INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

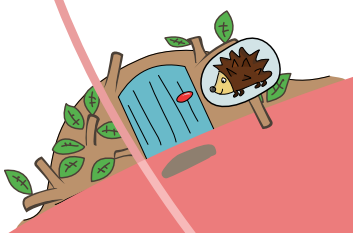
Aufgaben und Lösungen 2025

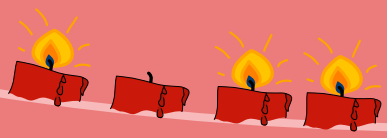
Schuljahre 11/12/13

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

Herausgeber:

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





01010011010101100100100
10000001001011010101001
10101001101001010010101
00101101010100110101001
01001001010010010010001

SV!A

www.svia-ssie-ssii.ch
schweizerischerverein für informatikind
erausbildung//société suisse pour l'infor
matiquedans l'enseignement//società sviz
zera per l'informatica nell'insegnamento

HASLERSTIFTUNG ETH zürich | ABZ Ausbildungs- und Beratungszentrum für Informatikunterricht Scuola universitaria professionale della Svizzera italiana SUPSI hep/ PH LUZERN Pädagogische Hochschule

Mitarbeit Informatik-Biber 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:

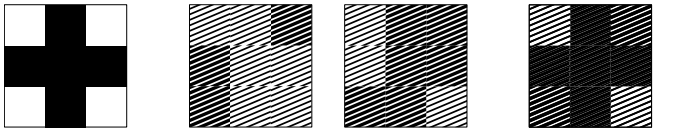
Schweizerischer Verein für Informatik in der Ausbildung
SVIA-SSIE-SSII
Informatik-Biber
Nora A. Escherle

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

Inhaltsverzeichnis

1. 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.

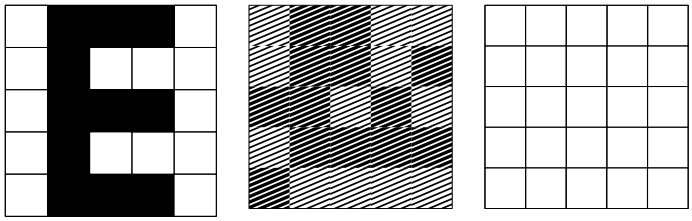


Geheimes Bild | Folie 1 | Folie 2 | Folien 1 und 2 übereinander

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 ▨).

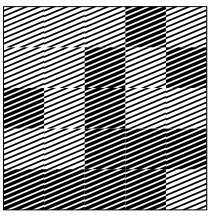
Für das folgende geheime Bild wurde Folie 1 bereits erzeugt. Erstelle nun Folie 2.



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

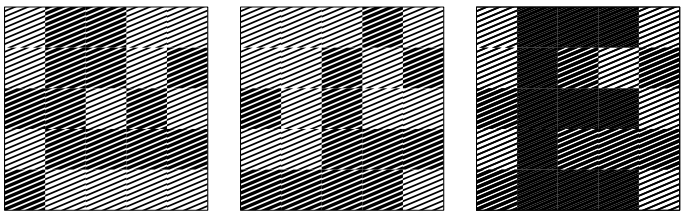
Lösung

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!

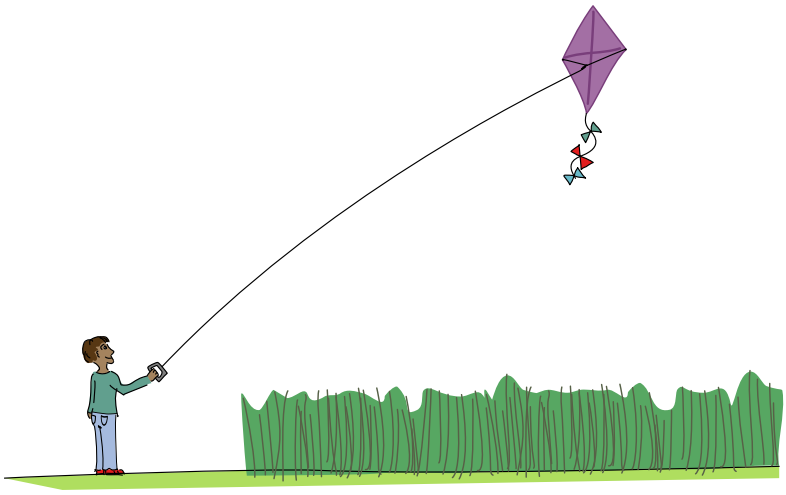
Der Kurierdienst verwendet ein *kryptografisches Verfahren*, das auf visueller Wahrnehmung beruht und daher *visuelle Kryptografie* genannt wird. Eine solche Technik wurde 1994 von den israelischen Wissenschaftlern Moni Naor und Adi Shamir entwickelt. In Bezug auf die einzelnen Folien ist das Verfahren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

2. Der Drachen ist weg!

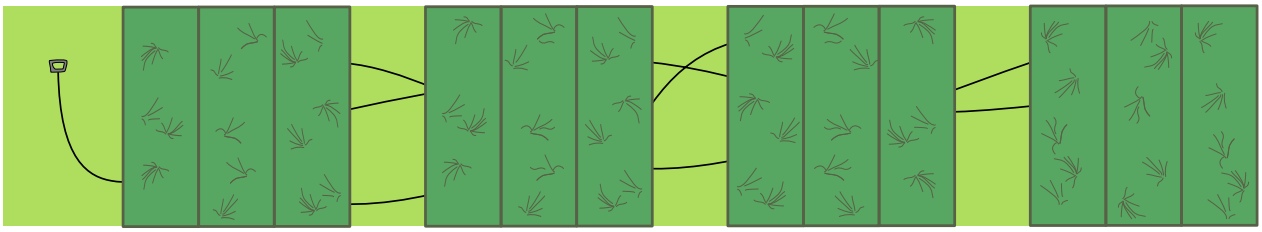


So ein Pech! Asterios hat seinen Drachen auf der Wiese verloren. Die Drachenschnur hat sich im hohen Gras verfangen, und es ist gar nicht so einfach, den Drachen wiederzufinden.

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.

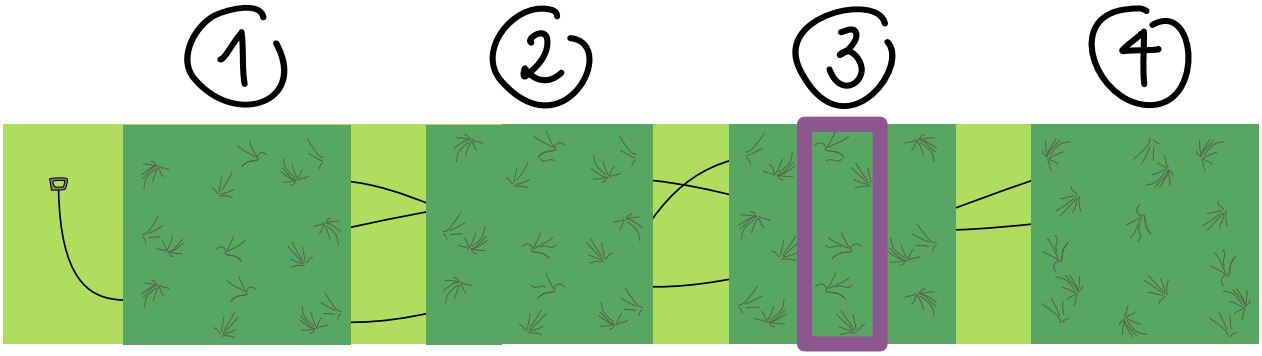
Welches Feld ist das?



