INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Aufgaben und Lösungen 2025

Schuljahre 7/8

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

Herausgeber:

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

SVIA
www.svia-ssie-ssii.ch
schweizerischervereinfürinformatikind erausbildung//sociétésuissepourl'informatiquedansl'enseignement//societàsvizzeraperl'informaticanell'insegnamento

HASLERSTIFTUNG ETHzürich 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

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 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.

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

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/`

Inhaltsverzeichnis

Mitarbeit Informatik-Biber 2025 i

Vorwort iv

Inhaltsverzeichnis vi

1. Zahlenmaschine 1

2. Blätter im Wind 5

3. Lichterstern 11

4. Dein Bauwerk 15

5. Blumentöpfe 19

6. Biberholz 23

7. Bebrasien 27

8. Lefty II 31

9. Ein Tag im Nebel 35

10. Stammbaum 39

11. Kurierdienst 41

12. Der Drachen ist weg! 45

13. Brennende Kerzen 49

14. Helligkeitskarte 53

15. Seoul entdecken! 57

16. Bergseen 61

17. Parkplätze 65

18. Noch mehr Holz 71

A. Aufgabenautoren 74

B. Akademische Partner 75

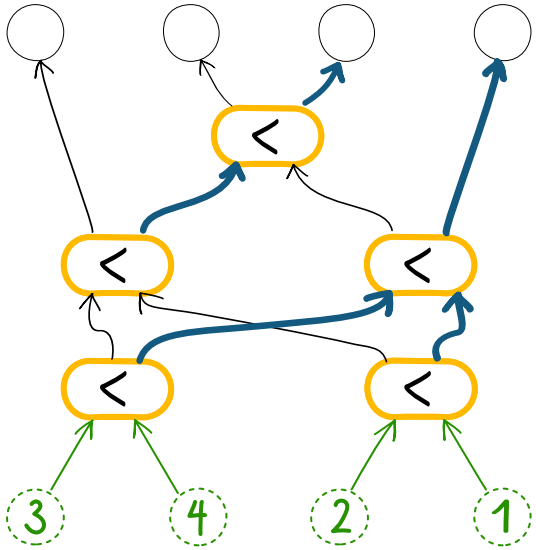
C. Sponsoring 76

1. Zahlenmaschine

Die Biber haben eine Zahlenmaschine.

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

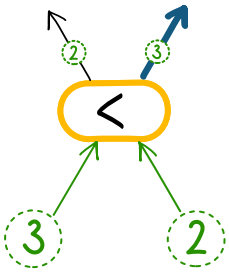
Entlang von Pfeilen und Schaltern < wandern die Zahlen durch die Maschine nach oben bis zu den Ausgabefeldern ◌.



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

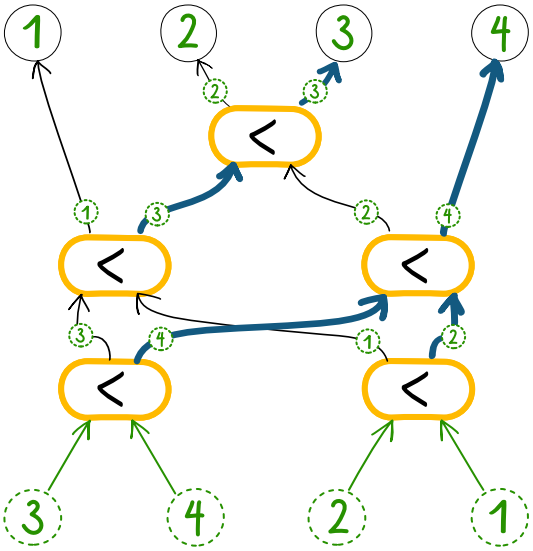
C) Sie gibt die Zahlen in derselben Reihenfolge aus. Ergebnis: 3, 4, 2, 1

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.

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`

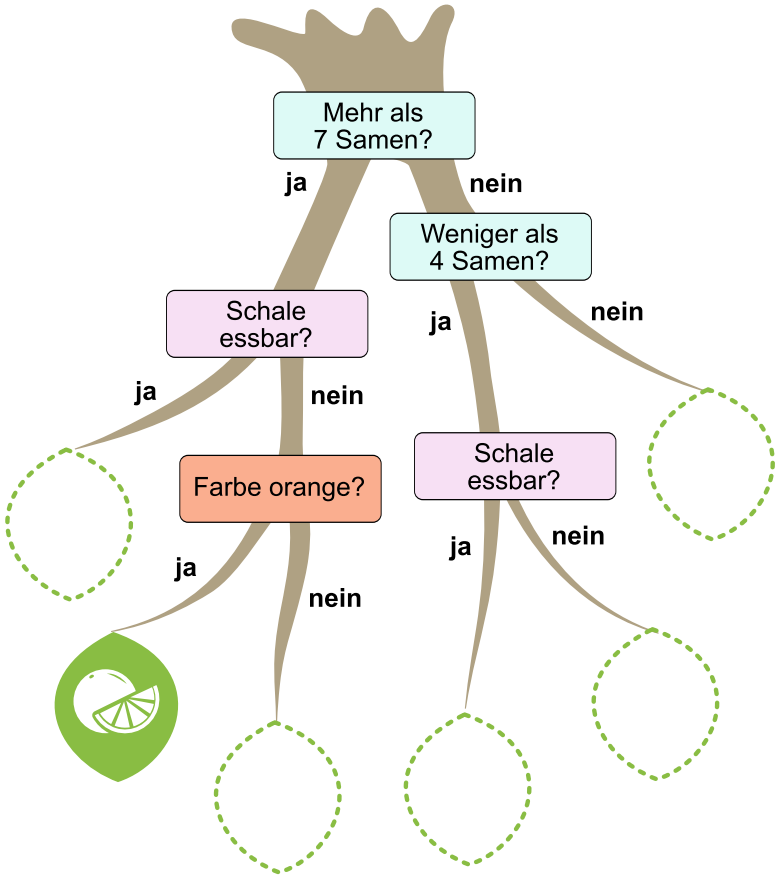
2. Blätter im Wind

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.

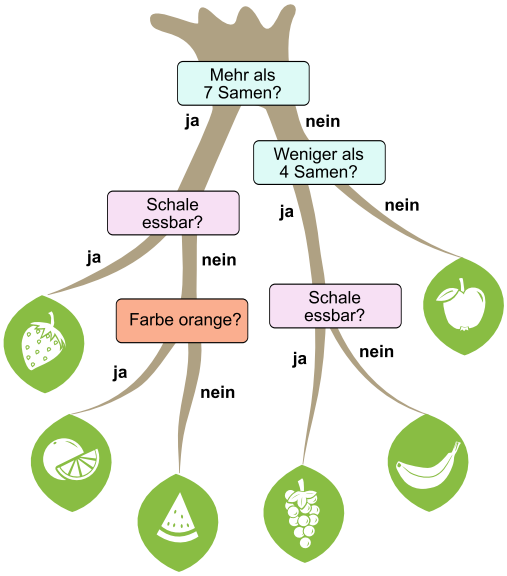


Nach den zehn Früchten ist der Entscheidungsbaum leider kaputt gegangen: Fast alle Blätter sind abgefallen.

An welchen Stellen waren die Blätter?

Lösung

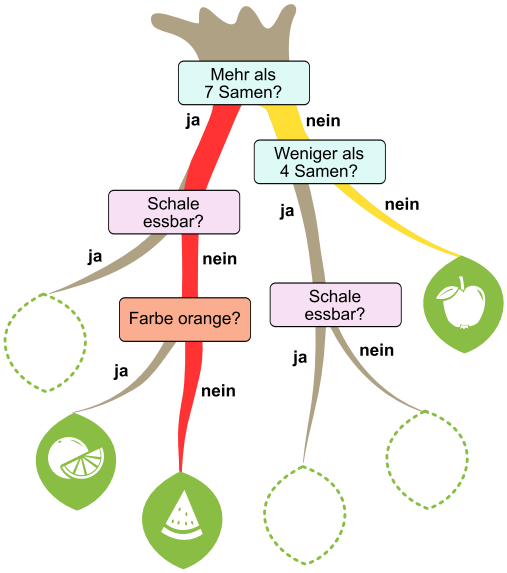
So ist es richtig:



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.

