INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

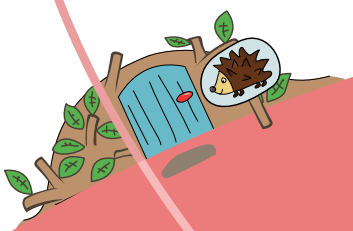
Aufgaben und Lösungen 2025

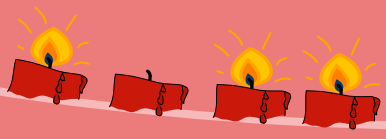
Schuljahre 5/6

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

Herausgeber:

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





01010011010101100100100
10000010010110101010011
01010011010010010100101
00101101010100110101001
01001001010010010010001

SV!A

www.svia-ssie-ssii.ch
schweizerischerverein für informatikin
derausbildung//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:

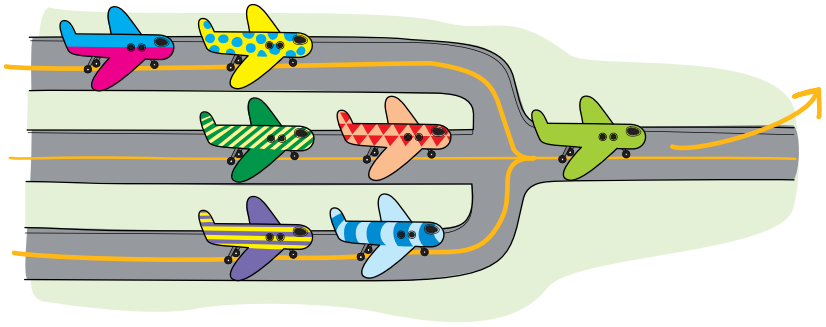
Schweizerischer Verein für Informatik in der Ausbildung
SVIA-SSIE-SSII
Informatik-Biber
Nora A. Escherle

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

Inhaltsverzeichnis

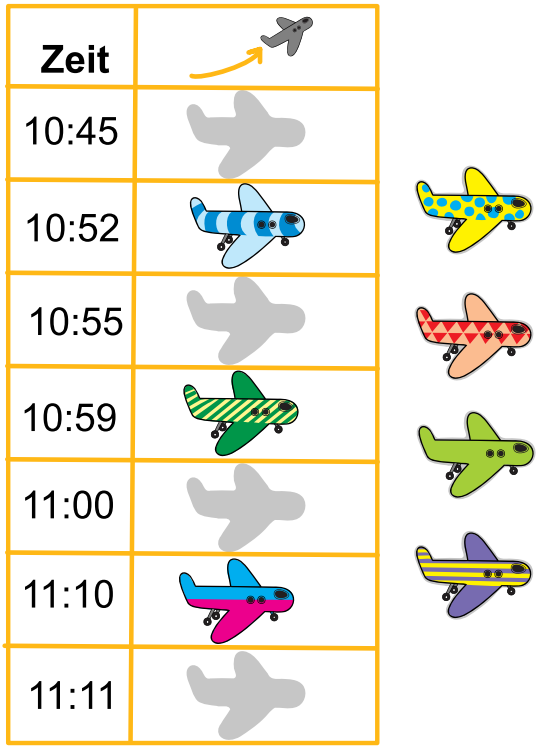
1. Flugzeuge

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



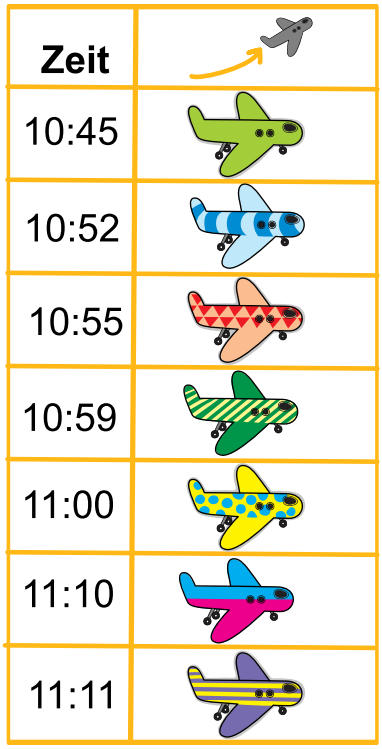
Der Startplan zeigt, in welcher Reihenfolge die sieben Flugzeuge starten. Einige Flugzeuge fehlen aber noch.

Ergänze die fehlenden Flugzeuge im Startplan.



Lösung

So ist es richtig:



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

Dies ist Informatik!

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

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

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

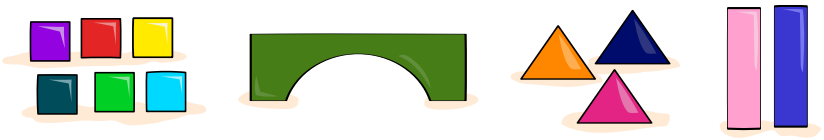
In der Informatik gilt Scheduling als besonders schwieriges Problem. Die Herausforderung besteht darin, für jede denkbare Situation einen optimalen Plan zu berechnen. Wenn viele Parameter und Bedingungen berücksichtigt werden müssen, kann selbst ein sehr schneller Computer dafür enorm viel Zeit benötigen.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

Stichwörter und Webseiten

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

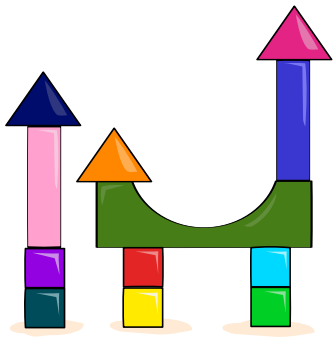
2. Baue mit Bauklötzen

Ali hat diese Bauklötze:



6 Würfel 1 Brücke 3 Pyramiden 2 Quader

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



In welcher Reihenfolge hat Zaha die Anweisungen gegeben?

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

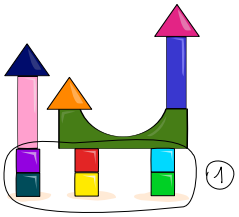
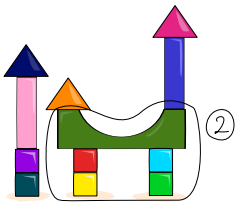
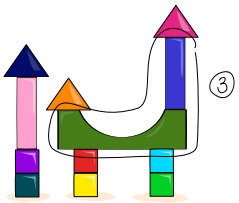
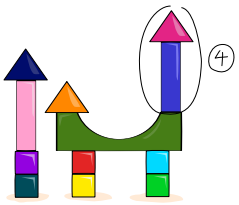
- Stelle beide Quader auf dein Bauwerk.
- Stelle die Würfel-Türme in eine Reihe.
- Lege die Pyramiden auf dein Bauwerk.
- Baue 3 Türme mit jeweils 2 Würfeln.
- Lege die Brücke auf dein Bauwerk.

Lösung

Zaha hat die Anweisungen in dieser Reihenfolge gegeben:

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

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

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

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

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

Dies ist Informatik!