Lösung

So ist es richtig:

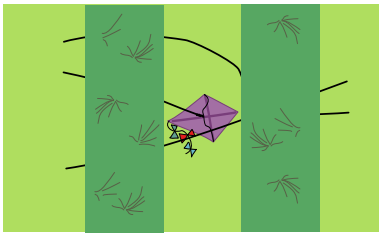
Asterios muss das lila Feld durchsuchen, um sicher zu wissen, wo der Drachen ist.



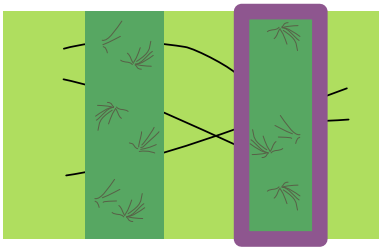
Die Lösung kannst du in zwei Schritten finden. Zuerst überlegst du dir, in welchem der vier Blöcke 1, 2, 3 und 4 der Drachen liegen muss. Denke daran, dass die Drachenschnur links beginnt und der Drachen am anderen Ende der Schnur hängt.

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.

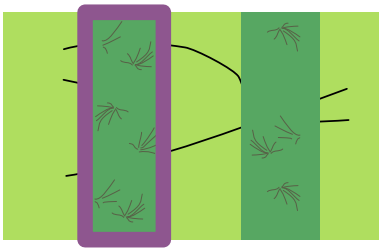
Wenn Asterios das mittlere Feld in Block 3 durchsucht, lassen sich drei Fälle unterscheiden:



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`

3. Brennende Kerzen

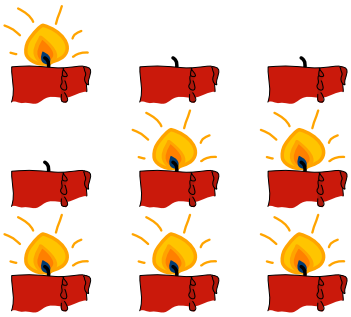
Es gibt eine Tradition, an den vier Sonntagen vor Weihnachten Kerzen anzuzünden: 1 Kerze am ersten Sonntag, 2 Kerzen am zweiten Sonntag und so weiter.

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

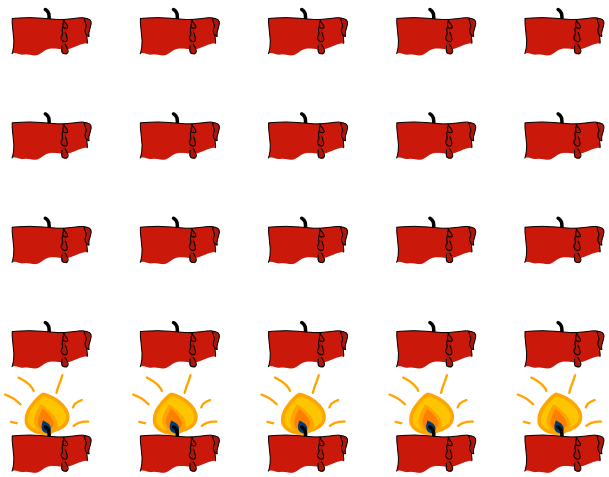


Auch mit fünf Sonntagen (und Kerzen) wäre es möglich.

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

Für den fünften Sonntag haben wir die Kerzen bereits angezündet.

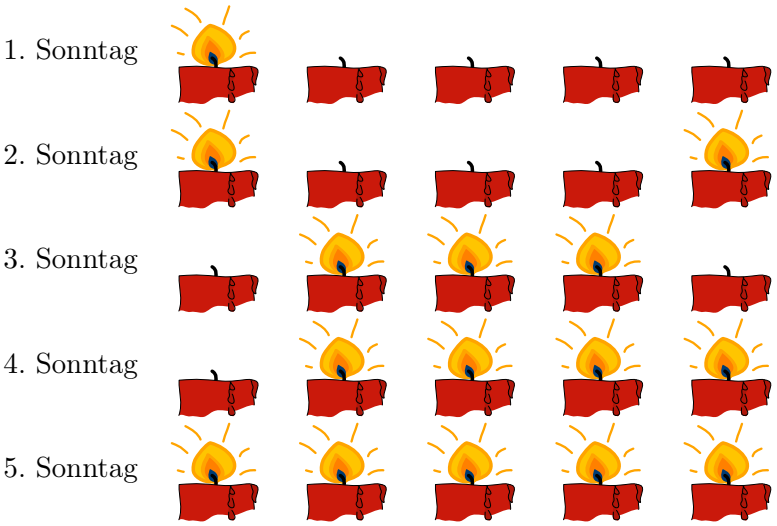
1. *Sonntag*
2. *Sonntag*
3. *Sonntag*
4. *Sonntag*
5. *Sonntag*



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.

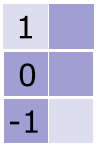
Die Entwicklung von Algorithmen zur Problemlösung ist eine der wichtigsten Aufgaben von Informatikerinnen und Informatikern. Wenn ein Algorithmus auf solider mathematischer Modellierung beruht, lassen sich seine gewünschten Eigenschaften leichter beweisen als ohne eine solche Modellierung. Für den obigen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

Stichwörter und Webseiten

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

4. 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:

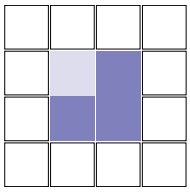


1, falls das Pixel heller ist als sein rechtes Nachbarpixel.

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

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

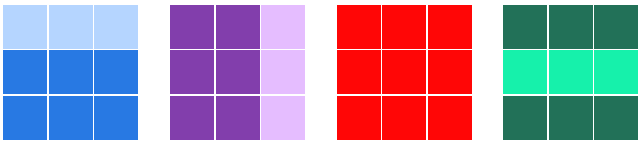
Hier siehst du ein Bild aus vier Pixeln (plus die zusätzlichen weissen Pixel) und seine Helligkeitskarte.



1	-1
0	-1

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

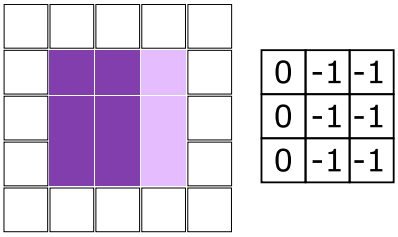
*Welches der Bilder hat als einziges eine **andere** Helligkeitskarte?*



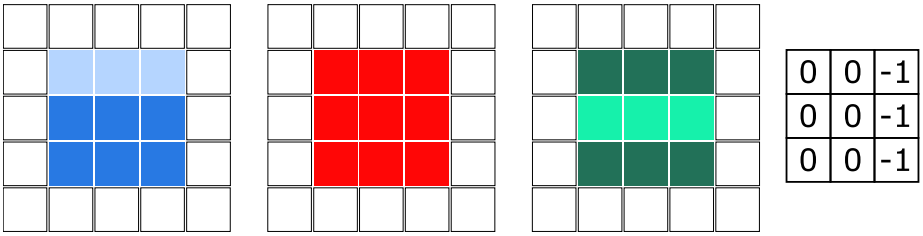
A) B) C) D)

Lösung

Antwort B ist richtig:



Die übrigen drei Bilder haben alle die gleiche Helligkeitskarte:



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.

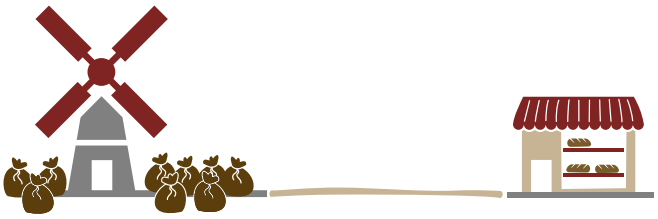
Die Helligkeitskarte in dieser Biberaufgabe ist ähnlich zum Konzept einer *Faltung* zwischen dem Bild und einem *Filter*. Je nach Filter können durch eine solche Faltung verschiedene strukturelle Eigenschaften des Bildes erkennbar gemacht werden, wie beispielsweise Ecken, Kanten oder Bereiche uniformer 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`

5. Mehltransport



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

Ihre Leistungen beim Mehl holen sind unterschiedlich:

Albert holt 13 kg Mehl in 1 Stunde.