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`

3. 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.

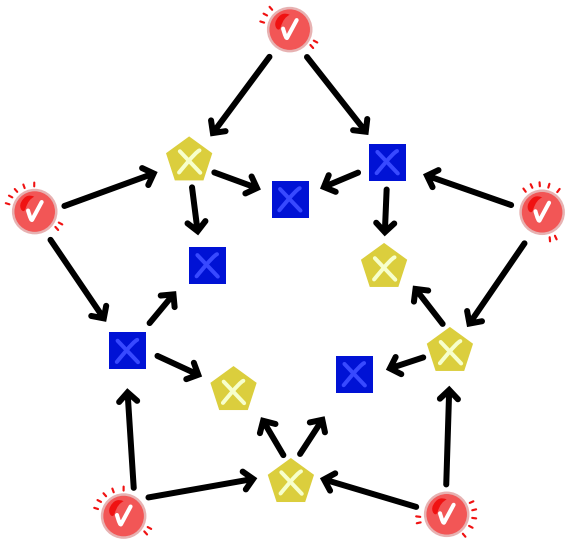
Die blauen und gelben Lichter werden über die Kabel gesteuert, in Pfeilrichtung. Jedes blaue oder gelbe Licht hat also genau zwei «Steuer-Lichter».

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.

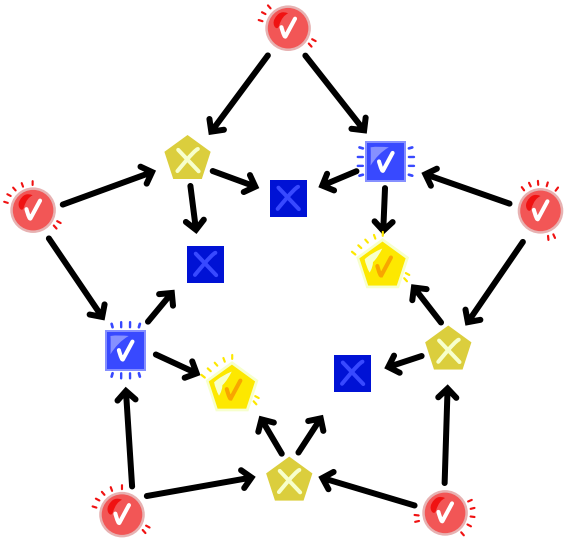
Sophie schaltet alle roten Lichter an.

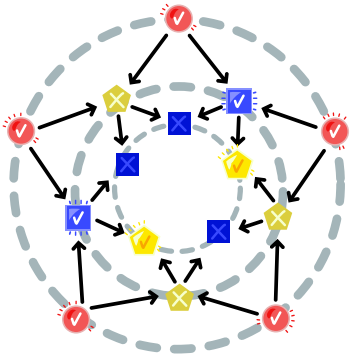
Welche anderen Lichter sind dann auch an?



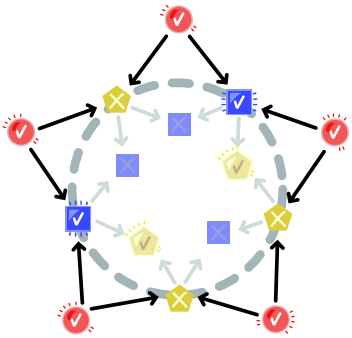
Lösung

So ist es richtig:

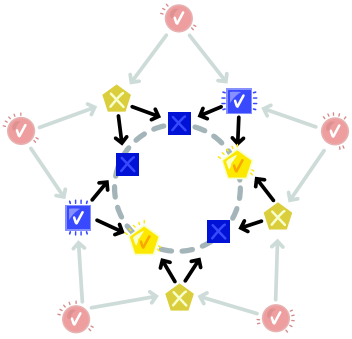




Die Lichter bilden drei Ringe: einen äusseren mit allen roten Lichtern sowie einen mittleren und einen inneren mit blauen und gelben Lichtern.



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.

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`

4. Dein Bauwerk

Du hast diese Bauklötze:

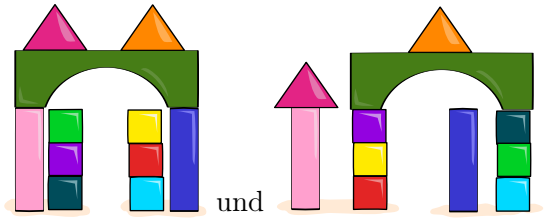
6 Würfel 2 Quader 1 Brücke 2 Pyramiden

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.

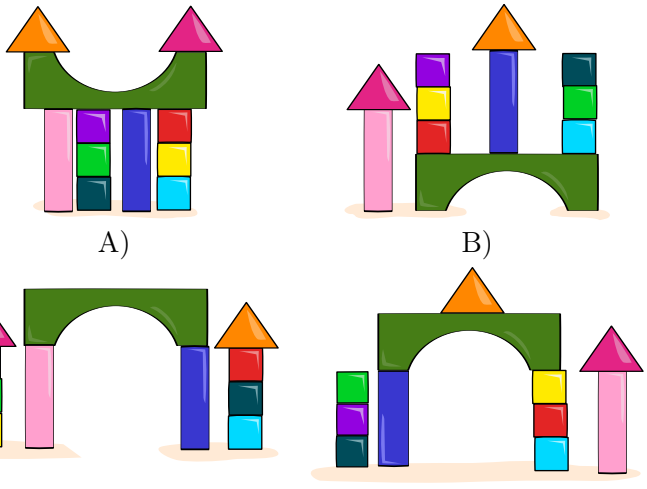
Beim Bauen musst du dich an die Reihenfolge der 6 Anweisungen halten. Mit der Bauanleitung kannst du trotzdem viele verschiedene Bauwerke bauen.

Beispiele:



und

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

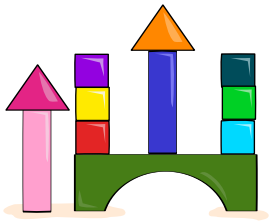


A) B)

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.

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`

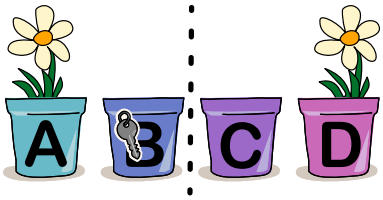
5. Blumentöpfe

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

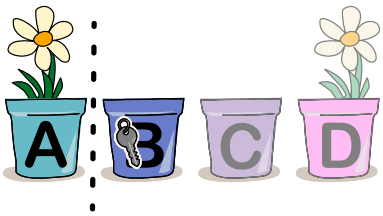
In einem Topf ist ein Schlüssel versteckt. Florian erklärt seine Methode, wie man den Schlüssel finden kann.

