Reimerei

Aysun möchte für ihren kleinen Bruder Reimrätsel erstellen. Dafür benötigt sie Wörter, die sich reimen. Um mehr Auswahl zu haben, lädt sie sich eine umfangreiche Liste von Wörtern aus dem Internet herunter. Nun brauchst sie nur noch ein Programm, das ihr aus dieser Liste passende Wortpaare heraussucht.

Junioraufgabe 1

Schreibe ein Programm, das eine Liste von Wörtern einliest und daraus alle passenden Wortpaare berechnet. Ein Wortpaar ist passend, wenn es die folgenden Regeln erfüllt:

1) Die beiden Worte enden gleich: Sie haben dieselbe maßgebliche Vokalgruppe, und nach der maßgeblichen Vokalgruppe enthalten beide Wörter dieselben Buchstaben in derselben Reihenfolge. Dabei ist eine Vokalgruppe eine längstmögliche Folge von unmittelbar aufeinanderfolgenden Vokalen (z.B. hat das Wort *Taifun* die Vokalgruppen ‚ai' und ‚u'), und die maßgebliche Vokalgruppe eines Wortes ist seine vorletzte Vokalgruppe, wenn das Wort zwei oder mehr Vokalgruppen enthält. Enthält ein Wort nur eine Vokalgruppe, ist seine maßgebliche Vokalgruppe die eine vorhandene Vokalgruppe.

2) In jedem der beiden Wörter enthält die maßgebliche Vokalgruppe und was ihr folgt mindestens die Hälfte der Buchstaben.

3) Keines der beiden Wörter darf mit dem kompletten anderen Wort enden.

Passende Wortpaare wären zum Beispiel *Baum*, *Traum* und *singen*, *klingen*; aber *Tanne*, *Rinne* verletzt Regel 1, *Informatik*, *Akrobatik* verletzt Regel 2, und *kaufen*, *verkaufen* verletzt Regel 3.

Wende dein Programm mindestens auf alle Beispiele an, die du auf den BWINF-Webseiten findest, und dokumentiere die Ergebnisse.

Container

Mehrere Container werden per Kran auf ein Schiff verladen. Der schwerste Container soll zuerst auf das Schiff platziert werden; das ist bei Seegang gut für den Schwerpunkt.

Jeder Container ist vorher mindestens einmal mit einem anderen Container auf der Container-Paar-Waage gewesen. Für jedes gewogene Container-Paar ist danach nur bekannt, welcher der beiden Container der schwerere ist. Von den Containern haben keine zwei das gleiche Gewicht.

Steht damit fest, welchen Container die Kranführerin zuerst verladen soll?

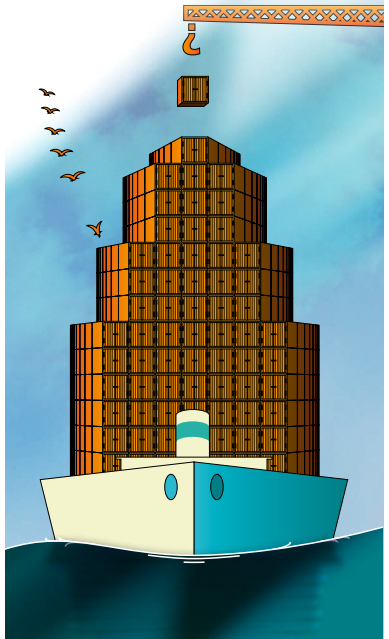
Junioraufgabe 2

Schreibe ein Programm, das die Kranführerin unterstützt.

Das Programm soll für die gewogenen Container-Paare einlesen, welcher Container jeweils der schwerere ist.

Anschließend soll das Programm ausgeben, ob daraus eindeutig der schwerste Container bestimmt werden kann. Falls ja, soll das Programm außerdem ausgeben, welcher Container das ist.

Wende dein Programm mindestens auf alle Beispiele an, die du auf den BWINF-Webseiten findest, und dokumentiere die Ergebnisse.