Marco holt 5 kg Mehl in 30 Minuten.

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

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

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

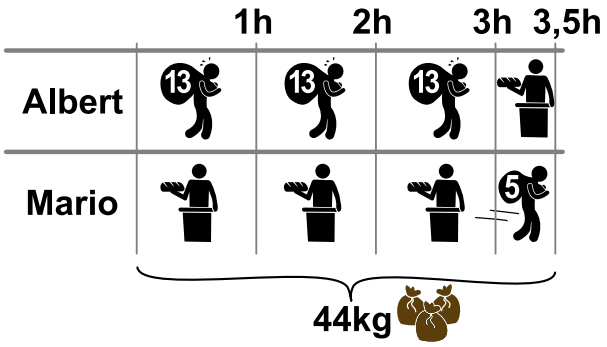
Lösung

Antwort A ist richtig.

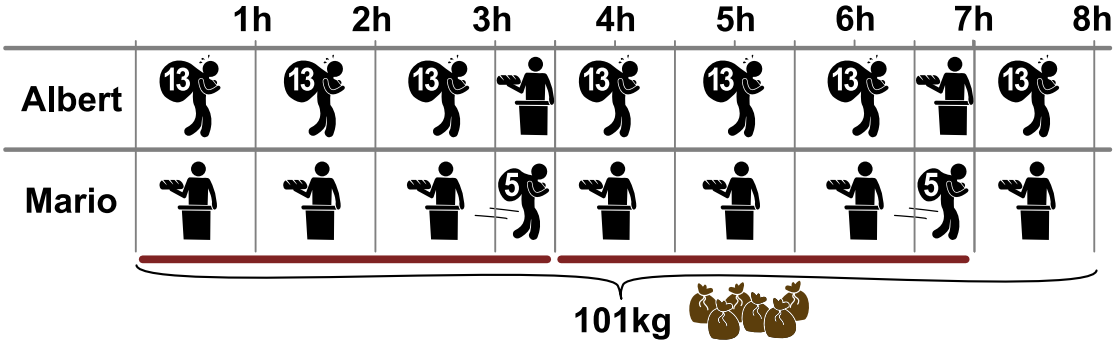
Wir wissen:

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

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



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



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

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

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

Dies ist Informatik!

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

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

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

Stichwörter und Webseiten

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

6. Schwarz - Weiss

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

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

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

`W`

`xWS`

`xxSWS`

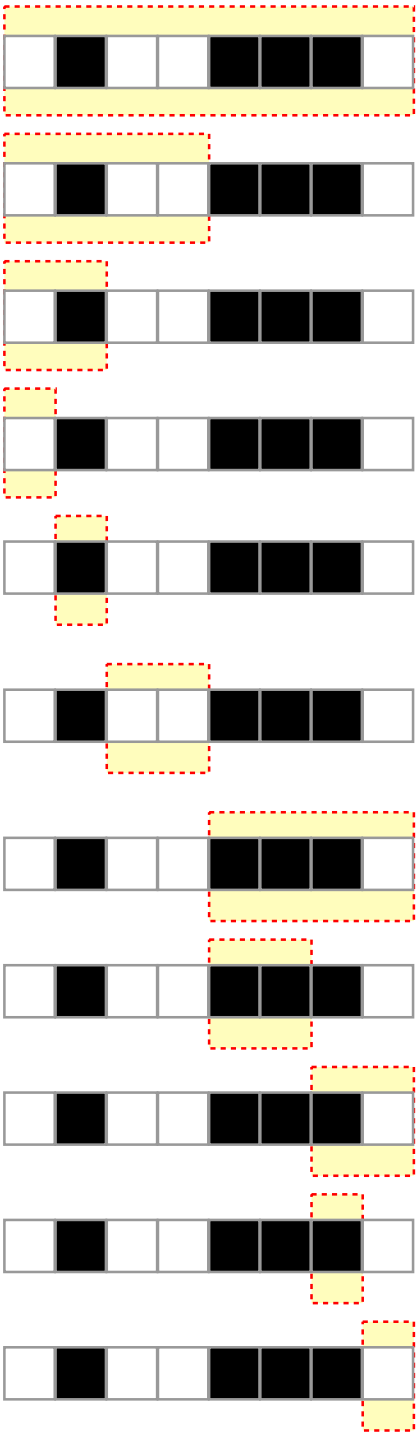
`xSxWxSW`

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

Lösung

So ist es richtig: `xxxWSWxSxSW`.

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



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

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

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

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

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

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

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

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

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

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

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

Dies ist Informatik!

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

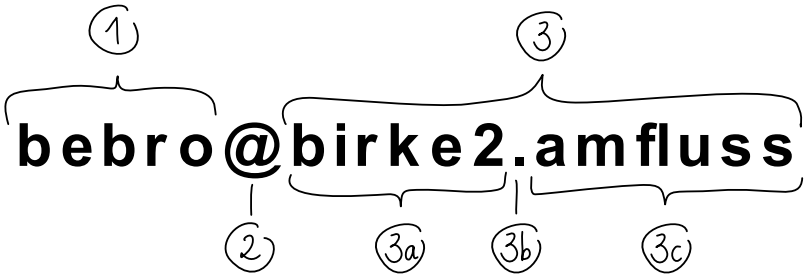
Sarahs Algorithmus beschreibt bzw. *kodiert* die Kästchen-Folgen mit Buchstaben. Eine Kodierung muss umkehrbar sein; es muss also ein Verfahren geben, die Buchstaben wieder in die originale Kästchenfolge umzuwandeln. Wenn die Buchstaben-Beschreibung um die Anzahl der Kästchen in der beschriebenen Folge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

Stichwörter und Webseiten

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

7. Biber-Mail-Adressen

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



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

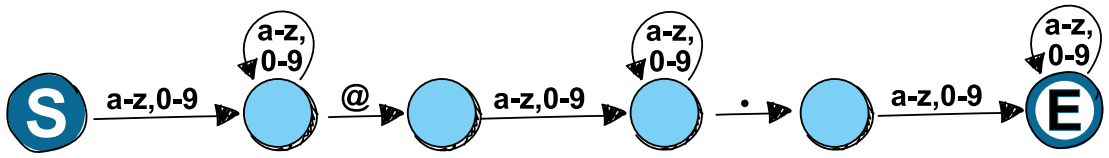
«Nicht leere» Zeichenfolgen bestehen aus mindestens einem Zeichen. «bebro@birke2.amfluss» ist ein Beispiel für eine Biber-Mail-Adresse.