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

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

Stichwörter und Webseiten

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

3. Nach Hause

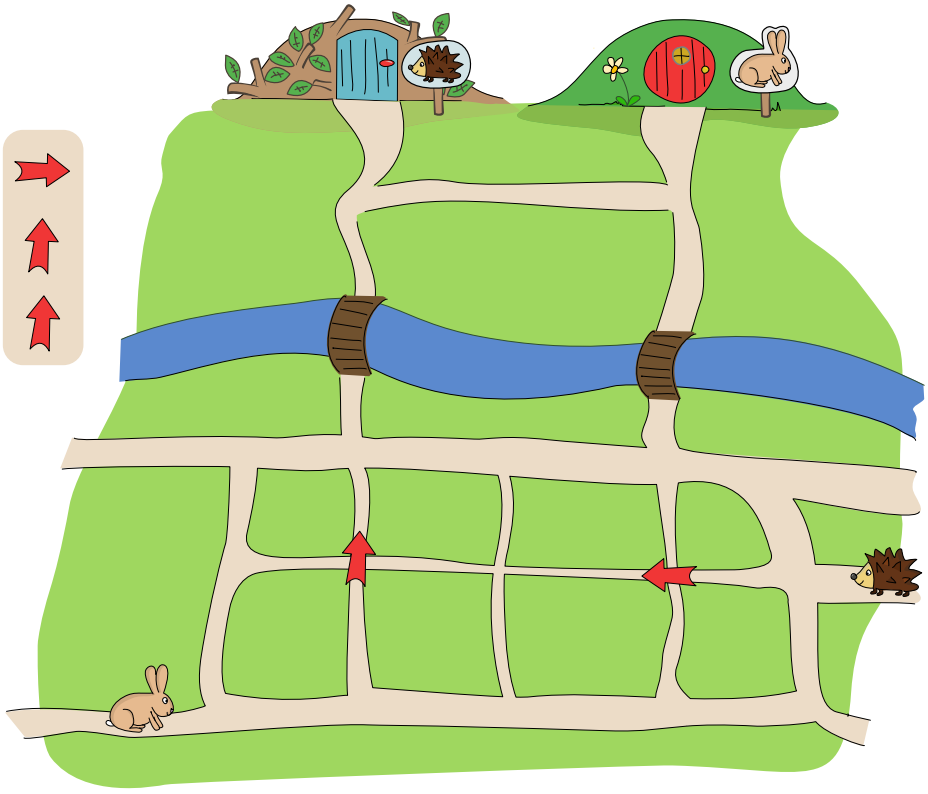
Ein Hase  und ein Igel  wollen nach Hause.

Jeder hat ein eigenes Haus:  und  .

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

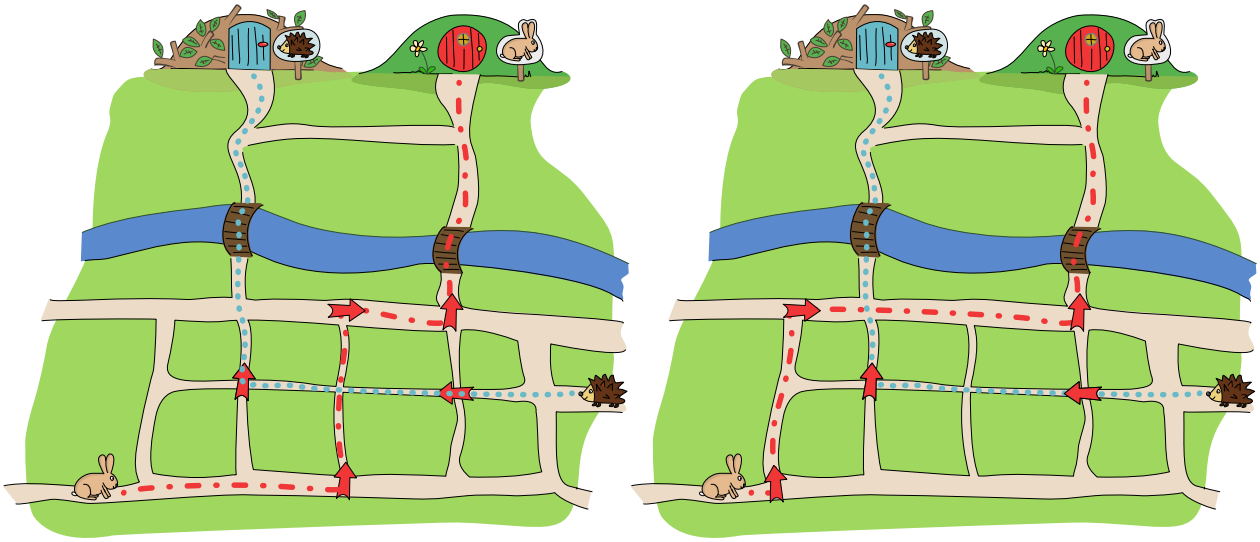
An einigen Kreuzungen liegen schon Pfeile.
Entlang der Pfeile findet der Igel nach Hause, der Hase aber nicht. Zum Glück sind noch drei Pfeile übrig.

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



Lösung

Es gibt zwei richtige Antworten:



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

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

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

Dies ist Informatik!

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

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

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

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

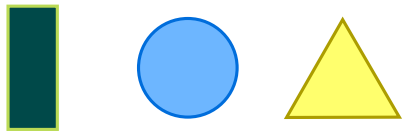
Diese Beispiele zeigen, dass es kritisch ist, wenn die Qualität der Ausgaben eines Algorithmus von der Qualität der Eingaben abhängt. Das ist zum Beispiel bei vielen KI-Systemen der Fall. KI-Systeme arbeiten mit Modellen der Wirklichkeit, und diese Modelle werden aus Daten gebildet. Sind diese Daten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

Stichwörter und Webseiten

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

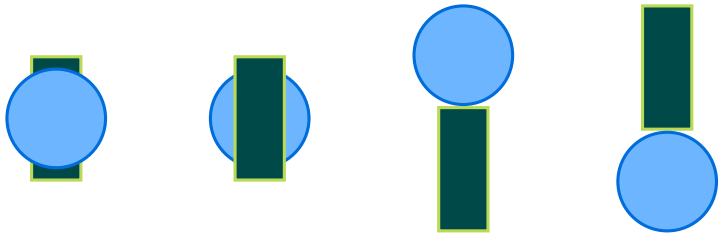
4. Hivobu

Im Land Hivobu heissen diese drei Formen:



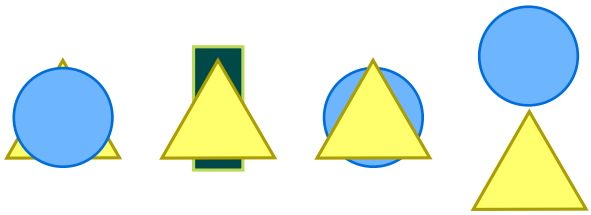
RAH OH TEH

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



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

Was heisst in Hivobu: TEH OH CO?



A) B) C) D)

Lösung

Antwort C ist richtig.

Wir wissen, wie die einzelnen Formen heissen:

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

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

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

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

Aber wie heissen die anderen Antworten in Hivobu?

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

Dies ist Informatik!