«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:

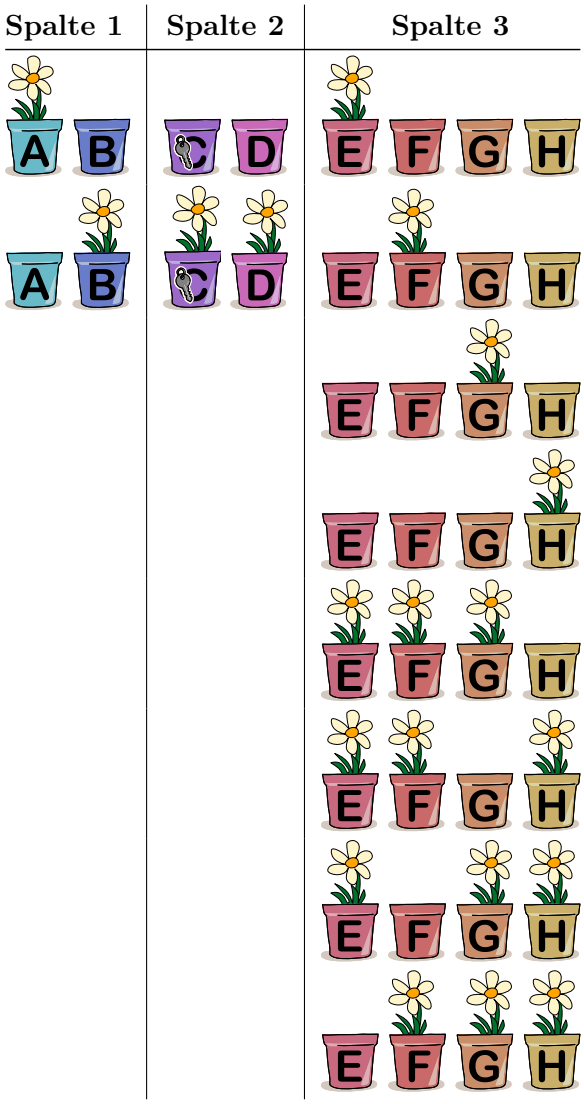
Eine andere richtige Antwort ist:

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.



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.

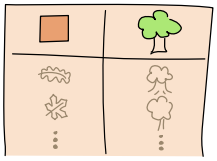
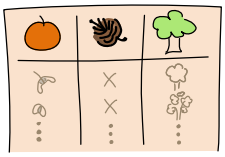
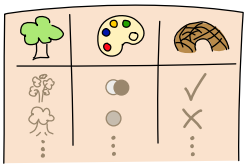
Die richtige Antwort lässt sich bei dieser Biberaufgabe am besten mit einem sogenannten *Bottom-up Ansatz* finden. Dabei betrachtet man die kleinsten Teile der Antwort zuerst, da die grösseren Teile davon abhängen. Statt alle möglichen Bepflanzungen der Töpfe durchzuprobieren, was hier schon $2^8 = 256$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

6. 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.

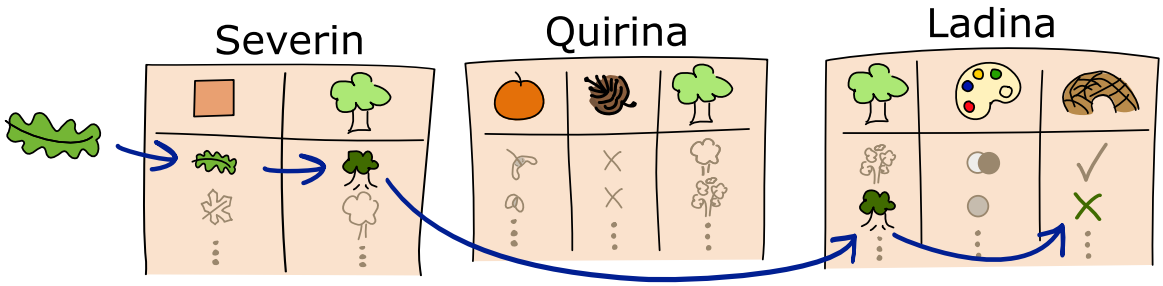
Lösung

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.

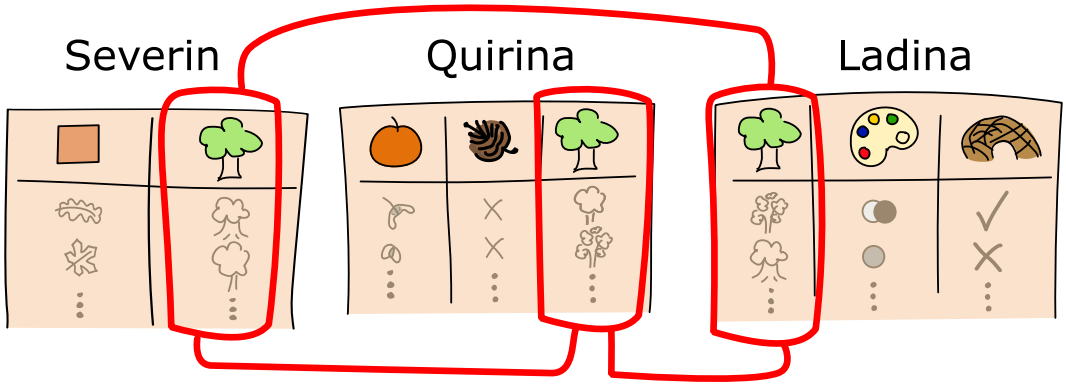


Dies ist Informatik!

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.

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



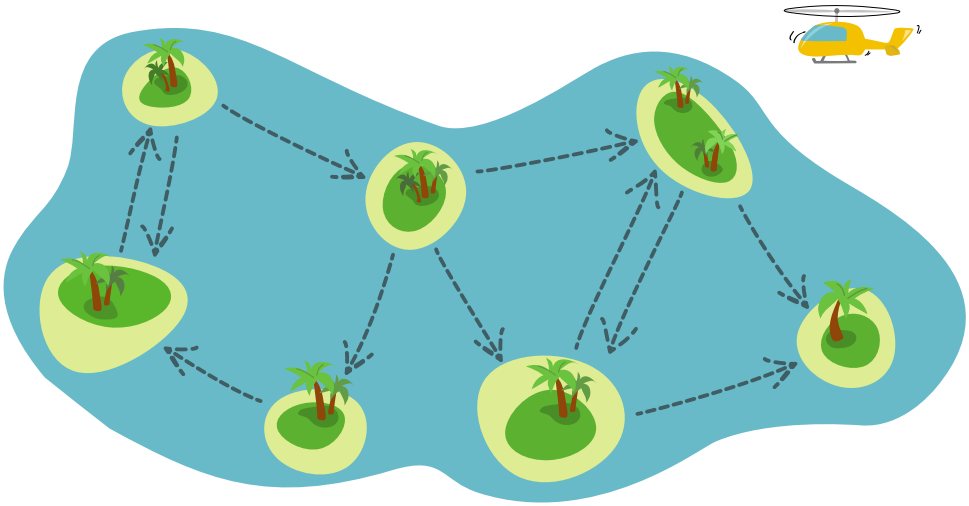
Relationale Datenbanken sind so wichtig, dass es in der Informatik für Abfragen und andere Operationen auf Datenbanken eine eigene Sprache gibt, nämlich *SQL (Structured Query Language)*.

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`

7. 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

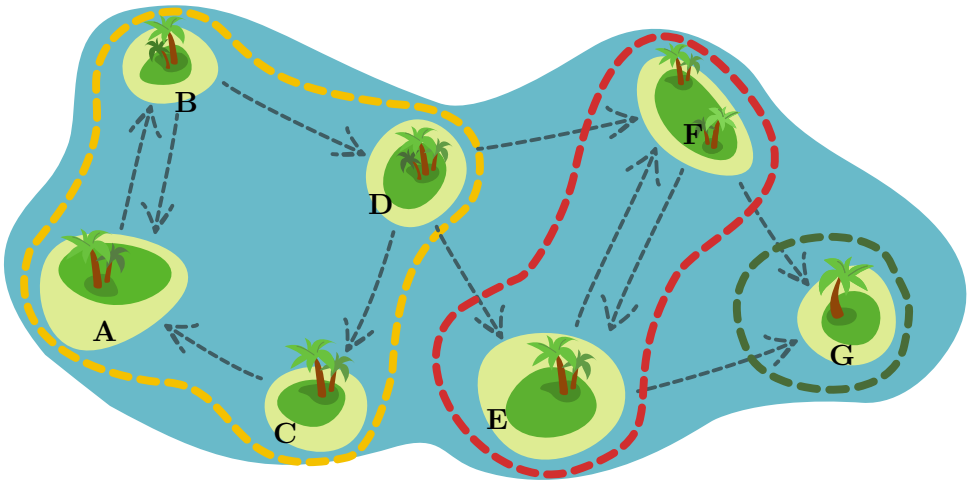
Lösung

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.