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

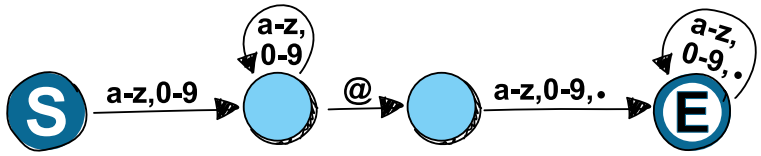
Ein System untersucht eine Zeichenfolge Zeichen für Zeichen, von links nach rechts, und ist zu Beginn im Zustand S. Der nächste Zustand hängt vom aktuell untersuchten Zeichen ab: Das System folgt dem Zustandswechsel, der zum aktuellen Zeichen passt. Ist das System nach dem letzten Zeichen im Zustand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

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

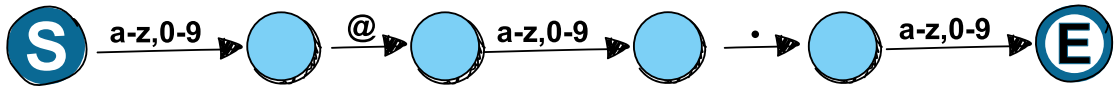
A)



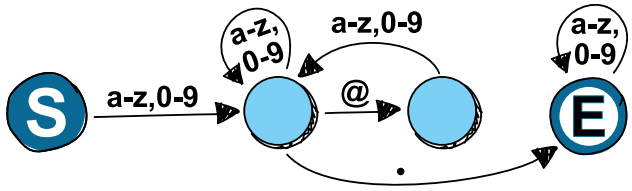
B)



C)

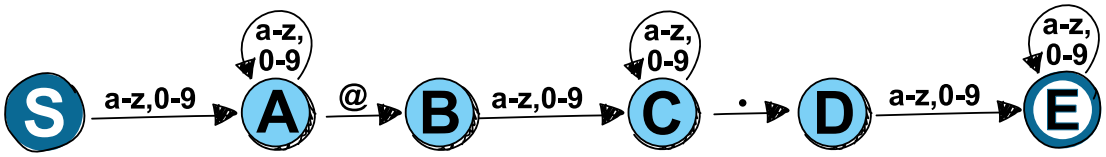


D)



Lösung

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



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

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

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

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

Dies ist Informatik!

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

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

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

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

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

Stichwörter und Webseiten

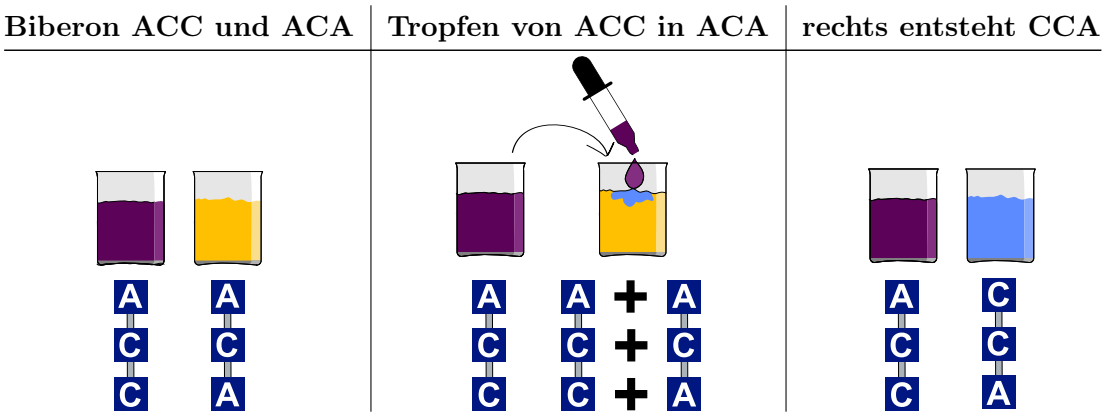
- Endlicher Automat: `https://de.wikipedia.org/wiki/Endlicher_Automat`
- Reguläre Grammatik: `https://de.wikipedia.org/wiki/Reguläre_Grammatik`
- Reguläre Sprache: `https://de.wikipedia.org/wiki/Reguläre_Sprache`
- Regulärer Ausdruck: `https://de.wikipedia.org/wiki/Regulärer_Ausdruck`
- Chomsky-Hierarchie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

8. Biberone

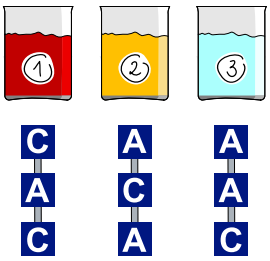
Ein Biberon ist ein besonderer chemischer Stoff: Seine Moleküle sind jeweils eine Folge von genau drei Bausteinen der Typen A und C. Diese Folge wird auch als Name des Biberons verwendet, zum Beispiel: ACA.

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

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

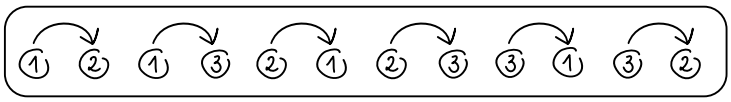


Im Labor gibt es drei Behälter 1, 2 und 3 mit den Biberonen CAC, ACA und AAC:



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

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

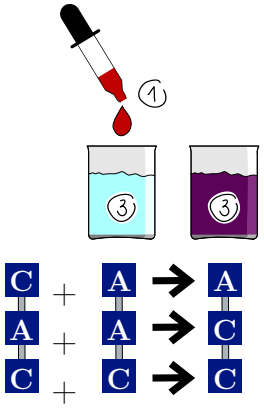


Lösung

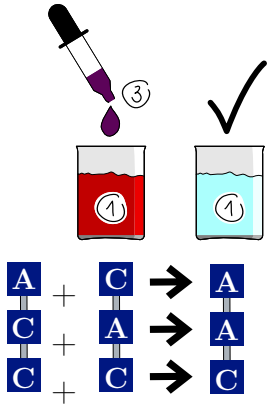
So ist es richtig:

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

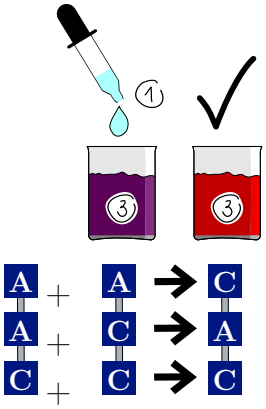
Wir veranschaulichen die erste Antwort 1→3, 3→1, 1→3. Zu Beginn sind die Stoffe in den Behältern: 1: CAC, 2: ACA, 3: AAC.



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



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



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

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

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

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

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

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

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

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

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

Dies ist Informatik!

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

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

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

Stichwörter und Webseiten

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

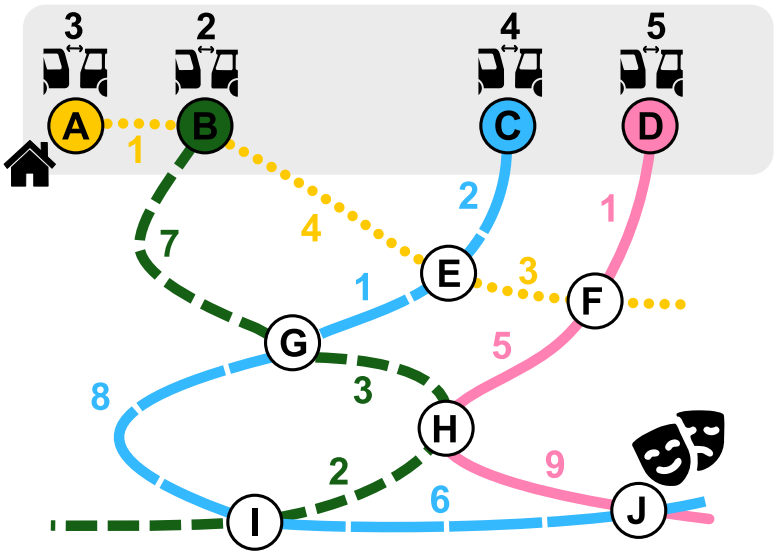
9. ÖV

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

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

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

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



Welche Haltestellen liegen auf dieser Route?

Lösung

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

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

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

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

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

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

Dies ist Informatik!