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

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

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

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

Stichwörter und Webseiten

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

5. Holz für den Damm

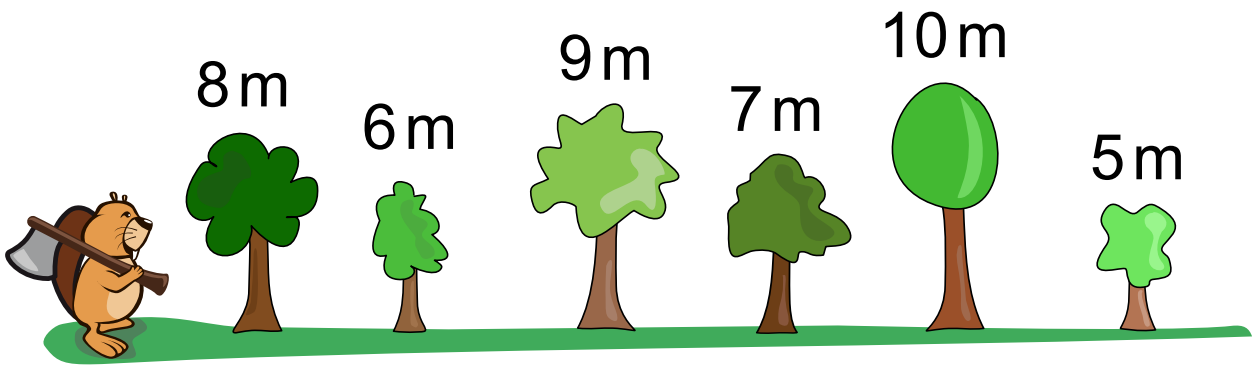
Für ihren nächsten Dammbau müssen die Biber einige Bäume fällen.

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

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

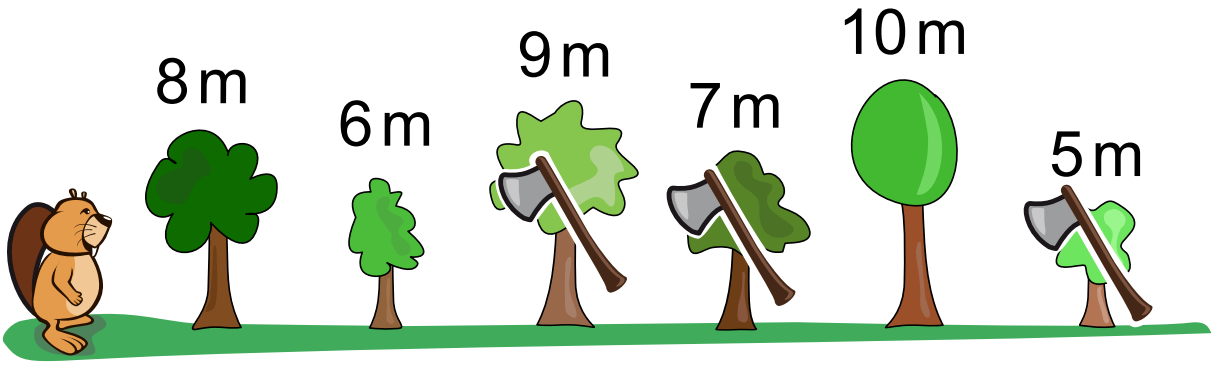
Wenn sie beispielsweise den 6m-Baum fällen, dürfen sie danach nur noch den 5m-Baum fällen. Dann haben sie am Ende insgesamt 11 Meter Holz.

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



Lösung

So ist es richtig:



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

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

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

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

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

Dies ist Informatik!

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

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

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

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

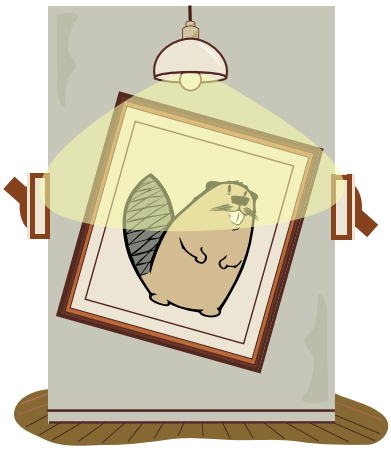
Für besonders Interessierte: Das Problem in dieser Aufgabe kennt die Informatik auch als «Maximum Sum Decreasing Subsequence». Schau dir einmal `diese Webseite` an. Dort kannst du nachverfolgen, wie man von einfachen, aber schlechten Ansätzen zur Lösung eines Optimierungsproblems nach und nach immer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

Stichwörter und Webseiten

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

6. Verrückte Lampe

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



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

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

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

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

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

Lösung

So ist es richtig:

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

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

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

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

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

Dies ist Informatik!

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

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

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

Stichwörter und Webseiten

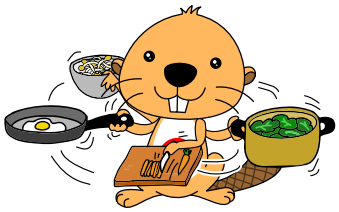
- Logik: `https://de.wikipedia.org/wiki/Logik`
- Logikgatter: `https://de.wikipedia.org/wiki/Logikgatter`
- Boolesche Algebra: `https://de.wikipedia.org/wiki/Boolesche_Algebra`

7. Bibimbap

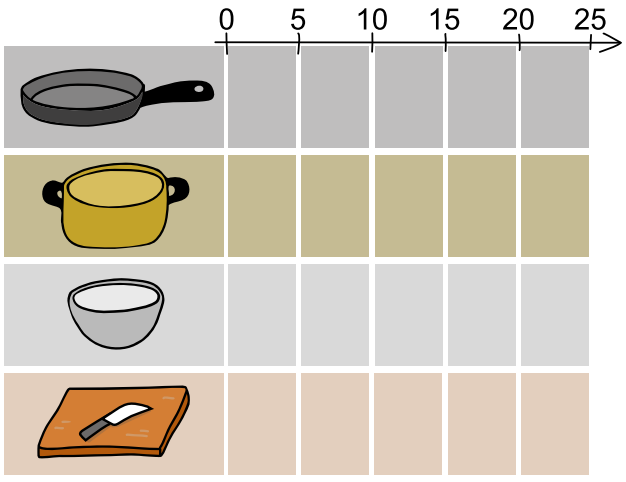
Ein Koch möchte das traditionelle koreanische Gericht Bibimbap (비빔밥) zubereiten. Er benutzt dazu u.a. vier Geräte, nämlich Kochtopf, Bratpfanne, Schneidebrett und Schüssel. Damit bereitet er die vier Zutaten für Bibimbap so vor:

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

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

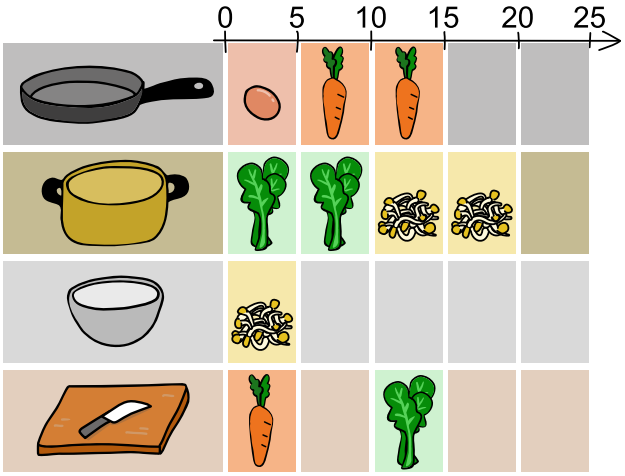


Erstelle einen Plan, mit dem der Koch die Zutaten für Bibimbap in kürzester Zeit vorbereiten kann.