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

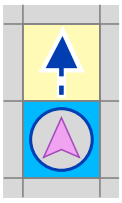
- *Gerichteter Graph*: `https://de.wikipedia.org/wiki/Gerichteter_Graph`
- *stark zusammenhängende Komponente*: `https://de.wikipedia.org/wiki/Zusammenhang_(Graphentheorie)`

8. Lefty II

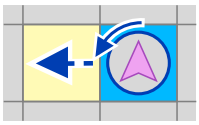
Roboter *Lefty* bewegt sich über ein Raster mit quadratischen Feldern. Zwischen Feldern kann es rote Mauern geben. Lefty soll das grüne Ziel erreichen.

Lefty kann sich auf genau zwei Arten bewegen:

Ein Feld vorwärts fahren

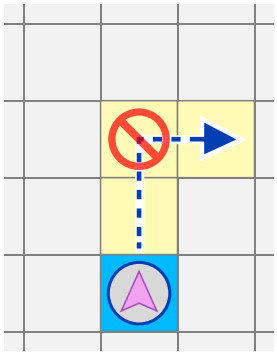


Nach links drehen und dann sofort ein Feld vorwärts fahren

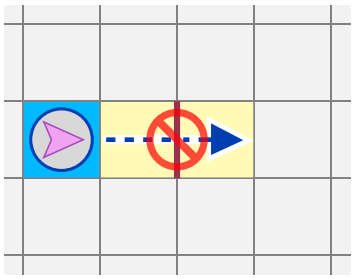


Lefty kann aber nicht alles. Zum Beispiel kann er

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

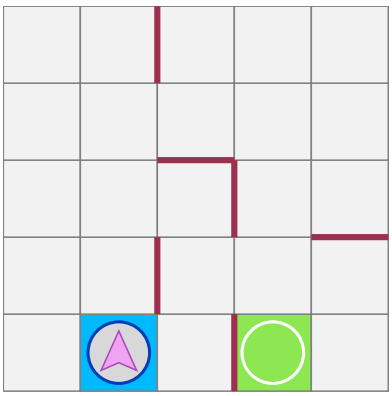


... **nicht** durch Mauern fahren.



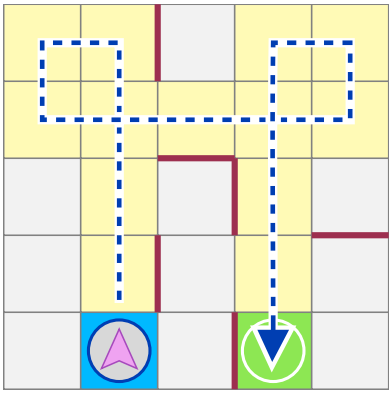
*Über welche Felder **muss** Lefty fahren, um das Ziel zu erreichen?*

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



Lösung

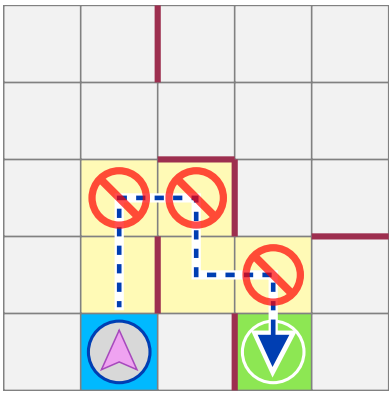
So ist es richtig:



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.

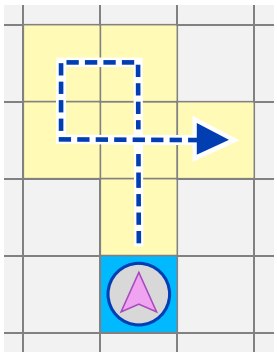


Dies ist Informatik!

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*

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

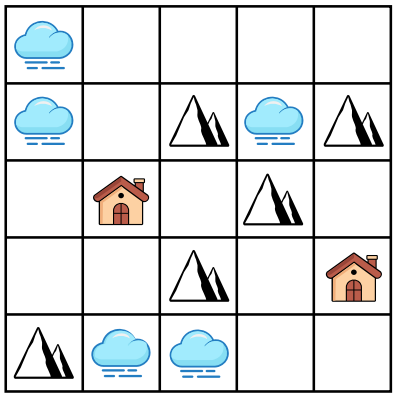
- Prozessor: `https://de.wikipedia.org/wiki/Prozessor`
- Befehlssatz: `https://de.wikipedia.org/wiki/Befehlssatz`
- CISC: `https://de.wikipedia.org/wiki/Complex_Instruction_Set_Computer`
- RISC: `https://de.wikipedia.org/wiki/Reduced_Instruction_Set_Computer`

9. Ein Tag im Nebel

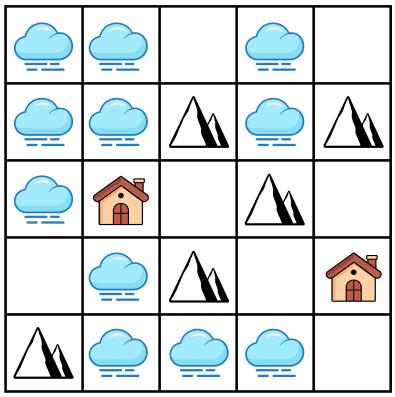
Im Land der Berge ist heute Nebel, und der breitet sich mit jeder Stunde weiter aus.

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.

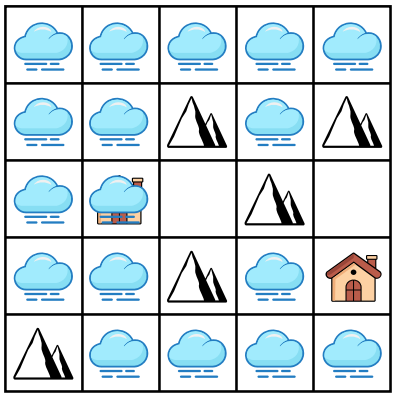
Ein Beispiel:



Sonnenaufgang

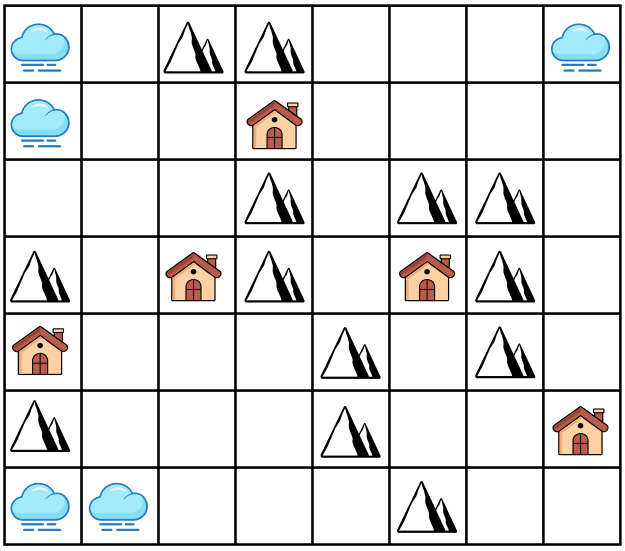


Nach 1 Stunde



Nach 2 Stunden

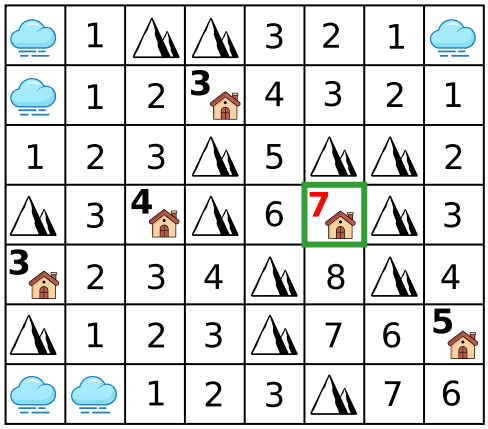
*Welches Haus im Land wird als **letztes** vom Nebel bedeckt?*



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.