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

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

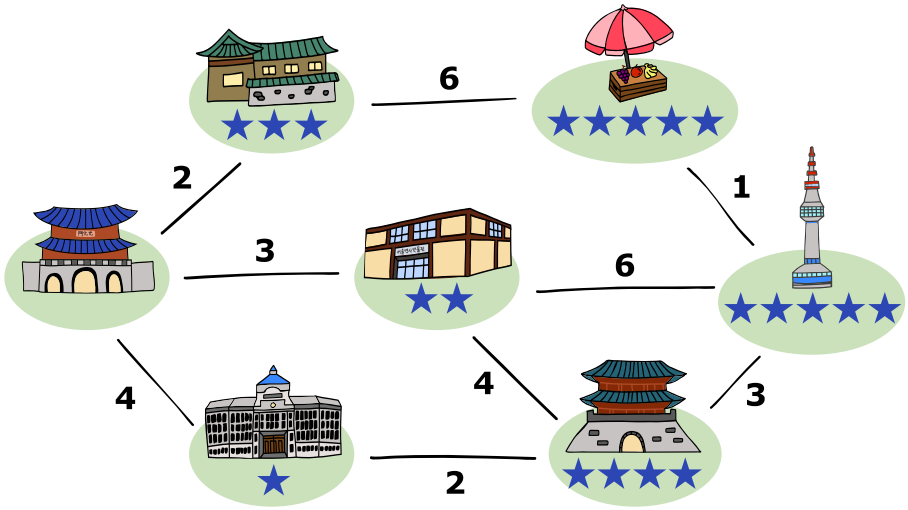
Diese Strategie heisst in der Informatik *Dynamische Programmierung*. Man kann sie für Optimierungsprobleme nutzen, wie für die Suche nach Marcus' frühester Ankunftszeit beim Theater in dieser Biberaufgabe. Dynamische Programmierung funktioniert genau dann, wenn das *Optimalitätsprinzip von Bellman*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

Stichwörter und Webseiten

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

10. 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.



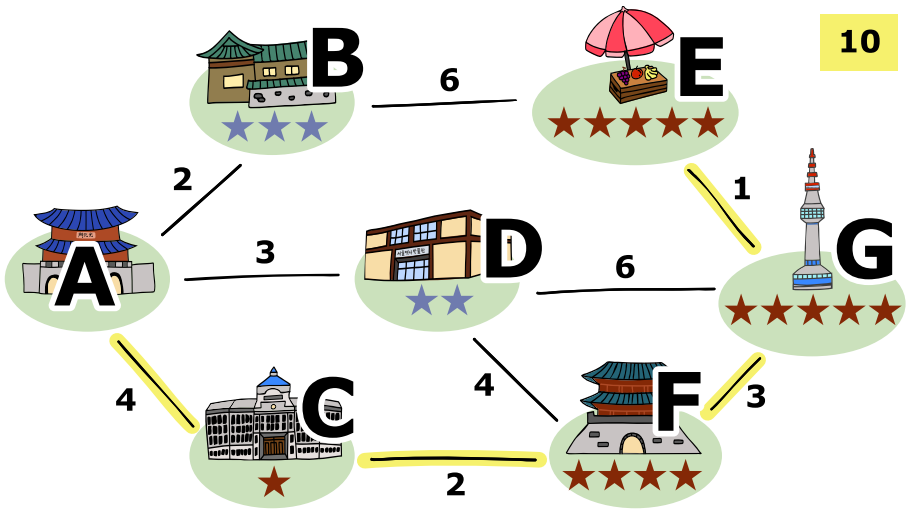
Lotte besucht zuerst den Palast  . Von dort aus möchte sie mit den Bussen weitere Orte besuchen. Lotte hat eine Fahrkarte, mit der sie höchstens 10 Kilometer weit fahren kann. Damit möchte sie über die Verbindungen Orte erreichen, die insgesamt möglichst viele Sterne haben! Sie besucht einen Ort natürlich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

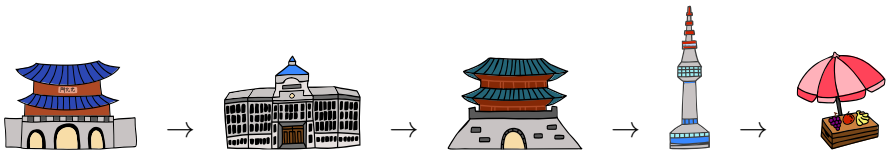
Eine Route nennen wir *gültig*, wenn sie höchstens 10 km lang ist. Eine gültige Route führt beispielsweise

- 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`

11. Bergseen

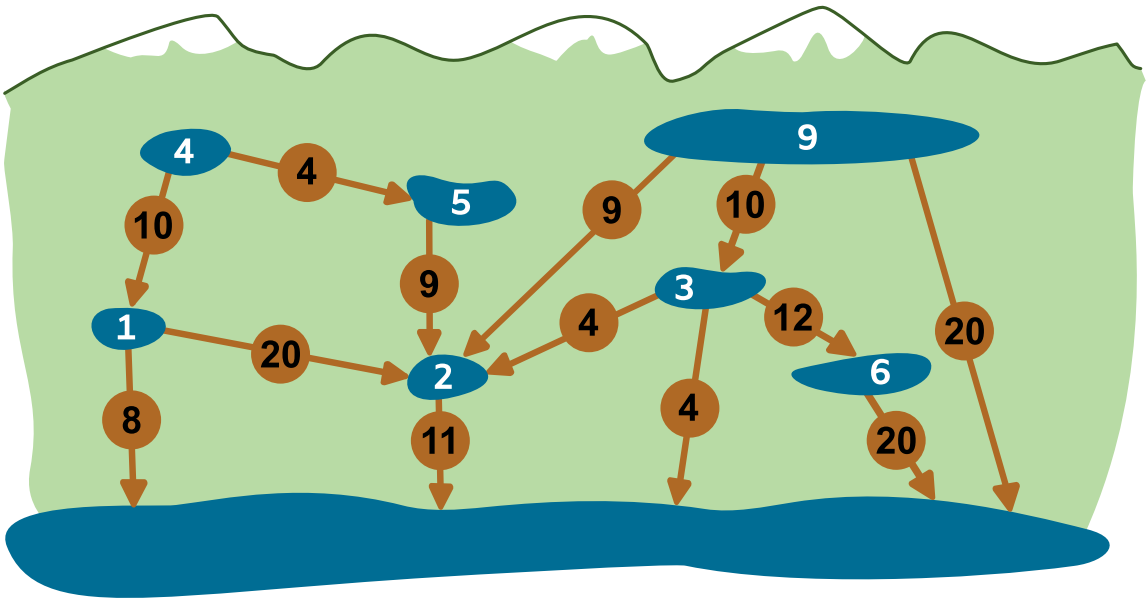
Am Bergmassiv hinter dem Stausee gibt es mehrere kleine Bergseen. Bei starkem Regen könnten sie überlaufen, und das ist gefährlich. Deshalb sollen zwischen einigen Seen Kanäle gebaut werden. Diese Kanäle sollen alles überschüssige Wasser aus den Bergseen in den Stausee ableiten können. Gleichzeitig soll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

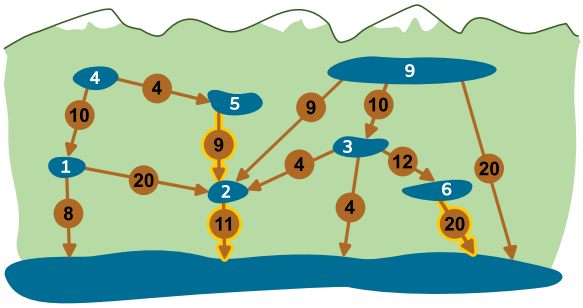
Beachte: Wenn ein Kanal Wasser von einem kleinen Bergsee in einen zweiten ableitet, sammelt sich im zweiten See das überschüssige Wasser aus beiden Seen.

An welchen Stellen sollen Kanäle gebaut werden?