Lösung

So ist es richtig:



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

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

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

Dies ist Informatik!

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

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

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

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

Stichwörter und Webseiten

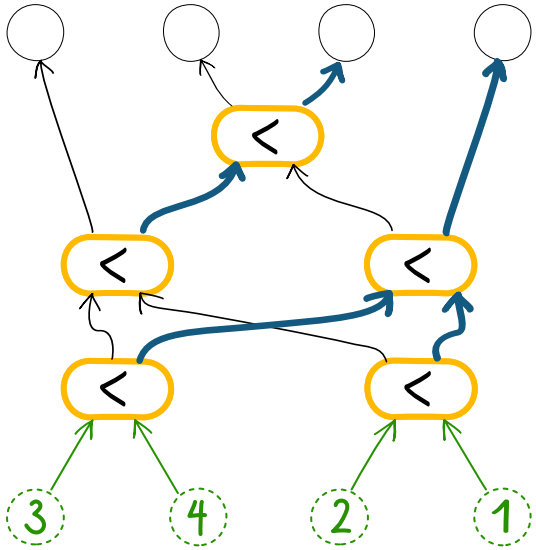
- Scheduling: `https://www.swisseduc.ch/informatik/theoretische_-informatik/scheduling/algorithmus1.html`
- Critical Path Method: `https://de.wikipedia.org/wiki/Methode_des_kritischen_Pfades`
- Stanford Technical Report John Fondahl (siehe Vorwort): `https://catalog.hathitrust.org/Record/005766951`

8. Zahlenmaschine

Die Biber haben eine Zahlenmaschine.

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

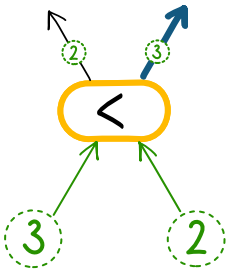
Entlang von Pfeilen und Schaltern wandern die Zahlen durch die Maschine nach oben bis zu den Ausgabefeldern.



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

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

Beispiel:



Welche Aufgabe führt die Maschine aus?

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

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

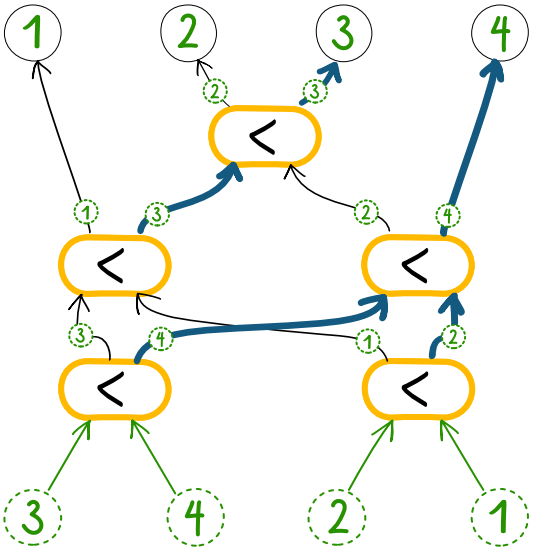
C) Sie gibt die Zahlen in derselben Reihenfolge aus. Ergebnis: 3, 4, 2, 1

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

Lösung

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

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



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

Dies ist Informatik!

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

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

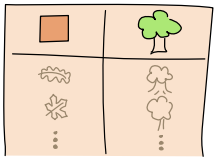
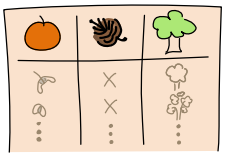
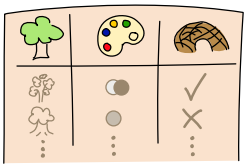
Sortiernetzwerke sind in der Informatik von praktischer Bedeutung, weil Komparatoren als sehr einfache und daher kostengünstige elektronische Bauteile und damit Sortiernetzwerke auch insgesamt gut in Hardware realisiert werden können. Ausserdem können die Komparatoren zum Teil gleichzeitig arbeiten,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

Stichwörter und Webseiten

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

9. Biberholz

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

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

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

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

A) Nur Ladina.

B) Erst Severin, dann Quirina.

C) Erst Severin, dann Ladina.

D) Erst Quirina, dann Severin, dann Ladina.

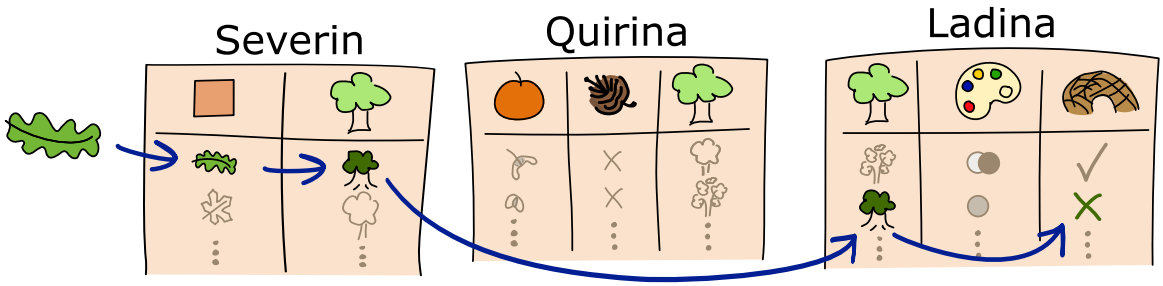
Lösung

Antwort C ist richtig: Erst Severin, dann Ladina.

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

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

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

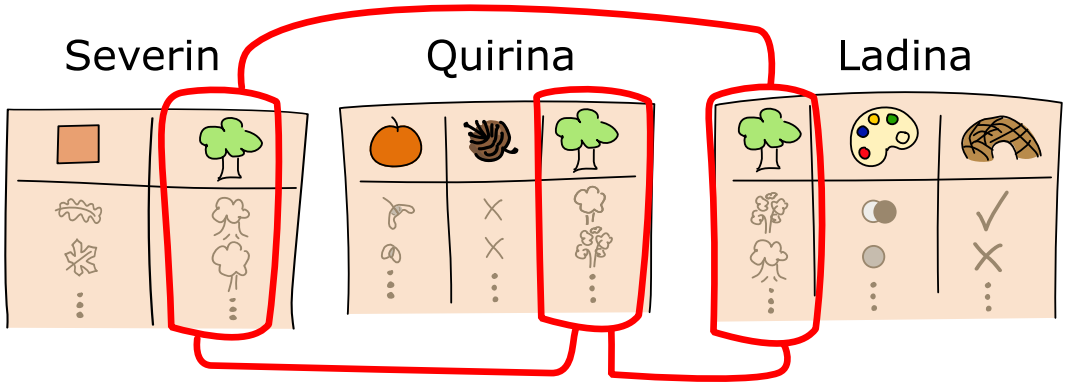


Dies ist Informatik!