Störung

Alice und Bob tauschen in ihrem persönlichen Chat gerne Sätze aus, die Zitate aus ihrem Lieblingsbuch sind. Leider hat Trudi einen Weg gefunden, Lücken in die Sätze zu reißen und ganze Wörter verschwinden zu lassen. So empfing Bob vor Kurzem diesen Lückensatz von Alice, den er nicht verstehen konnte:

das … mir … … … vor

Er musste lange im Buch blättern, bis er diese passende Stelle darin fand:

das kommt mir gar nicht richtig vor

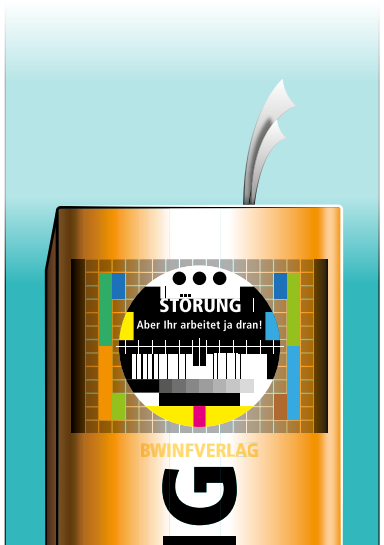
Aber er ist unsicher, denn schließlich könnte es ja noch andere passende Stellen geben.

Alice und Bob überlegen, wie sie das Problem lösen können. Den Text des Lieblingsbuches gibt es auch digital. Da könnte es doch möglich sein, sich die Suche nach passenden Stellen von einem Computerprogramm abnehmen zu lassen.

Aufgabe 1

Hilf Alice und Bob und schreibe ein Programm, das für einen Lückensatz nach passenden Stellen im Lieblingsbuch sucht.

Auf den BWINF-Webseiten findest du den digitalen Text des Lieblingsbuchs und einige Lückensätze. Wende dein Programm mindestens auf diese Beispiele an und dokumentiere die Ergebnisse.



Verzinkt

Muster wie im Bild unten sieht man an feuerverzinkten Metallen, zum Beispiel an den Masten von Laternen oder Ampeln. Diese Muster entstehen, indem sich beim Erkalten des flüssigen Zinks, von zufällig vorhandenen Kristallisationskeimen aus, Kristalle bilden. Jeder Keim hat eine bestimmte Orientierung. Um ihn herum wächst ein Kristall mit der gleichen Orientierung, bis er auf andere Kristalle trifft. Kristalle unterschiedlicher Orientierung reflektieren das Licht unterschiedlich und erscheinen so unterschiedlich hell.

Aufgabe 2

Versuche, solche Kristallmuster mit dem Computer zu generieren. Schreibe dazu ein Programm, das die Entstehung solcher Muster simuliert.

Deine Simulation soll sich an folgende Vorgaben halten:

- Der Ort eines Kristallisationskeims wird als Punkt in einem zweidimensionalen Raster dargestellt.
- Von einem Keim aus wächst der Kristall schrittweise in die vier Raster-Richtungen, also nach links, rechts, oben und unten.
- Für jede dieser Richtungen hat ein Kristall eigene Wachstumsgeschwindigkeiten.
- Ein Kristall wächst so weit wie möglich, aber nicht in die Fläche eines anderen Kristalls hinein.
- Die aus der jeweiligen Orientierung resultierenden unterschiedlichen Lichtreflektonen der Kristalle werden als Grautöne repräsentiert.

Deine Simulation soll in einigen Parametern variiert werden, zum Beispiel in der Anzahl und den Orten der Keime sowie den Entstehungszeitpunkten und den Wachstumsgeschwindigkeiten der Kristalle.

Visualisiere die durch die Simulation entstandenen Muster als Pixelbilder. Mit welchen Werten für die Parameter erhältst du Bilder, die dem Bild oben ähneln?

