felder fährt, erreicht er das Ziel. Dabei bewegt er sich nur auf die beiden Arten, die er kann.

Es gibt keinen anderen Weg mit weniger oder gleich vielen Feldern, auf dem Lefty das Ziel erreicht.

Den direkten Weg kann Lefty nicht nehmen, weil er dafür mehrmals rechts abbiegen müsste.



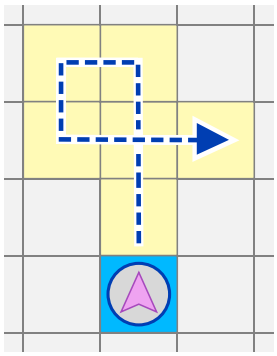
Dies ist Informatik!

Leftys Funktionsweise ist stark eingeschränkt. Wenn er doch nur andere Bewegungen machen könnte! Wenn er sich nach rechts drehen und vielleicht sogar über Mauern klettern könnte, wäre sein Leben im Raster viel einfacher. Lefty würde sich viel selbstbewusster fühlen, wenn er einen *komplexeren Befehlssatz* hätte. (Bei einem Roboter werden die grundlegenden Dinge, die er tun kann, durch entsprechende Befehle in der Software des Roboters gesteuert).

Aber ist das wirklich notwendig? Lefty könnte sich zum Beispiel nach rechts drehen, indem er sich dreimal hintereinander nach links dreht. Wir müssen nur die Regel abschaffen, dass sich Lefty nach der Linksdrehung sofort nach vorne bewegen muss. Dann könnte er sich in alle Richtungen drehen und fahren. Und anstatt über eine Mauer zu klettern, könnte er um sie herumfahren, wenn dafür genug Platz ist. Das heisst, ein *reduzierter Befehlssatz* kann für einen Roboter durchaus genügen. Um komplexeres Verhalten zu implementieren, das seltener vorkommt, kann man *Unterprogramme*

 © Informatik-Biber 2025, SVIA

entwerfen, die mehrere einfache Befehle zu einem komplexeren kombinieren. Zum Beispiel könnte ein Unterprogramm beschreiben (und in der Antwort oben gleich zweimal verwendet werden), wie Lefty es unter bestimmten Bedingungen schaffen kann, seine Richtung nach rechts zu ändern:



In der Informatik sind genau diese beiden Ansätze, den Befehlssatz eines *Prozessors* zu gestalten, am weitesten verbreitet: Manche Prozessoren sind ein CISC (Complex Instruction Set Computer / deutsch: Computer mit komplexem Befehlssatz), andere sind ein RISC (Reduced Instruction Set Computer / Computer mit reduziertem Befehlssatz) - wie jener von Lefty in dieser Biberaufgabe. Ein CISC hat in der Regel viele verschiedene Befehle, die sehr mächtig sein können (wie das Klettern über eine Mauer), aber dafür seltener verwendet werden. Ein RISC hingegen hat nur wirklich nötige Befehle mit eher einfacher Wirkung, die dann häufig verwendet werden.

Beide Arten von *Architekturen* haben ihre Vor- und Nachteile. Die Prozessoren bekannter Marken sind entweder CISC- oder RISC-Prozessoren, wobei RISC-Prozessoren in letzter Zeit etwas beliebter geworden sind.

Stichwörter und Webseiten

- Prozessor: `https://de.wikipedia.org/wiki/Prozessor`
- Befehlssatz: `https://de.wikipedia.org/wiki/Befehlssatz`
- CISC: `https://de.wikipedia.org/wiki/Complex_Instruction_Set_Computer`
- RISC: `https://de.wikipedia.org/wiki/Reduced_Instruction_Set_Computer`

© Informatik-Biber 2025, SVIA

19. Momos Spiel

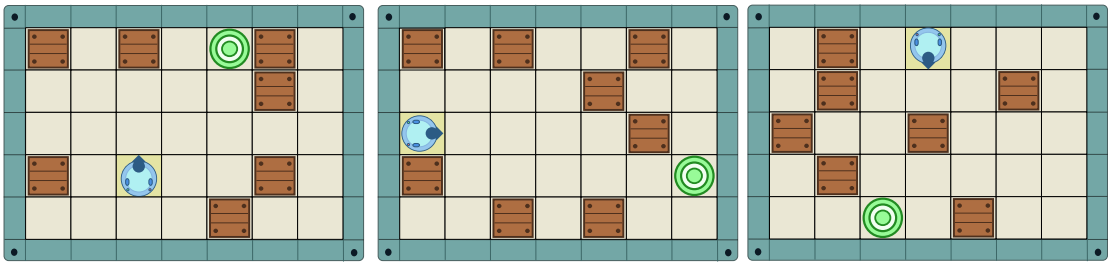
In einem von Momos Computerspielen kann ein Roboter über Felder fahren. Wenn er vor ein Hindernis oder die Wand fährt, geht es nicht mehr weiter. Dann muss der Roboter die Richtung wechseln.

In jedem Level des Spiels können die Hindernisse und auch das Ziel woanders sein. Ein Level ist geschafft, wenn der Roboter das Ziel erreicht.

Momo kann den Roboter mit Programmen steuern. Für seine Programme kann er diese vier Befehle benutzen:

Fahre ein Feld vorwärts.
Fahre vorwärts, bis es nicht mehr weitergeht.
Drehe dich um 90 Grad gegen den Uhrzeigersinn.
Drehe dich um 90 Grad im Uhrzeigersinn.

Momo kennt die nächsten 3 Level. Er möchte ein Programm mit möglichst wenigen Befehlen haben, das alle 3 Level schafft.



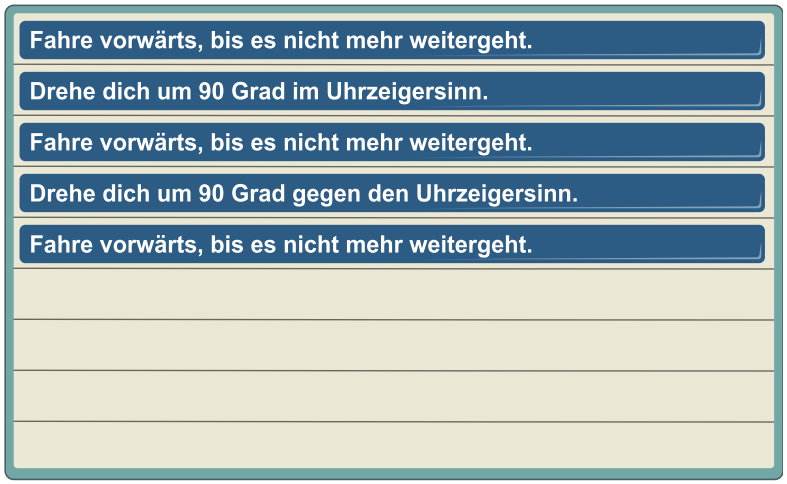
Erstelle so ein Programm für Momo!

Fahre ein Feld vorwärts.
Fahre vorwärts, bis es nicht mehr weitergeht.
Drehe dich um 90 Grad gegen den Uhrzeigersinn.
Drehe dich um 90 Grad im Uhrzeigersinn.

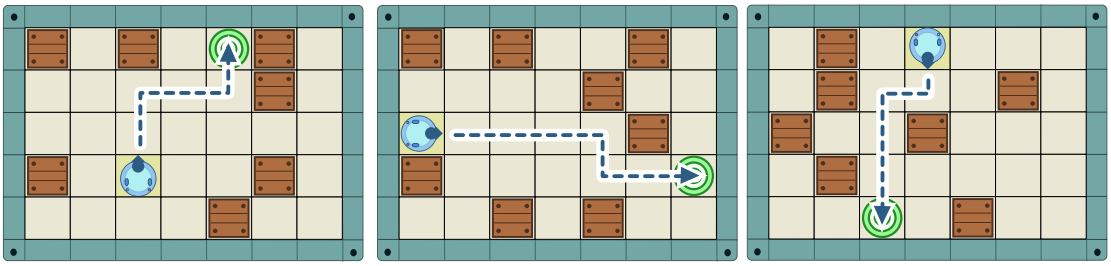
Lösung

So ist es richtig:

Die einzelnen Level kann man jeweils mit mehreren verschiedenen Programmen schaffen. Aber nur ein Programm schafft alle drei Level:



Die Pfeile zeigen, wie das Programm den Roboter in den drei Leveln steuert:



In Level 1 (links) befindet sich das Ziel rechts oberhalb vom Roboter. Deshalb muss der Befehl «Fahre vorwärts, bis es nicht mehr weitergeht» mindestens zweimal verwendet werden: einmal, um nach rechts zu gelangen, und einmal, um nach oben zu gelangen. Der Roboter blickt anfangs nach oben und soll am Ende auch wieder nach oben blicken. Er muss sich also zuerst im Uhrzeigersinn drehen, um nach rechts zu gehen, und später gegen den Uhrzeigersinn, um wieder nach oben zu blicken. Ein Programm, das Level 1 schafft, hat also mindestens vier Befehle.

In Level 2 (mitte) muss der Roboter auf dem Weg nach rechts ein Hindernis umgehen und benötigt dafür den Befehl «Fahre vorwärts, bis es nicht mehr weitergeht» einmal mehr. Ein Programm, das Level 2 schafft, hat also mindestens fünf Befehle.

Das Programm mit fünf Befehlen schafft auch die Level 1 und 3 (rechts). Ein kürzeres Programm für alle drei Level gibt es nicht.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Die vier Roboter-Befehle in dieser Biberaufgabe bilden eine kleine Programmiersprache. Durch Aneinanderreihung der Befehle kann der Roboter gesteuert werden. In der Informatik nennt man solche Aneinanderreihungen *Befehlsfolgen* oder *Sequenzen*. Die Sequenz ist die einfachste und grundlegendste Kontrollstruktur in der strukturierten Programmierung.

Der Befehl «Fahre ein Feld vorwärts.» wird für die richtige Antwort nicht benötigt. Das Programm enthält stattdessen mehrmals den Befehl «Fahre vorwärts, bis es nicht mehr weitergeht». Mit diesem Befehl wird der Vorwärtsschritt so lange wiederholt, bis der Roboter auf ein Hindernis oder den Spielfeldrand stösst. In der strukturierten Programmierung sind *Wiederholungen* eine weitere grundlegende Kontrollstruktur. Es gibt Wiederholungen mit einer festen Anzahl («Fahre fünf Felder vorwärts.»), aber wichtiger sind Wiederholungen mit Abbruchbedingungen, wie in «Fahre vorwärts, bis es nicht mehr weitergeht». Weil der Roboter mit diesem Befehl unterschiedlich lange gerade Strecken fahren kann, kann das Programm alle drei Level schaffen.

Du kannst dir sicher Level ausdenken, die das Programm nicht schaffen kann, z. B. wenn der Roboter mehrmals die Richtung wechseln muss, um das Ziel zu erreichen. Informatikerinnen und Informatiker versuchen, die *Algorithmen* für ihre Programme so zu entwerfen, dass sie nicht auf bestimmte Fälle zugeschnitten sind, sondern *allgemein* funktionieren, also für alle denkbaren Fälle. Dazu schauen sie auch, ob es für verwandte Probleme schon Lösungsalgorithmen gibt. Die Level in Momos Spiel sind ein wenig wie Irrgärten, und für Irrgärten gibt es solche allgemeinen Lösungsalgorithmen. Die bestimmen zwar in manchen Fällen Wege zum Ziel, die komplizierter sind als nötig, sind aber für alle Fälle erfolgreich.

Stichwörter und Webseiten

- Strukturierte Programmierung:
  `https://de.wikipedia.org/wiki/Strukturierte_Programmierung`
- Lösungsalgorithmen für Irrgärten:
  `https://de.wikipedia.org/wiki/Lösungsalgorithmen_für_Irrgärten`

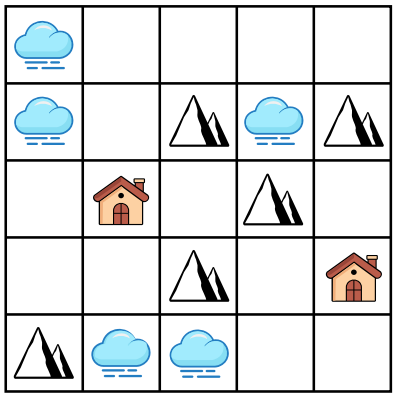
 © Informatik-Biber 2025, SVIA

20. Ein Tag im Nebel

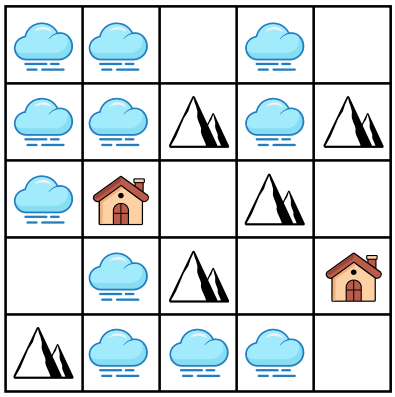
Im Land der Berge ist heute Nebel, und der breitet sich mit jeder Stunde weiter aus.

Bei Sonnenaufgang bedeckt der Nebel nur einige Regionen. In jeweils einer Stunde breitet sich der Nebel von jeder bisherigen Nebelregion in alle ihr benachbarten Regionen aus; nach rechts, links, oben oder unten. Dadurch werden auch Häuser vom Nebel bedeckt. Nur die Bergregionen kann der Nebel nicht bedecken.

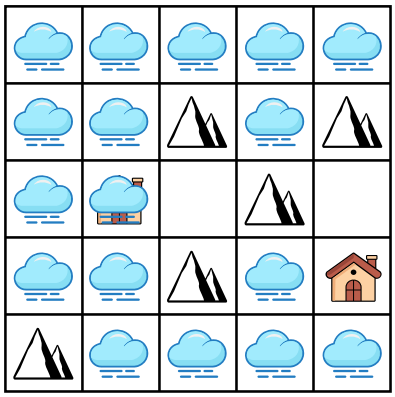
Ein Beispiel:



Sonnenaufgang

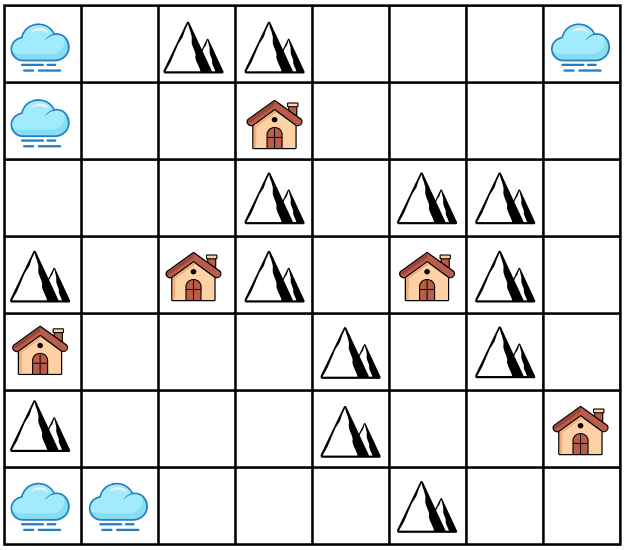


Nach 1 Stunde



Nach 2 Stunden

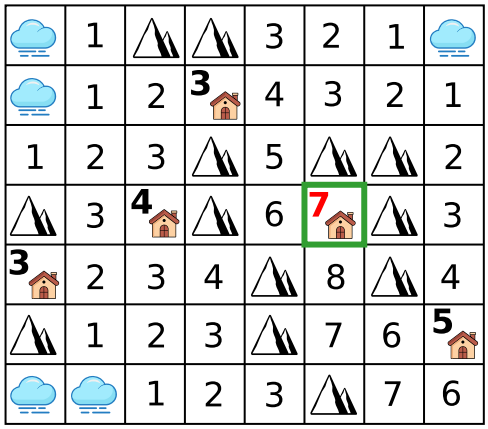
*Welches Haus im Land wird als **letztes** vom Nebel bedeckt?*



Lösung

Der Nebel verbreitet sich wie eine Flut über das Land: zuerst auf die benachbarten Regionen der Nebelregionen, dann auf die Nachbarn der Nachbarn (die noch nicht benebelt sind), und so weiter. Eigentlich müsste man nur zusehen, bis alle Häuser ausser einem unter dem Nebel verschwunden sind. Dann ist die richtige Antwort gefunden.

Im Bild unten ist die Region des Hauses grün markiert, welches als letztes vom Nebel bedeckt wird. Es zeigt auch für jede Region, wie viele Stunden es gedauert hat, bis sie vom Nebel bedeckt war. Man sieht: Das markierte Haus hat von allen Häusern die höchste Zahl, und es gibt nur ein Haus mit dieser Zahl, so dass die Antwort eindeutig ist: 7.



Man kann sich auch etwas anderes überlegen: Dasjenige Haus wird als letztes vom Nebel bedeckt, das zu Beginn am weitesten von einer Nebelregion entfernt ist. Also kann man für alle Häuser diese Entfernung messen und so die richtige Antwort bestimmen. Dabei ist zu beachten: Die Entfernung von einem Haus zum Nebel misst sich entlang der Ausbreitung des Nebels, über die nicht-diagonal benachbarten Regionen und um die Bergregionen herum. Dieses Vorgehen würde aber deutlich länger dauern als das obige, denn man hätte für n Häuser und m ursprüngliche Nebelregionen $n \times m$ Entfernungen zu berechnen.

Interessant ist, dass ein schneller, menschlicher Blick sich hier täuschen lassen kann: Auf Anhieb könnte man das Haus rechts unten als am weitesten entfernt von allen ursprünglichen Nebelregionen vermuten. Weil auf dem Weg zu diesem Haus dem Nebel keine Berge im Weg sind, ist es aber schneller erreicht als das Haus der richtigen Antwort.

Dies ist Informatik!

Der Nebel in dieser Biberaufgabe flutet nach und nach die Regionen des Landes, die von mindestens einer der ursprünglichen Nebelregionen entlang des Ausbreitungswegs des Nebels erreichbar sind. Ein Gebiet aus allen Regionen, die von einer einzigen Nebelregion aus erreicht werden kann, können wir auch als *zusammenhängendes* Gebiet bezeichnen. Im Nebel-Land dieser Aufgabe bilden alle Regionen ein einziges zusammenhängendes Gebiet. Ein einziger weiterer Berg in der zweiten Zeile von oben,

viertes Feld von rechts würde dafür sorgen, dass die Regionen des Landes in zwei zusammenhängende Nebel-Gebiete aufgeteilt wären.

Zusammenhängende Gebiete sind auch für die Informatik interessant, und zwar in unterschiedlichen Bereichen. Ein einfarbiger Bereich in einem (Computer-)Bild ist ein zusammenhängendes Gebiet gleichfarbiger Pixel; man kann es mit einem *Floodfill-Algorithmus* bestimmen, der so ähnlich funktioniert wie die Nebelausbreitung in dieser Biberaufgabe. Eine Gruppe von Jugendlichen, in denen jeder mindestens mit einem anderen Jugendlichen der Gruppe befreundet ist, ist ebenfalls ein zusammenhängendes Gebiet, wenn man die Freundschaftsbeziehung als Nachbarschaft betrachtet. Auf ganz ähnliche Weise kann man die «Blasen» in einem sozialen Netzwerk als zusammenhängende Gebiete betrachten. Auch für solche Netzwerke kennt die Informatik Methoden, zusammenhängende Gebiete zu bestimmen, etwa die Breitensuche oder die Tiefensuche. Mit Methoden zur Bestimmung zusammenhängender Gebiete können zum Beispiel Bereiche in Bildern umgefärbt oder Gruppierungen in sozialen Netzwerken ermittelt werden.

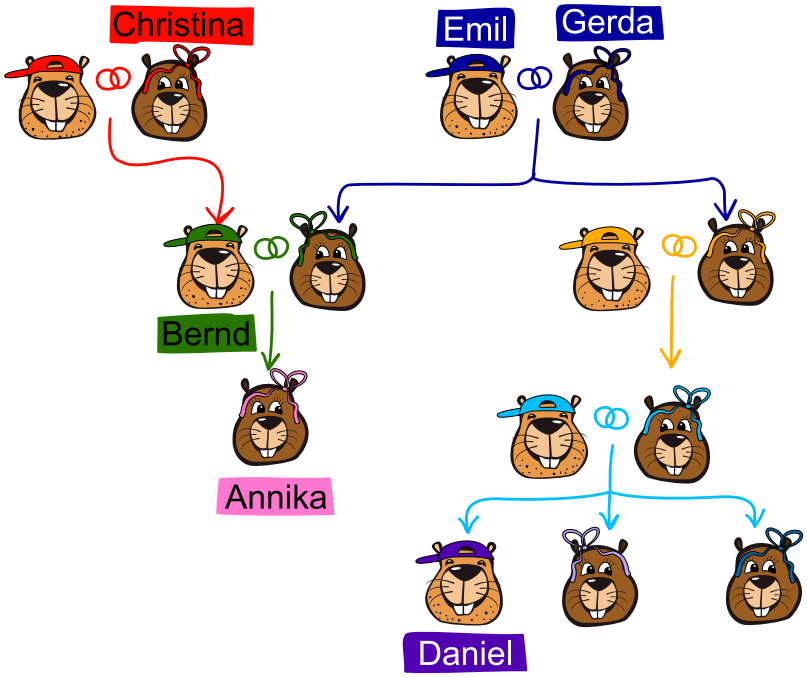
Stichwörter und Webseiten

- Flooding-Algorithmus: `https://de.wikipedia.org/wiki/Flooding-Algorithmus`
- Flooding-Algorithmus (Englisher Wikipedia Eintrag mit mehr Details): `https://en.wikipedia.org/wiki/Flooding_algorithm`
- Floodfill Algorithmus: `https://de.wikipedia.org/wiki/Floodfill`

© Informatik-Biber 2025, SVIA

21. Stammbaum

Die Biber Annika und Daniel wollen wissen, wie sie miteinander verwandt sind. Annika hat einen Stammbaum ihrer gemeinsamen Familie. Darin tragen die männlichen Biber eine Kappe und die weiblichen eine Schleife.



Annika verwendet eine Kurzschreibweise:

- `Vater(X)` steht für «Vater von Biber X»
- `Mutter(X)` steht für «Mutter von Biber X»

Annikas Vater ist Bernd, und Bernds Mutter ist Christina. Das beschreibt Annika mit Hilfe von Gleichungen so:

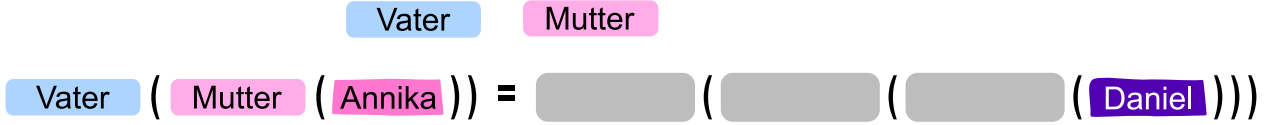
- `Vater(Annika) = Bernd`
- `Mutter(Bernd) = Christina`

Ihre Verwandtschaft mit Christina kann Annika auch mit nur einer Gleichung beschreiben:

- `Mutter(Vater(Annika)) = Christina` steht für «Mutter vom Vater von Annika ist Christina»

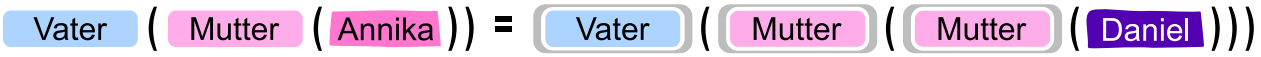
Nun hätte sie gerne eine Gleichung für ihre Verwandtschaft mit Daniel.