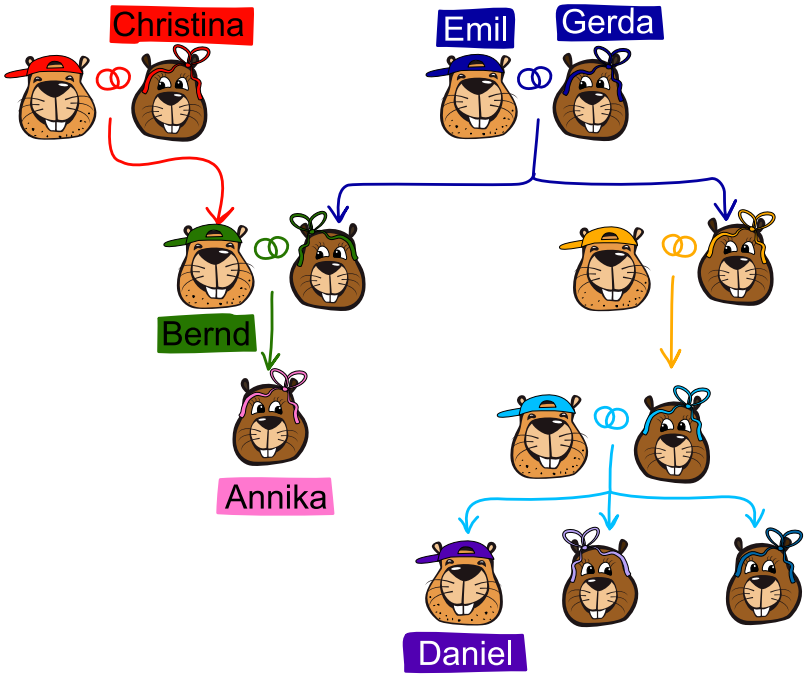
Zusammenhängende Gebiete sind auch für die Informatik interessant, und zwar in unterschiedlichen Bereichen. Ein einfarbiger Bereich in einem (Computer-)Bild ist ein zusammenhängendes Gebiet gleichfarbiger Pixel; man kann es mit einem *Floodfill-Algorithmus* bestimmen, der so ähnlich funktioniert wie 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`

10. 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:

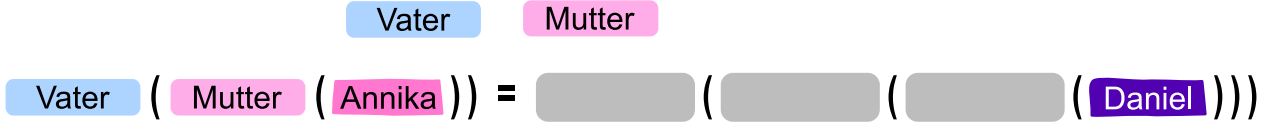
- `Vater(Annika) = Bernd`
- `Mutter(Bernd) = Christina`

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»

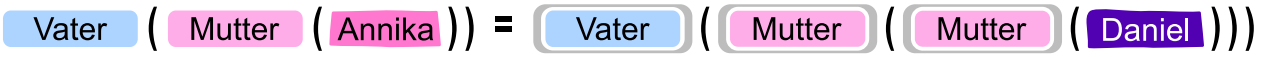
Nun hätte sie gerne eine Gleichung für ihre Verwandtschaft mit Daniel.

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.

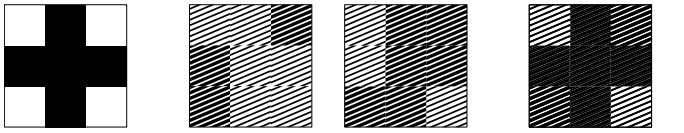
Diese Biberaufgabe zeigt, wie Funktionsaufrufe für komplexere Berechnungen zusammengesetzt (oder: «verschachtelt») werden können. In einem *verschachtelten Funktionsaufruf* dienen die Ergebnisse der inneren Funktionsaufrufe als Eingabe für äussere Aufrufe. Ein verschachtelter Funktionsaufruf wird von 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)`

11. 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.

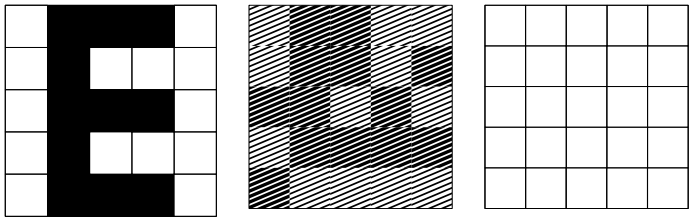


Geheimes Bild | Folie 1 | Folie 2 | Folien 1 und 2 übereinander

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 ▨).

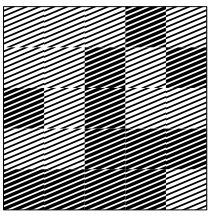
Für das folgende geheime Bild wurde Folie 1 bereits erzeugt. Erstelle nun Folie 2.



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

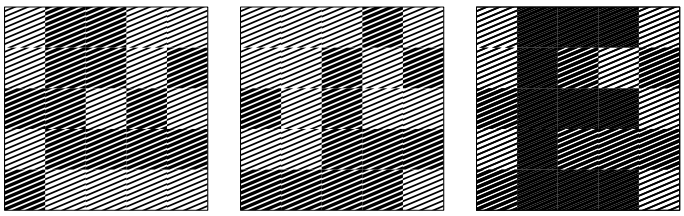
Lösung

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!

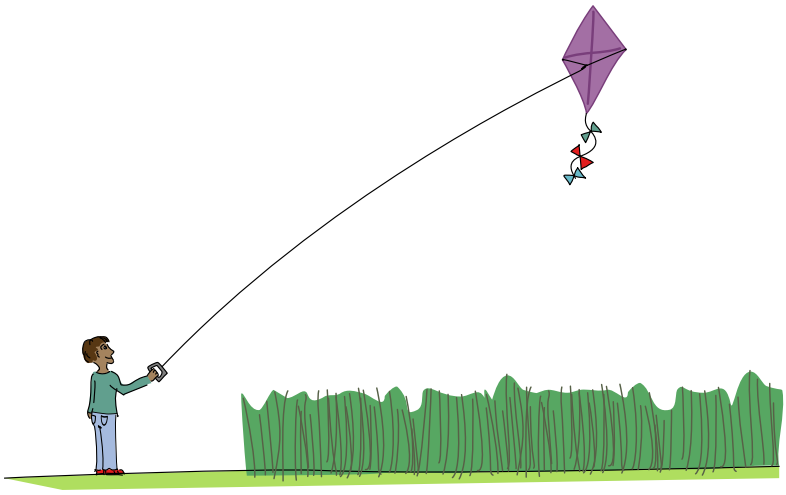
Der Kurierdienst verwendet ein *kryptografisches Verfahren*, das auf visueller Wahrnehmung beruht und daher *visuelle Kryptografie* genannt wird. Eine solche Technik wurde 1994 von den israelischen Wissenschaftlern Moni Naor und Adi Shamir entwickelt. In Bezug auf die einzelnen Folien ist das Verfahren 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.

Stichwörter und Webseiten

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

12. Der Drachen ist weg!

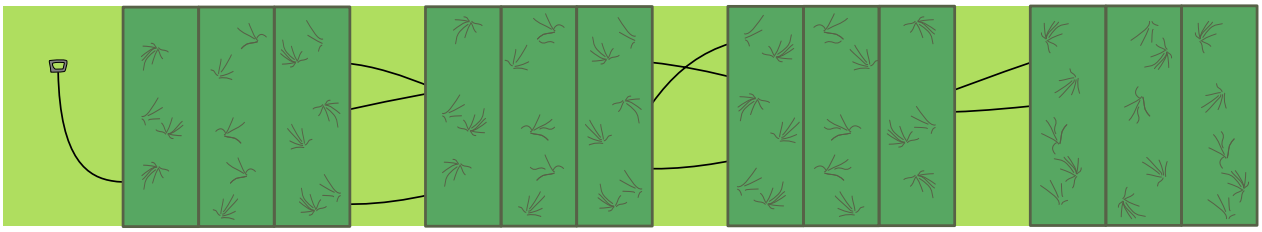


So ein Pech! Asterios hat seinen Drachen auf der Wiese verloren. Die Drachenschnur hat sich im hohen Gras verfangen, und es ist gar nicht so einfach, den Drachen wiederzufinden.