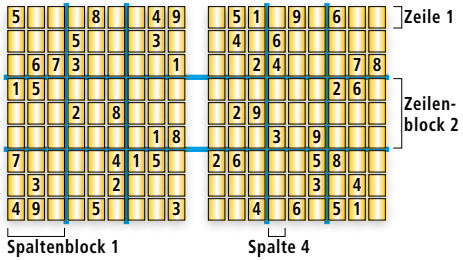
Sudokopie

Rudi ist gar nicht mehr begeistert von den Sudokus seiner Rätselzeitung: Er hat den starken Verdacht, dass die Redaktion die Sudoku-Aufgaben immer wiederverwendet! Die Rätsel sehen zwar unterschiedlich aus, aber der Lösungsweg wiederholt sich erkennbar. Er möchte nun seinen Verdacht nachweisen.

Rudi überlegt sich, dass man aus einem Sudoku sehr viele Varianten erzeugen kann, indem man eine oder mehrere der folgenden Umformungen durchführt:

- Permutation (Umordnung) der drei Spalten innerhalb der Spaltenblöcke.
- Permutation der drei Spaltenblöcke.
- Permutation der drei Zeilen innerhalb der Zeilenblöcke.
- Permutation der drei Zeilenblöcke.
- 90-Grad-Rotation im Uhrzeigersinn.
- Umbenennen der Ziffern 1 bis 9 (z.B. alle ‚8'en und ‚3'en vertauschen).

So sind zum Beispiel diese beiden Rätsel Varianten voneinander:



Hierbei wurden der 1. und der 3. Spaltenblock sowie die 5. und die 6. Zeile vertauscht und die Ziffern wie folgt umbenannt:
1 → 2, 2 → 3, … , 8 → 9, 9 → 1.

Aufgabe 3

Hilf Rudi, indem du ein Programm schreibst, das zwei Sudokus einliest und überprüft, ob sie Varianten voneinander sind. Wenn ja, soll das Programm Umformungen ausgeben, mit denen das eine aus dem anderen Sudoku erzeugt werden kann.

Wende dein Programm mindestens auf alle Beispiele an, die du auf den BWINF-Webseiten findest, und dokumentiere die Ergebnisse.

BUNDES WEIT INFORMATIK NACHWUCHS FÖRDERN

Fahrradwerkstatt

Marc hat sich mit einer kleinen Fahrradwerkstatt selbstständig gemacht. Aufgrund des Fahrrad-Booms wird es zunehmend schwierig, die ganzen Reparaturaufträge von Kunden abzuarbeiten.

Daher arbeitet Marc bereits jeden Tag von 9 bis 17 Uhr. Wenn er einen Auftrag beginnt, erledigt er ihn vollständig, bevor er einen neuen Auftrag beginnt. Am Ende eines Arbeitstages unterbricht er gegebenenfalls den aktuellen Auftrag und nimmt ihn am nächsten Arbeitstag wieder auf.

Bislang erledigt er die Aufträge einfach in der Reihenfolge des Eingangs. Damit sind manche Kunden unzufrieden: Kurze Aufträge müssen teilweise sehr lange warten, weil die Werkstatt durch lange Aufträge blockiert ist. Marc hat sich bereits ein anderes Verfahren überlegt: Wann immer er einen neuen Auftrag beginnt, wählt er den kürzesten unter den bereits vorliegenden Aufträgen.

Aufgabe 4

1) Unterstütze Marc und schreibe ein Programm, das die beiden Verfahren simuliert. Dazu bekommt es eine Liste von Aufträgen. Für jeden Auftrag sind Eingangszeitpunkt und Bearbeitungsdauern in Minuten angegeben. Für die beiden Verfahren soll das Programm die durchschnittliche und die maximale Wartezeit der Aufträge (in Minuten) berechnen. Die Wartezeit eines Auftrags ist die Differenz zwischen dem Zeitpunkt seiner Fertigstellung und seinem Eingangszeitpunkt.

2) Begründe, warum auch bei dem zweiten Verfahren vermutlich nicht alle Kunden zufrieden sein werden.