Ergänze die folgende Gleichung so, dass sie die Verwandtschaft zwischen Annika und Daniel beschreibt.



Lösung

So ist es richtig:



Zuerst betrachten wir die linke Seite der Gleichung und entdecken, dass Emil der Vater von Annikas Mutter ist, also: `Vater(Mutter(Annika)) = Emil`. Um die Lücken auf der rechten Seite zu füllen, muss man herausfinden, wie Daniel mit Emil verwandt ist. Dazu betrachten wir den Stammbaum und gehen von Daniel aus schrittweise in Richtung Emil:

1. Daniels Mutter, also `Mutter(Daniel)`, ist mit Emil verwandt; Daniels Vater nicht.
2. Die Mutter von Daniels Mutter, also Daniels Grossmutter bzw. `Mutter(Mutter(Daniel))`, ist mit Emil verwandt, denn ...
3. ... Emil ist der Vater dieser Grossmutter: `Emil = Vater(Mutter(Mutter(Daniel)))`.

Dies ist Informatik!

Annika verwendet ihre Kurzschreibweise für die Vater- und Mutter-Beziehungen im Stammbaum, nämlich `Vater()` und `Mutter()` wie mathematische *Funktionen*, die für ein *Argument* (hier: eine Person im Stammbaum) einen *Wert* haben (eine andere Person). Auch in Computerprogrammen gibt es Funktionen; das sind eigenständige Module des Programm-Codes, mit denen man mathematische Funktionen direkt umsetzen kann: Eine solche Funktion wird mit einem oder mehreren Argumenten (in der Informatik häufig auch: *Parameter*) aufgerufen und liefert, nach Ausführung des Codes, einen Wert als Ergebnis.

Diese Biberaufgabe zeigt, wie Funktionsaufrufe für komplexere Berechnungen zusammengesetzt (oder: «verschachtelt») werden können. In einem *verschachtelten Funktionsaufruf* dienen die Ergebnisse der inneren Funktionsaufrufe als Eingabe für äussere Aufrufe. Ein verschachtelter Funktionsaufruf wird von innen nach aussen ausgewertet: Zum Beispiel wird bei der Auswertung von `Vater(Mutter(Mutter(Daniel)))` zuerst die Mutter von Daniel ermittelt, dann die Mutter seiner Mutter und schliesslich der Vater der Mutter seiner Mutter. Die Zusammensetzung von Funktionen bzw. Funktionsaufrufen (auch *Funktionskomposition* genannt) steht in den meisten Programmiersprachen zur Verfügung, ist aber besonders wichtig für sogenannte *funktionale Programmiersprachen*.

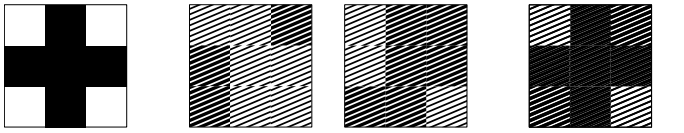
Stichwörter und Webseiten

- Funktion: `https://de.wikipedia.org/wiki/Funktion_(Programmierung)`
- Geschachtelte Funktionen: `https://en.wikipedia.org/wiki/Function_composition_(computer_science)`

 © Informatik-Biber 2025, SVIA

22. Kurierdienst

Ein geheimes Bild, das aus schwarzen ■ und weissen □ Pixeln besteht, soll sicher übertragen werden. Hierfür erstellt der Kurierdienst auf transparenten Folien zwei Bilder aus dunklen ▨ und hellen ▨ Pixeln. Das geheime Bild wird erst dann erkennbar, wenn die beiden Folien übereinander gelegt werden.

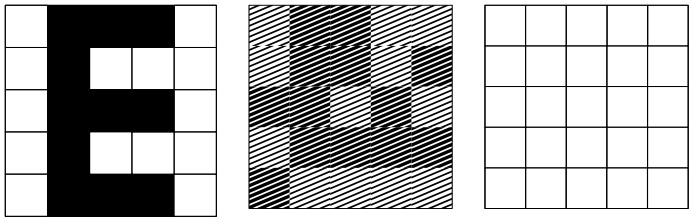


Geheimes Bild | Folie 1 | Folie 2 | Folien 1 und 2 übereinander

Die Bilder für die beiden Folien werden so erstellt: Zuerst wird für Folie 1 ein zufälliges Muster aus dunklen ▨ und hellen ▨ Pixeln erzeugt. Die Pixel im Bild für Folie 2 werden dann nach der folgenden Regel gesetzt, abhängig von den Pixeln an der gleichern Stelle im geheimen Bild und in Folie 1:

- Ist das Pixel im geheimen Bild schwarz ■, dann müssen die Pixel in Folie 1 und Folie 2 verschieden sein (das eine dunkel ▨, das andere hell ▨).
- Ist das Pixel im geheimen Bild weiss □, dann müssen die Pixel in Folie 1 und Folie 2 gleich sein (beide ▨ oder beide ▨).

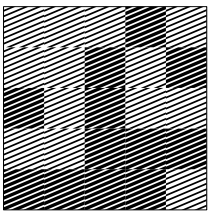
Für das folgende geheime Bild wurde Folie 1 bereits erzeugt. Erstelle nun Folie 2.



Geheimes Bild | *Folie 1* | *Folie 2*

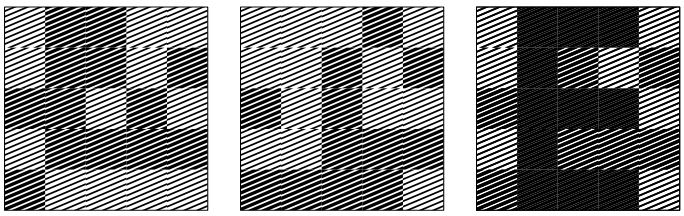
Lösung

So ist es richtig:



In diesem Bild für Folie 2 wurde jedes Pixel entsprechend der oben beschriebenen Regel – abhängig vom geheimen Bild und von Folie 1 – gesetzt: Das Bild unterscheidet sich nur genau an den Stellen, wo das geheime Bild schwarze Pixel hat, vom Bild für Folie 1.

Hier siehst du, wie durch Übereinanderlegen von Folie 1 und dieser Folie 2 das geheime Bild erkennbar wird:



Folie 1 | Folie 2 | Folien 1 und 2 übereinander

Dies ist Informatik!

Der Kurierdienst verwendet ein *kryptografisches Verfahren*, das auf visueller Wahrnehmung beruht und daher *visuelle Kryptografie* genannt wird. Eine solche Technik wurde 1994 von den israelischen Wissenschaftlern Moni Naor und Adi Shamir entwickelt. In Bezug auf die einzelnen Folien ist das Verfahren sehr sicher. Da sie auf einer zufälligen Pixelmatrix basieren, kann man aus jeder Folie alleine keine Information gewinnen, selbst mit Computerhilfe nicht. Die Entschlüsselung ist nur möglich – und dann sogar recht einfach – wenn beide Folien vorhanden sind.

Zur Übertragung geheimer Nachrichten könnte eine zufällige, aber feste Folie 1 sowohl beim Absender als auch beim Empfänger als «Schlüssel» gespeichert werden. Dann müsste für jede neue Nachricht nur noch Folie 2 erzeugt und übermittelt werden. Wenn man den Schlüssel, also die Folie 1 jedoch mehrfach verwendet, ist das Verfahren nicht mehr ganz sicher. Dieses Problem hat man generell bei Verschlüsselungen, die nach dem Prinzip des *One-Time-Pad* funktionieren. Dabei muss der Schlüssel (mindestens) genau so lang sein wie die geheime Nachricht und muss zufällig erzeugt werden. Die Verschlüsselung wird durch eine umkehrbare Verknüpfung der passenden Zeichen aus Nachricht und Schlüssel erzeugt. In der Informatik wird für die Nachrichten aus Bits, die Computer miteinander austauschen, in der Regel die Operation XOR als eine solche umkehrbare Verknüpfung verwendet. Die Kombination der hellen und dunklen Pixel in dieser Biberaufgabe entspricht genau dieser Operation.

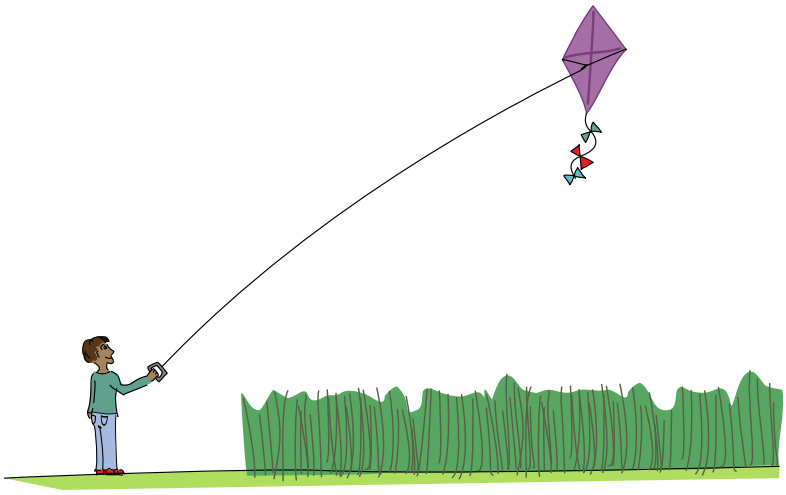
 © Informatik-Biber 2025, SVIA

Stichwörter und Webseiten

- visuelle Kryptographie: `https://de.wikipedia.org/wiki/Visuelle_Kryptographie`

© Informatik-Biber 2025, SVIA

23. Der Drachen ist weg!

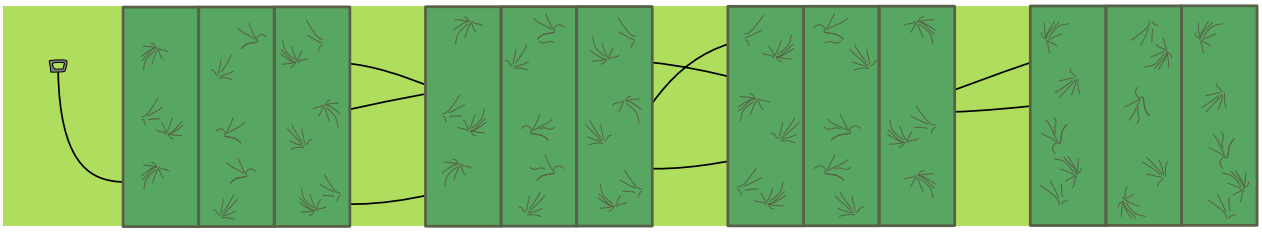


So ein Pech! Asterios hat seinen Drachen auf der Wiese verloren. Die Drachenschnur hat sich im hohen Gras verfangen, und es ist gar nicht so einfach, den Drachen wiederzufinden.

Die Wiese ist in 15 Felder unterteilt, die man einzeln durchsuchen kann.

Asterios hat schon 3 Felder der Wiese durchsucht. Er schaut sich genau an, wie die Schnur in diesen Feldern verläuft, und erkennt: Jetzt muss er nur noch ein weiteres Feld durchsuchen, um sicher zu wissen, wo der Drachen ist.

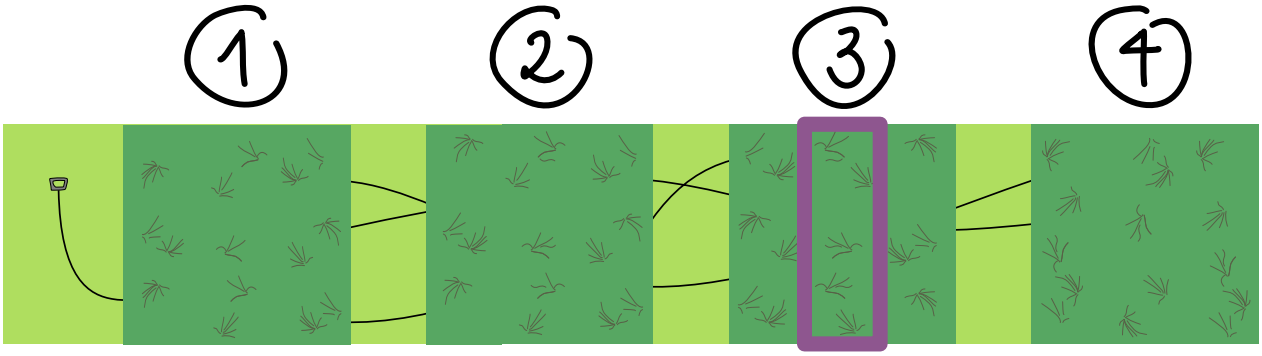
Welches Feld ist das?



Lösung

So ist es richtig:

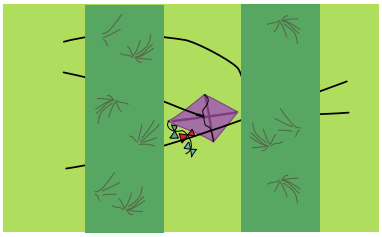
Asterios muss das lila Feld durchsuchen, um sicher zu wissen, wo der Drachen ist.



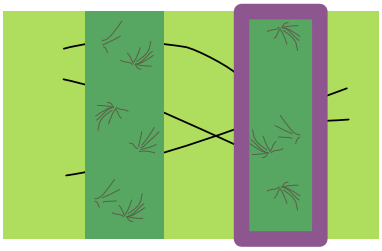
Die Lösung kannst du in zwei Schritten finden. Zuerst überlegst du dir, in welchem der vier Blöcke 1, 2, 3 und 4 der Drachen liegen muss. Denke daran, dass die Drachenschnur links beginnt und der Drachen am anderen Ende der Schnur hängt.

Der Drachen kann sich nicht in Block 4 befinden, weil die Drachenschnur in Block 4 hineingeht und dort auch wieder herauskommt. Der Drachen muss sich rechts von Block 2 befinden, weil man zwischen Block 2 und Block 3 drei Schnursegmente sieht. Zwei Schnursegmente müssen zu einer Schlaufe im rechten Teil der Wiese gehören, und ein Schnursegment führt zum Drachen.

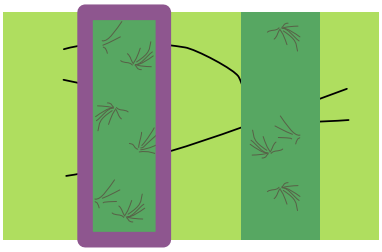
Wenn Asterios das mittlere Feld in Block 3 durchsucht, lassen sich drei Fälle unterscheiden:



Fall 1: In dem Feld liegt der Drachen. Super! Der Drachen ist gefunden und die Suche ist beendet.



Fall 2: In dem Feld sieht man keinen Drachen, aber drei Schnursegmente, die in das rechte Feld gehen. Weil aus diesem Feld nur zwei Schnursegmente nach rechts herausführen, muss der Drachen im rechten Feld liegen.



Fall 3: In dem Feld sieht man keinen Drachen, aber zwei Schnursegmente, die vom linken Feld zum rechten Feld gehen. Dann muss der Drachen links von diesem Feld liegen. Denn die beiden Schnursegmente gehören zu einer Schlaufe im rechten Teil der Wiese.

 © Informatik-Biber 2025, SVIA

In jedem Fall wissen wir also nach dem Durchsuchen des lila Felds, in welchem Feld der Drachen ist.

Dies ist Informatik!

Beim systematischen Durchsuchen der Felder wird eine deterministische Suche durchgeführt: Jede neue Information – die Beobachtungen zu einem durchsuchten Feld – ermöglicht gezielte Schlussfolgerungen über benachbarte Felder. Eine wichtige Rolle spielt dabei eine *topologische* Eigenschaft der Drachenschnur: Die Schnur bildet geschlossene Schlaufen, und jeder durch zwei Geraden begrenzter Bereich (wie die Felder in dieser Biberaufgabe) enthält entweder eine gerade Anzahl von Segmenten oder die Wendestelle einer Schlaufe.

Topologische Eigenschaften beschreiben Strukturen, die durch die Anordnung und Verbindung von Elementen entstehen – unabhängig von Grössen, Abständen oder Winkeln. Es geht also nicht darum, wie gross oder wie lang etwas ist, sondern wie Dinge miteinander verbunden sind und wie viele Wege oder Durchgänge es gibt.

Systematisches Durchsuchen mit topologischer Interpretation ist nicht nur ein spannendes Denkspiel, sondern hat auch praktische Bedeutung in der Informatik:

Ein Strassennetz kann zum Beispiel als System aus Punkten und Verbindungen betrachtet werden – etwa Kreuzungen und Strassen. Diese Struktur hilft Suchalgorithmen, zielgerichtet Wege zu finden, Umleitungen zu erkennen und zu zeigen, wie man möglichst rasch zum Ziel gelangt.

Auch bei der Fehlersuche in Stromnetzen liefert deren Topologie entscheidende Hinweise: Parallelschaltungen, Schleifen oder Unterbrechungen zeigen oft, wo der Fehler liegen könnte – ganz ohne genaue Masse, nur durch die logische Struktur des Netzwerks.

Stichwörter und Webseiten

- Topologie: `https://de.wikipedia.org/wiki/Topologie_(Mathematik)`
- Suchverfahren: `https://de.wikipedia.org/wiki/Suchverfahren`

 © Informatik-Biber 2025, SVIA

24. Brennende Kerzen

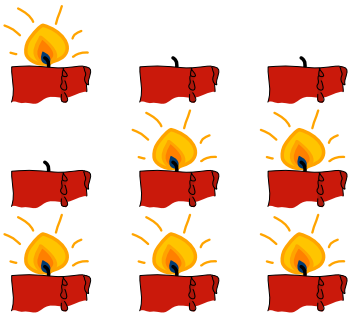
Es gibt eine Tradition, an den vier Sonntagen vor Weihnachten Kerzen anzuzünden: 1 Kerze am ersten Sonntag, 2 Kerzen am zweiten Sonntag und so weiter.

Chris liebt diese Tradition. Alle vier seiner Kerzen sind gleich lang. Chris' Weihnachtsfest wäre wunderschön, wenn auch nach dem letzten Sonntag alle Kerzen noch gleich lang wären. Dafür müsste er jede Kerze insgesamt gleich oft anzünden.

Leider findet Chris keine Möglichkeit, ein wunderschönes Weihnachtsfest zu haben.

Wenn die Tradition nur drei Sonntage (und Kerzen) umfassen würde, wäre es möglich. Dann würde Chris jede Kerze genau zweimal anzünden:

1. Sonntag
2. Sonntag
3. Sonntag

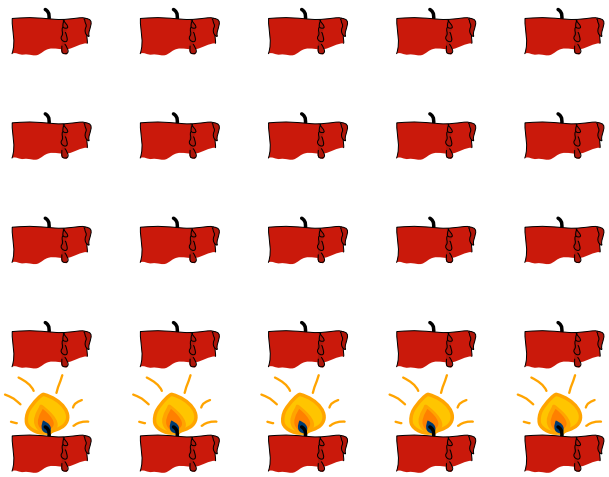


Auch mit fünf Sonntagen (und Kerzen) wäre es möglich.

Zeige Chris, wie er jede Kerze gleich oft anzünden kann.

Für den fünften Sonntag haben wir die Kerzen bereits angezündet.

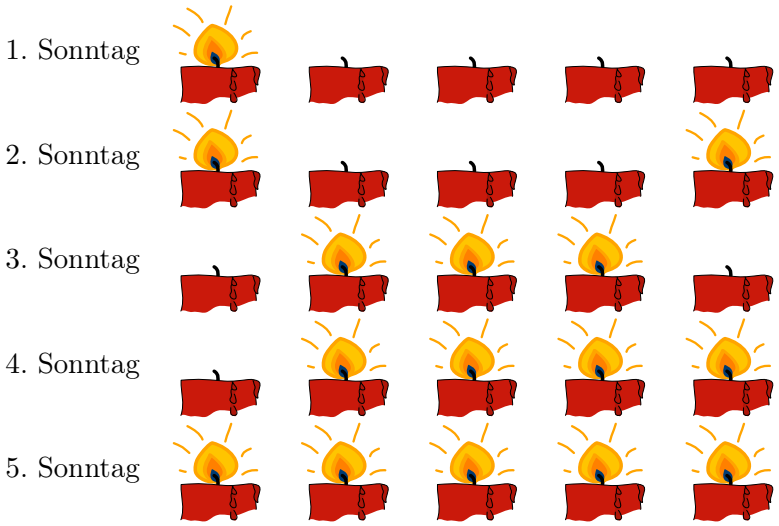
1. *Sonntag*
2. *Sonntag*
3. *Sonntag*
4. *Sonntag*
5. *Sonntag*



Lösung

So ist es richtig:

Chris kann die Kerzen so anzünden, jede einzelne genau dreimal:



Es gibt viele Möglichkeiten, dies zu erreichen. In allen Fällen kann man so vorgehen:

1. Man gruppiert die Sonntage in Paare, deren Nummern zusammen die Gesamtanzahl der Sonntage ergeben; bei fünf Sonntagen sind das die Paare 1 und 4 sowie 2 und 3.
2. Für jedes dieser Paare zündet man die Kerzen so an, dass die beiden Mengen an Kerzen, die an den jeweiligen Sonntagen angezündet werden, disjunkt sind – z.B. an Sonntag 1 die Kerze 1 (von links) und an Sonntag 4 die Kerzen 2 bis 5. (Betrachtet man jede Kerze als ein Bit mit 1 = angezündet, 0 = nicht angezündet, dann müssen für jedes Sonntage-Paar die beiden Kerzen-Bitfolgen der Sonntage bitweise komplementär zueinander sein.) So wird an den beiden Sonntagen eines Paares jede Kerze genau einmal angezündet.
3. Zusätzlich wird jede Kerze ein weiteres Mal am letzten Sonntag (Sonntag 5) angezündet.

Daher wird jede Kerze insgesamt gleich oft angezündet.

Dies ist Informatik!

Auf den ersten Blick scheint diese Bebras-Aufgabe ein ziemlich mathematisches Problem zu sein. Es gibt tatsächlich einen Beweis dafür, dass Chris' Kerzenproblem für jede ungerade Anzahl von Sonntagen lösbar ist. (Man erkennt, dass die in der Lösungserklärung vorgestellte Strategie generell für ungerade Sonntaganzahlen funktioniert.)

Wenn man jedoch konkret herausfinden will, wie man die Kerzen anzünden soll, muss man einen Algorithmus beschreiben, der die Anzündreihenfolge vorgibt - oder noch besser: einen, der alle möglichen Lösungen aufzählt. Dieser Algorithmus basiert auf der obigen Erklärung; sei n die (ungerade) Anzahl der Sonntage:

 © Informatik-Biber 2025, SVIA

1. Am n-ten Sonntag: Zünde alle n Kerzen an.
2. Für $i = 1$ bis $(n - 1)/2$:
   a) Zünde am Sonntag i die ersten i Kerzen an und am Sonntag $n - i$ die letzten $n - i$ Kerzen.