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

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

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



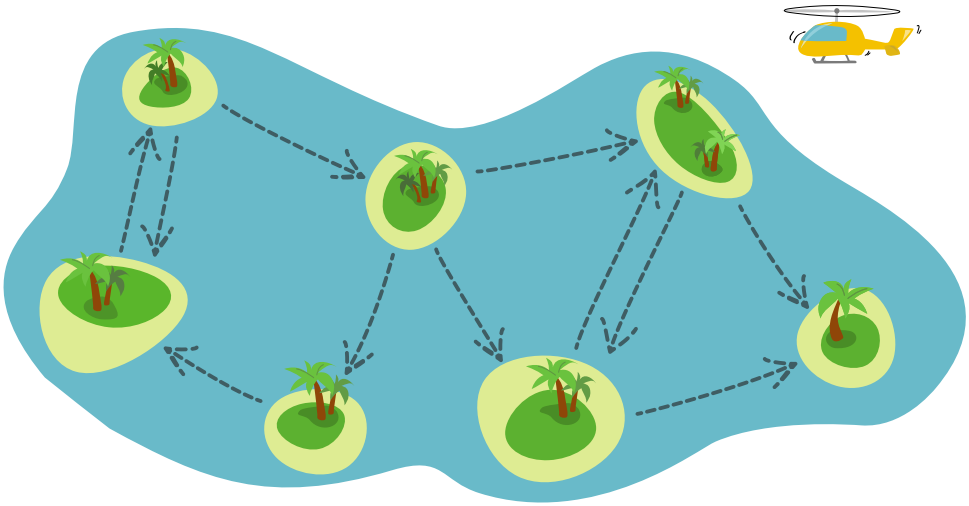
Relationale Datenbanken sind so wichtig, dass es in der Informatik für Abfragen und andere Operationen auf Datenbanken eine eigene Sprache gibt, nämlich *SQL (Structured Query Language)*.

Stichwörter und Webseiten

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

10. Bebrasien

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



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

Das Team ...

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

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

Wieviele Ausflüge muss das Team dazu mindestens machen?

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

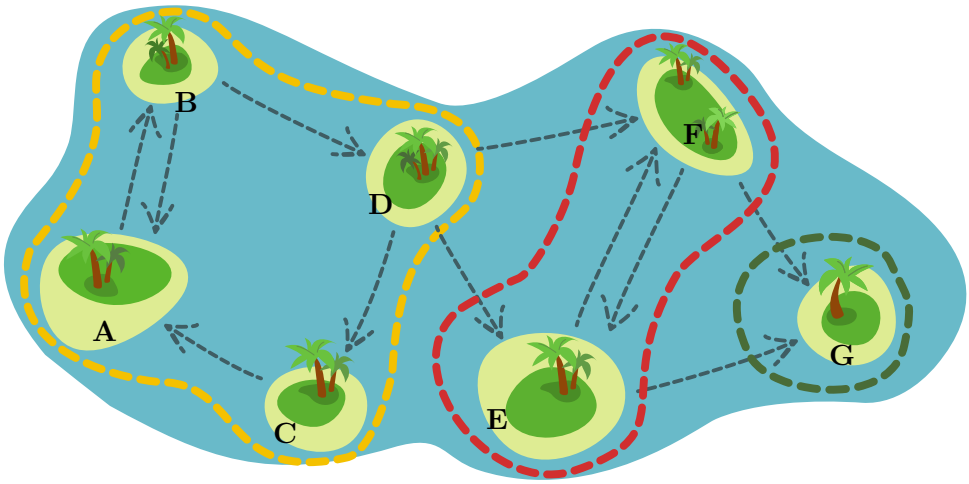
Lösung

3 ist die richtige Antwort.

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

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

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



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

Dies ist Informatik!

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

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

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

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

Stichwörter und Webseiten

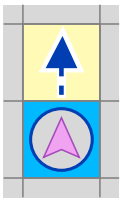
- *Gerichteter Graph*: `https://de.wikipedia.org/wiki/Gerichteter_Graph`
- *stark zusammenhängende Komponente*: `https://de.wikipedia.org/wiki/Zusammenhang_(Graphentheorie)`

11. Lefty II

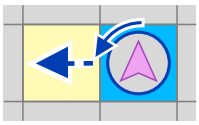
Roboter *Lefty* bewegt sich über ein Raster mit quadratischen Feldern. Zwischen Feldern kann es rote Mauern geben. Lefty soll das grüne Ziel erreichen.

Lefty kann sich auf genau zwei Arten bewegen:

Ein Feld vorwärts fahren

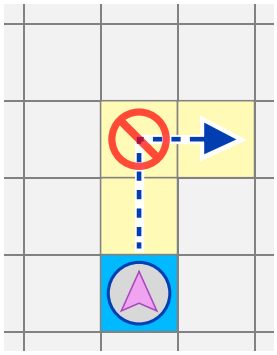


Nach links drehen und dann sofort ein Feld vorwärts fahren

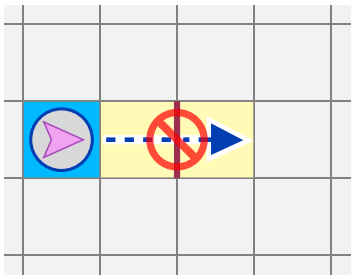


Lefty kann aber nicht alles. Zum Beispiel kann er

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

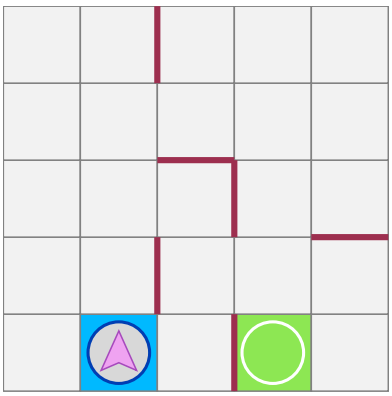


... **nicht** durch Mauern fahren.



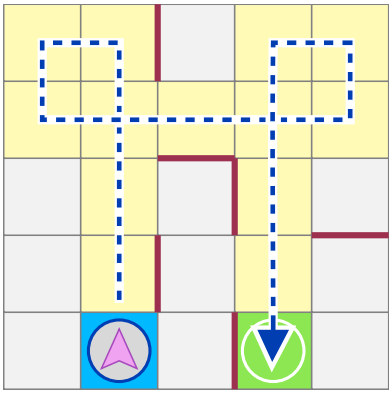
*Über welche Felder **muss** Lefty fahren, um das Ziel zu erreichen?*

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



Lösung

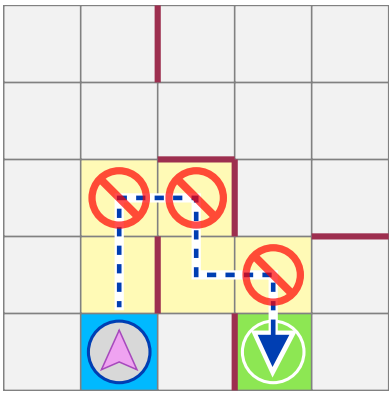
So ist es richtig:



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

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

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

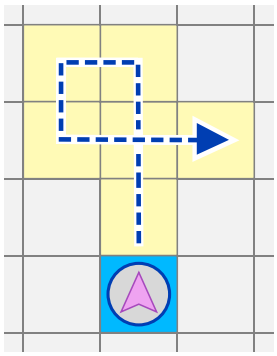


Dies ist Informatik!