3) Gibt es Verfahren, die du für besser geeignet hältst? Hierzu kannst du dir auch weitere Kennzahlen als nur die durchschnittliche und maximale Wartezeit der Aufträge überlegen. Verändere dein Programm entsprechend und beschreibe, inwiefern dein Verfahren deine Erwartungen erfüllt.

Wende dein Programm mindestens auf alle Beispiele an, die du auf den BWINF-Webseiten findest, und dokumentiere die Ergebnisse.

Hüpfburg

Sasha und Mika durchlaufen miteinander nach folgenden Regeln einen mit Kreide auf den Schulhof gemalten Parcours:

1) Sasha startet auf Feld 1 und Mika auf Feld 2.

2) In jedem Schritt springen beide gleichzeitig, jeweils entlang eines Pfeils, auf ein neues Feld.

3) Der Parcours gilt als erfolgreich absolviert, sobald beide auf demselben Feld landen.

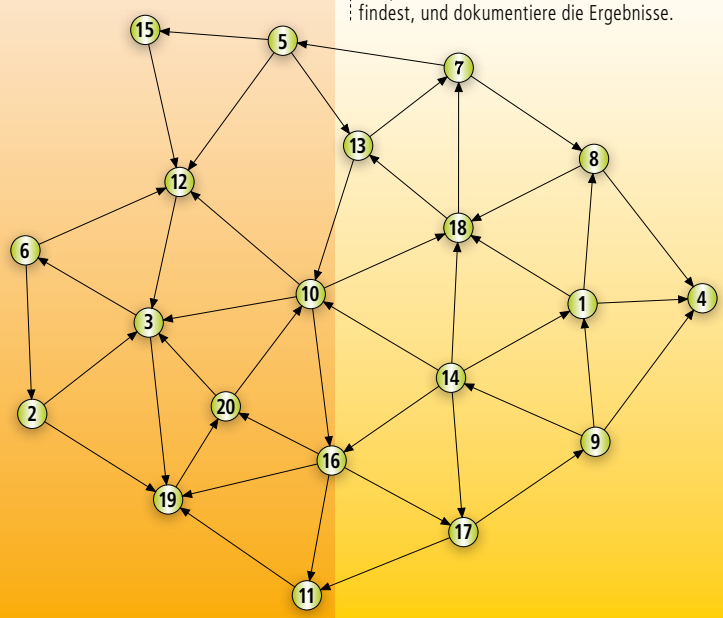
Den folgenden Parcours können sie tatsächlich erfolgreich absolvieren, aber man kann auch solche zeichnen, für die das nicht möglich ist.

Aufgabe 5

Überlege dir, wie man feststellen kann, ob Sasha und Mika einen Parcours erfolgreich absolvieren können. Schreibe ein Programm das einen Parcours einliest und dann ausgibt, ob dieser Parcours erfolgreich absolviert werden kann. Falls es möglich ist, soll außerdem eine erfolgreiche Folge von Sprüngen für Sasha und Mika ausgegeben werden.

Den obigen Parcours können Sasha und Mika in drei Schritten erfolgreich absolvieren, indem sie gleichzeitig auf Feld 10 landen.

Wende dein Programm mindestens auf alle Beispiele an, die du auf den BWINF-Webseiten findest, und dokumentiere die Ergebnisse.



Teilnehmen

Dieses Blatt enthält die Aufgaben der 1. Runde des 41. Bundeswettbewerbs Informatik. Die Junioraufgaben sind gleichzeitig die Aufgaben der 3. und letzten Runde des Jugendwettbewerbs Informatik 2022.

Einsendeschluss für beide Wettbewerbe: 21. November 2022.

Anmelden

online unter: login.bwinf.de

Sobald du dort registriert bist, kannst du dich dort auch zur Teilnahme anmelden: für Jugendwettbewerb (3. Runde), Bundeswettbewerb oder beides. Bei der Anmeldung zum Jugendwettbewerb musst du deine Kennung der Online-Runden (E-Mail-Adresse oder Logincode) angeben.