Beachte, dass Schritt 2a dieses Algorithmus auf alle Arten durchgeführt werden kann, welche die oben in der Erklärung unter Punkt 2 genannte Bedingung erfüllen.

Die Entwicklung von Algorithmen zur Problemlösung ist eine der wichtigsten Aufgaben von Informatikerinnen und Informatikern. Wenn ein Algorithmus auf solider mathematischer Modellierung beruht, lassen sich seine gewünschten Eigenschaften leichter beweisen als ohne eine solche Modellierung. Für den obigen Algorithmus kann man beweisen, dass er bei jeder ungeraden Anzahl von Sonntagen funktioniert.

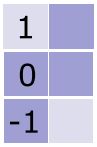
Stichwörter und Webseiten

- Algorithmen: `https://de.wikipedia.org/wiki/Algorithmus`

© Informatik-Biber 2025, SVIA

25. Helligkeitskarte

Digitale Bilder bestehen häufig aus Pixeln. Sandra erstellt Helligkeitskarten für solche Pixelbilder. Dazu legt sie um ein Bild zuerst einen Rahmen aus zusätzlichen weissen Pixeln. Dann bestimmt sie für jedes Pixel des Bildes einen Helligkeitswert, und zwar:

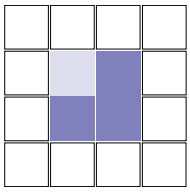


1, falls das Pixel heller ist als sein rechtes Nachbarpixel.

0, falls das Pixel gleich hell ist wie sein rechtes Nachbarpixel.

-1, falls das Pixel dunkler ist als sein rechtes Nachbarpixel.

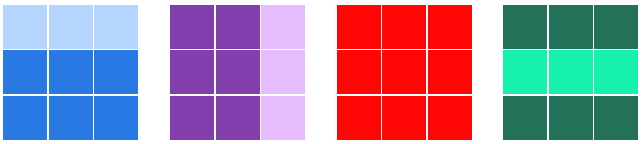
Hier siehst du ein Bild aus vier Pixeln (plus die zusätzlichen weissen Pixel) und seine Helligkeitskarte.



1	-1
0	-1

Unten siehst du vier Bilder mit je neun Pixeln. Genau drei davon haben die gleiche Helligkeitskarte.

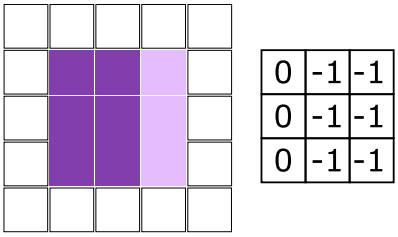
*Welches der Bilder hat als einziges eine **andere** Helligkeitskarte?*



A) B) C) D)

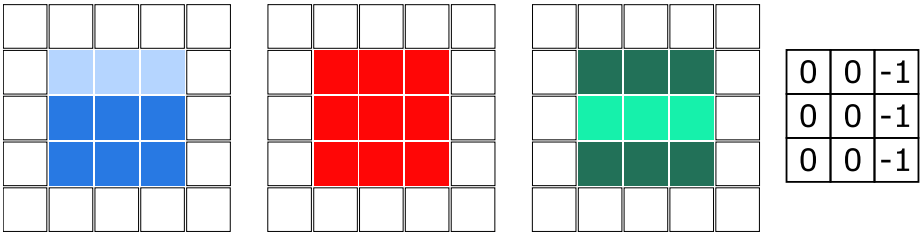
Lösung

Antwort B ist richtig:



0	-1	-1
0	-1	-1
0	-1	-1

Die übrigen drei Bilder haben alle die gleiche Helligkeitskarte:



0	0	-1
0	0	-1
0	0	-1

Um die richtige Antwort zu finden, kann man die Helligkeitskarten aller vier Bilder berechnen und diese vergleichen. Oder man kann feststellen, dass eine Helligkeitskarte nur Helligkeitsunterschiede in horizontaler Richtung erfasst. Da die drei Bilder der Antworten A, C und D in horizontaler Richtung keine Helligkeitsunterschiede haben, müssen ihre Helligkeitskarten gleich sein.

Dies ist Informatik!

Für die Darstellung von Bildern in Computern kennt die Informatik viele verschiedene Formate. Grundsätzlich werden Bilder entweder als *Vektorgrafiken* oder als *Rastergrafiken* gespeichert. Bei Letzteren nennt man die einzelnen Rasterpunkte auch *Pixel* (kurz für: picture element). Jedes Pixel kann im einfachsten Fall schwarz oder weiss sein; dann kann man jedes Pixel durch einziges Bit mit den Werten 0 oder 1 darstellen. Farbige Pixel werden durch mehrere Werte beschrieben, die z.B. jeweils den Anteil der Farben Rot, Grün und Blau in der Farbe des Pixels angeben.

Die Helligkeitskarte in dieser Biberaufgabe ist ähnlich zum Konzept einer *Faltung* zwischen dem Bild und einem *Filter*. Je nach Filter können durch eine solche Faltung verschiedene strukturelle Eigenschaften des Bildes erkennbar gemacht werden, wie beispielsweise Ecken, Kanten oder Bereiche uniformer Helligkeit. Dies erleichtert einem Computer die Interpretation der im Bild vorhandenen Information.

Sogenannte *Convolutional Neural Networks* (CNN), häufig zentrale Bestandteile von KI-Systemen, nutzen das Konzept der Faltung zur Bilderkennung. Um komplexe Objekte in einem Bild zu erkennen, lernt ein CNN, selbst geeignete Filter zu entwickeln, mit denen es das Bild faltet. Damit kann es beispielsweise Katzenbilder von Hundebildern unterscheiden oder Tumore in medizinischen Scans entdecken.

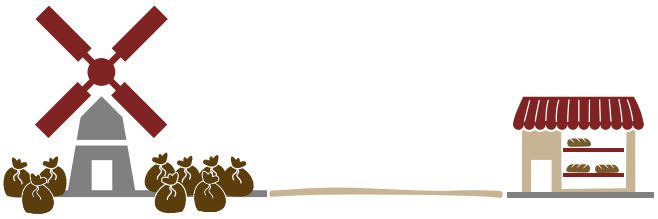
 © Informatik-Biber 2025, SVIA

Stichwörter und Webseiten

- Computer Vision: `https://de.wikipedia.org/wiki/Computer_Vision`
- Pixel: `https://de.wikipedia.org/wiki/Pixel`
- Faltung: `https://de.wikipedia.org/wiki/Faltung_(Mathematik)#Diskrete_Faltung`
- Filter: `https://de.wikipedia.org/wiki/Faltungsmatrix`
- Convolutional Neural Network:
  `https://de.wikipedia.org/wiki/Convolutional_Neural_Network`

 © Informatik-Biber 2025, SVIA

26. Mehltransport



Albert und Marco arbeiten in einer Bäckerei.
Sie müssen immer wieder Mehl von der Mühle holen gehen. Es darf aber immer nur einer gehen, damit der andere die Kunden bedienen kann.

Ihre Leistungen beim Mehl holen sind unterschiedlich:

Albert holt 13 kg Mehl in 1 Stunde.

Marco holt 5 kg Mehl in 30 Minuten.

Für jeden der beiden gilt aber: Nach dreimal Mehl holen sind 30 Minuten Erholung nötig. In dieser Zeit können Kunden bedient werden.

Albert und Marco möchten in 8 Stunden so viel Mehl wie möglich holen. Unter genau einer der folgenden Bedingungen schaffen sie das. Unter welcher?

A) Albert muss zuerst gehen.
B) Marco muss zuerst gehen.
C) Marco muss zuletzt gehen.
D) Albert darf nicht zuletzt gehen.
E) Marco muss genau einmal gehen.

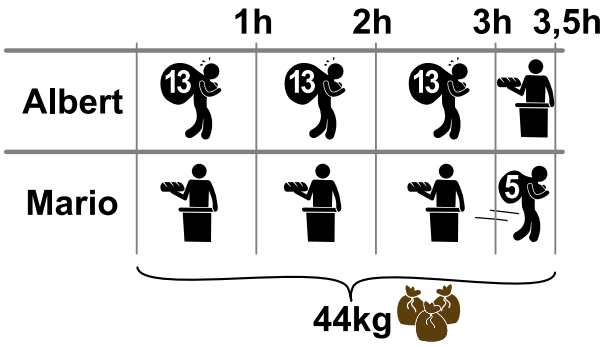
Lösung

Antwort A ist richtig.

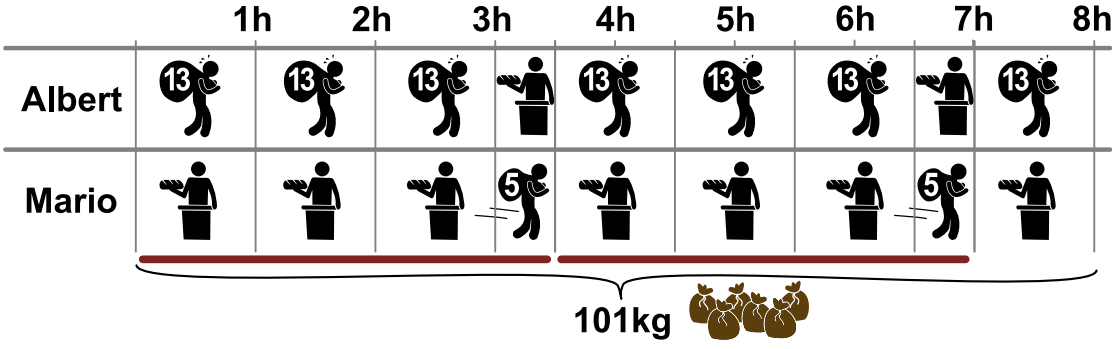
Wir wissen:

- Albert holt 13 kg pro Stunde.
- Marco holt 5 kg in 30 Minuten, also 10 kg pro Stunde.
- Einer muss immer in der Bäckerei bleiben.
- Jeder darf höchstens dreimal hintereinander gehen, danach muss er 30 Minuten in der Bäckerei bleiben.
- Während einer in der Bäckerei bleibt, kann der andere weiter Mehl holen gehen.

Albert schafft mehr Mehl pro Stunde als Marco. Deshalb sollte er so oft wie möglich gehen. Marco geht nur, wenn Albert sich nach dreimal Gehen erholt  . Auf diese Weise können sie in 3,5 Stunden maximal 44 kg Mehl holen:



Weil Albert danach erholt ist, können sie dieses Muster (kurz: A, A, A, M) wiederholen und in 7 Stunden 88 kg Mehl holen. Dann kann Albert noch einmal gehen. Insgesamt holen sie also in 8 Stunden maximal 101 kg Mehl, wenn Albert zuerst geht.



Antwort B: Auch wenn Marco zuerst geht, können sie in 3,5 Stunden 44 kg Mehl holen, diesmal aber nach dem Muster (M, A, A, A). Nach zwei Wiederholungen und 88 kg Mehl ist noch eine Stunde übrig. Weil Albert sich dann erholen muss, muss Marco in dieser Stunde zweimal gehen und 10 kg Mehl holen. Insgesamt können sie nur unter dieser Bedingung weniger Mehl holen als oben, nämlich $44 + 44 + 10 = 98$ kg.

 © Informatik-Biber 2025, SVIA

Antwort C und D: Wenn Marco zuletzt geht, kann Albert nicht zuletzt gehen. Also sind Antwort C und D gleichbedeutend. Auch unter diesen Bedingungen können sie in 7 Stunden 88 kg Mehl holen, aber in der letzten Stunde wieder nur 10 kg, insgesamt also wieder nur 44 + 44 + 10 = 98 kg.

Antwort E: Wenn Marco genau einmal geht, können sie in den ersten 3,5 Stunden wieder 44 kg Mehl holen. In den zweiten 3,5 Stunden muss Alberg sich aber 30 Minuten lang erholen; und weil Marco nicht noch einmal gehen darf, holen sie in dieser Zeit 39 kg. Danach kann Alberg noch einmal gehen, so dass sie unter dieser Bedingung nur 44 + 39 + 13 = 96 kg Mehl holen können.

Dies ist Informatik!

Wenn Albert und Marco in einer bestimmten Zeit möglichst viel Mehl holen wollen, müssen sie ein Planungs- und Optimierungsproblem lösen:

- Sie müssen ihre Gänge planen und dabei Beschränkungen (Ruhezeiten und Belegungsbeschränkung) einhalten.
- Sie müssen die Gänge zwischen Albert und Marco aufteilen.
- Sie wollen die Gesamtmenge des geholten Mehls maximieren (Optimierungsziel).

Planungs- und Optimierungsprobleme treten in vielen Bereichen des Lebens auf. Das wichtigste Ziel dabei ist *Effizienz*; das bedeutet einfach gesagt, unter gegebenen Bedingungen möglichst viel zu schaffen. In Produktionsanlagen soll mit den vorhandenen Personen und Maschinen möglichst viel produziert werden, an einem Flughafen sollen an den vorhandenen Check-In-Schaltern und Flugsteigen möglichst viele Flüge und Passagiere abgefertigt werden, und so weiter. Sobald Planungs- und Optimierungsprobleme grösser werden, helfen Computerprogramme dank Methoden der Informatik, sie zu lösen. Solche Probleme, bei denen Bedingungen zu beachten sind, kommen aber auch in der Informatik selbst vor, denn auch Computerprozessoren sollen effizient sein. Zwei Beispiele: Bei der Ausführung von Befehlen muss berücksichtigt werden, dass arithmetische Logikeinheiten jeweils nur einen Befehl gleichzeitig ausführen können (Belegungsbeschränkung). Befehle, die Daten aus dem Speicher lesen, müssen auf das Eintreffen der Daten warten und haben dann «Ruhezeiten» - so wie Albert und Marco nach dreimal Gehen in dieser Biberaufgabe.

Stichwörter und Webseiten

- Optimierung: `https://de.wikipedia.org/wiki/Optimierungsproblem`
- Scheduling: `https://de.wikipedia.org/wiki/Prozess-Scheduler`

© Informatik-Biber 2025, SVIA

27. Schwarz - Weiss

Sarah möchte Folgen von schwarzen und weissen Kästchen mit Buchstaben beschreiben. Sie wendet dazu diesen Algorithmus auf eine Kästchen-Folge an:

- Wenn alle Kästchen der Folge weiss sind, schreibe `W`.
- Wenn alle Kästchen der Folge schwarz sind, schreibe `S`.
- Wenn die Folge schwarze und weisse Kästchen enthält, schreibe `x` und mache Folgendes:
  - Wende den Algorithmus auf die linke Hälfte der Folge an.
  - Wende den Algorithmus auf die rechte Hälfte der Folge an.

Hier siehst du für einige Kästchen-Folgen, welche Buchstaben-Beschreibung der Algorithmus ausgibt:

`W`

`xWS`

`xxSWS`

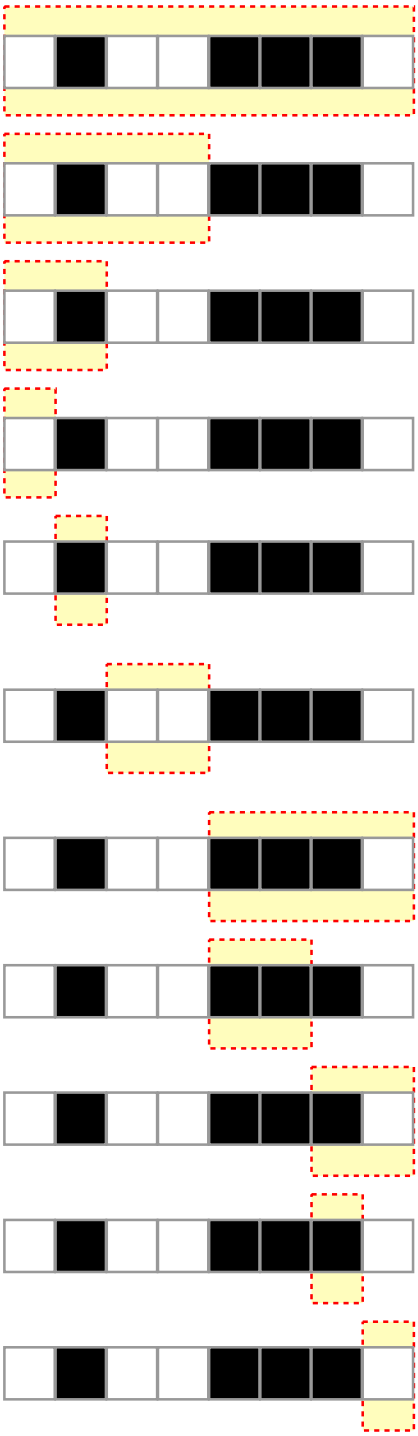
`xSxWxSW`

Welche Buchstaben-Beschreibung gibt Sarahs Algorithmus für diese Kästchen-Folge aus?

Lösung

So ist es richtig: `xxxWSWxSxSW`.

Wir wenden den Algorithmus auf die Kästchen-Folge an und konstruieren die Buchstaben-Beschreibung Schritt für Schritt. In den Bildern ist die Kästchenfolge, auf die der Algorithmus gerade angewendet wird, gelb markiert.



`x` - Die Folge enthält schwarze und weisse Kästchen, also schreibt der Algorithmus `x` und wendet sich selbst auf die linke Hälfte der Folge an.

`xx` - Die Folge enthält schwarze und weisse Kästchen, also schreibt der Algorithmus `x` und wendet sich selbst auf die linke Hälfte der Folge an.

`xxx` - Die Folge enthält schwarze und weisse Kästchen, also schreibt der Algorithmus `x` und wendet sich selbst auf die linke Hälfte der Folge an.

`xxxW` – Alle Kästchen der Folge sind weiss, also schreibt der Algorithmus `W`. Damit wird der Algorithmus für diese Folge beendet und nun auf die rechte Hälfte der vorigen Folge angewendet.

`xxxWS` - Alle Kästchen der Folge sind schwarz, also schreibt der Algorithmus `S` und wird danach auf die rechte Hälfte der vorigen Folge angewendet.

`xxxWSW` - Alle Kästchen der Folge sind weiss, also schreibt der Algorithmus `W`. Jetzt haben wir die linke Hälfte der gesamten Folge verarbeitet und können nun den Algorithmus auf die rechte Hälfte anwenden.

`xxxWSWx` - Die Folge enthält schwarze und weisse Kästchen, also schreibt der Algorithmus `x` und wendet sich selbst auf die linke Hälfte der Folge an.

`xxxWBWxS` - Alle Kästchen der Folge sind schwarz, also schreibt der Algorithmus `S` und wird danach auf die rechte Hälfte der vorigen Folge angewendet.

`xxxWSWxSx` - Die Folge enthält schwarze und weisse Kästchen, also schreibt der Algorithmus `x` und wendet sich selbst auf die linke Hälfte der Folge an.

`xxxWSWxSxS` - Alle Kästchen der Folge sind schwarz, also schreibt der Algorithmus `S` und wird danach auf die rechte Hälfte der vorigen Folge angewendet.

`xxxWSWxSxSW` - Alle Kästchen der Folge sind weiss, also schreibt der Algorithmus `W`. Jetzt haben wir auch die rechte Hälfte der gesamten Folge verarbeitet und sind fertig.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Sarahs Algorithmus hat eine ganz besondere Eigenschaft: Wenn die eingegebene Kästchen-Folge farblich gemischt ist, wendet er sich sozusagen selbst an, und zwar nacheinander auf die linke und die rechte Hälfte der Eingabe. Damit wird die Aufgabe, die Eingabe als Buchstabenfolge zu beschreiben, in zwei kleinere Teilaufgaben zerlegt. Das ist unter anderem dann sinnvoll, wenn die Teilaufgaben leichter zu bearbeiten sind als die gesamte Aufgabe. Die Aufteilung von Problemen in (hoffentlich leichtere) Teilprobleme ist in der Informatik unter dem Namen «divide and conquer» bekannt, gemäss der im antiken Rom bekannten Herrschaftsstrategie «divide et impera», auf Deutsch «teile und herrsche». Wenn ein Lösungsverfahren die Teilaufgaben auf die gleiche Weise angeht wie die gesamte Aufgabe, sich also selbst auf die Teilaufgaben anwendet und dann insbesondere die Teilaufgaben wieder in kleinere Teile zuerlegt, folgt das Verfahren gleichzeitig dem Prinzip der *Rekursion* - so wie Sarahs Algorithmus in dieser Biberaufgabe. Rekursion wird in der Informatik sehr häufig genutzt, ob beim Sortieren von Daten, bei der Konstruktion von Dateisystemen und vieles mehr.

Sarahs Algorithmus beschreibt bzw. *kodiert* die Kästchen-Folgen mit Buchstaben. Eine Kodierung muss umkehrbar sein; es muss also ein Verfahren geben, die Buchstaben wieder in die originale Kästchenfolge umzuwandeln. Wenn die Buchstaben-Beschreibung um die Anzahl der Kästchen in der beschriebenen Folge ergänzt wird, ist das problemlos möglich. Überlege dir, wie! Möglicherweise benötigst du auch dafür einen rekursiven Algorithmus. Im Idealfall ist die Kodierung, die Sarahs Algorithmus ausgibt, sehr viel kürzer als die eingegebene Kästchenfolge. Zum Beispiel wird eine Folge von 1024 weissen Kästchen mit einem einzigen `W` beschrieben. Sarahs Algorithmus betreibt also nicht nur Kodierung, sondern auch *Datenkompression* (platzsparende Beschreibung von Daten), und folgt dabei ähnlichen Ideen wie Verfahren zur Kompression von Bilddaten (etwa ins Foto-Format JPEG) oder Videodaten.

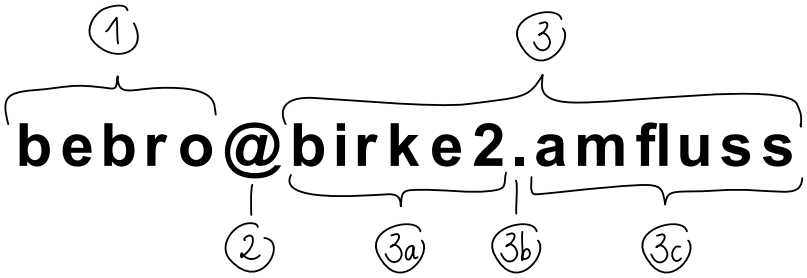
Stichwörter und Webseiten

- Rekursion: `https://de.wikipedia.org/wiki/Rekursion`
- Datenkompression: `https://de.wikipedia.org/wiki/Datenkompression`
- Quadtree: `https://de.wikipedia.org/wiki/Quadtree`

© Informatik-Biber 2025, SVIA

28. Biber-Mail-Adressen

Die IT-Biber brauchen ein System, das erkennt, ob eine Zeichenfolge eine Biber-Mail-Adresse ist. Eine Biber-Mail-Adresse besteht aus drei Teilen:



	Bezeichnung	Erläuterung	Zulässige Werte
1	**Benutzername**	eine beliebige, nicht leere Zeichenfolge aus Kleinbuchstaben und Ziffern	0–9, a-z
2	**AT-Zeichen**		@
3	**Servername**		
3a	**Domänename**	eine beliebige, nicht leere Zeichenfolge aus Kleinbuchstaben und Ziffern	0–9, a-z
3b	**Punkt**		.
3c	**Top-Level-Domänename**	eine beliebige, nicht leere Zeichenfolge aus Kleinbuchstaben und Ziffern	0–9, a-z

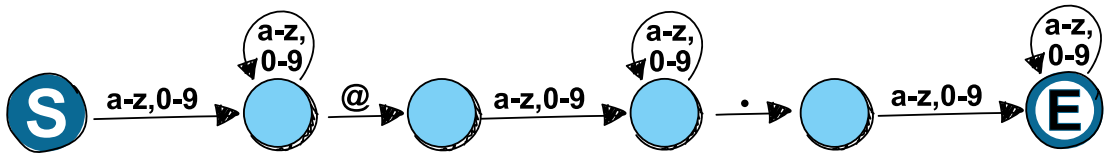
«Nicht leere» Zeichenfolgen bestehen aus mindestens einem Zeichen. «bebro@birke2.amfluss» ist ein Beispiel für eine Biber-Mail-Adresse.