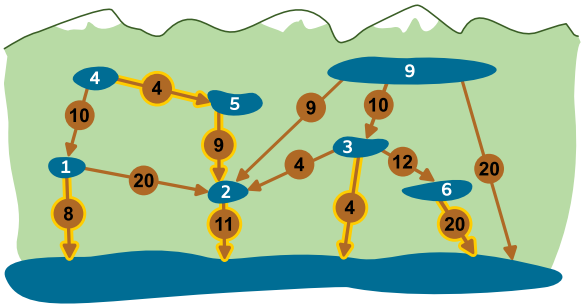


Lösung

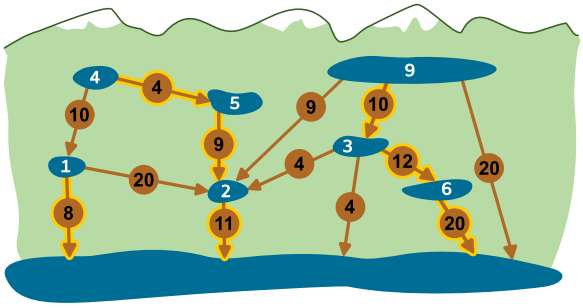
So ist es richtig:



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`

12. 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:

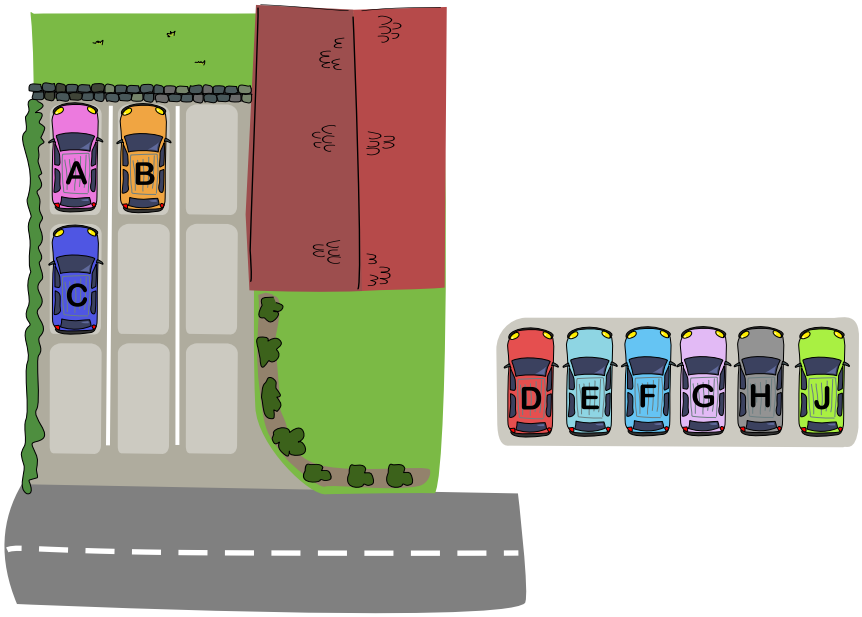
Anja, **B**eate, **C**lara, **D**avid, **E**lia, **F**rank, **G**abi, **H**arald und zuletzt **J**ulia.

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.

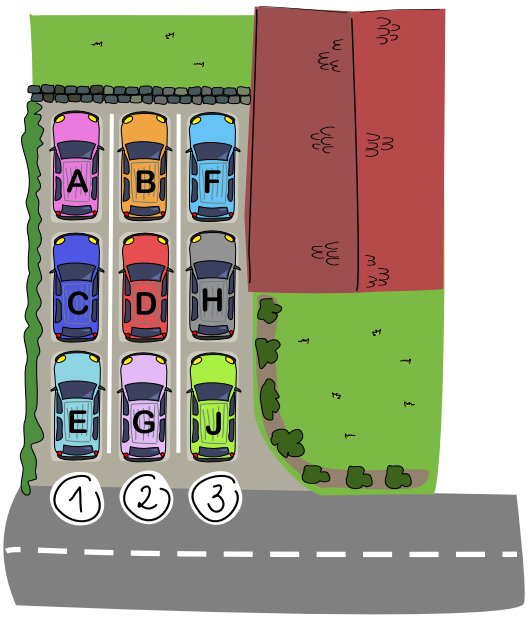


Zeige den Gästen, wie sie so parkieren können!

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`

13. Biber Jones

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

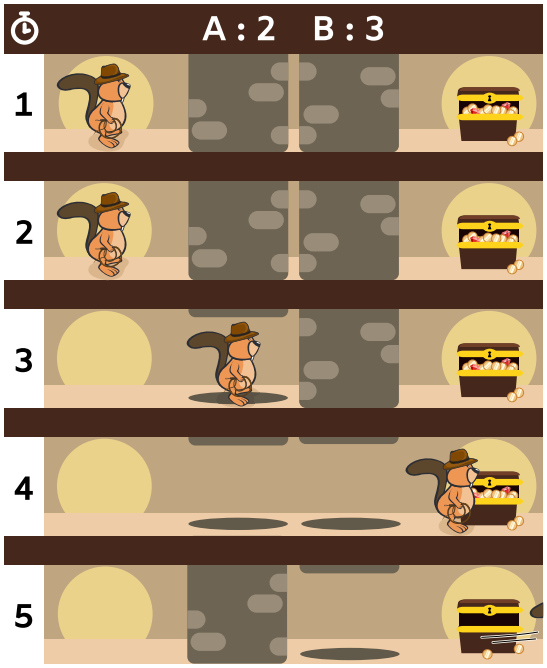
Jedoch ist jeder Gang durch eine Reihe von Fallblöcken gesichert. Am Anfang sind alle Blöcke unten. Sobald jemand den Gang betritt, beginnen die Blöcke sich zu bewegen. Jeder Block bewegt sich in festem Takt nach oben und unten. Zum Beispiel schnellt ein Block mit Takt 2 nach 2 Minuten hoch und kracht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

Jones steht vor seiner ersten Herausforderung:

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

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

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

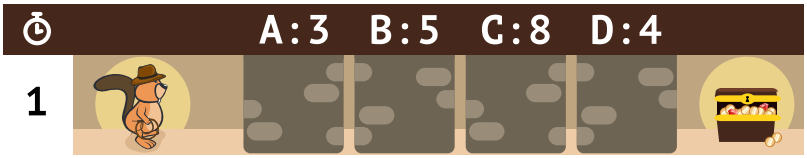


Jones bemerkt, dass er auch mit weniger Anweisungen dem Schatz erreicht hätte, und zwar gleich schnell:

```
wait(3)
goto_treasure
```

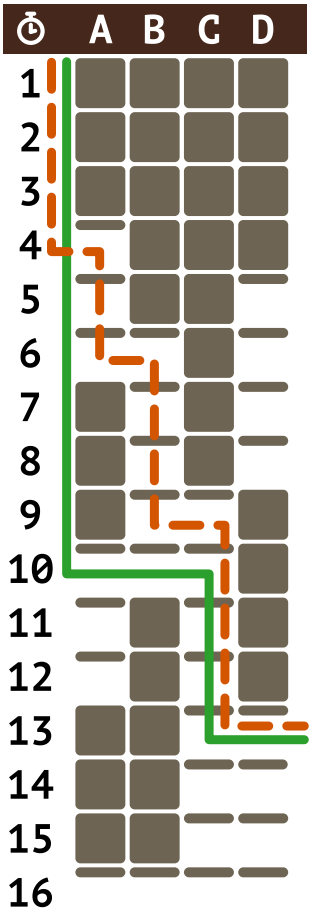
Er kommt zum nächsten Gang. Der hat vier Blöcke, mit Takt 3, 5, 8 und 4.

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



Lösung

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



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

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

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

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

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

Dies ist Informatik!