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

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

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



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

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

Stichwörter und Webseiten

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

12. Momos Spiel

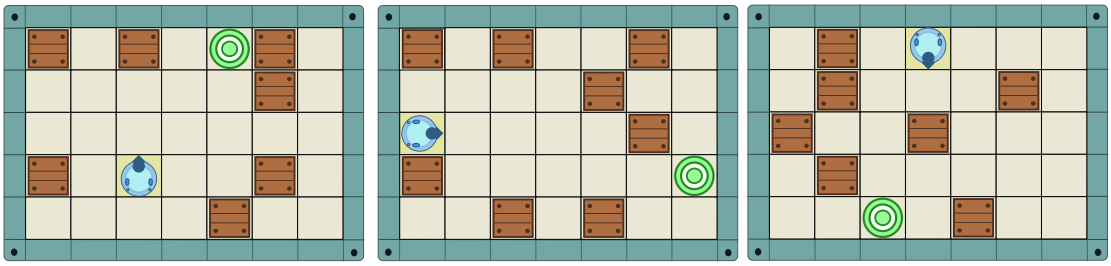
In einem von Momos Computerspielen kann ein Roboter über Felder fahren. Wenn er vor ein Hindernis oder die Wand fährt, geht es nicht mehr weiter. Dann muss der Roboter die Richtung wechseln.

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

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

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

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



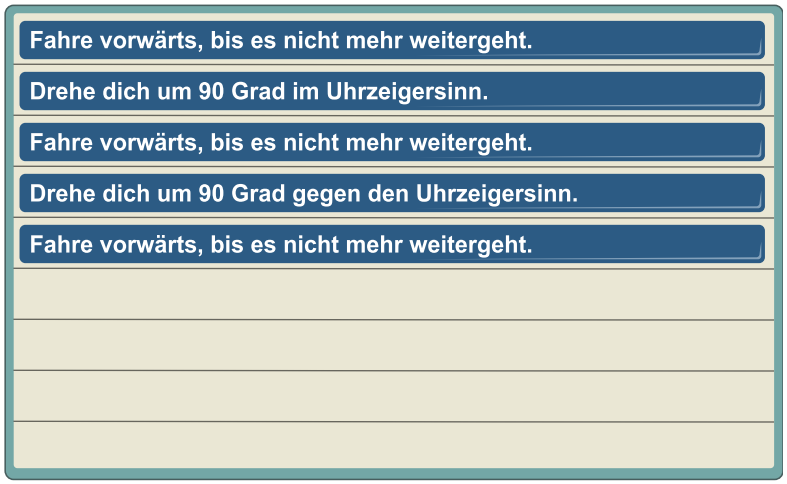
Erstelle so ein Programm für Momo!

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

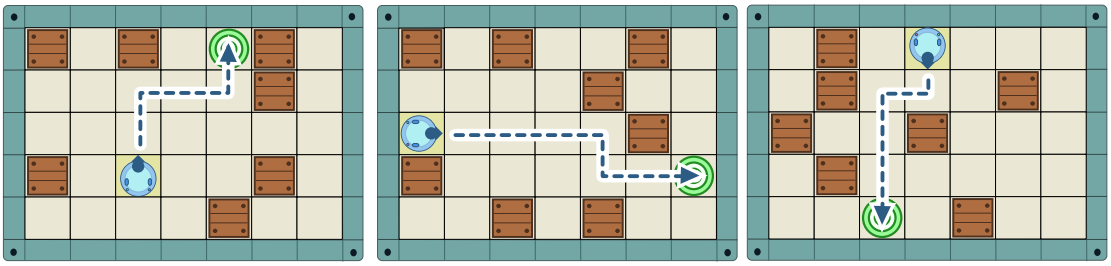
Lösung

So ist es richtig:

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



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



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

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

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

Dies ist Informatik!

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

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

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

Stichwörter und Webseiten

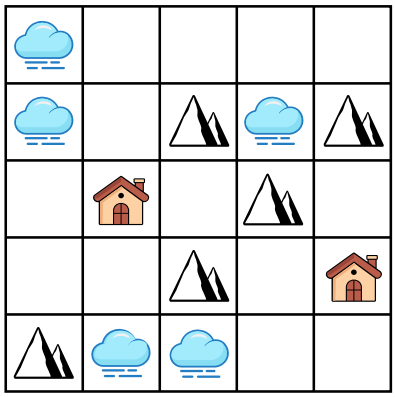
- Strukturierte Programmierung:
  `https://de.wikipedia.org/wiki/Strukturierte_Programmierung`
- Lösungsalgorithmen für Irrgärten:
  `https://de.wikipedia.org/wiki/Lösungsalgorithmen_für_Irrgärten`

13. Ein Tag im Nebel

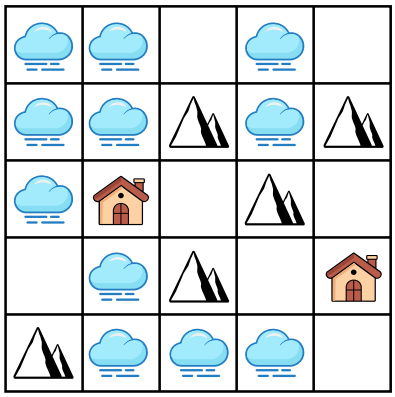
Im Land der Berge ist heute Nebel, und der breitet sich mit jeder Stunde weiter aus.

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

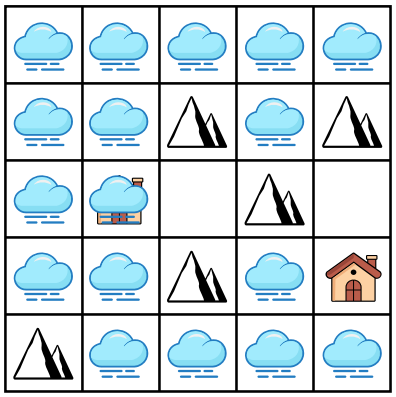
Ein Beispiel:



Sonnenaufgang

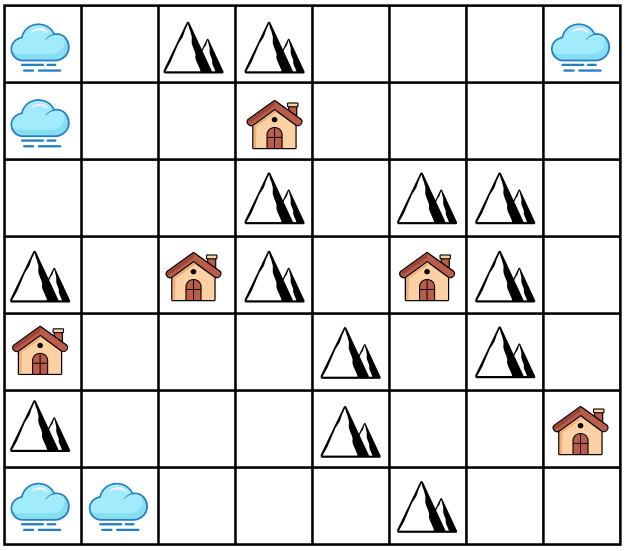


Nach 1 Stunde



Nach 2 Stunden

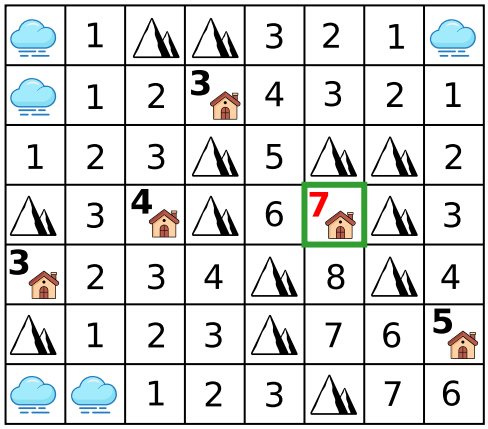
*Welches Haus im Land wird als **letztes** vom Nebel bedeckt?*



Lösung

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

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



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

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

Dies ist Informatik!