Die IT-Biber überlegen sich vier Systeme und beschreiben sie mit Diagrammen aus Kreisen und Pfeilen. Ein Kreis steht für einen der Zustände, in denen das System sein kann. Ein Pfeil beschreibt den Wechsel zu einem nächsten, möglicherweise gleichen Zustand. Die Beschriftung des Pfeils sagt, zu welchen Zeichen dieser Zustandswechsel passt.

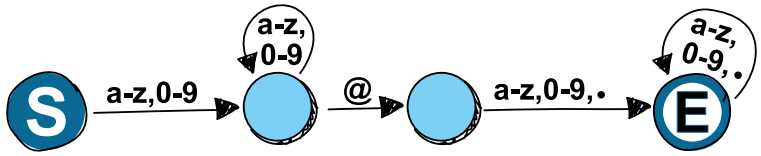
Ein System untersucht eine Zeichenfolge Zeichen für Zeichen, von links nach rechts, und ist zu Beginn im Zustand S. Der nächste Zustand hängt vom aktuell untersuchten Zeichen ab: Das System folgt dem Zustandswechsel, der zum aktuellen Zeichen passt. Ist das System nach dem letzten Zeichen im Zustand E, hat es die Zeichenfolge als Biber-Mail-Adresse erkannt.

Nur eines dieser Systeme erkennt alle möglichen Biber-Mail-Adressen korrekt. Welches?

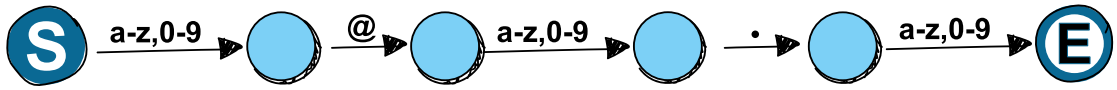
A)



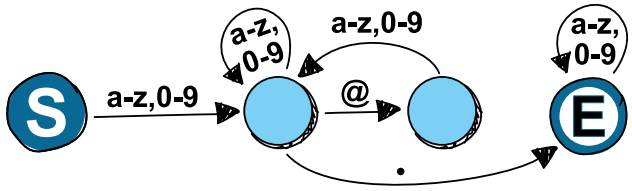
B)



C)



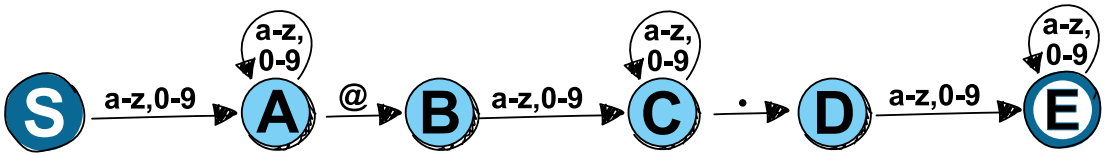
D)



 © Informatik-Biber 2025, SVIA

Lösung

Antwort A ist richtig. Zur Erläuterung bezeichnen wir alle Zustände mit Buchstaben:



Der Wechsel (in der Informatik-Fachsprache: Übergang) von S zu A erkennt einen Kleinbuchstaben oder eine Ziffer, so dass der Benutzername nicht leer sein kann. Der Übergang von A zu A erlaubt, beliebig viele weitere Kleinbuchstaben oder Ziffern zu erkennen. Der Übergang von A zu B erkennt ein At-Zeichen. Der Übergang von B zu C erkennt einen Kleinbuchstaben oder eine Ziffer, so dass der Domänenname nicht leer sein kann. Der Übergang von C zu C erlaubt, beliebig viele weitere Kleinbuchstaben oder Ziffern zu erkennen. Der Übergang von C zu D erkennt einen Punkt. Der Übergang von D zu E erkennt einen Kleinbuchstaben oder eine Ziffer, so dass der Top-Level-Domänenname nicht leer sein kann. Der Übergang von E zu E erlaubt, beliebig weitere Kleinbuchstaben oder Ziffern zu erkennen. Zusammen entspricht das genau der Definition von Biber-Mail-Adressen aus der Aufgabenstellung.

Antwort B ist falsch. Dieses System erkennt beispielsweise auch «biber@internet» oder «biber@.internet.com». Diese Zeichenfolgen sind aber keine Biber-Mail-Adressen.

Antwort C ist ebenfalls falsch. Dieses System kann nur wenige Biber-Mail-Adressen erkennen, nämlich nur solche deren Benutzername, Domänenname und Top-Level-Domänenname jeweils aus genau einem Buchstaben bestehen.

Antwort D ist auch falsch. Dieses System erkennt beispielsweise auch «biber@biberbau@amwasser.», und diese Zeichenfolge ist keine Biber-Mail-Adresse.

Dies ist Informatik!

Die in dieser Biberaufgabe verwendeten Systeme sind *endliche Automaten* (engl. *finite-state automata*). Endliche Automaten sind sogenannte erkennende Maschinen, und zwar die einfachste Form. Sie bestehen aus einer Menge von Zuständen, von denen einer der *Startzustand* ist und eine Teilmenge die *Endzustände* enthält. Wenn ein endlicher Automat nach Untersuchung der gesamten Zeichenfolge in einem Enzustand ist, gilt das untersuchte Wort als erkannt. Mithilfe von Übergängen, die von den untersuchten Zeichen abhängen, bestimmt der endliche Automat den nächsten Zustand.

Zu jedem endlichen Automaten gibt es eine passende *reguläre Grammatik*, die genau alle Wörter erzeugen kann (die dazugehörige *reguläre Sprache*), die der endliche Automat erkennt. Ähnliche Analogien zwischen Grammatiken und Sprachen existieren auch für andere erkennende Maschinen, die in der *Chomsky-Hierarchie* für *formale Sprachen* systematisiert werden.

Die Sprachen, die von endlichen Automaten erkannt werden, können ausserdem mit regulären Ausdrücken beschrieben werden. Der reguläre Ausdruck für die Biber-Mail-Adressen, wie sie in dieser Biberaufgabe definiert sind, lautet:

```
[a-z0-9]+@[a-z0-9]+.[a-z0-9]+
```

Endliche Automaten oder reguläre Ausdrücke werden an vielen Stellen eingesetzt, um Zeichenfolgen zu überprüfen: ob sie eine korrekt geformte Mail- oder Web-Adresse sind, ob sie bestimmten Anforderungen an ein Passwort entsprechen und vieles mehr. Zudem werden sie aufgrund ihrer einfachen Struktur auch in vielen eingebetteten Systemen verwendet, wo sehr wenig Strom verbraucht werden soll, beispielsweise weil die eingebetteten Systeme nur mit einer Batterie betrieben werden und mehrere Jahre laufen sollen.

Stichwörter und Webseiten

- Endlicher Automat: `https://de.wikipedia.org/wiki/Endlicher_Automat`
- Reguläre Grammatik: `https://de.wikipedia.org/wiki/Reguläre_Grammatik`
- Reguläre Sprache: `https://de.wikipedia.org/wiki/Reguläre_Sprache`
- Regulärer Ausdruck: `https://de.wikipedia.org/wiki/Regulärer_Ausdruck`
- Chomsky-Hierarchie `https://de.wikipedia.org/wiki/Chomsky-Hierarchie`

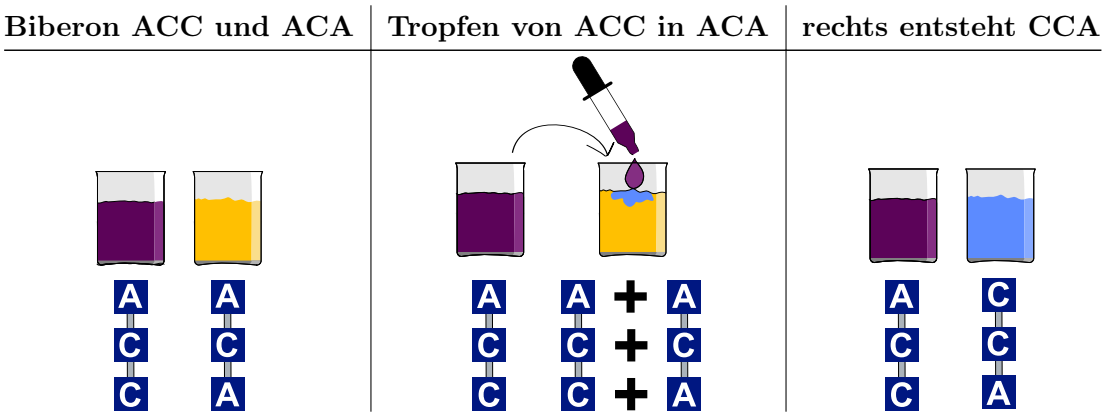
 © Informatik-Biber 2025, SVIA

29. Biberone

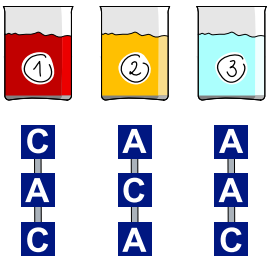
Ein Biberon ist ein besonderer chemischer Stoff: Seine Moleküle sind jeweils eine Folge von genau drei Bausteinen der Typen A und C. Diese Folge wird auch als Name des Biberons verwendet, zum Beispiel: ACA.

Wenn ein Biberon zu einem anderen Biberon hinzugefügt wird, entsteht wieder ein Biberon. Aus den Bausteinen in den Molekülen der beiden Biberone entstehen die Bausteine des neuen Biberons nach diesen Regeln:

Ein Beispiel: Wenn man einen Tropfen von ACC zu ACA hinzufügt, entsteht CCA:

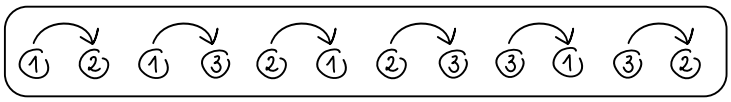


Im Labor gibt es drei Behälter 1, 2 und 3 mit den Biberonen CAC, ACA und AAC:



Mit einer solchen Anweisung kannst du bestimmen, dass ein Tropfen des Biberons in einem Behälter (z.B. Behälter 2) dem Biberon in einem anderen Behälter (z.B. Behälter 3) hinzugefügt wird.

Unten siehst du sechs verschiedene Anweisungen. Verwende möglichst wenige davon, um die Biberone in den Behältern 1 und 3 zu vertauschen: Am Ende soll in Behälter 1 AAC sein, in Behälter 2 bleibt ACA, und in Behälter 3 soll CAC sein.

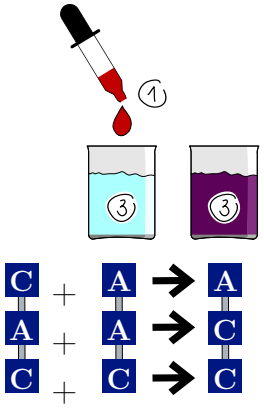


Lösung

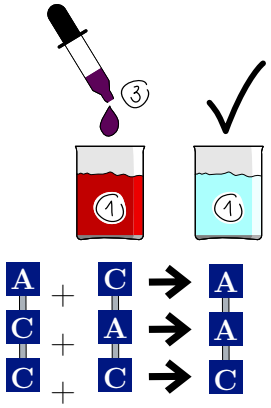
So ist es richtig:

Die Biberone in den Behältern 1 und 3 können mit drei Anweisungen vertauscht werden. Es gibt zwei richtige Antworten:

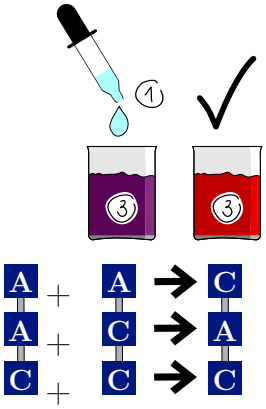
Wir veranschaulichen die erste Antwort 1→3, 3→1, 1→3. Zu Beginn sind die Stoffe in den Behältern: 1: CAC, 2: ACA, 3: AAC.



1→3: In Behälter 3 entsteht CAC + AAC → ACC. Die Stoffe in den Behältern sind nun: 1: CAC, 2: ACA, 3: ACC.



3→1: In Behälter 1 entsteht ACC + CAC → AAC. Die Stoffe in den Behältern sind nun: 1: AAC, 2: ACA, 3: ACC.



1→3: In Behälter 3 entsteht AAC + ACC → CAC. Die Stoffe in den Behältern sind nun: 1: AAC, 2: ACA, 3: CAC. Das ist die gewünschte Reihenfolge der Stoffe.

Nun zeigen wir, dass es nicht möglich ist, die Biberone in den Behältern 1 und 3 mit weniger als drei Anweisungen zu vertauschen. Eine einzige Anweisung reicht offensichtlich nicht aus: Damit kann man nur einen Stoff ändern, und wir müssen zwei Stoffe ändern.

Falls zwei Anweisungen ausreichen, muss durch die eine Anweisung in Behälter 1 das Biberon AAC und durch die andere Anweisung in Behälter 3 das Biberon CAC entstehen.

Betrachten wir zunächst den Fall, dass zuerst in Behälter 1 das Biberon AAC entstehen soll. Die beiden möglichen Anweisungen wirken so:

- 2 → 1: ACA + CAC → AAA
- 3 → 1: AAC + CAC → ACC

Es ist also nicht möglich, dass in Behälter 1 durch eine Anweisung AAC entsteht. Weil die zweite Anweisung nicht zwei Stoffe ändern kann, reichen in diesem Fall zwei Anweisungen nicht aus.

 © Informatik-Biber 2025, SVIA

Betrachten wir nun den Fall, dass zuerst in Behälter 3 das Biberon CAC entstehen soll. Die beiden möglichen Anweisungen wirken so:

- 1 → 3: CAC + AAC → ACC
- 2 → 3: ACA + AAC → CAA

Es ist also nicht möglich, dass in Behälter 3 durch eine Anweisung CAC entsteht. Auch in diesem Fall reichen zwei Anweisungen nicht aus. Es werden also mindestens drei Anweisungen benötigt.

Dies ist Informatik!

Die Regeln, nach der aus den Bausteinen A und C der Baustein im neuen Molekül entsteht, entsprechen der logischen Operation XOR (engl.: exclusive OR, deutsch: exklusives Oder). Das kann man erkennen, wenn man A und C als die Wahrheitswerte «wahr» bzw. «falsch» auffasst: XOR liefert genau dann «wahr», wenn seine beiden Operanden *unterschiedliche* Wahrheitswerte haben.

XOR hat in der Informatik eine besondere Bedeutung. Die «Swapping»-Eigenschaften dieser Operation können genutzt werden, um die Werte zweier Variablen auszutauschen, ohne dass eine dritte, temporäre Variable erforderlich ist - wie bei den Behältern in dieser Biberaufgabe.

XOR ist auch eine wichtige Operation in der *Kryptografie*. Stelle dir vor, du hast eine Nachricht, die als Binärcode dargestellt ist. Du kannst sie verschlüsseln, indem du sie per XOR mit einem Schlüssel gleicher Länge verknüpfst. Ein Beispiel: 01011 (Nachricht) XOR 11001 (Schlüssel) = 10010 (Chiffretext). Man kann die Nachricht wiederherstellen, indem man den Chiffretext mit dem Schlüssel per XOR verknüpft: 10010 (Chiffretext) XOR 11001 (Schlüssel) = 01011 (Nachricht) Ohne den Schlüssel bleibt die Nachricht aber vollständig verborgen, da jedes Bit des Chiffretexts gleichermassen von einer 0 oder 1 in der Nachricht abgeleitet werden kann. Mit XOR arbeitet zum Beispiel die *Stromverschlüsselung*, die sich unter anderem für den Einsatz im Mobilfunk eignet.

Stichwörter und Webseiten

- XOR-Operation:
  - `https://de.wikipedia.org/wiki/Exklusiv-Oder-Gatter`
  - `https://de.wikipedia.org/wiki/Kontravalenz`
- XOR swap algorithm: `https://en.wikipedia.org/wiki/XOR_swap_algorithm`
- Stromverschlüsselung: `https://de.wikipedia.org/wiki/Stromverschlüsselung`

© Informatik-Biber 2025, SVIA

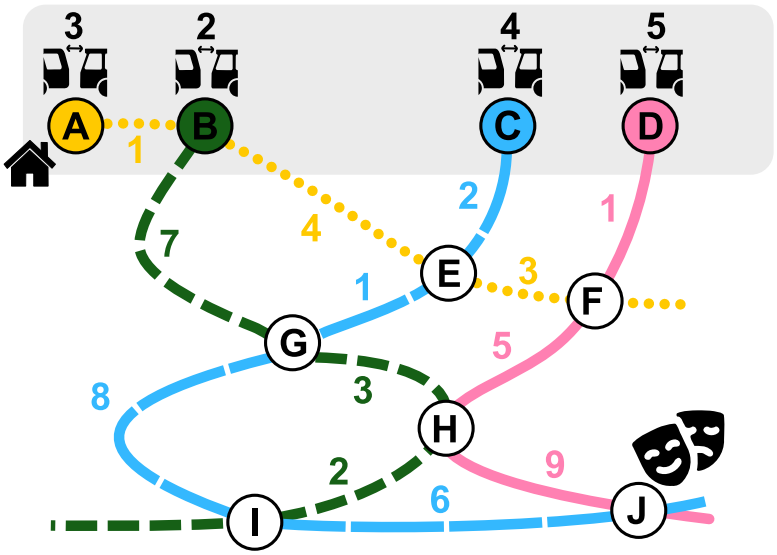
30. ÖV

In Marcus' Stadt gibt es vier Buslinien. Sie starten an den Haltestellen A, B, C und D. Ihre Fahrstrecken kannst du unten in der Karte sehen.

Die ersten Busse jeder Linie fahren zur gleichen Zeit von den Starthaltestellen (A, B, C, D) ab: das ist Minute 0. Danach fahren die Busse in den angegebenen Zeit-Abständen. Von Haltestelle A zum Beispiel fahren Busse alle 3 Minuten, also an den Minuten 0, 3, 6, 9 ...

Für jeden Streckenabschnitt zwischen zwei Haltestellen gibt eine Zahl an, wie viele Minuten ein Bus für den Abschnitt benötigt. Der erste Bus, der von A abfährt, hat also an Minute $0 + 1 + 4 + 3 = 8$ die Haltestelle F erreicht.

Marcus wohnt bei Haltestelle A. Von dort aus möchte er mit dem ersten Bus zum Theater fahren. An Haltestellen, an denen sich Buslinien kreuzen, kann er innerhalb von 0 Minuten umsteigen. Er kann also mit jedem Bus weiterfahren, der zur gleichen Zeit oder später wie er an der Umsteige-Haltestelle ankommt. Marcus weiss, auf welcher Route er fahren und umsteigen muss, damit er so früh wie möglich beim Theater ist.



Welche Haltestellen liegen auf dieser Route?

Lösung

So ist es richtig: Die Route führt über die Haltestellen A, B, E, G, H und J. Die früheste Zeit, zu der Marcus das Theater erreichen kann, ist Minute 20.

Wie kommt man auf diese früheste Zeit und damit auf die Route? Die früheste Zeit, zu der Marcus Haltestelle J und damit das Theater erreicht ist das Minimum von

- der frühesten Zeit, zu der er Haltestelle H erreicht, plus Wartezeit für den von D aus kommenden Bus plus 9 Minuten, und
- der frühesten Zeit, zu der er Haltestelle I erreicht, plus Wartezeit für den von C aus kommenden Bus plus 6 Minuten.

Nun kann man weiter rückwärts diese beiden frühesten Zeiten bestimmen. Alternativ kann man von Marcus' Starthaltestelle A aus für alle erreichbaren Haltestellen diese frühesten Zeiten ermitteln – und sich dabei jeweils die vorherige Haltestelle für diese früheste Zeit merken. Dann weiss man am Ende die früheste Zeit, zu der Marcus das Theater erreicht, und kann die auf der Route liegenden Haltestellen angeben.

- Haltestelle **A**: Marcus erreicht A an Minute **0**.
- Haltestelle **B**: Marcus erreicht B an Minute 1 (wir sagen kurz: **um 1**), **von A aus**.
- Die Haltestellen C und D kommen für Marcus' Route nicht in Frage.
- Haltestelle **E**: Marcus erreicht E nur **von B aus**, ohne Wartezeit bei B, **um 5**.
- Haltestelle **F**: Marcus erreicht F (ohne Umwege) nur **von E aus**, ohne Wartezeit bei E, **um 8**.
- Haltestelle **G**: Marcus kann G auf zwei Wegen erreichen, nämlich von B oder E aus. (1) In B ist Marcus um 1, kann um 2 dort losfahren und ist um 9 bei G. (2) In E ist Marcus um 5, kann um 6 dort losfahren (der Bus von C aus fährt um 2, 6, 10, … bei E ab) und ist um 7 bei G. Die früheste Zeit, zu der Marcus G erreichen kann, ist also **7, von E aus**.
- Haltestelle **H**: Marcus kann H von F oder G aus erreichen. (1) In F ist er um 8, kann um 11 dort losfahren (der Bus von D aus fährt um 6, 11, 16, … bei F ab) und ist um 16 bei H. (2) In G ist er um 7, kann direkt losfahren (der Bus von B aus fährt um 7, 9, 11, … bei G ab) und ist um 10 bei H. Die früheste Zeit, zu der Marcus H erreichen kann, ist also **10, von G aus**.
- Haltestelle **I**: Marcus kann I von H oder G aus erreichen. (1) In H ist Marcus um 10, kann direkt losfahren (der Bus von B aus fährt um 10, 12, 14, … bei H ab) und ist um 12 bei I. (2) In G ist Marcus um 7, kann direkt dort losfahren (der Bus von C aus fährt um 3, 7, 11, … bei G ab) und ist um 15 bei I. Die früheste Zeit, zu der Marcus I erreichen kann, ist also **12, von H aus**.
- Haltestelle **J / Theater**: Marcus kann J von H oder I aus erreichen. (1) In H ist Marcus um 10, kann um 11 dort losfahren (der Bus von D aus fährt um 6, 11, 16, … bei H ab) und ist um 20 bei J. (2) In I ist Marcus um 12, kann um 15 dort losfahren (der Bus von C aus führt um 11, 15, 19, … bei I ab) und ist um 21 bei J. Die früheste Zeit, zu der Marcus J erreichen kann, ist also **20, von H aus**.