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

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

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

Stichwörter und Webseiten

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

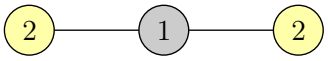
14. Prüfungsplan

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

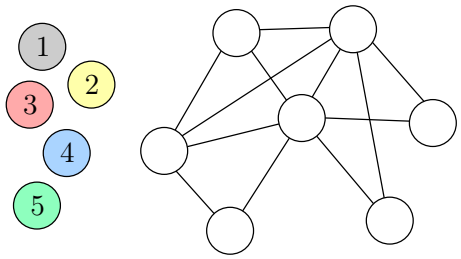
Um die Planung zu erleichtern, werden die Tageskonflikte in einem Diagramm aus Kreisen und Linien dargestellt:

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

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



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

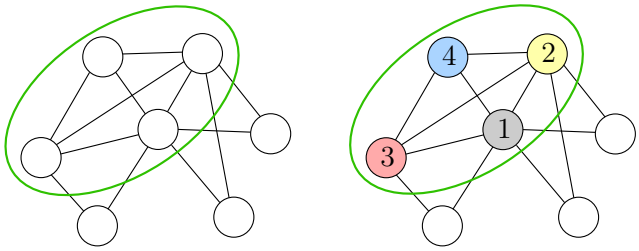


Verteile die Prüfungen auf möglichst wenige der 5 Tage und beachte dabei die Tageskonflikte. Es gibt mehrere richtige Antworten.

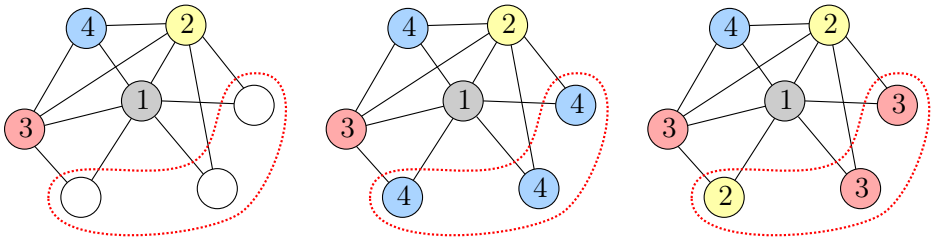
Lösung

So ist es richtig:

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



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



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

Dies ist Informatik!

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

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

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

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

Stichwörter und Webseiten

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

15. Prüf-Biber

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

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

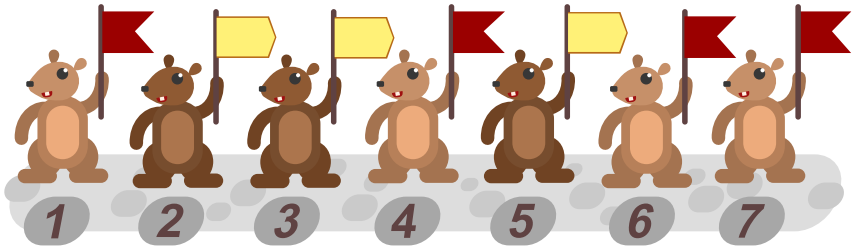
Jeder Prüf-Biber prüft drei Nachrichten-Biber. Wenn diese drei eine ungerade Anzahl an roten Flaggen hochhalten sollen, dann soll ihr Prüf-Biber auch eine rote Flagge hochhalten, sonst eine gelbe. Wenn alle die richtigen Flaggen hochhalten, dann halten ein Prüf-Biber und seine Nachrichten-Biber zusammen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

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

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

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

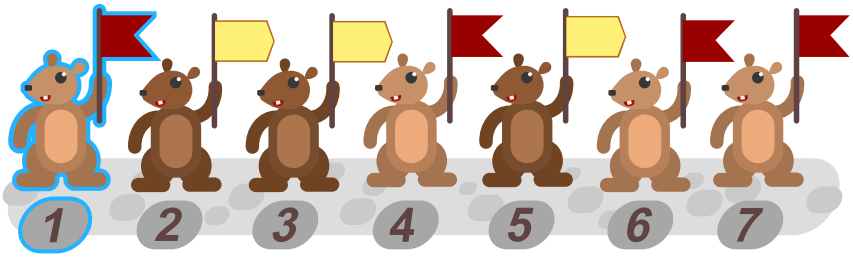
Welcher Biber hält die falsche Flagge hoch?



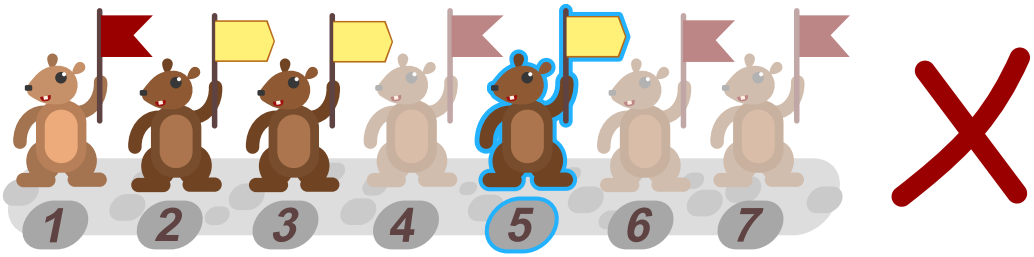
Lösung

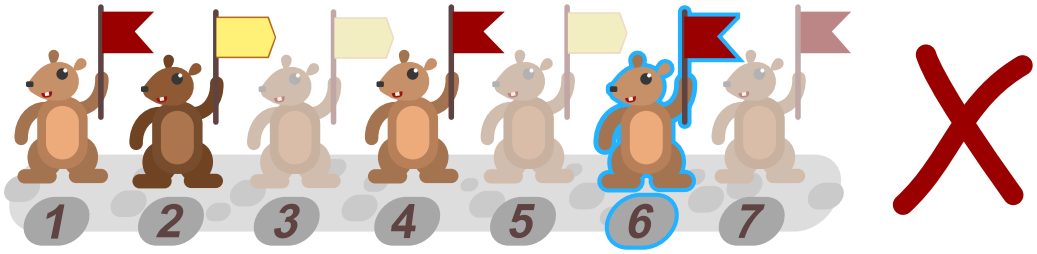
So ist es richtig:

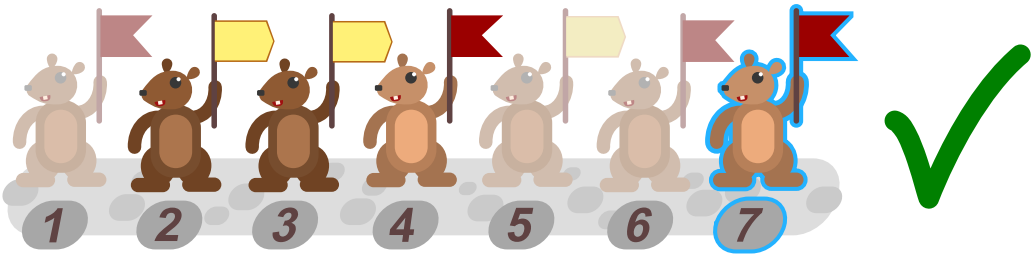
Biber Nummer 1 hält die falsche Flagge hoch.



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







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

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

Dies ist Informatik!