Bearbeiten

In der 3. Runde des Jugendwettbewerbs bearbeitest du eigenständig die beiden Junioraufgaben. Im Bundeswettbewerb sind die Junioraufgaben SchülerInnen vor der Qualifikationsphase des Abiturs vorbehalten; wer in die 2. Runde kommen will, muss drei oder mehr Aufgaben bearbeiten, einzeln oder im Team.

Einsenden

Für jede bearbeitete Aufgabe sollst du im schriftlichen Teil deiner Einsendung (**Dokumentation**)
- deine **Lösungsidee** beschreiben;
- die **Umsetzung** der Idee in ein Programm erläutern;
- an genügend **Beispielen** zeigen, dass und wie deine Lösung funktioniert; und
- die wichtigsten Teile des Quelltextes anfügen.

Achtung: eine gute Dokumentation muss nicht lang sein, aber unbedingt die **Beispiele** enthalten!

Der praktische Teil deiner Einsendung ist die **Implementierung** und umfasst den kompletten Quelltext und das ausführbare Programm (Windows, Linux, MacOS X oder Android).

Die **Einsendung** wird über login.bwinf.de als ZIP-Dateiarchiv abgegeben. Ein Team gibt gemeinsam nur eine Einsendung ab.

Weitere Informationen unter: bwinf.de/teilnehmen

Doppelteilnahme: Teilnehmende am Jugendwettbewerb vor der Qualifikationsphase können ihre Bearbeitungen der Junioraufgaben auch zur 1. Runde des Bundeswettbewerbs einsenden, gemeinsam mit der Bearbeitung mindestens einer weiteren Aufgabe.

Fragen?

Wende dich an BWINF:
- E-Mail: bundeswettbewerb@bwinf.de bzw. jugendwettbewerb@bwinf.de
- Telefon: 0228 378646
- Chat: bwinf.de/chat

Diskutiere mit den Mitgliedern der EI Community: einstieg-informatik.de/community

Tipps und Infos

Unter bwinf.de/bundeswettbewerb/tipps findest du
- genauere Hinweise zur Einsendung;
- Beispiele für Aufgabenbearbeitungen;
- Tipps zu Informatik und Programmierung.

Deine Chancen

Mit einer Teilnahme am Bundeswettbewerb Informatik kannst du nur gewinnen. In allen Runden gibt es **Urkunden** sowie kleine **Geschenke** für alle.

Bei erfolgreicher Teilnahme an der 1. Runde kannst du zu **Informatik-Workshops** eingeladen werden, die von vielen BWINF-Partnern wie dem Hasso-Plattner-Institut und der DB Systel ausgerichtet werden. Google lädt Teilnehmerinnen zum **Girls@Google Day** ein.

Nach deiner Teilnahme an der 2. Runde winken die **Forschungstage Informatik** des Max-Planck-Instituts für Informatik und einige Buchpreise. Die Einsendung zur 2. Runde kann in einigen Bundesländern als **besondere Lernleistung** in die Abiturwertung eingebracht werden.

Die Besten der 2. Runde erreichen die **Endrunde**. Dort werden Bundessieger und Preisträger ermittelt; sie werden mit **Geldpreisen** belohnt. Die Bundessieger werden in der Regel ohne weiteres Auswahlverfahren in die **Studienstiftung des deutschen Volkes** aufgenommen.

Siehe auch: bwinf.de/bundeswettbewerb/chancen

bwinf.de/teilnehmen

BUNDES WEIT INFORMATIK NACHWUCHS FÖRDERN

- Triff Teilnehmerinnen und Teilnehmer in der Community auf einstieg-informatik.de!

twitter.com/_BWINF instagram.com/bwinf

Aufgabe 2: Verzinkt

Team-ID: 00878

Team: Advanced Beginners

Bearbeiter/-innen dieser Aufgabe:

Luis Liebenstein

5.11.2022

Inhaltsverzeichnis

Lösungsidee:

Ziel dieser Aufgabe ist es Kristallmuster zu erzeugen. Die Erzeugung soll hier mehreren Regeln folgen, welche später genauer ausgeführt werden.