Da wir uns jeweils gemerkt haben, von wo aus Marcus eine Haltestelle zur frühesten Zeit erreicht, können wir seine Route rückwärts verfolgen: J ← H ← G ← E ← B ← A.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Die entscheide Idee bei der Ermittlung der richtigen Antwort war, das ursprüngliche Problem (nämlich: früheste Ankunft bei J) auf zwei kleinere Probleme (früheste Ankünfte bei I und H) zurückzuführen. Diese Methode hätten wir weiterführen können: Die früheste Ankunft bei I ergibt sich wiederum aus den frühesten Ankünften bei G und H, und so weiter. Diese Strategie, eine Berechnungsfunktion (früheste Ankunft) auf sich selbst zurückzuführen, kennt die Informatik als *Rekursion*.

Bei der Erklärung der richtigen Antwort haben wir aber die Rekursion aufgelöst und die Ergebnisse der Funktion vom einfachsten Fall aus nach und nach (in der Informatik sagt man: *iterativ*) aufgebaut. So konnten wir letztlich nicht nur die Berechnung für J auf die Berechnungen für I und H zurückführen, sondern das Ergebnis für J auf die bereits bekannten Ergebnisse für I und H.

Diese Strategie heisst in der Informatik *Dynamische Programmierung*. Man kann sie für Optimierungsprobleme nutzen, wie für die Suche nach Marcus' frühester Ankunftszeit beim Theater in dieser Biberaufgabe. Dynamische Programmierung funktioniert genau dann, wenn das *Optimalitätsprinzip von Bellman* gilt, wenn man also die optimale Lösung eines Problems aus optimalen Lösungen für Teilprobleme zusammensetzen kann. Sie funktioniert dann häufig gut, weil man beim iterativen Aufbau der (Teil-)Lösungen jede (Teil-)Lösung nur einmal berechnen muss.

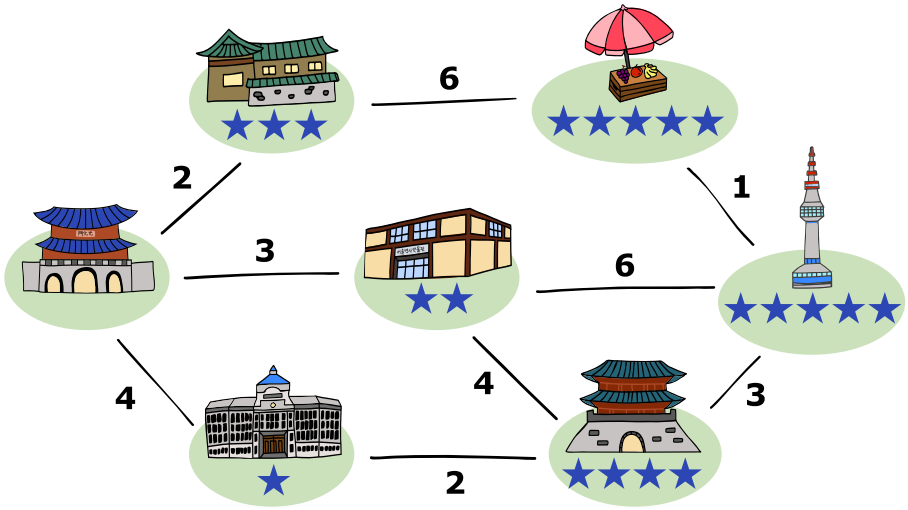
Stichwörter und Webseiten

- Dynamische Programmierung:
  `https://de.wikipedia.org/wiki/Dynamische_Programmierung`
- Optimalitätsprinzip von Ballman:
  `https://de.wikipedia.org/wiki/Optimalitätsprinzip_von_Bellman`
- Rekursion: `https://de.wikipedia.org/wiki/Rekursion`
- Iteration: `https://de.wikipedia.org/wiki/Iteration#Informatik`

© Informatik-Biber 2025, SVIA

31. Seoul entdecken!

In Seoul in Korea gibt es Busse für Touristen, die sehenswerte Orte miteinander verbinden. Das Bild zeigt die wichtigsten Orte von Seoul. Die Sterne sagen, wie beliebt die Orte sind. Die Linien zeigen die Busverbindungen. An jeder Linie steht, wie viele Kilometer lang die Verbindung ist.



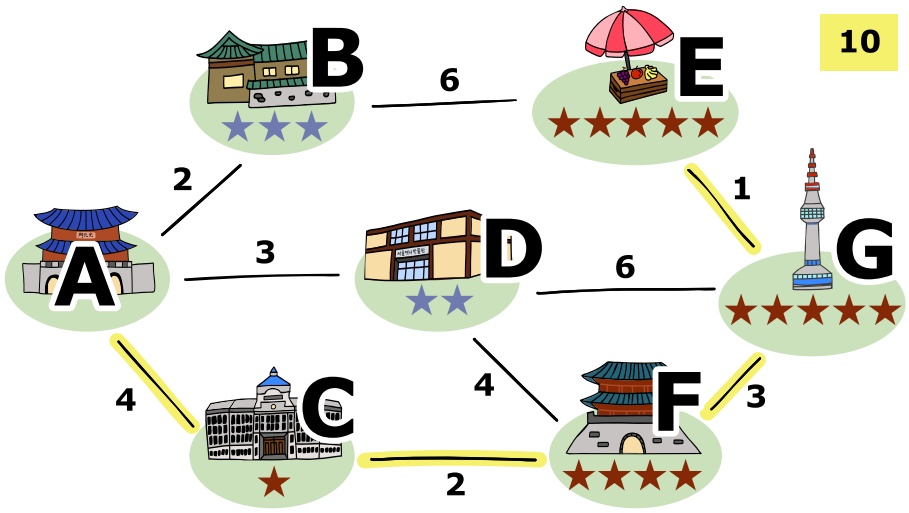
Lotte besucht zuerst den Palast  . Von dort aus möchte sie mit den Bussen weitere Orte besuchen. Lotte hat eine Fahrkarte, mit der sie höchstens 10 Kilometer weit fahren kann. Damit möchte sie über die Verbindungen Orte erreichen, die insgesamt möglichst viele Sterne haben! Sie besucht einen Ort natürlich nur einmal und muss nicht zum Palast zurück.

Welche Verbindungen muss Lotte mit ihrer Fahrkarte fahren, um möglichst viele Sterne zu sammeln?

Lösung

Wir wollen für Lotte eine Busroute finden, die vom Palast ausgeht, höchstens 10 km lang ist und mit der sie Orte erreicht, die insgesamt möglichst viele Sterne haben. Deshalb sollten wir bei der Beschreibung von Routen nicht nur die einzelnen Orte, sondern auch die Entfernung vom Palast und die Anzahl der auf der Route gesammelten Sterne berücksichtigen.

Zunächst bezeichnen wir die einzelnen Orte mit den Buchstaben A bis G.



Einen Zwischenstopp auf einer Route bei einem Ort können wir nun beschreiben durch:

- den Buchstaben der Orte,
- die Gesamtentfernung vom Start (A) zu dieses Ortes und
- die Gesamtzahl der auf dem Weg vom Start zu diesem Ort gesammelten Sterne.

Eine Route nennen wir *gültig*, wenn sie höchstens 10 km lang ist. Eine gültige Route führt beispielsweise

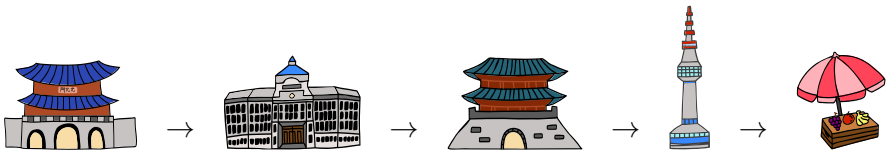
- von A nach B: B ist 2 km von A entfernt und hat 3 Sterne, so dass dieser Zwischenstopp mit B(2,3) bezeichnet wird;
- weiter von B nach E: die Gesamtentfernung wächst um 6 km auf 8 km, die Gesamtzahl der gesammelten Sterne steigt um 5 auf 8, so dass dieser Zwischenstopp mit E(8,8) bezeichnet wird;
- zuletzt von E nach G: es kommen 1 km Entfernung und 5 Sterne hinzu, so dass dieser Zwischenstopp, der gleichzeitig das Ende dieser gültigen Route ist, mit G(9,13) bezeichnet wird.

Das sind alle gültigen Routen:

```
A → B(2,3) → E(8,8) → G(9,13)
A → D(3,2) → G(9,7) → E(10,12)
A → D(3,2) → F(7,6) → G(10,11)
A → D(3,2) → F(7,6) → C(9,7)
A → C(4,1) → F(6,5) → D(10,7)
A → C(4,1) → F(6,5) → G(9,10) → E(10,15)
```

 © Informatik-Biber 2025, SVIA

Die letzte gültige Route ist die beste, da ihr Endpunkt E(10,15) die höchste Gesamtzahl an Sternen aufweist. Lotte muss mit ihrer Fahrkarte also diese Verbindungen fahren, um möglichst viele Sterne zu sammeln:



Dies ist Informatik!

Lotte hat in dieser Biberaufgabe ein Ziel: Sie möchte auf ihrer Route möglichst viele Sterne sammeln. Sie will also einen bestimmten Wert, nämlich die Gesamtzahl der Sterne, *optimieren*, d.h. den grösstmöglichen Wert finden. Sprich: Lotte hat ein *Optimierungsproblem*. Beim Lösen dieses Problems ist sie durch ihre Fahrkarte eingeschränkt, die die Länge der Route begrenzt.

Die Informatik kennt viele Verfahren zur Lösung von Optimierungsproblemen, ob mit oder ohne Einschränkungen. Solche Probleme werden also häufig mit Hilfe von Informatiksystemen gelöst. Bei der Bestimmung optimaler Routen helfen Navigationssysteme. Sie erlauben ihren Benutzerinnen und Benutzern meist, den zu optimierenden Wert zu verändern: Soll die Route eine möglichst kurze Strecke haben oder soll möglichst wenig Energie verbraucht werden? Auch Einschränkungen sind möglich; zum Beispiel können Autobahnen, kostenpflichtige Strassen oder Fährverbindungen vermieden werden.

Informatik-Verfahren zur Routenfindung verwenden Datenstrukturen, die ganz ähnlich gezeichnet werden können wie das Busnetz in dieser Biberaufgabe, nämlich *Graphen*. Diese bestehen aus einer Menge von *Knoten* und einer Menge von Knotenpaaren, den *Kanten*. Mit Graphen lassen sich Beziehungen zwischen Dingen sehr gut modellieren; zum Beispiel Verkehrsverbindungen zwischen Orten. Entfernungen können dann als *Gewichte* mit den Kanten verbunden werden. Die Informatik kennt viele Algorithmen zur Lösung von Problemen, deren Daten als Graphen verwaltet werden – zum Beispiel die *Tiefensuche*, mit der wir oben die verschiedenen Routen für Lotte ermittelt haben.

Stichwörter und Webseiten

- *Optimierungsproblem*: `https://de.wikipedia.org/wiki/Optimierungsproblem`
- *Graphentheorie*: `https://de.wikipedia.org/wiki/Graphentheorie`
- *Tiefensuche*: `https://de.wikipedia.org/wiki/Tiefensuche`

 © Informatik-Biber 2025, SVIA

32. Bergseen

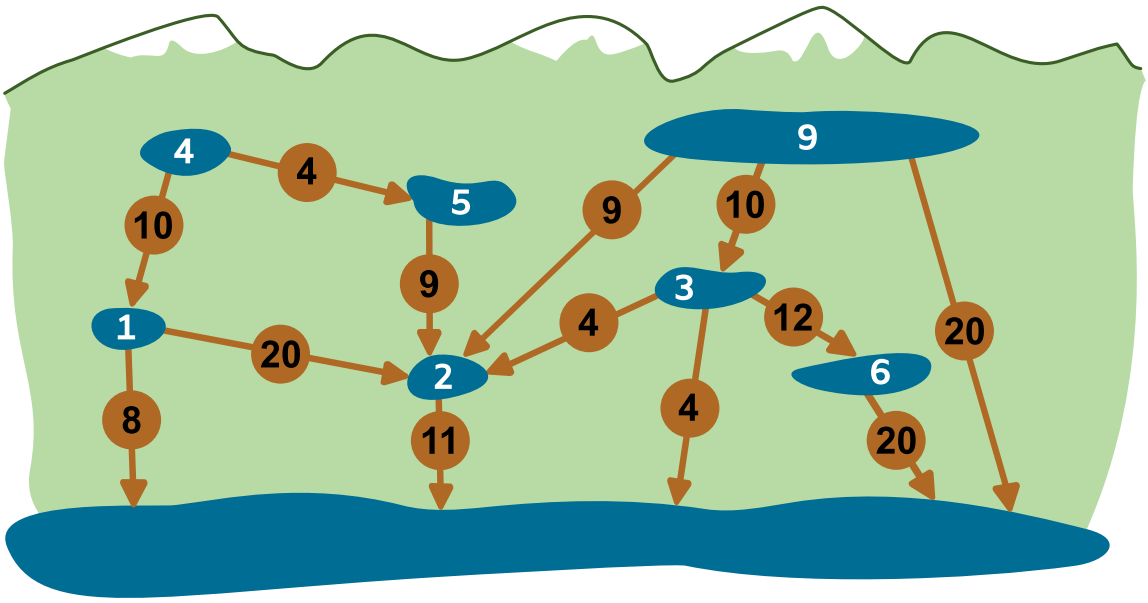
Am Bergmassiv hinter dem Stausee gibt es mehrere kleine Bergseen. Bei starkem Regen könnten sie überlaufen, und das ist gefährlich. Deshalb sollen zwischen einigen Seen Kanäle gebaut werden. Diese Kanäle sollen alles überschüssige Wasser aus den Bergseen in den Stausee ableiten können. Gleichzeitig soll ihr Bau möglichst wenig kosten.

Für jeden Bergsee gibt eine Zahl an, wieviel überschüssiges Wasser aus dem See abgeleitet werden muss.

An jeder Stelle zwischen zwei Seen, an der ein Kanal gebaut werden kann, ist ein Pfeil. Er zeigt, in welche Richtung ein Kanal das Wasser dort ableiten würde. Die Zahl an einem Pfeil gibt die Kapazität des Kanals an, also wieviel überschüssiges Wasser er ableiten kann. Die Kapazität bestimmt auch die Kosten für den Bau eines Kanals an dieser Stelle.

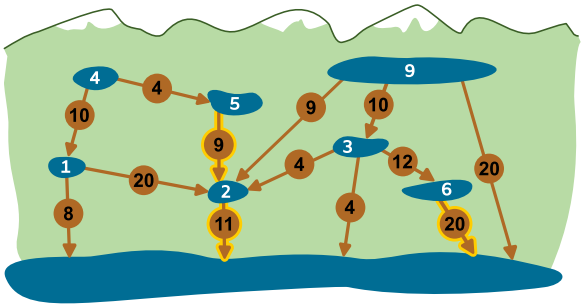
Beachte: Wenn ein Kanal Wasser von einem kleinen Bergsee in einen zweiten ableitet, sammelt sich im zweiten See das überschüssige Wasser aus beiden Seen.

An welchen Stellen sollen Kanäle gebaut werden?

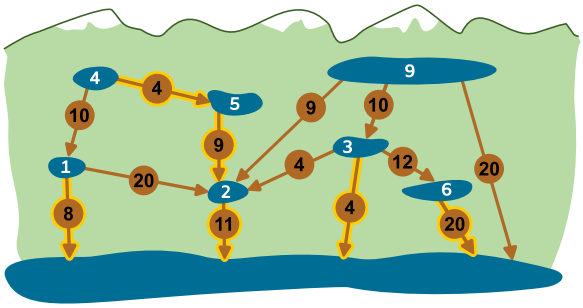


Lösung

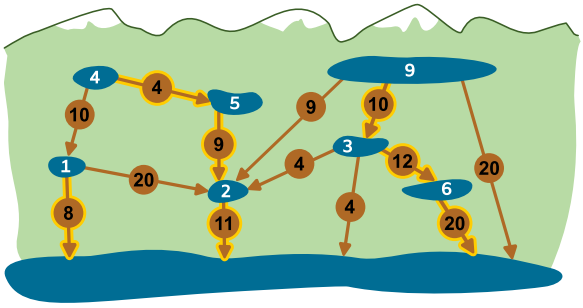
So ist es richtig:



Für einige Seen gibt es nur eine Stelle, an der ein Kanal Wasser ableiten kann. Das sind die Seen mit Wasserüberschuss 2 unten Mitte (kurz: See 2), 5 und 6. An diesen Stellen *muss* also ein Kanal gebaut werden. Die Kapazitäten dieser Kanäle genügen für diese drei Seen, auch wenn man bedenkt, dass aus See 2 durch den Zufluss aus See 5 ein Gesamtüberschuss von 7 abgeleitet werden muss. Der Bau dieser drei Kanäle kostet 11 + 9 + 20 = 40.



Für die Seen 1 und 4 gibt es jeweils zwei Kanalbau-Stellen. (a) Wenn der Kanal von 4 nach 5 gebaut wird, muss der Kanal von 2 in den grossen See einen Überschuss von 4 + 5 + 2 = 11 ableiten. Damit ist seine Kapazität erschöpft, so dass See 1 nicht auch noch in See 2, sondern in den Stausee abgeleitet werden muss. Die Kosten sind dann insgesamt 4 + 8 = 12. (b) Alternativ kann der Kanal von 4 nach 1 gebaut werden. Wenn dann der Kanal von 1 nach 2 gebaut würde, müssten aus See 2 insgesamt 4 + 1 + 7 = 12 abgeleitet werden, was nicht möglich ist. Also muss in diesem Fall der Kanal von 1 in den Stausee gebaut werden. Die Kosten sind dann insgesamt 10 + 8 = 18, also höher.



Bleiben die Seen 3 und 9. (a) See 9 kann nicht in See 2 abgeleitet werden, weil das die Kapazität des Kanals von 2 in den Stausee übersteigt (übrigens auch, wenn es den Kanal von 4 nach 5 nicht gäbe). (b) Wird See 9 direkt in den Stausee abgeleitet, hätte die günstigste Gesamtlösung für die Seen 3 und 9 die Kosten 20 + 4 = 24. (c) Wird See 9 in See 3 abgeleitet, entsteht dort ein Überschuss von 12, der nur in See 6 abgeleitet werden kann. Der dort entstehende Überschuss 18 kann durch den bereits gebauten Kanal in den Stausee abgeleitet werden. Die Kosten für die Seen 3 und 9 sind dann 10 + 12 = 22, also günstiger.

Die gewählten Kanäle können alles überschüssige Wasser aus den Bergseen in den Stausee ableiten, und das zu minimalen Kosten von 40 + 12 + 22 = 74.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Seen (Bergseen und Stausee) und Kanalbaustellen können als *Graph* modelliert werden. Ein Graph hat *Knoten* (hier: die Seen), von denen jeweils 2 durch Kanten (hier: die Kanalbaustellen) miteinander verbunden sein können. Wie in dieser Biberaufgabe können Kanten eine Richtung haben und ausserdem ein *Gewicht* wie hier die potenzielle Kanal-Kapazität an den Baustellen.

Ein Problem mit Hilfe bekannter Strukturen zu modellieren, ist besonders sinnvoll, wenn man auch Algorithmen zur Lösung heranziehen kann, die die Informatik für Probleme auf diesen Strukturen kennt. Für Graphen sind viele Probleme gut beschrieben und für viele davon auch effiziente Lösungsalgorithmen bekannt. Dazu gehören auch Flussprobleme, wie etwa das «minimum cost flow problem», die mit dem Problem in dieser Aufgabe verwandt sind.

Stichwörter und Webseiten

- Graph: `https://de.wikipedia.org/wiki/Graph_(Graphentheorie)`
- Gerichteter Graph: `https://de.wikipedia.org/wiki/Graph_-(Graphentheorie)#Gerichteter_Graph_(Digraph)`
- Gewichteter Graph: `https://de.wikipedia.org/wiki/Graph_(Graphentheorie)#Gewichtete_Graphen`
- Graphenprobleme: `https://de.wikipedia.org/wiki/Graphentheorie#Probleme`
- Flussprobleme: `https://de.wikipedia.org/wiki/Flüsse_und_Schnitte_in_Netzwerken`
- Lokale Optimierung: `https://de.wikipedia.org/wiki/Lokale_Suche`

© Informatik-Biber 2025, SVIA

33. Parkplätze

Zur Party kommen 9 Gäste mit ihren Autos. Vor dem Haus können 9 Autos so parkieren, dass in 3 Parkspuren jeweils 3 Autos hintereinander stehen. Die Gäste kommen in dieser Reihenfolge:

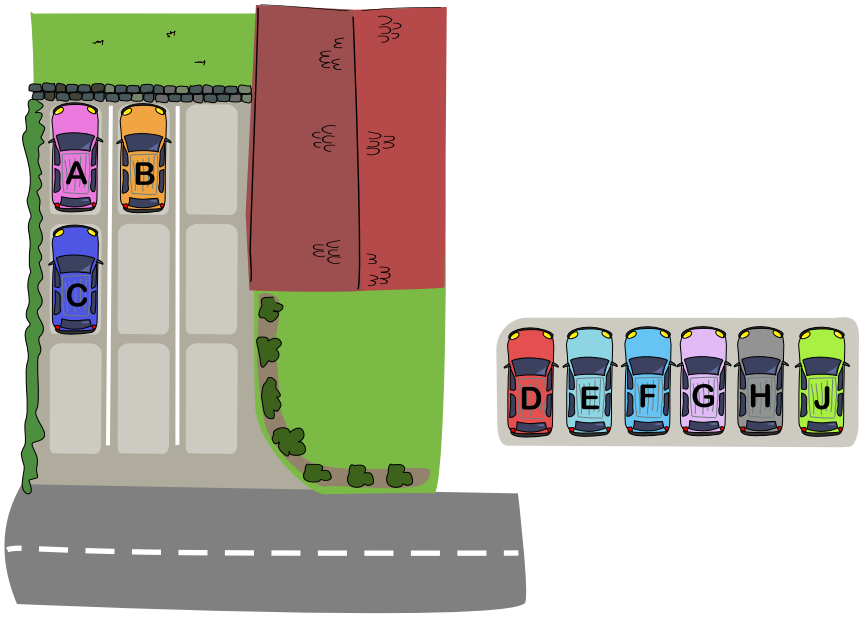
Anja, **B**eate, **C**lara, **D**avid, **E**lia, **F**rank, **G**abi, **H**arald und zuletzt **J**ulia.

Beim Einparkieren wählt jeder eine Parkspur aus und fährt darin so weit wie möglich nach vorne.

Die Gäste wollen in dieser Reihenfolge von der Party wegfahren:

Gabi, **D**avid, **B**eate, **E**lia, **J**ulia, **C**lara, **H**arald, **A**nja und zuletzt **F**rank.

Die Autos von **A**nja, **B**eate und **C**lara sind bereits parkiert. Nun parkieren die anderen Gäste nach und nach ein. Sie wollen so parkieren, dass beim Wegfahren kein Auto von einem anderen blockiert ist, das später wegfährt.