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

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

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

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

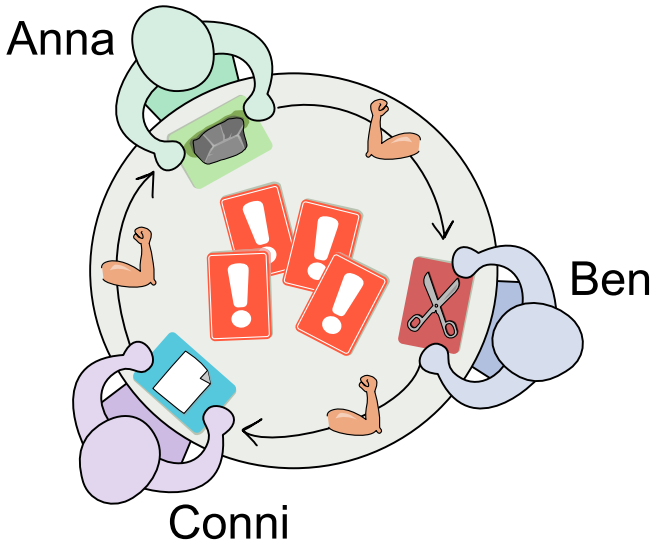
Stichwörter und Webseiten

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

16. Schere, Stein, Papier

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

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



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

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

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

Nur eine der vier Aktionen garantiert das. Welche?

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

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

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

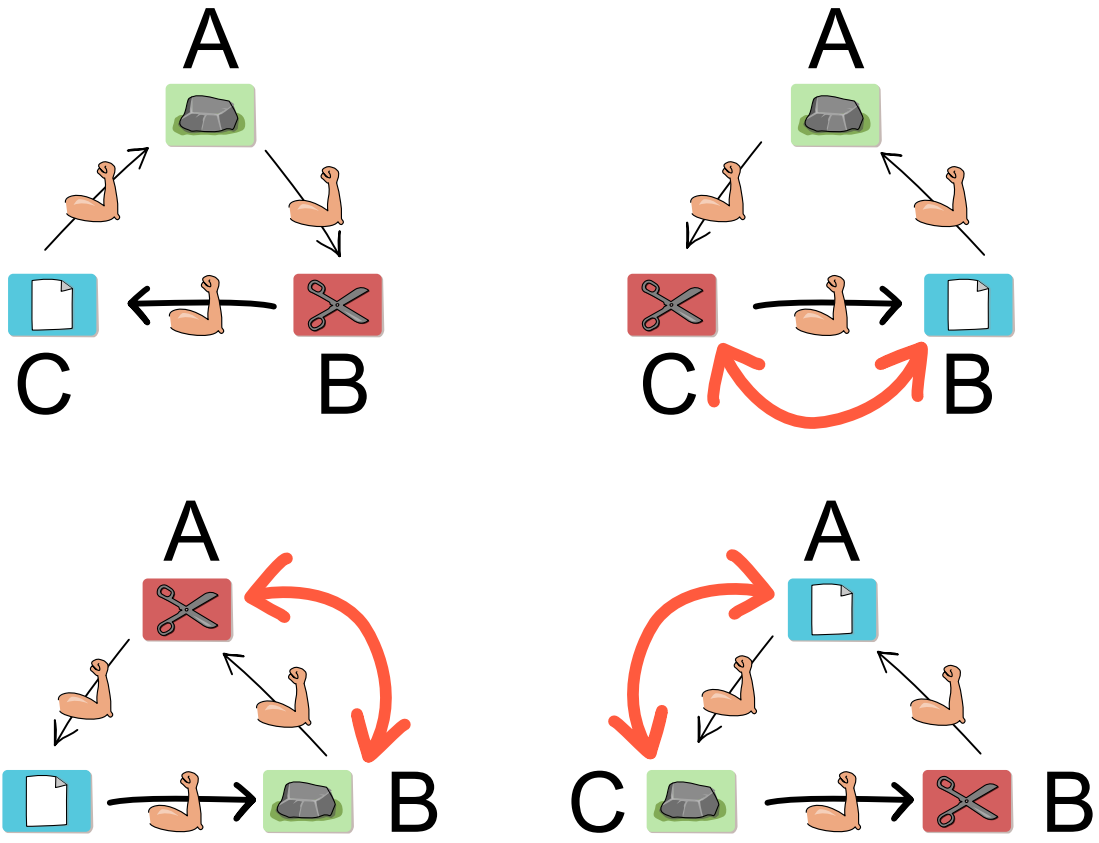
D) Eine gerade Anzahl Male tauschen 2 Spieler ihre Karten.

Lösung

Antwort D ist richtig.

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

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



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

Dies ist Informatik!

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

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

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

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

Stichwörter und Webseiten

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

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.

17. Hin und Her

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

Verwende dazu die folgenden Anweisungen:

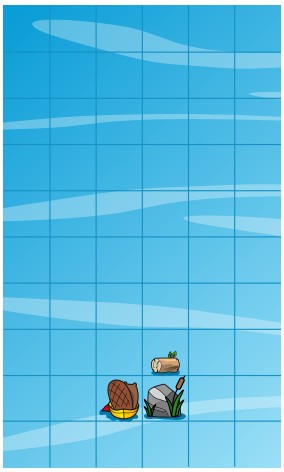
Anweisung	**Beschreibung**
`move()`	Benno bewegt sich genau ein Feld in Blickrichtung nach vorne.
`turnRight()` / `turnLeft()`	Benno dreht sich an Ort um 90 Grad nach rechts / links.
`removeLog()`	Benno entfernt den Baumstamm von dem Feld, auf dem er steht.

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

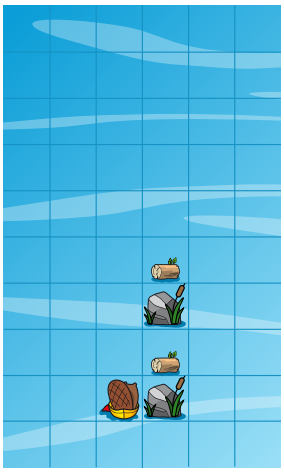
`rockRight()`	*Bedingung*: Prüft, ob rechts von Benno ein Fels ist.
`rockLeft()`	*Bedingung*: Prüft, ob links von Benno ein Fels ist.

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

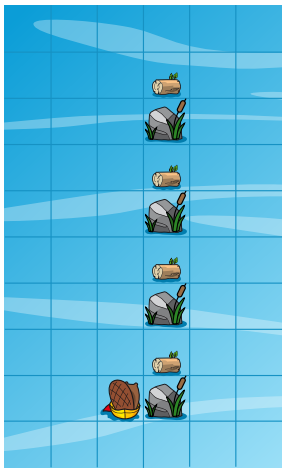
```
while rockLeft():
    move()
turnLeft()
```



See 1



See 2



See 3

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

Lösung

Die richtige Lösung lautet wie folgt:

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

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

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

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

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

Dies ist Informatik!

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

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

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

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

Stichwörter und Webseiten

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

A. Aufgabenautoren

James Atlas
Masiar Babazadeh
Leonardo Barichello
Wilfried Baumann
Susanne Berchtold
Maksim Bolonkin
Maria Cepeda
Špela Cerar
Gi Soong Chee
Anton Chukhnov
Darija Dasović
Christian Datzko
Justina Dauksaite
Diane Dowling
Nora A. Escherle
Gerald Futschek
Vernon Gutierrez
Silvan Horvath
Alisher Ikramov
Thomas Ioannou
Asterios Karagiannis
Blaž Kelvišar
David Khachatryan
Doyong Kim
Jihye Kim
Dong Yoon Kim
Vaidotas Kinčius
Jia-Ling Koh
Sophie Koh
Víctor Koleszar
Lukas Lehner
Gunwoong Lim
Yoshiaki Matsuzawa
Hamed Mohebbi
Mattia Monga
Anna Morpurgo
A-Yeong Park
Suchan Park
Jean-Philippe Pellet
Emmanuel Plan
Zsuzsa Pluhár
Wolfgang Pohl
Cesar F. Bolanos Revelo
Pedro Ribeiro
Rokas Rimkus
Kirsten Schlüter
Dirk Schmerenbeck
Jacqueline Staub
Alieke Stijf
Supawan Tasanaprasert
Susanne Thut
Christine Vender

 Michael Weigend

 Hsu Sint Sint Yee

 Kyra Willekes

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/