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

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

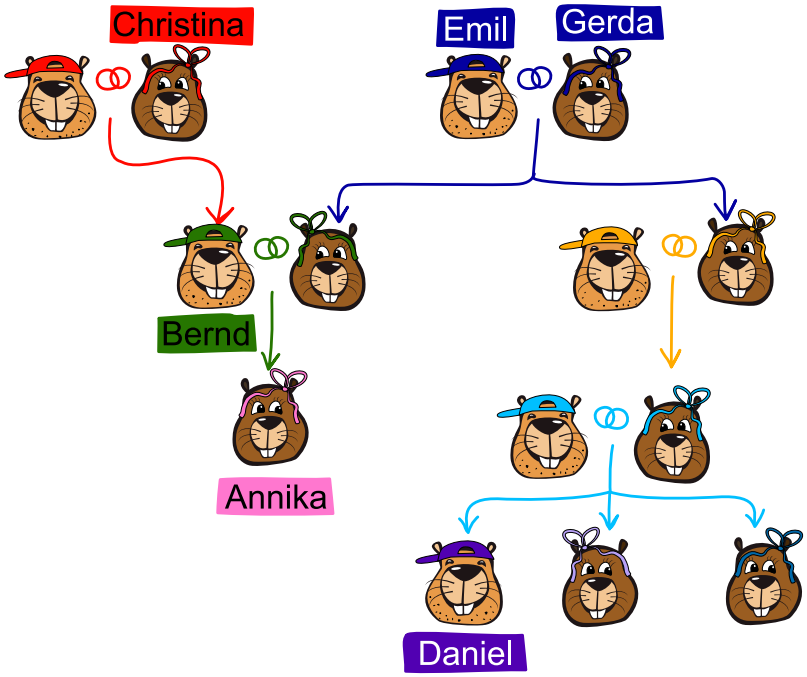
Zusammenhängende Gebiete sind auch für die Informatik interessant, und zwar in unterschiedlichen Bereichen. Ein einfarbiger Bereich in einem (Computer-)Bild ist ein zusammenhängendes Gebiet gleichfarbiger Pixel; man kann es mit einem *Floodfill-Algorithmus* bestimmen, der so ähnlich funktioniert wie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

Stichwörter und Webseiten

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

14. Stammbaum

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



Annika verwendet eine Kurzschreibweise:

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

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

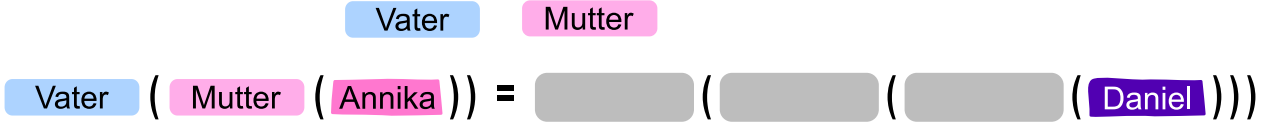
- `Vater(Annika) = Bernd`
- `Mutter(Bernd) = Christina`

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

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

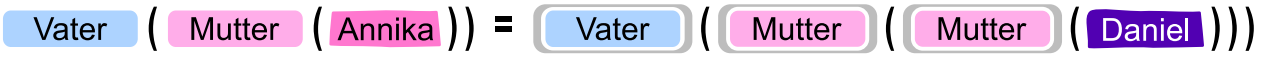
Nun hätte sie gerne eine Gleichung für ihre Verwandtschaft mit Daniel.

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



Lösung

So ist es richtig:



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

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

Dies ist Informatik!

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

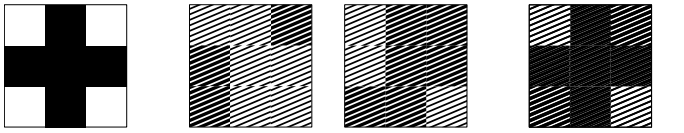
Diese Biberaufgabe zeigt, wie Funktionsaufrufe für komplexere Berechnungen zusammengesetzt (oder: «verschachtelt») werden können. In einem *verschachtelten Funktionsaufruf* dienen die Ergebnisse der inneren Funktionsaufrufe als Eingabe für äussere Aufrufe. Ein verschachtelter Funktionsaufruf wird von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

Stichwörter und Webseiten

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

15. 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.

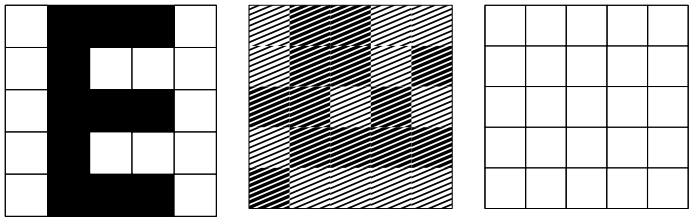


Geheimes Bild | Folie 1 | Folie 2 | Folien 1 und 2 übereinander

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 ▨).

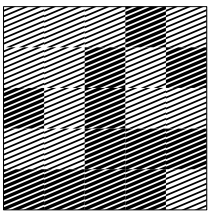
Für das folgende geheime Bild wurde Folie 1 bereits erzeugt. Erstelle nun Folie 2.



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

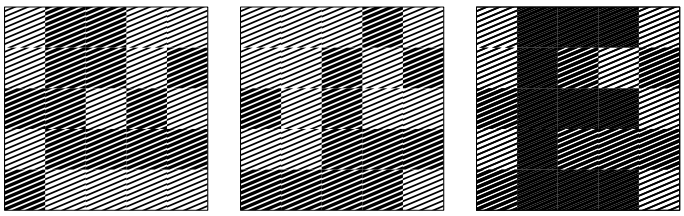
Lösung

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`

16. 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`

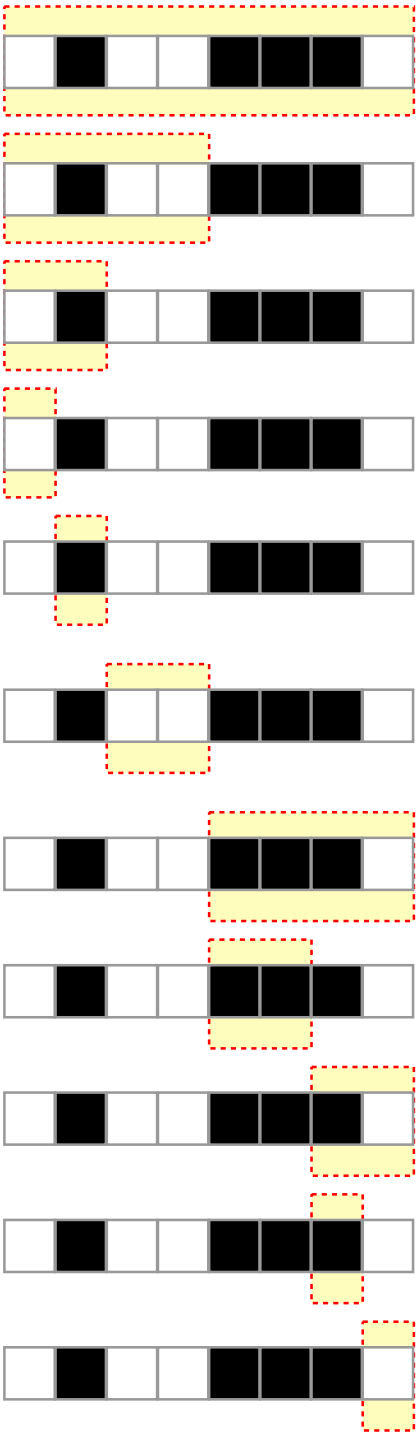
`xSxWxSW`

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`

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.