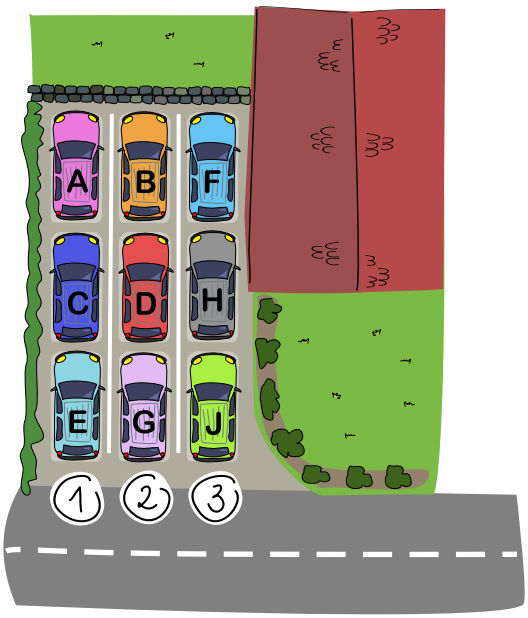


Zeige den Gästen, wie sie so parkieren können!

Platziere die restlichen 6 Autos in den Parkspuren. Du musst die Reihenfolgen beim Ankommen ***und*** *beim Wegfahren berücksichtigen.*

Lösung

So ist es richtig:



Wir beobachten zunächst:

a) **F**rank fährt zuletzt weg, muss daher ganz vorne parkieren.

b) **G**abi fährt zuerst weg, muss daher in einer der 3 Spuren ganz hinten parkieren.

c) **B**eate fährt als Dritte weg, daher müssen **D**avid und **G**abi hinter ihr parkieren.

Damit können wir weitere Überlegungen anstellen:

- Wegen a) bleibt für **F**rank nur der Platz neben **B**eate, vorne in Parkspur 3.
- Nach **A**nna, **B**eate und **C**lara kommt **D**avid an. Wegen c) muss **D**avid hinter **B**eate in Parkspur 2 parkieren.
- **G**abi muss wegen c) hinter **D**avid in Parkspur 2 parkieren.
- Für **E**mil bleibt nur Parkspur 1 hinter **C**lara. **E**mil fährt als Vierter (nach **G**abi, **D**avid, **B**eate) ab und muss deswegen ganz hinten in einer Spur stehen. Wenn er ankommt, kommt dafür nur Parkspur 1 in Frage, denn in Parkspur 3 steht bisher nur **F**rank, so dass **E**mil dort auf den mittleren Platz müsste.
- Schliesslich bleibt nur Parkspur 3 für **H**arald (mittlerer Platz) und **J**ulia (hinten) übrig. Zum Glück will **J**ulia vor **H**arald wegfahren - sonst gäbe es keine richtige Antwort.

Weil für jedes einzelne Auto genau eine feste Parkposition in Frage kommt, gibt es nur die eine richtige Antwort.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Die parkierten Autos in den 3 Parkspuren verhalten sich so, dass nur das zuletzt parkierte Auto wegfahren kann. Das ist so wie bei einem Stapel Teller, bei dem man nur den zuletzt auf den Stapel gelegten Teller gefahrlos wegnehmen kann.

Auch die Informatik kennt einen *Stapel* (engl.: stack), und zwar als Datenstruktur. Sie funktioniert analog zu Parkspuren oder Tellerstapeln: Die Operation *push* fügt ein Daten-Objekt dem Stapel hinzu. Die Funktion *top* liefert das zuletzt hinzugefügte Objekt, und die Operation *pop* entfernt es aus dem Stapel. In der theoretischen Informatik wiederum gibt es Berechnungsmodelle, die Stapel verwenden. Ein Automat mit einem Stapel (in der theoretischen Informatik auch *Keller* oder *Kellerspeicher* genannt) entspricht den sogenannten *kontextfreien Sprachen*, die von Computern gut verarbeitet werden können; Beispiele dafür sind Programmiersprachen oder Markup-Sprachen wie HTML.

Mehrere Stapel – so wie die mehreren Parkspuren in dieser Biberaufgabe – werden zum Beispiel in Computer-Betriebssystemen verwendet, um Aufgaben an mehrere Prozessoren zu verteilen. Wenn man dem Stapel-Automaten aus der theoretischen Informatik mindestens einen weiteren Stapel hinzufügt, kann man damit beliebige Berechnungen modellieren, so wie mit einer Turingmaschine. Der zweite Stapel macht den Unterschied!

Stichwörter und Webseiten

- Stapel (engl. stack): `https://de.wikipedia.org/wiki/Stapelspeicher`
- Partitionsproblem: `https://de.wikipedia.org/wiki/Partitionsproblem`
- Mehrprozessorsystem: `https://de.wikipedia.org/wiki/Mehrprozessorsystem`
- Kellerautomat: `https://de.wikipedia.org/wiki/Kellerautomat`

 © Informatik-Biber 2025, SVIA

34. Biber Jones

Biber Jones  ist in einer perfiden Pyramide mit gefährlichen Gängen. Am Ende jedes Ganges befindet sich ein sagenhafter Schatz. Jones will jeden Schatz *so schnell wie möglich* erreichen.

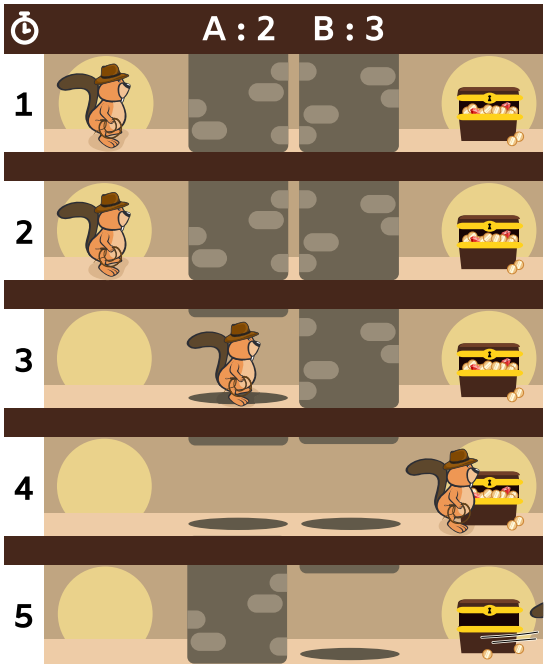
Jedoch ist jeder Gang durch eine Reihe von Fallblöcken gesichert. Am Anfang sind alle Blöcke unten. Sobald jemand den Gang betritt, beginnen die Blöcke sich zu bewegen. Jeder Block bewegt sich in festem Takt nach oben und unten. Zum Beispiel schnellt ein Block mit Takt 2 nach 2 Minuten hoch und kracht nach weiteren 2 Minuten wieder herunter.

Jones steht vor seiner ersten Herausforderung:

Dieser Gang hat zwei Blöcke: Block A hat Takt 2, Block B Takt 3. Glücklicherweise hat Jones eine Schriftrolle mit Anweisungen gefunden, wie man so schnell wie möglich sicher den Schatz erreicht:

```
wait(2)
goto_block(A)
wait(1)
goto_treasure
```

Jones befolgt die Anweisungen: Er wartet 2 Minuten, geht dann zu Block A, wartet dort 1 Minute und geht dann zum Schatz. Er erreicht den Schatz nach 3 Minuten:

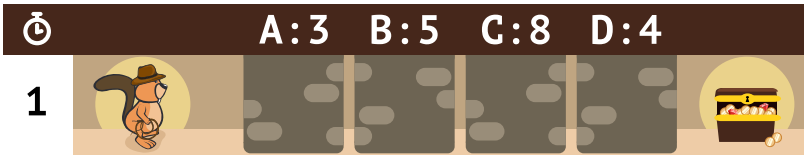


Jones bemerkt, dass er auch mit weniger Anweisungen dem Schatz erreicht hätte, und zwar gleich schnell:

```
wait(3)
goto_treasure
```

Er kommt zum nächsten Gang. Der hat vier Blöcke, mit Takt 3, 5, 8 und 4.

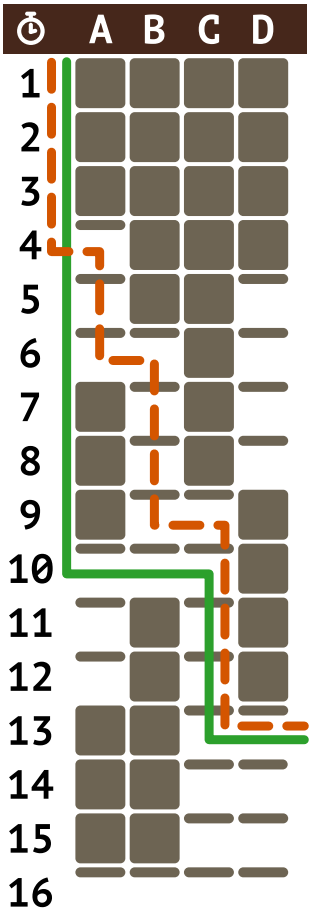
Was ist die kürzeste Folge von Anweisungen, mit denen Jones so schnell wie möglich den Schatz erreicht?



 © Informatik-Biber 2025, SVIA

Lösung

Das Bild zeigt, wie sich Jones durch den Gang bewegen kann:



Um sich immer so schnell wie möglich weiterzubewegen, kann er den folgenden Anweisungen folgen und den Schatz nach 12 Minuten (siehe orange Linie) erreichen:

```
wait(3)
goto_block(A)
wait(2)
goto_block(B)
wait(3)
goto_block(C)
wait(4)
goto_treasure
```

Es gibt aber eine kürzere Folge von Anweisungen, mit denen er ebenfalls nach 12 Minuten den Schatz erreicht (siehe grüne Linie):

```
wait(9)
goto_block(C)
wait(3)
goto_treasure
```

Es gibt keine Möglichkeit, mit noch weniger Anweisungen den Schatz zu erreichen, ohne Zeit zu verlieren. Die einzige Möglichkeit bestünde darin, dass Jones den Gang in einem Zug passiert. Dazu muss er aber 15 Minuten warten (erst dann sind alle Blöcke gleichzeitig oben) und wäre später beim Schatz.

Dies ist Informatik!

Biber Jones, der atemlose Abenteurer, will natürlich so schnell wie möglich den sagenhaften Schatz erreichen. Nicht dass ihm jemand zuvorkommt! Aber als ordentlich organisierter Ober-Optimierer ist Jones nicht nur mit irgendwelchen Instruktionen zufrieden, die ihn flugs voranbringen. Nein, er sucht die kürzeste Anweisungsfolge, die das leistet. Er optimiert also mit gleich zwei Zielen bzw. fügt der Optimierung der Geschwindigkeit eine Nebenbedingung hinzu. Kein Problem für Jones!

Etwas schwieriger wäre, wenn Jones sich Optimierungsziele ausgesucht hätte, die nicht unbedingt miteinander verträglich sind: einerseits so schnell wie möglich beim Schatz zu sein, anderseits sein Hollywood-reifes Abenteurer-Outfit möglichst in Form zu halten. In Informatik und Mathematik spricht man in solchen Fällen von *Mehrzieloptimierung*. Dabei gibt es meist kein einziges Optimum, sondern mehrere *Pareto-Optima*. Ein Pareto-Optimum (benannt nach dem Ökonomen Vilfredo Pareto) ist ein Zustand, in dem es nicht möglich ist, einen Zielwert zu verbessern, ohne zugleich einen anderen zu verschlechtern.

Wie allen guten Informatikerinnen und Informatikern geht es Jones sehr um Qualität. Bei diesem Stichwort geht es in der Informatik nicht immer um die Qualität bzw. Optimalität von Berechnungsergebnissen. Es gibt zum Beispiel auch die Qualität von Programmcode, die sich an verschiedenen Kriterien wie der Strukturiertheit des Codes oder den enthaltenen Kommentaren bemessen kann. Auch kann unter verschiedenen Codes mit gleicher Funktion derjenige mit den wenigsten Anweisungen bevorzugt werden – so wie in dieser Biberaufgabe.

Stichwörter und Webseiten

- Optimierungsproblem: `https://de.wikipedia.org/wiki/Optimierungsproblem`
- Mehrzieloptimierung: `https://de.wikipedia.org/wiki/Mehrzieloptimierung`

 © Informatik-Biber 2025, SVIA

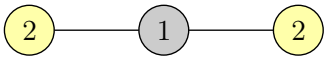
35. Prüfungsplan

Am Euler-Gymnasium stehen die Matura-Prüfungen an. Sie sollen an höchstens 5 verschiedenen Tagen geschrieben werden. Weil man nur eine Prüfung pro Tag mitschreiben darf, dürfen 2 Prüfungen, bei denen mindestens eine Person beide mitschreibt, nicht für den gleichen Tag geplant werden. Solche 2 Prüfungen haben einen «Tageskonflikt».

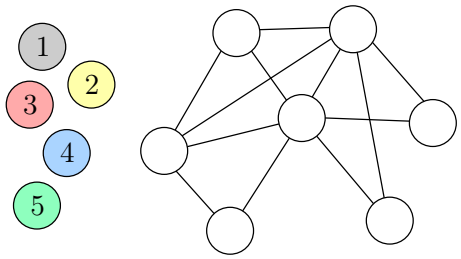
Um die Planung zu erleichtern, werden die Tageskonflikte in einem Diagramm aus Kreisen und Linien dargestellt:

- Für jede Prüfung wird ein Kreis gezeichnet.
- Zwischen 2 Kreisen wird genau dann eine Linie gezeichnet, wenn diese beiden Prüfungen einen Tageskonflikt haben, also nicht für den gleichen Tag geplant werden dürfen.

Hier ist ein Beispiel mit 3 Prüfungen: Die Prüfung in der Mitte hat 2 Tageskonflikte, nämlich mit jeder der beiden anderen Prüfungen. Die Nummern zeigen eine Möglichkeit, die Prüfungen auf 2 Tage (1 und 2) zu verteilen.



Innerhalb der nächsten 5 Tage sollen 7 Prüfungen geschrieben werden. Das Diagramm zeigt ihre Tageskonflikte.

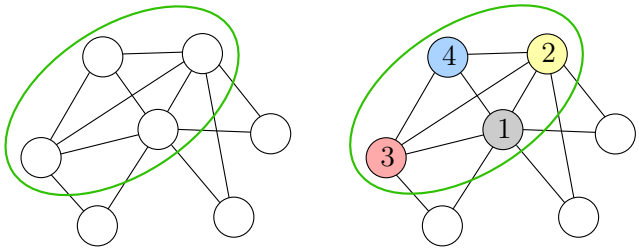


Verteile die Prüfungen auf möglichst wenige der 5 Tage und beachte dabei die Tageskonflikte. Es gibt mehrere richtige Antworten.

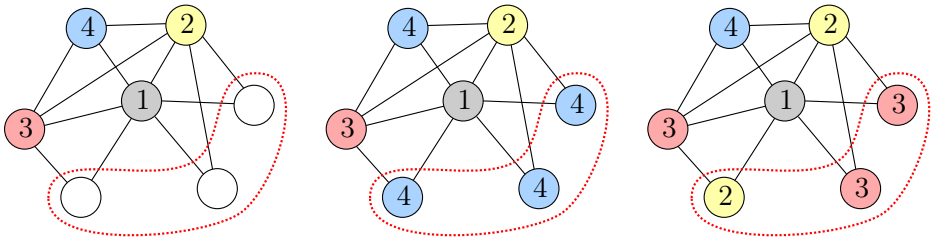
Lösung

So ist es richtig:

Bei genauer Betrachtung des Diagramms erkennen wir eine besondere Konstellation bei den oben links eingezeichneten Prüfungen (Bild links). Jede dieser Prüfungen hat mit jeder anderen einen Tageskonflikt, so dass sie an vier unterschiedlichen Tagen geschrieben werden müssen. Im Bild rechts siehst du eine Möglichkeit, diese Prüfungen auf vier Tage zu verteilen. Jede andere der $1 \times 2 \times 3 \times 4 = 24$ Möglichkeiten, die Prüfungen auf vier Tage zu verteilen, ist genauso richtig.



Für diese Verteilung betrachten wir nun die verbleibenden drei Prüfungen (Bild links). Keine davon kann an Tag 1 geschrieben werden, aber alle an Tag 4 (Bild mitte). Wenn man die Prüfungen an den Tagen 2 oder 3 schreiben möchte, geht das wegen der Tageskonflikte nur so wie im Bild rechts gezeigt. Ausserdem hat man in dieser Verteilung noch sechs Möglichkeiten, eine oder zwei der für die Tage 2 und 3 angesetzten Prüfungen stattdessen an Tag 4 schreiben zu lassen.



Für jede der 24 Möglichkeiten für die vier Klausuren oben links gibt es also 8 Möglichkeiten, die drei Klausuren unten rechts zu verteilen. Es gibt also 192 richtige Antworten mit den Tagen {1,2,3,4} und jeweils noch einmal so viele mit den Tagen {2,3,4,5}, {1,3,4,5}, {1,2,4,5} und {1,2,3,5}. Insgesamt sind das $5 \times 192 = 960$ richtige Antworten.

Dies ist Informatik!

Bei der Verteilung der Prüfungen ist es hilfreich, dass die Tageskonflikte in einem übersichtlichen Diagramm dargestellt sind. Auch wenn solche «Verteilungsprobleme» grösser sind und sie mit Hilfe von Informatiksystemen gelöst werden sollen, ist es wichtig, eine hilfreiche Darstellung bzw. Modellierung der Daten zu nutzen. Das Tageskonflikt-Diagramm in dieser Biberaufgabe ist die bildhafte Darstellung eines *Graphen*, einer in der Informatik besonders wichtigen Struktur zur Modellierung von Relationen, also Beziehungen zwischen Objekten. Ein Graph besteht aus *Knoten* (hier als Kreise angezeigt) und *Kanten*, die je zwei Knoten miteinander verbinden (die Linien zwischen den Kreisen).

 © Informatik-Biber 2025, SVIA

Das «Verteilungsproblem», die Knoten eines Graphen mit möglichst wenigen Werten so zu markieren, dass zwei durch eine Kante verbundene Knoten unterschiedliche Werte haben, kennt die Informatik als *Färbungsproblem* (engl.: graph coloring) - die Markierungswerte kann man sich als unterschiedliche Farben vorstellen. Färbungsprobleme können in vielen Anwendungsbereichen vorkommen, zum Beispiel bei ...

- ... der Färbung von Landkarten, bei denen benachbarte Länder unterschiedliche Farben bekommen sollen;
- ... der Planung von Prüfungen, wie in dieser Biberaufgabe;
- ... der Frequenzplanung in Mobilfunknetzen, wenn Funkstörungen zwischen benachbarten Sendemasten vermieden werden sollen; oder
- ... der Verteilung von Zügen auf die Gleise eines Bahnhofs, da Züge mit überlappenden Aufenthaltszeiten verschiedene Gleise benutzen müssen.

Im Allgemeinen gehören Färbungsprobleme zu den schwierigsten Problemen, die in der Informatik bekannt sind, den sogenannten *NP-schweren Problemen*. Je nach Anwendungsfall ergeben sich aber spezielle Färbungsprobleme, die deutlich leichter zu lösen sind. Und in den wirklich schwierigen Fällen können Verfahren genutzt werden, die nicht garantiert optimale, aber meist sehr gute Lösungen finden.

Stichwörter und Webseiten

- Graphen färben: `https://de.wikipedia.org/wiki/Färbung_(Graphentheorie)`

 © Informatik-Biber 2025, SVIA

36. Prüf-Biber

Der Biber-Boss setzt vier sogenannte *Nachrichten-Biber* ein: Sie halten Flaggen hoch, um Nachrichten zu senden. Jeder Nachrichten-Biber hält entweder eine rote oder eine gelbe Flagge hoch.

Manchmal passiert es, dass ein Biber die falsche Flagge hochhält. Das möchte der Biber-Boss erkennen können. Deshalb bestimmt er zusätzlich drei *Prüf-Biber*.

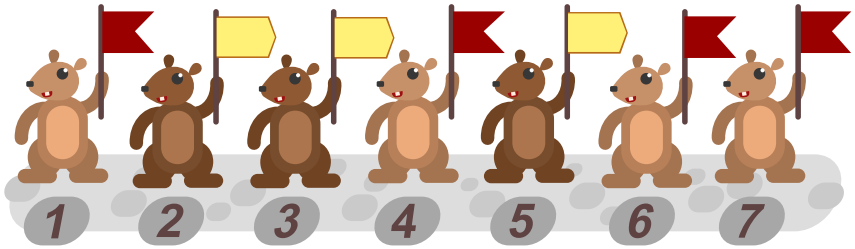
Jeder Prüf-Biber prüft drei Nachrichten-Biber. Wenn diese drei eine ungerade Anzahl an roten Flaggen hochhalten sollen, dann soll ihr Prüf-Biber auch eine rote Flagge hochhalten, sonst eine gelbe. Wenn alle die richtigen Flaggen hochhalten, dann halten ein Prüf-Biber und seine Nachrichten-Biber zusammen eine gerade Anzahl von roten Flaggen hoch.

Der Boss nummeriert die Nachrichten- und Prüf-Biber und ordnet sie so einander zu:

Nachrichten-Biber	**Prüf-Biber**
1, 2, 3	5
1, 2, 4	6
2, 3, 4	7

Insgesamt werden in einer Nachricht nun sieben Flaggen hochgehalten. Der Biber-Boss sieht die Nachricht unten. Er bemerkt, dass genau einer der sieben Biber die falsche Flagge hochhält.

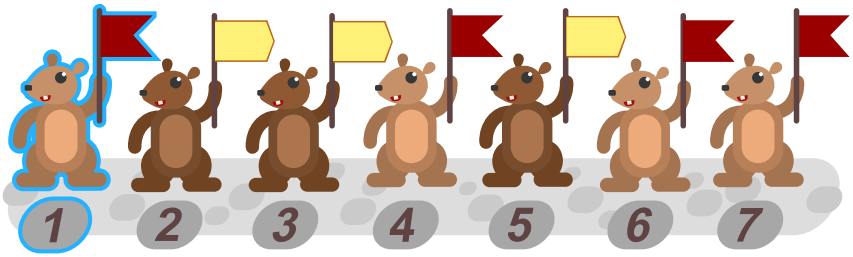
Welcher Biber hält die falsche Flagge hoch?



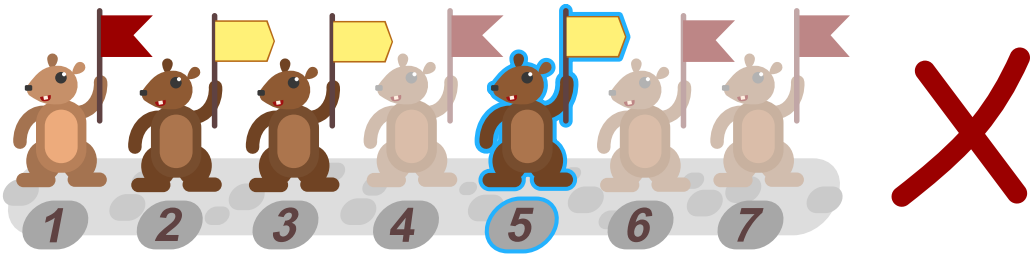
Lösung

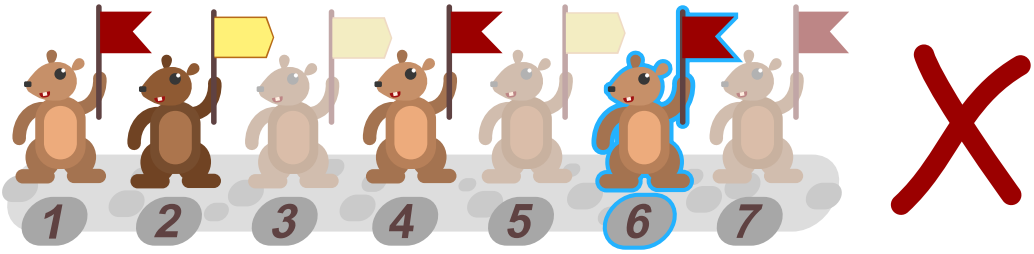
So ist es richtig:

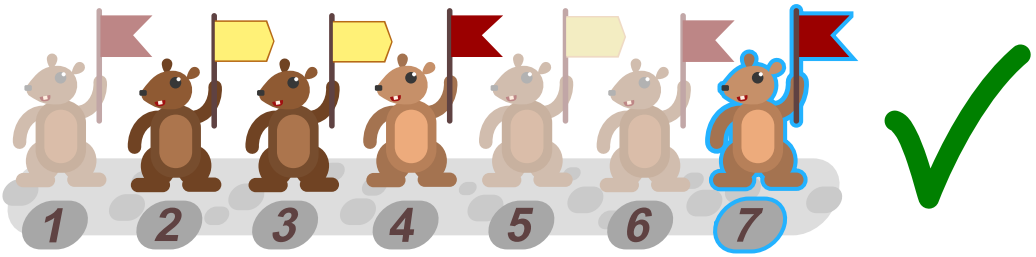
Biber Nummer 1 hält die falsche Flagge hoch.