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.

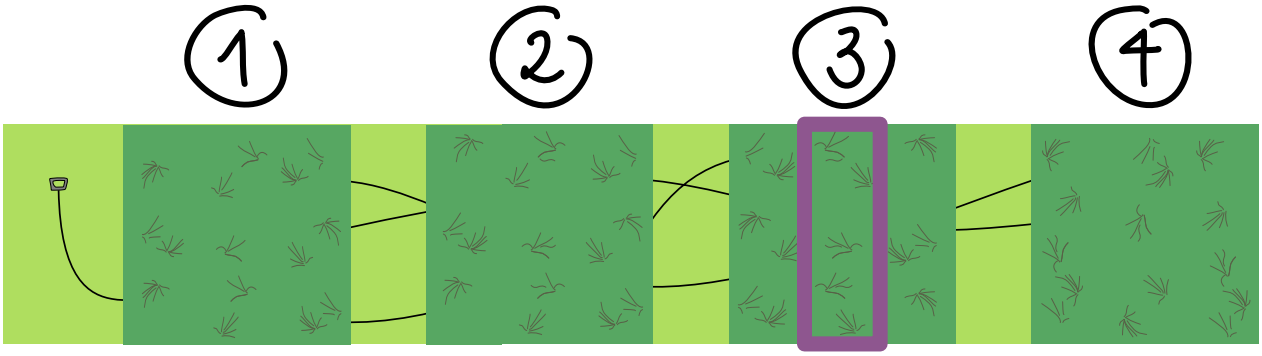
Welches Feld ist das?



Lösung

So ist es richtig:

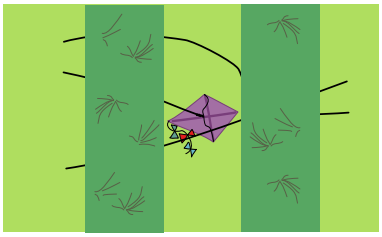
Asterios muss das lila Feld durchsuchen, um sicher zu wissen, wo der Drachen ist.



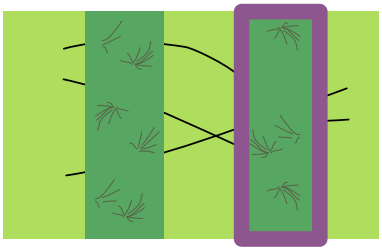
Die Lösung kannst du in zwei Schritten finden. Zuerst überlegst du dir, in welchem der vier Blöcke 1, 2, 3 und 4 der Drachen liegen muss. Denke daran, dass die Drachenschnur links beginnt und der Drachen am anderen Ende der Schnur hängt.

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.

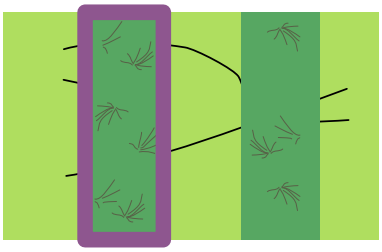
Wenn Asterios das mittlere Feld in Block 3 durchsucht, lassen sich drei Fälle unterscheiden:



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.

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`

13. Brennende Kerzen

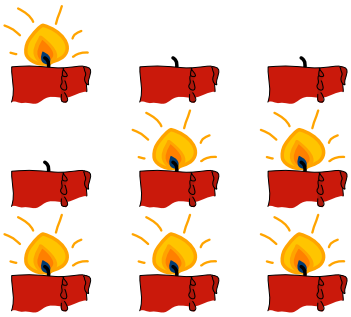
Es gibt eine Tradition, an den vier Sonntagen vor Weihnachten Kerzen anzuzünden: 1 Kerze am ersten Sonntag, 2 Kerzen am zweiten Sonntag und so weiter.

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

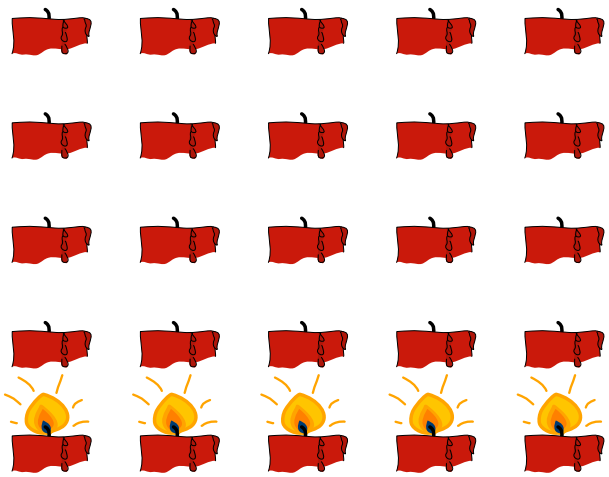


Auch mit fünf Sonntagen (und Kerzen) wäre es möglich.

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

Für den fünften Sonntag haben wir die Kerzen bereits angezündet.

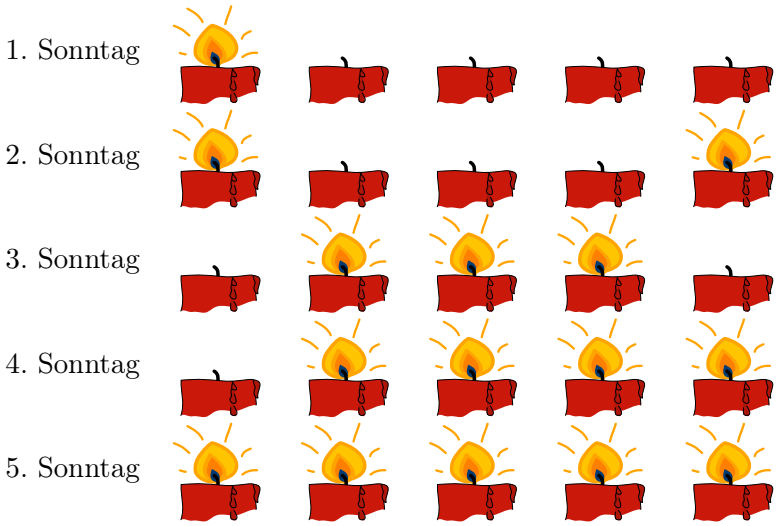
1. *Sonntag*
2. *Sonntag*
3. *Sonntag*
4. *Sonntag*
5. *Sonntag*



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:

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.

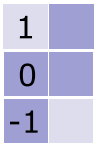
Die Entwicklung von Algorithmen zur Problemlösung ist eine der wichtigsten Aufgaben von Informatikerinnen und Informatikern. Wenn ein Algorithmus auf solider mathematischer Modellierung beruht, lassen sich seine gewünschten Eigenschaften leichter beweisen als ohne eine solche Modellierung. Für den obigen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

Stichwörter und Webseiten

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

14. 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:

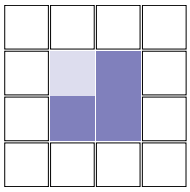


1, falls das Pixel heller ist als sein rechtes Nachbarpixel.

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

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

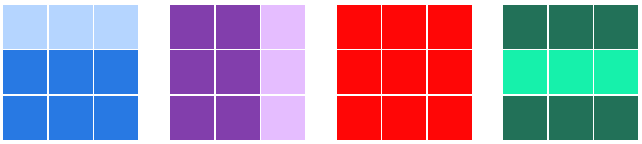
Hier siehst du ein Bild aus vier Pixeln (plus die zusätzlichen weissen Pixel) und seine Helligkeitskarte.