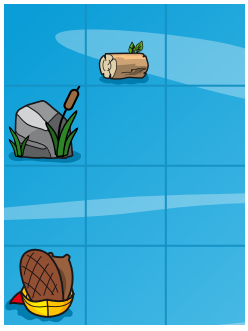
17. Verrückte Sandbank

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

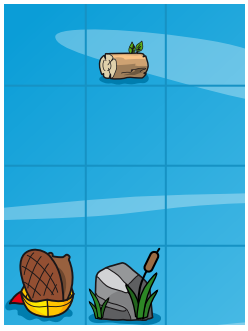
Du kannst folgende Anweisungen verwenden:

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

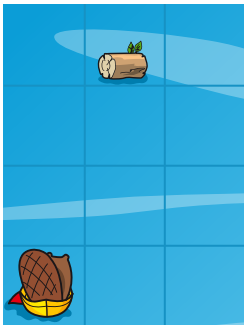
See 1



See 2



See 3



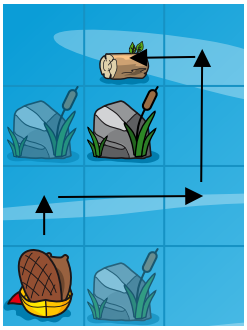
See 4

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

Lösung

Die richtige Lösung lautet wie folgt:

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



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

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

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

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

Dies ist Informatik!

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

Stichwörter und Webseiten

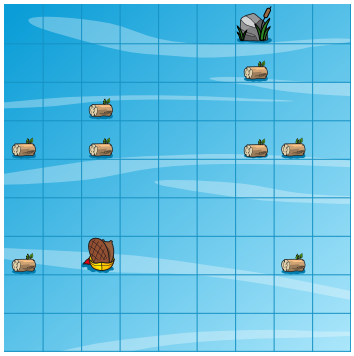
- Programm: `https://de.wikipedia.org/wiki/Programmierung`
- Sequenz: `https://de.wikipedia.org/wiki/Kontrollstruktur`

18. Wertvoller Baumstamm

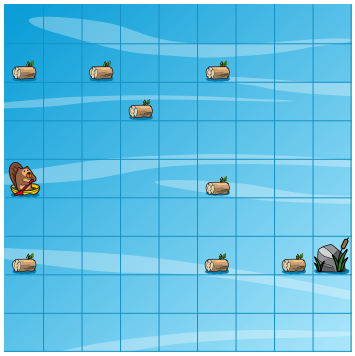
Biber Petunia ist auf der Suche nach wertvollen Baumstämmen im Seeland. Der wertvollste Baumstamm ist immer direkt bei einem Felsen. Hilf Petunia eine Anleitung zu schreiben, mit der sie in allen drei Seen bis auf das Feld mit dem wertvollen Baumstamm gelangt. Nutze möglichst wenige Anweisungen.

Du kannst folgende Anweisungen verwenden:

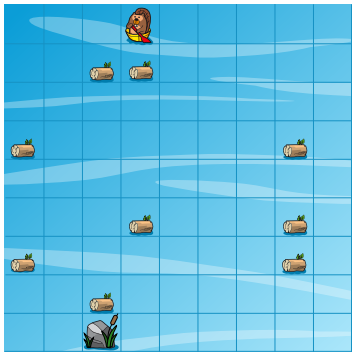
Anweisung	**Beschreibung**
`move()`	Petunia bewegt sich genau ein Feld in Blickrichtung nach vorne.
`turnRight()` / `turnLeft()`	Petunia dreht sich an Ort um 90 Grad nach rechts / links.
`goToLog()`	Petunia fährt vorwärts, bis sie auf einem Feld mit einem Baumstamm ist.



See 1



See 2



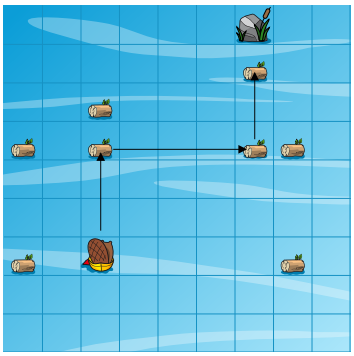
See 3

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

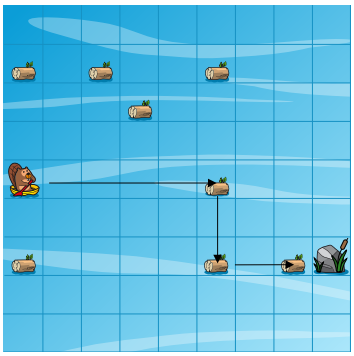
Lösung

Die richtige Lösung lautet wie folgt:

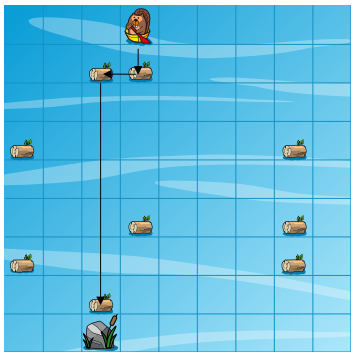
```
goToLog()
turnRight()
goToLog()
turnLeft()
goToLog()
```



See 1



See 2



See 3

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

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

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

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

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

Dies ist Informatik!

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

Stichwörter und Webseiten

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

A. Aufgabenautoren

Masiar Babazadeh
Wilfried Baumann
Gi Soong Chee
Byeonggyu Cho
Vladimir Costas
Valentina Dagienė
Christian Datzko
Nora A. Escherle
Abeer Eshra
Gerald Futschek
Silvan Horvath
Alisher Ikramov
David Khachatryan
Doyong Kim
Jihye Kim
Vaidotas Kinčius
Stefan Koch
Lukas Lehner
Taina Lehtimäki
Gunwoong Lim
Mattia Monga
Anna Morpurgo
Kamohelo Motloung
Justina Oostendorp
Jean-Philippe Pellet
Emiliano Pereiro
Zsuzsa Pluhár
Wolfgang Pohl
Pedro Ribeiro
Kirsten Schlüter
Dirk Schmerenbeck
Vipul Shah
Jacqueline Staub
Nikolaos Stratis
Susanne Thut
Christine Vender
Florentina Voboril
Michael Weigend
Philip Whittington
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/