Der Einfachheit halber werden ein Prüf-Biber zusammen mit seinen Nachrichten-Bibern als Gruppe mit der Nummer des jeweiligen Prüf-Bibers bezeichnet. Es gibt also die drei Gruppen 5, 6 und 7. Eine Gruppe soll also eine gerade Anzahl an roten Flaggen hochhalten. Hier sind die Gruppen, der Prüf-Biber der Gruppe ist markiert.







Man sieht: Bei Gruppe 5 gibt es einen Fehler, sie hält genau eine rote Flagge hoch, also eine ungerade Anzahl. Bei Gruppe 6 gibt es ebenfalls einen Fehler, sie hält drei rote Flaggen hoch, ebenfalls eine ungerade Anzahl. Bei Gruppe 7 ist alles in Ordnung, denn sie hält zwei rote Flaggen hoch, eine gerade Anzahl.

 © Informatik-Biber 2025, SVIA

Die Biber aus Gruppe 7, also die Biber 2, 3, 4 und 7, machen alles richtig. In Gruppe 5 könnte also entweder Biber 1 oder Biber 5 die falsche Flagge hochhalten. In Gruppe 6 könnte entweder Biber 1 oder Biber 6 die falsche Flagge hochhalten. Da insgesamt nur genau ein Biber die falsche Flagge hochhält, muss der gleiche Biber für die Fehler in den Gruppen 5 und 6 verantwortlich sein. Das kann nur Biber 1 sein.

Dies ist Informatik!

In der digitalen Welt werden Nachrichten und auch andere Daten *binär* codiert, also als Folgen aus *Bits* (0 und 1). Wenn diese über Kommunikationskanäle übertragen werden, kann es zu Störungen kommen, welche die Bits vertauschen: aus 0 wird 1 oder umgekehrt. Für eine korrekte Informationsverarbeitung möchte man feststellen, ob eine Übertragung von einer Störung betroffen war, und die entstandenen Fehler korrigieren. Dazu wurden *fehlerkorrigierende Codes* entwickelt.

Eine einfache Möglichkeit wäre, jedes Bit dreimal hintereinander zu senden. Eine Störung an einem der drei Bits könnte leicht erkannt und korrigiert werden, denn die zwei anderen Bits tragen noch immer die richtige Information; die Mehrheit entscheidet dann. Dieses einfache Verfahren führt aber dazu, dass drei mal so viele Daten übertragen werden müssen. Damit die Datenleitungen nicht verstopfen, ist es also wichtig, möglichst wenige zusätzliche Bits zur Fehlerkorrektur zu verwenden.

Der Biber-Boss setzt dazu einen *Hamming-Code* ein. In einem Hamming-Code gibt es für mehrere Gruppen der eigentlichen Datenbits jeweils ein Prüfbit. Das Prüfbit wird so aus den Datenbits der passenden Gruppe berechnet, dass Datenbits und Prüfbit zusammen eine gerade Anzahl von 1en enthalten; diese Eigenschaft nennt man auch «gerade Parität». Wenn in einer Nachricht für alle Gruppen aus Datenbits und zugehörigem Prüfbit die Parität stimmt, dann kann man annehmen, dass die Nachricht richtig übertragen wurde. Wurde nur ein Bit einer mit Hamming-Code kodierten Nachricht durch eine Störung verändert, lassen sich das gestörte Bit und die Originalnachricht korrekt bestimmen, indem man schaut, bei welchen Gruppen die Parität nicht stimmt.

Mit drei Prüfbits kontrolliert der Hamming-Code vier Datenbits, wie in dieser Biberaufgabe. Mit vier Prüfbits können schon 11 Datenbits kontrolliert werden, das Codewort ist dann 15 Bits lang. Mit k Prüfbits kann das Codewort $2^k - 1$ Bits lang sein. Für längere Nachrichten werden also nur wenige zusätzliche Bits benötigt. Dass man mit Hilfe des Hamming-Codes nur einen Bitfehler pro Codewort korrigieren kann, ist in vielen Fällen gut genug. In der Informatik sind auch Codes bekannt, mit deren Hilfe man mehrere Fehler korrigieren kann.

Stichwörter und Webseiten

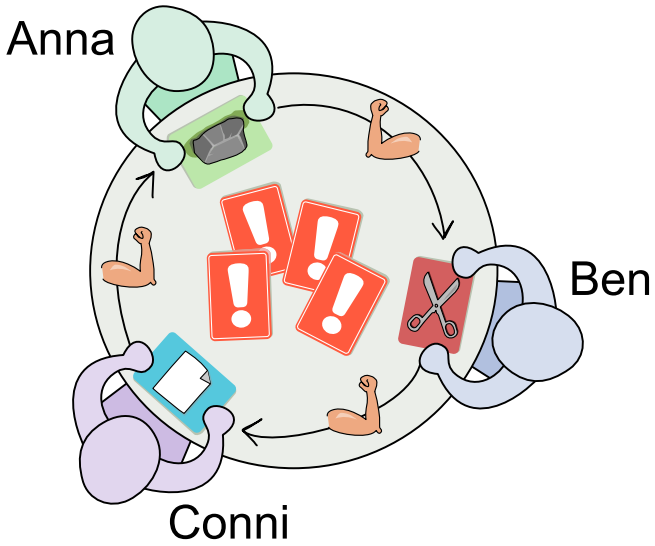
- Hamming Codes: `https://de.wikipedia.org/wiki/Hamming-Code`
- Paritätsbit: `https://de.wikipedia.org/wiki/Paritätsbit`

 © Informatik-Biber 2025, SVIA

37. Schere, Stein, Papier

Anna, Ben und Conni spielen gemeinsam eine Runde Schere-Stein-Papier. Sie spielen gleichzeitig paarweise gegeneinander.

Jeder hat eine Spielkarte gezogen: Anna hat Stein  , Ben hat Schere  , und Conni hat Papier  :



Es gelten die klassischen Regeln: Stein schlägt Schere, Schere schlägt Papier, Papier schlägt Stein. So wie die Karten jetzt verteilt sind, würde jeder einmal gewinnen und einmal verlieren. Zum Beispiel gewinnt Ben gegen Conni und verliert gegen Anna.

Bevor das Ergebnis für jedes Spielerpaar ausgewertet wird, muss aber noch 1 von 4 Aktionskarten gewählt und deren Aktion ausgeführt werden. Bei den Aktionen geht es darum, mehrfach Spielkarten zwischen je 2 Spielern zu tauschen. Bei jedem Tausch kann neu entschieden werden, welche 2 Spieler miteinander tauschen - es sei denn, die Aktion sagt etwas anderes.

Ben möchte unbedingt gegen Conni gewinnen, egal wie die Aktion auf der gewählten Aktionskarte ausgeführt wird.

Nur eine der vier Aktionen garantiert das. Welche?

A) Ben und Conni tauschen ihre Karten eine ungerade Anzahl Male.

B) Beliebig oft tauschen 2 Spieler ihre Karten, aber nie Ben mit Conni.

C) Beliebig oft tauschen 2 Spieler ihre Karten, darunter Ben mindestens einmal mit Conni.

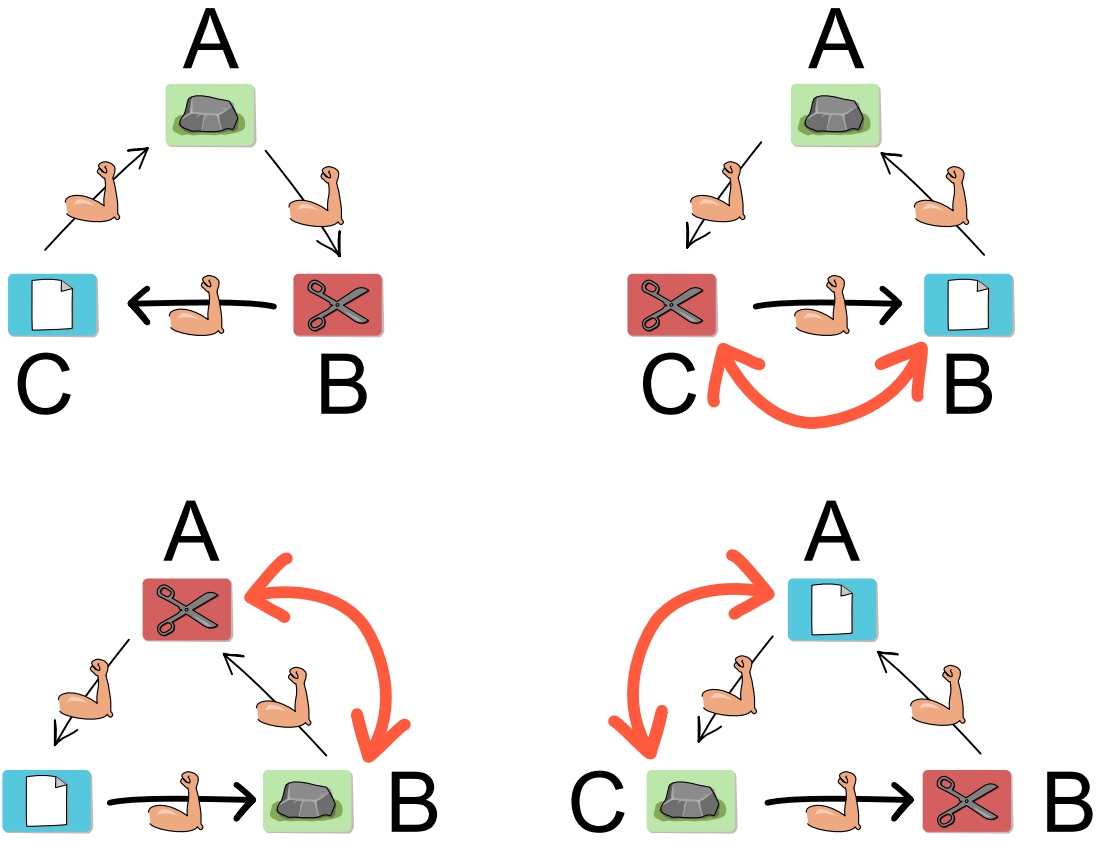
D) Eine gerade Anzahl Male tauschen 2 Spieler ihre Karten.

Lösung

Antwort D ist richtig.

Zunächst beobachten wir, dass Ben mit seiner ursprünglichen Spielkarte (Schere) gegen Conni gewinnen würde (Schere schlägt Papier). Es gilt, dies beizubehalten. Drei der vier Aktionskarten bergen das Risiko, dass Ben am Ende gegen Conni verliert. Nur die Aktionskarte von Antwort D führt **unter keinen Umständen** dazu.

Eine Erkenntnis ist hilfreich: Alle drei Spielkarten sind verschieden, und die Regeln sagen, dass jede Karte gegen eine andere Karte gewinnt und gegen eine andere verliert. Wenn die Spieler im Kreis sitzen, gewinnt jeder gegen den einen Nachbarn und verliert gegen den anderen. Durch einen Tausch ändern sich die Ergebnisse aller Paarungen, egal welche zwei Spieler ihre Karten tauschen:



Nach einer geraden Anzahl Tausche sind also die ursprünglichen Ergebnisse wieder hergestellt. Bei einer ungeraden Anzahl Tausche hingegen sind alle Ergebnisse geändert. Da Ben zu Beginn gegen Conni gewinnt, können wir nur dann sicherstellen, dass er nach Ausführung der Aktion immer noch gegen Conni gewinnt, wenn eine gerade Anzahl Tausche stattfindet. Und das ist nur bei der Aktion von Antwort D garantiert der Fall.

Dies ist Informatik!

Bei der Beantwortung dieser Biberaufgabe war zunächst wichtig zu erkennen, dass sich die Ergebnisse für alle Spielerpaare durch einen einzelnen Tausch komplett umdrehen. Entscheidend: Das passiert

 © Informatik-Biber 2025, SVIA

auch dann, wenn das Spielerpaar selbst seine Karten nicht tauscht. Das Ergebnis zwischen Ben und Conni ändert sich also auch dann, wenn Ben und Conni zwar nicht miteinander tauschen, aber eben Ben mit Anna oder Conni mit Anna tauscht. Eine *lokale Veränderung* zwischen einem Paar hat also auch einen *globalen Effekt* auf alle Paare. Bei der Entwicklung von Computerprogrammen können solche *Seiteneffekte* zu Problemen führen, insbesondere wenn sie unbeabsichtigt auftreten. Dann ist die Sicherheit und Verlässlichkeit der Programme gefährdet.

Sicherheit und Verlässlichkeit von Informatiksystemen spielen für die Informatik eine zentrale Rolle. Besonders offensichtlich ist das bei Systemen, deren Versagen unmittelbar zu grossen Schäden führen kann, wie bei Systemen zur Steuerung von Verkehrsmitteln, Kraftwerken oder schweren Waffen. Aber auch Systeme, die persönliche Daten verarbeiten, sollten dies sicher und verlässlich tun.

Dafür ist wichtig, dass die beim Betrieb des Systems eingesetzten Programme *korrekt* sind. Ein Programm ist korrekt, wenn es für jede Eingabe terminiert und ein den Vorgaben entsprechendes Ergebnis liefert (und so ähnlich kann man Korrektheit auch für Algorithmen beschreiben). Zumindest für wichtige Programme wird versucht, die Korrektheit auch zu beweisen. Dabei spielen u.a. *Invarianten* eine Rolle. So heissen per Logik formulierbare Bedingungen, die z.B. bei Wiederholungsanweisungen zu Beginn jeder einzelnen Wiederholung gültig sind. In dieser Biberaufgabe gilt die Invariante «Ben gewinnt gegen Conni», wenn jede Wiederholung von Kartentauschen zwei Tausche enthält und dadurch insgesamt eine gerade Anzahl von Kartentauschen durchgeführt wird.

Stichwörter und Webseiten

- Invariante: `https://de.wikipedia.org/wiki/Invariante_(Informatik)`
- Spieltheorie: `https://de.wikipedia.org/wiki/Spieltheorie`
- Korrektheit: `https://inf-schule.de/algorithmen/grundlagen/eigenschaften/korrektheit/konzept_korrektheit`

 © Informatik-Biber 2025, SVIA

Programmieraufgaben

Die folgenden Aufgaben zum Programmieren sind Bonusaufgaben des Wettbewerbs.

Für die Wettbewerbsaufgaben sind keine Vorkenntnisse notwendig. Diese Programmieraufgaben lassen sich jedoch mit Programmierkenntnissen einfacher lösen.

Da das Programmieren online viel mehr Spass macht und das Ergebnis direkt ausprobiert werden kann, sind diese Aufgaben unter folgendem QR-Code online zum Bearbeiten verfügbar.

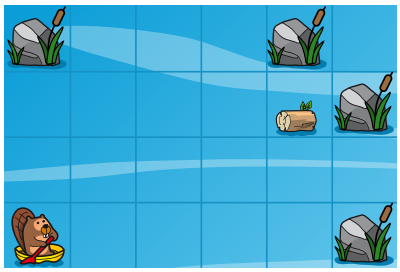
 © Informatik-Biber 2025, SVIA

38. Nur ein Weg

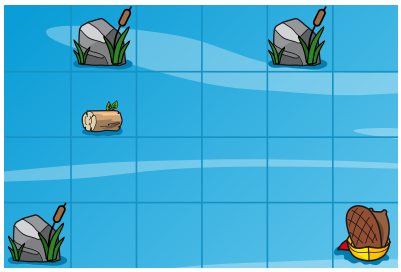
Biber Bea hat im Seeland Baumstämme entdeckt, die sie aufsammeln möchte. Damit sie sich nicht zu viel merken muss, möchte sie immer den gleichen Weg fahren. Hilf Bea eine Anleitung zu schreiben.

Du kannst folgende Anweisungen verwenden:

Anweisung	**Beschreibung**
`turnRight()` / `turnLeft()`	Bea dreht sich an Ort nach rechts / links.
`paddle()`	Bea paddelt solange, bis sie direkt vor einem Fels steht. Ist sie auf einem Feld mit einem Baustamm, sammelt sie diesen auf.



See 1



See 2

Schreibe eine Anleitung, um in beiden Seen den Baumstamm aufzusammeln.

Lösung

Die richtige Lösung lautet wie folgt:

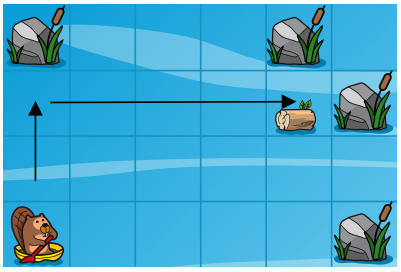
```
turnLeft()
paddle()
turnRight()
paddle()
```

Um im ersten See zum Baumstamm zu gelangen, gibt es neben der Lösung noch eine weitere Möglichkeit:

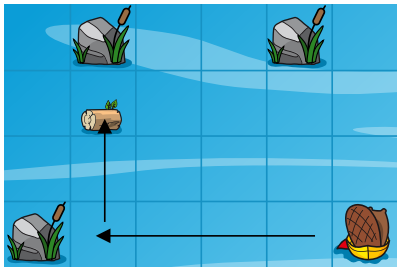
```
paddle()
turnRight()
paddle()
```

Diese Lösung scheint auf den ersten Blick die einfachere Lösung zu sein, allerdings kann mit dieser Lösung der Baumstamm im zweiten See nicht eingesammelt werden, da der Biber aufgrund der Anweisung `paddle()` direkt an das Ufer fährt.

Eine gute Lösungsstrategie bei mehreren Seen ist daher immer, sich zuerst die Seen anzuschauen, um bei der Planung der Strecke die Lage der Felsen und Baumstämme in beiden Seen zu berücksichtigen.



See 1



See 2

Dies ist Informatik!

Informatik befasst sich häufig mit Abstraktion, also der Vereinfachung komplexer Systeme und Prozesse. Programmieren erlaubt es, komplexe Probleme in kleinere Teilprobleme zu zerlegen und diese systematisch zu lösen. Programmieren lehrt eine strukturierte Denkweise, die eine systematische Annäherung an Probleme fördert. Oft versuchen wir, eine Lösung zu finden, die für mehrere ähnliche Probleme einsetzbar ist. In diesem Beispiel soll daher eine Lösung gefunden werden, die nicht nur für einen, sondern für mehrere Fälle funktionieren soll.

Stichwörter und Webseiten

- Programm: `https://de.wikipedia.org/wiki/Programmierung`
- Sequenz: `https://de.wikipedia.org/wiki/Kontrollstruktur`

 © Informatik-Biber 2025, SVIA

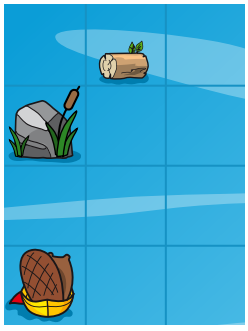
39. Verrückte Sandbank

Biber Benno möchte einen Baumstamm im See aufsammeln. Benno hat herausgefunden, dass eine Sandbank im See einen Fels immer an verschiedene Stellen verschiebt. Hilf Benno eine Anleitung zu schreiben, mit der er den Baumstamm aufsammeln kann, egal wo der Fels ist.

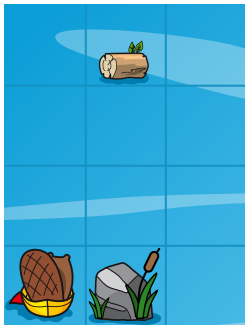
Du kannst folgende Anweisungen verwenden:

Anweisung	**Beschreibung**
`move()`	Benno bewegt sich genau ein Feld in Blickrichtung nach vorne.
`turnRight()` / `turnLeft()`	Benno dreht sich an Ort um 90 Grad nach rechts / links.
`removeLog()`	Benno entfernt den Baumstamm von dem Feld, auf dem er steht.

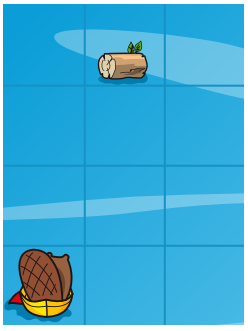
See 1



See 2



See 3



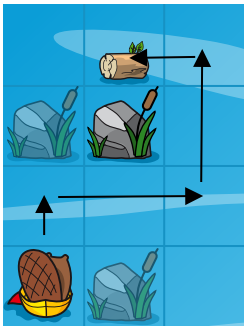
See 4

Schreibe eine Anleitung, mit der Benno den Baumstamm immer aufsammeln kann.

Lösung

Die richtige Lösung lautet wie folgt:

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



Da sich die Sandbank veschiebt, ist es nicht möglich den direkten Weg zum Baumstamm zu nehmen:

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

Diese Lösung scheint auf den ersten Blick die kürzeste Lösung zu sein, da sie nicht nur im See 1, sondern auch im See 2 und 3 zum Baumstamm führt. Allerdings kann der Baumstamm im vierten See nicht aufgesammelt werden, da der Biber direkt gegen einen Fels fährt. Natürlich könnten wir jetzt das Programm anpassen, sodass der Baumstamm im vierten See eingesammelt werden kann. Dabei kann es aber passieren, dass wir eine Lösung finden, bei der der Baumstamm in einem der anderen Seen nicht mehr eingesammelt werden kann.

Eine gute Lösungsstrategie bei mehreren Seen ist daher immer, sich zuerst die Seen anzuschauen, um bei der Planung der Strecke die Lage der Felsen und Baumstämme in beiden Seen zu berücksichtigen.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Informatik befasst sich häufig mit Abstraktion, also der Vereinfachung komplexer Systeme und Prozesse. Programmieren erlaubt es, komplexe Probleme in kleinere Teilprobleme zu zerlegen und diese systematisch zu lösen. Programmieren lehrt eine strukturierte Denkweise, die eine systematische Annäherung an Probleme fördert. Oft versuchen wir, eine Lösung zu finden, die für mehrere ähnliche Probleme einsetzbar ist. In diesem Beispiel soll daher eine Lösung gefunden werden, die nicht nur für einen, sondern für mehrere Fälle funktionieren soll.