1	-1
0	-1

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

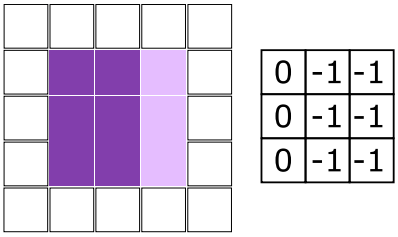
*Welches der Bilder hat als einziges eine **andere** Helligkeitskarte?*



A) B) C) D)

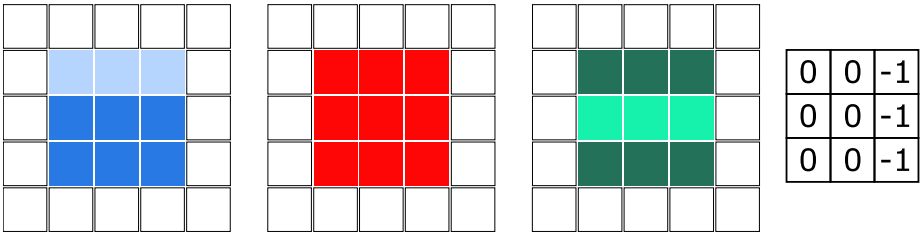
Lösung

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.

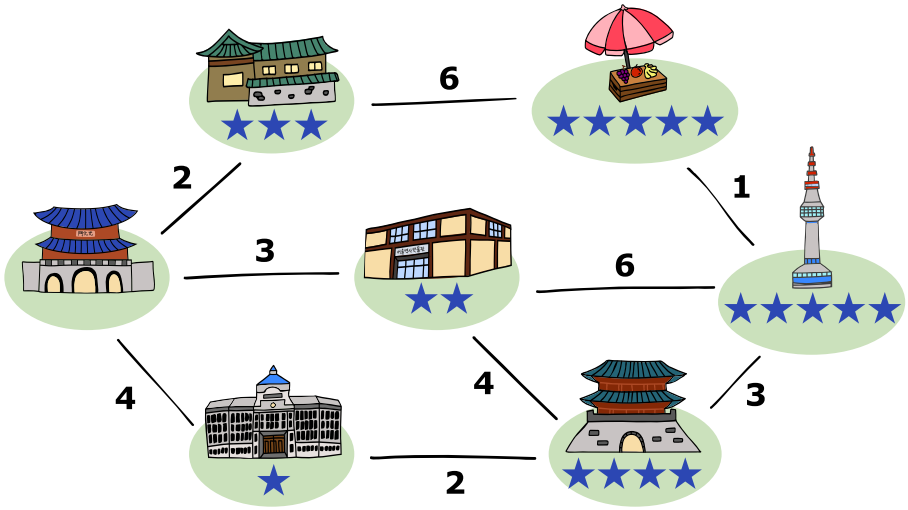
Sogenannte *Convolutional Neural Networks* (CNN), häufig zentrale Bestandteile von KI-Systemen, nutzen das Konzept der Faltung zur Bilderkennung. Um komplexe Objekte in einem Bild zu erkennen, lernt ein CNN, selbst geeignete Filter zu entwickeln, mit denen es das Bild faltet. Damit kann es beispielsweise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

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`

15. 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.



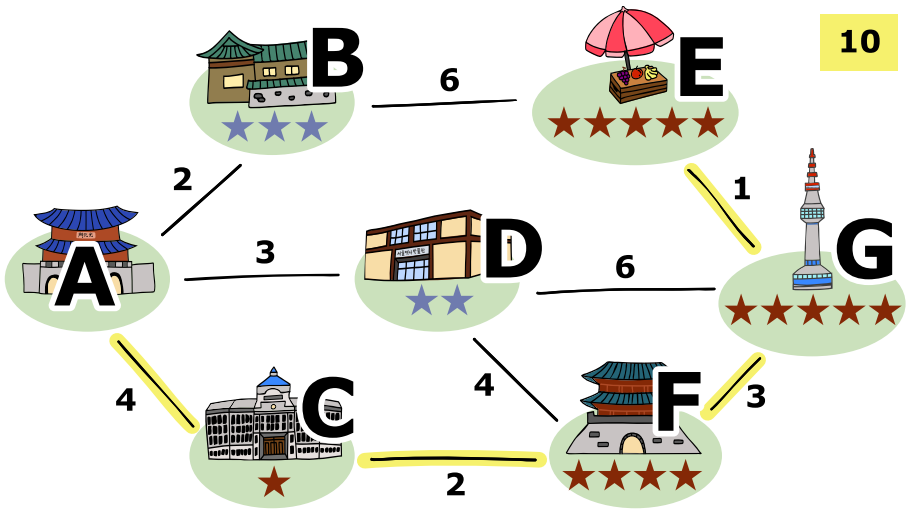
Lotte besucht zuerst den Palast  . Von dort aus möchte sie mit den Bussen weitere Orte besuchen. Lotte hat eine Fahrkarte, mit der sie höchstens 10 Kilometer weit fahren kann. Damit möchte sie über die Verbindungen Orte erreichen, die insgesamt möglichst viele Sterne haben! Sie besucht einen Ort natürlich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

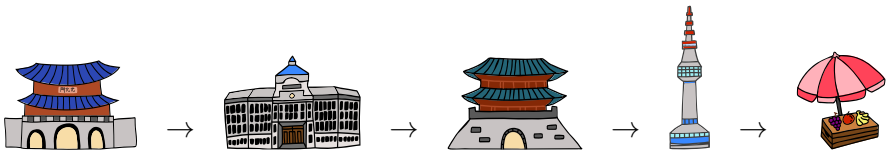
Eine Route nennen wir *gültig*, wenn sie höchstens 10 km lang ist. Eine gültige Route führt beispielsweise