Stichwörter und Webseiten

- Programm: `https://de.wikipedia.org/wiki/Programmierung`
- Sequenz: `https://de.wikipedia.org/wiki/Kontrollstruktur`

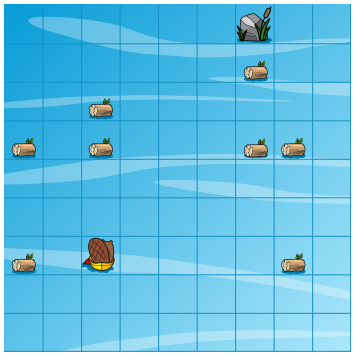
 © Informatik-Biber 2025, SVIA

40. Wertvoller Baumstamm

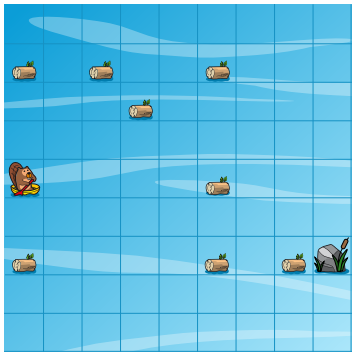
Biber Petunia ist auf der Suche nach wertvollen Baumstämmen im Seeland. Der wertvollste Baumstamm ist immer direkt bei einem Felsen. Hilf Petunia eine Anleitung zu schreiben, mit der sie in allen drei Seen bis auf das Feld mit dem wertvollen Baumstamm gelangt. Nutze möglichst wenige Anweisungen.

Du kannst folgende Anweisungen verwenden:

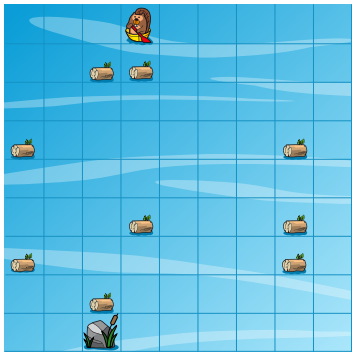
Anweisung	**Beschreibung**
`move()`	Petunia bewegt sich genau ein Feld in Blickrichtung nach vorne.
`turnRight()` / `turnLeft()`	Petunia dreht sich an Ort um 90 Grad nach rechts / links.
`goToLog()`	Petunia fährt vorwärts, bis sie auf einem Feld mit einem Baumstamm ist.



See 1



See 2



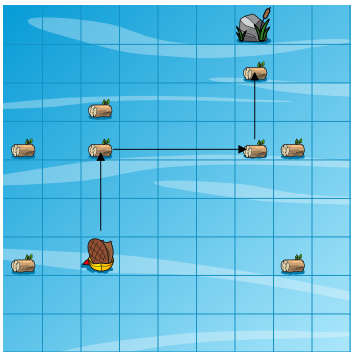
See 3

Schreibe eine Anleitung, um in allen drei Seen zum wertvollen Baumstamm am Fels zu gelangen. Nutze möglichst wenige Anweisungen.

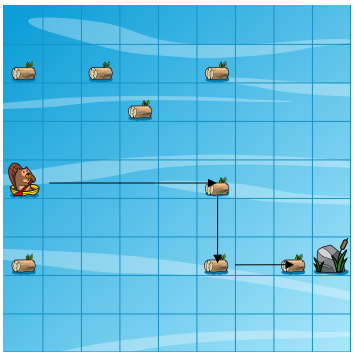
Lösung

Die richtige Lösung lautet wie folgt:

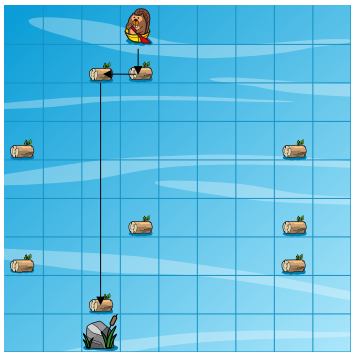
```
goToLog()
turnRight()
goToLog()
turnLeft()
goToLog()
```



See 1



See 2



See 3

Um möglichst wenige Anweisungen zu nutzen, ist die Verwendung des Befehls goToLog() nötig. Damit kommen wir immer von einem Baumstamm zum nächsten, unabhängig davon, wie weit diese voneinander entfernt liegen.

Würden wir für See 1 stattdessen den Befehl move() nehmen, so gelangen wir auch direkt zum wertvollen Baumstamm:

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

Allerdings haben wir damit nicht das Programm gefunden, welches die wenigsten Anweisungen verwendet, da das Programm deutlich länger als die gesuchte Lösung ist.

Aber auch ein anderes Problem entsteht: Mit dem längeren Programm erreicht der Biber im zweiten und dritten See nicht mehr den wertvollen Baumstamm. Erst mit dem Befehl `goToLog()` schaffen wir es, mit der gleichen Reihenfolge von Befehlen in allen drei Seen zum Baumstamm zu gelangen, egal wie weit die einzelnen Abstände zwischen dem Biber und dem Baumstamm sind.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Informatik befasst sich häufig mit Abstraktion, also der Vereinfachung komplexer Systeme und Prozesse. Programmieren erlaubt es, komplexe Probleme in kleinere Teilprobleme zu zerlegen und diese systematisch zu lösen. Programmieren lehrt eine strukturierte Denkweise, die eine systematische Annäherung an Probleme fördert. Oft versuchen wir, eine Lösung zu finden, die für mehrere ähnliche Probleme einsetzbar ist. In diesem Beispiel soll daher eine Lösung gefunden werden, die nicht nur für einen, sondern für mehrere Fälle funktionieren soll. Programmatische Lösungen, die nur für einen bestimmten Fall funktionieren, nennen wir Hardcode. Oft versuchen wir zu verhindern, dass eine Lösung hardcodiert wird, und schreiben stattdessen Programme, die für mehrere ähnliche Probleme einsetzbar sind. Anstatt also abzuzählen, wie viele Felder Petunia nach vorne gehen soll, nutzen wir den Befehl `goToLog()`, dem eine Schleifenstruktur zugrunde liegt (hier: bewege dich so lange nach vorne, bis du auf einem Feld mit einem Baumstamm bist), die das Abzählen der Felder nicht mehr nötig macht und eine kürzere Version des Programms ermöglicht.

Stichwörter und Webseiten

- Programm: `https://de.wikipedia.org/wiki/Programmierung`
- Sequenz: `https://de.wikipedia.org/wiki/Kontrollstruktur`
- Schleife: `https://de.wikipedia.org/wiki/Schleife_(Programmierung)`

 © Informatik-Biber 2025, SVIA

41. Noch mehr Holz

Biber Linus braucht viel Holz. Leider kann Linus noch nicht so gut paddeln. Paddelt er einmal los, fährt er immer bis direkt vor den nächsten Felsen. Hilf Linus einen Weg zu finden, mit dem er die höchste Anzahl an Baumstämmen aufsammeln kann.

Du kannst folgende Anweisungen verwenden:

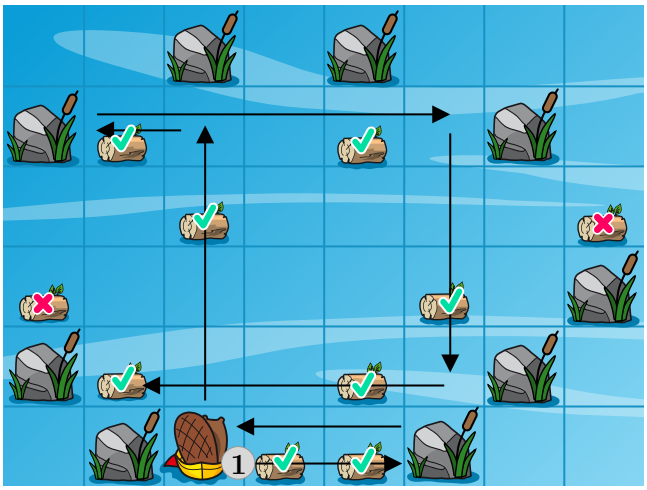
Anweisung	**Beschreibung**
`turnRight()` / `turnLeft()`	Linus dreht sich an Ort um 90 Grad nach rechts / links.
`paddle()`	Linus paddelt solange, bis er direkt vor einem Fels steht. Ist er auf einem Feld mit einem Baumstamm, sammelt er diesen auf.

Schreibe eine Anleitung, um die höchste Anzahl an Baumstämmen aufzusammeln.

Lösung

Die richtige Lösung lautet wie folgt:

```
turnRight()
paddle()
turnRight()
turnRight()
paddle()
turnRight()
paddle()
turnLeft()
paddle()
turnRight()
turnRight()
paddle()
turnRight()
paddle()
turnRight()
paddle()
```



Da wir eine Bewegung nach vorne nur über den Befehl `paddle()` durchführen können, endet jede Vorwärtsbewegung automatisch vor einem Fels. Durch diese Einschränkung können maximal 8 Baumstämme eingesammelt werden. Hätten wir zusätzlich zum `paddle()`-Befehl noch einen weiteren Befehl, der Linus einen Einzelschritt machen ließe, wäre es möglich gewesen, mehr Baumstämme aufzusammeln. Dazu hätten wir aber noch einen weiteren Befehl, nämlich zum Aufsammeln der Baumstämme, benötigt.

 © Informatik-Biber 2025, SVIA

Dies ist Informatik!

Das Problem umfasst mehrere Teile: Im ersten Teil befassen wir uns mit einer Wegsuche. Diese ist in der Mathematik wie in der Informatik oft als Graphenproblem beschrieben. Eng damit verbunden sind meist Optimierungsprobleme. Denn in der Aufgabe ist nicht nur die Suche nach dem richtigen Weg, sondern gleichzeitig nach der größtmöglichen Anzahl an Baumstämmen gefragt, die dabei eingesammelt werden können. Bekannte Optimierungsprobleme sind zum Beispiel das Problem des Handlungsreisenden.

Im zweiten Teil, der Programmierung, befassen wir uns mit der Frage, wie der zuvor gefundene Weg präzise beschrieben werden kann. Hierbei ist es zunächst wichtig, die Funktionsweise der Anweisung `paddle()` zu verstehen. Hinter dieser Anweisung ist eine bedingte Schleife versteckt. Das heißt, es werden dieselben Befehle so lange wiederholt ausgeführt, wie eine bestimmte Bedingung zutrifft. Biber Petunia kontrolliert vorab, ob der Weg noch frei ist. Ist dies der Fall (d. h. es befindet sich kein Fels auf dem Feld direkt vor ihr), bewegt sie sich um ein Feld vor und beginnt denselben Kontrollprozess erneut. Damit ist es möglich, eine beliebige Distanz bis zu einem Felsen zu überbrücken.

Stichwörter und Webseiten

- Programm: `https://de.wikipedia.org/wiki/Programmierung`
- Sequenz: `https://de.wikipedia.org/wiki/Kontrollstruktur`
- Schleife: `https://de.wikipedia.org/wiki/Schleife_(Programmierung)`
- TSP: `https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden`

 © Informatik-Biber 2025, SVIA

42. Um die Felsen

Biber Benno möchte Baumstämme im Seeland aufsammeln. Schreibe ein Programm, mit dem der Biber in allen Seen den Baumstamm einsammeln kann.

Du kannst folgende Anweisungen verwenden:

Anweisung	**Beschreibung**
`move()`	Benno bewegt sich genau ein Feld in Blickrichtung nach vorne.
`turnRight()` / `turnLeft()`	Benno dreht sich an Ort um 90 Grad nach rechts / links.
`removeLog()`	Benno entfernt den Baumstamm von dem Feld, auf dem er steht.

`while ...:`
`  Anweisung`
`  Anweisung`
	Benno wiederholt nachfolgend eingerückte Anweisungen solange, wie eine Bedingung erfüllt ist.

Im folgenden Beispiel bewegt ich Benno so lange ein Feld nach vorne, wie rechts von ihm ein Stein ist. Ist dies nicht (mehr) der Fall, führt er die nächste nicht eingerückte Anweisung – hier im Beispiel `turnRight()` – aus:

```
while rockRight():
    move()
turnRight()
```

See 1

See 2

See 3

Schreibe eine Anleitung, um in allen Seen den Baumstamm aufzusammeln. Zeilen 1 - 3 sind vorgegeben und können nicht verändert werden.

Lösung

Die richtige Lösung lautet wie folgt:

```
while rockRight():
    move()
turnRight()

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

while rockRight():
    move()
    move()

removeLog()
```

Ein erstes Ausführen des Programms mit den vorgegebenen Zeilen 1–3 zeigt, dass Benno so lange fährt, bis kein Fels mehr rechts von ihm ist und dann eine Rechtsdrehung durchführt. Den weiteren Programmcode können wir jetzt in zwei Teile unterteilen. Wir müssen Benno zunächst um die obere waagerechte Reihe der Felsen führen. Da diese in allen Seen genau zwei Felsen breit ist, können wir diese Distanz direkt mit einer dreifachen Ausführung des Befehls `move()` überbrücken. Für den Weg nach unten reicht es allerdings nicht, über Abzählen die Anzahl der Vorwärtsschritte zu bestimmen, da die Distanz zum Baumstamm in allen drei Seen unterschiedlich ist. Daher benötigen wir erneut die Kontrollstruktur `while`, um einen oder mehrere Befehle wiederholt auszuführen. Im Gegensatz zur linken Felsreihe befindet sich aber immer genau ein leeres Feld zwischen den Felsen der rechten Reihe, d. h. damit die Bedingung `rockRight()` wieder erfüllt ist, muss sich Benno zwei Felder nach vorne bewegen.

Dies ist Informatik!

Informatik befasst sich häufig mit Abstraktion, also der Vereinfachung komplexer Systeme und Prozesse. Programmieren erlaubt es, komplexe Probleme in kleinere Teilprobleme zu zerlegen und diese systematisch zu lösen. Programmieren lehrt eine strukturierte Denkweise, die eine systematische Annäherung an Probleme fördert. Oft versuchen wir, eine Lösung zu finden, die für mehrere ähnliche Probleme einsetzbar ist. In diesem Beispiel soll daher eine Lösung gefunden werden, die nicht nur für einen, sondern für mehrere Fälle funktionieren soll. Programmatische Lösungen, die nur für einen bestimmten Fall funktionieren, nennen wir Hardcode. Oft versuchen wir zu verhindern, dass eine Lösung hardcodiert wird, und schreiben stattdessen Programme, die für mehrere ähnliche Probleme einsetzbar sind. Über die Kontrollstruktur `while` in Zusammenhang mit einer Bedingung `rockRight()` ist es möglich, unabhängig von der Anzahl der Felsen eine Vorwärtsbewegung anzustoßen, die erst

 © Informatik-Biber 2025, SVIA

beendet wird, wenn die Bedingung nicht mehr erfüllt ist. Im konkreten Fall endet die Bewegung, wenn sich rechts neben Benno kein Fels mehr befindet. Wir sprechen bei dieser Kontrollstruktur auch von einer bedingten Schleife.

Stichwörter und Webseiten

- Programm: `https://de.wikipedia.org/wiki/Programmierung`
- Sequenz: `https://de.wikipedia.org/wiki/Kontrollstruktur`
- Schleife: `https://de.wikipedia.org/wiki/Schleife_(Programmierung)`

 © Informatik-Biber 2025, SVIA

43. Hin und Her

Biber Benno möchte Baumstämme im Seeland aufsammeln. Schreibe ein Programm, mit dem der Biber in allen Seen den Baumstamm einsammeln kann.

Verwende dazu die folgenden Anweisungen:

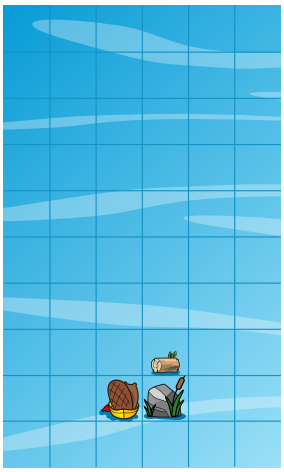
Anweisung	**Beschreibung**
`move()`	Benno bewegt sich genau ein Feld in Blickrichtung nach vorne.
`turnRight()` / `turnLeft()`	Benno dreht sich an Ort um 90 Grad nach rechts / links.
`removeLog()`	Benno entfernt den Baumstamm von dem Feld, auf dem er steht.

`while ...:`
`  Anweisung`
`  Anweisung`
	Benno wiederholt nachfolgend eingerückte Anweisungen solange, wie eine Bedingung erfüllt ist.

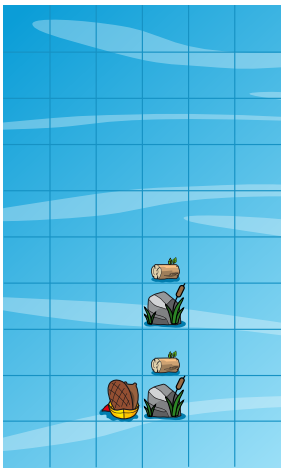
`rockRight()`	*Bedingung*: Prüft, ob rechts von Benno ein Fels ist.
`rockLeft()`	*Bedingung*: Prüft, ob links von Benno ein Fels ist.

Im folgenden Beispiel bewegt ich Benno so lange ein Feld nach vorne, wie links von ihm ein Fels ist. Ist dies nicht (mehr) der Fall, fährt er mit der nächsten nicht eingerückten Anweisung – hier im Beispiel `turnLeft()` – fort:

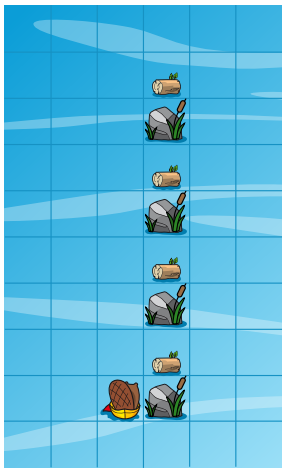
```
while rockLeft():
    move()
turnLeft()
```



See 1



See 2



See 3

Schreibe eine Anleitung, um in allen Seen den Baumstamm aufzusammeln.

Lösung

Die richtige Lösung lautet wie folgt:

```
while rockRight():
    move()
    turnRight()
    move()
    removeLog()
    turnLeft()
    turnLeft()
    move()
    turnRight()
    move()
```

Da wir ein Programm schreiben möchten, welches für alle drei Seen gültig ist, erkennen wir direkt, dass wir über reines Abzählen einzelner Schritte und Bewegungen nicht zur Lösung kommen. Sicherlich funktioniert diese Herangehensweise für einen einzelnen See, diese Lösung wird aber bei den anderen Seen nicht funktionieren, da sich die Anzahl der Felsen und der Baumstämme verändert.

Vergleichen wir die Seen untereinander, so erkennt man aber ein Muster, welches aus einem Fels und einem dahinterliegenden Baumstamm besteht. Dieses Muster wird dann in beiden Welten unterschiedlich oft wiederholt.

Das Erkennen des Musters führt zu der Verwendung der Kontrollstruktur `while`, mit welcher Befehle wiederholt ausgeführt werden können, solange eine Bedingung erfüllt ist. In seiner Startposition befindet sich in allen drei Welten rechts neben Benno ein Fels. Nehmen wir daher als Bedingung der Schleife `rockRight()`, so wissen wir, dass diese Bedingung direkt erfüllt ist und die eingerückten Anweisungen (Schleifenrumpf) mindestens einmal ausgeführt werden.

Da sich das Muster Fels–Baumstamm wiederholt, müssen wir im Schleifenrumpf lediglich dafür sorgen, dass Benno seine Bewegung so beendet, dass rechts von ihm wieder ein Fels ist, sodass die Schleifenbedingung `rockRight()` erneut erfüllt ist und der Schleifenrumpf ein weiteres Mal ausgeführt wird. Sobald in der Endposition kein Fels mehr rechts von Benno ist, also die Bedingung nicht mehr erfüllt ist, wird der Schleifenrumpf nicht mehr ausgeführt und die Programmausführung endet.

Dies ist Informatik!

Informatik befasst sich häufig mit Abstraktion, also der Vereinfachung komplexer Systeme und Prozesse. Programmieren erlaubt es, komplexe Probleme in kleinere Teilprobleme zu zerlegen und diese systematisch zu lösen. Programmieren lehrt eine strukturierte Denkweise, die eine systematische Annäherung an Probleme fördert. Oft versuchen wir, eine Lösung zu finden, die für mehrere ähnliche Probleme einsetzbar ist.

In diesem Beispiel soll eine Lösung gefunden werden, die nicht nur für einen, sondern für mehrere Fälle funktionieren soll. Programmatische Lösungen, die nur für einen bestimmten Fall funktionieren,

 © Informatik-Biber 2025, SVIA

nennen wir Hardcode. Oft versuchen wir zu verhindern, dass eine Lösung hardcodiert wird, und schreiben stattdessen Programme, die für mehrere ähnliche Probleme einsetzbar sind.

Über die Kontrollstruktur while in Zusammenhang mit der Bedingung ist es möglich, Befehle wiederholt ausführen zu lassen, solange die verwendete Bedingung nach Ausführung des Schleifenrumpfes erneut erfüllt ist (kopfgesteuerte Schleife). Erkennen wir zudem das zugrunde liegende Muster (hier: Fels und Baumstamm wechseln sich ab), können wir die Lösung verallgemeinern und damit eine Lösung finden, welche auch weitere Instanzen (hier: verschiedene Seen) des gleichen Problems lösen kann, ohne den Programmcode anpassen zu müssen.

Stichwörter und Webseiten

- Programm: `https://de.wikipedia.org/wiki/Programmierung`
- Sequenz: `https://de.wikipedia.org/wiki/Kontrollstruktur`
- Schleife: `https://de.wikipedia.org/wiki/Schleife_(Programmierung)`

A. Aufgabenautoren

James Atlas
Masiar Babazadeh
Angeni Bai
Leonardo Barichello
Wilfried Baumann
Susanne Berchtold
Maksim Bolonkin
Leonardo Cavalcante
Maria Cepeda
Špela Cerar
Gi Soong Chee
Byeonggyu Cho
Anton Chukhnov
Vladimir Costas
Andrew Csizmadia
Valentina Dagienė
Darija Dasović
Christian Datzko
Justina Dauksaite
Diane Dowling
Nora A. Escherle
Abeer Eshra
Gerald Futschek
Vernon Gutierrez
Silvan Horvath
Alisher Ikramov
Thomas Ioannou
Asterios Karagiannis
Blaž Kelvišar
David Khachatryan
Doyong Kim
Jihye Kim
Dong Yoon Kim
Vaidotas Kinčius
Stefan Koch
Jia-Ling Koh
Sophie Koh
Víctor Koleszar
Lukas Lehner
Taina Lehtimäki
Gunwoong Lim
Linda Mannila
Yoshiaki Matsuzawa
Hamed Mohebbi
Mattia Monga
Anna Morpurgo
Kamohelo Motloung
Justina Oostendorp
A-Yeong Park
Suchan Park
Gabriela Gomez Pasquali
Jean-Philippe Pellet

© Informatik-Biber 2025, SVIA

Emiliano Pereiro

Emmanuel Plan

Zsuzsa Pluhár

Wolfgang Pohl

Cesar F. Bolanos Revelo

Pedro Ribeiro

Rokas Rimkus

Kirsten Schlüter

Margareta Schlüter

Dirk Schmerenbeck

Vipul Shah

Jacqueline Staub

Alieke Stijf

Nikolaos Stratis

Supawan Tasanaprasert

Susanne Thut

Christine Vender

Florentina Voboril

Michael Weigend

Chris Wetherell

Philip Whittington

Kyra Willekes

Hsu Sint Sint Yee

B. Akademische Partner

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

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

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

Pädagogische Hochschule Zürich
https://www.phzh.ch/

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

© Informatik-Biber 2025, SVIA

C. Sponsoring

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

Abraxas Informatik AG
https://www.abraxas.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
http://senarclens.com/

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