- 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)
```


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`

16. Bergseen

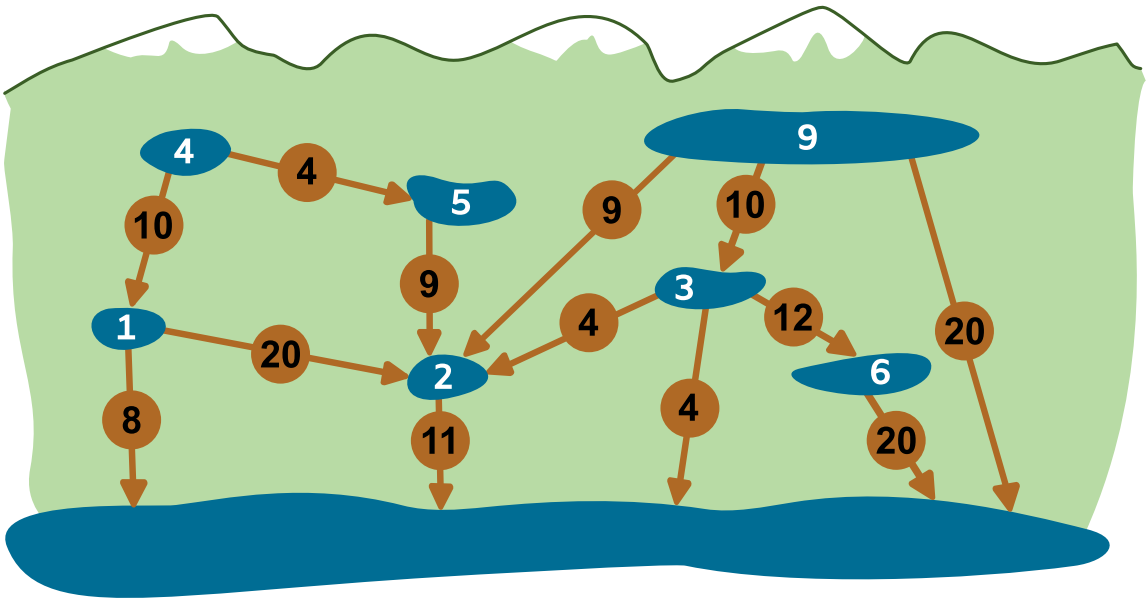
Am Bergmassiv hinter dem Stausee gibt es mehrere kleine Bergseen. Bei starkem Regen könnten sie überlaufen, und das ist gefährlich. Deshalb sollen zwischen einigen Seen Kanäle gebaut werden. Diese Kanäle sollen alles überschüssige Wasser aus den Bergseen in den Stausee ableiten können. Gleichzeitig soll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

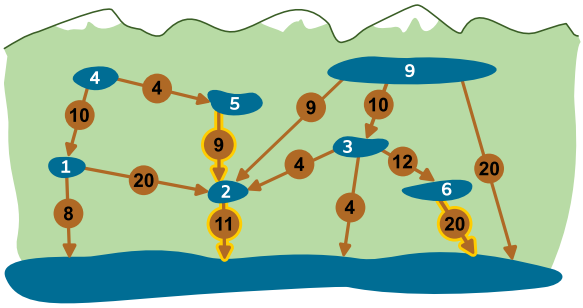
Beachte: Wenn ein Kanal Wasser von einem kleinen Bergsee in einen zweiten ableitet, sammelt sich im zweiten See das überschüssige Wasser aus beiden Seen.

An welchen Stellen sollen Kanäle gebaut werden?

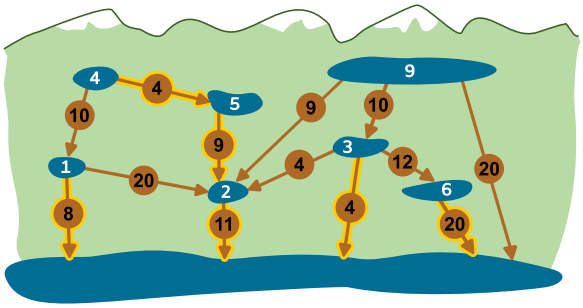


Lösung

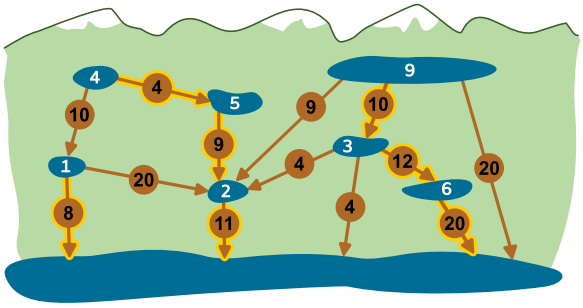
So ist es richtig:



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.

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`

17. 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:

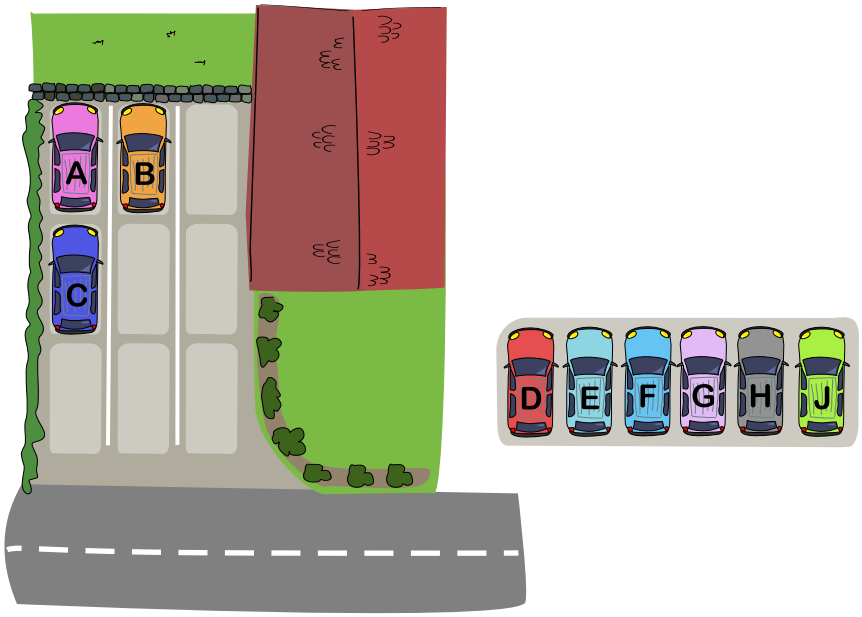
Anja, **B**eate, **C**lara, **D**avid, **E**lia, **F**rank, **G**abi, **H**arald und zuletzt **J**ulia.

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.

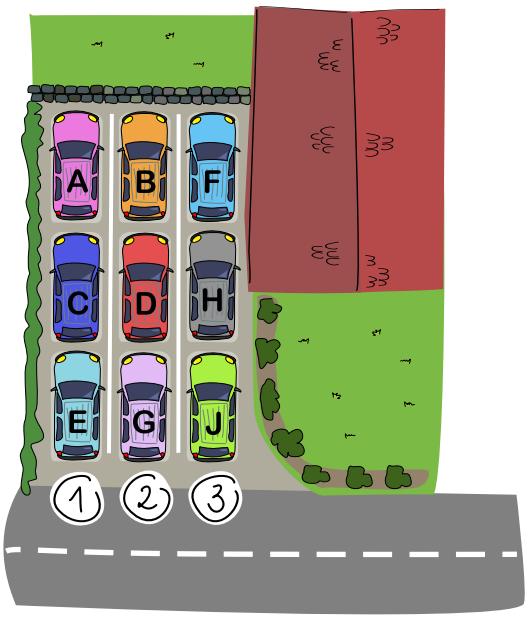


Zeige den Gästen, wie sie so parkieren können!

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.

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`

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.

18. 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:

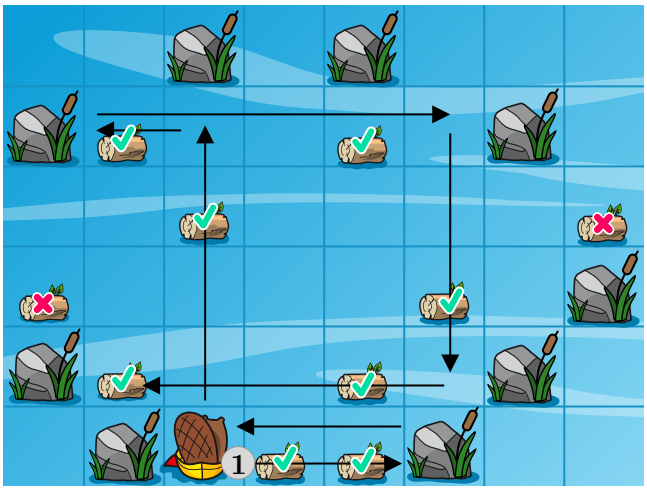
Anweisung	**Beschreibung**
`turnRight()` / `turnLeft()`	Linus dreht sich an Ort um 90 Grad nach rechts / links.
`paddle()`	Linus paddelt solange, bis er direkt vor einem Fels steht. Ist er auf einem Feld mit einem Baumstamm, sammelt er diesen auf.

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.

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`

A. Aufgabenautoren

James Atlas
Masiar Babazadeh
Wilfried Baumann
Leonardo Cavalcante
Špela Cerar
Andrew Csizmadia
Christian Datzko
Diane Dowling
Nora A. Escherle
Gerald Futschek
Silvan Horvath
Alisher Ikramov
Thomas Ioannou
Asterios Karagiannis
Blaž Kelvišar
David Khachatryan
Doyong Kim
Jihye Kim
Dong Yoon Kim
Vaidotas Kinčius
Stefan Koch
Lukas Lehner
Gunwoong Lim
Linda Mannila
Anna Morpurgo
A-Yeong Park
Suchan Park
Gabriela Gomez Pasquali
Jean-Philippe Pellet
Zsuzsa Pluhár
Wolfgang Pohl
Pedro Ribeiro
Kirsten Schlüter
Margareta Schlüter
Dirk Schmerenbeck
Jacqueline Staub
Susanne Thut
Christine Vender
Florentina Voboril
Michael Weigend
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/`

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/