Die Idee war es, Kristallkeime an zufälligen Punkten in einem Koordinatensystem zu erzeugen, die sich dann mit unterschiedlichen Geschwindigkeiten in alle Richtungen ausbreiten. Am Ende soll das Programm jeweils prüfen, in welche Richtung sich der Kristall ausgebreitet hat und ihm eine Helligkeitsstufe zuweisen.

Umsetzung:

Bei der Umsetzung habe ich mich an den gegebenen Vorgaben orientiert.

1. „*Der Ort eines Kristallisationskeims wird als Punkt in einem zweidimensionalen Raster dargestellt.*“ —> *D*as zweidimensionale Raster habe ich als zweidimensionalen Array umgesetzt und mit einer Schleife jedem Kristall einen zufälligen Punkt auf der Achse zugewiesen.

2. *„Von einem Keim aus wächst der Kristall schrittweise in die vier Raster-Richtungen, also nach links, rechts, oben und unten.“* —> Ich habe eine Anzahl an Epochen festgelegt (Zeit), pro Epoche wird jeder Punkt auf dem Array durchgegangen, falls auf diesem ein Kristall ist (der Wert somit nicht mehr 0 ist) wird dieser entlang jeder Richtung erweitert. Daraufhin werden die vier neuen Teile für diese Runde auf die „Blacklist“ gesetzt, das bedeutet sie können sich in dieser Epoche nicht mehr ausbreiten. Es wird also jeweils abgefragt, ob sich ein Kristallkeim

der nicht auf der Blacklist ist auf diesem Feld befindet, nur dann breitet er sich in alle freien Richtungen aus.

3. *„Ein Kristall wächst so weit wie möglich, aber nicht in die Fläche eines anderen Kristalls hinein.“* —> Dafür wird, bevor ein Kristall sich auf ein Feld ausbreitet abgefragt, ob dieses Feld auch leer ist. Dies verhindert, dass Kristalle ineinander wachsen. Dies bringt nur etwas, wenn man die Kristalle am Ende auch auseinanderhalten kann. Dafür habe ich jedem Kristallkeim am Anfang einen unterschiedlichen Wert zwischen 0 und der Anzahl der Kristalle zugewiesen. Dieser Wert wird bei der Ausbreitung an die Äste weitergegeben.

4. *„Für jede dieser Richtungen hat ein Kristall eigene Wachstumsgeschwindigkeiten.“* —> Hierfür habe ich am Anfang einen Höchst- und Tiefstwert für die Ausbreitung festgelegt. Daraufhin wird jeder Richtung eines Kristalls eine zufällige Ausbreitungsgeschwindigkeit in diesem Rahmen zugeordnet. Es gibt einen Wert (hier „a“ genannt), der bei 0 beginnt und pro Epoche immer weiter erhöht wird, bis er dem höchsten Geschwindigkeitswert entspricht. Dann wird er wieder 0 gesetzt. Ein Kristall breitet sich nur dann aus, wenn seine Ausbreitungsgeschwindigkeit in der Epoche kleiner als a ist. Niedrige Werte bedeuten also hohe Geschwindigkeit. Dadurch breitet sich jeder Kristall in jede Richtung mit unterschiedlicher Geschwindigkeit aus.

5. *„Die aus der jeweiligen Orientierung resultierenden unterschiedlichen Lichtreflektionen der Kristalle werden als Grautöne repräsentiert.“* —> Ich habe es so verstanden, dass einem kompletten Kristall, je nachdem in welche Richtung er sich am meisten ausgebreitet hat ein Grauton zugeordnet wird. Hierfür habe ich zuerst für jeden Kristall die durchschnittliche X-Koordinate und die Durchschnittliche Y-Koordinate ermittelt. Daraufhin wird von diesen Werten die X- und Y-Koordinate des Kristallursprungs abgezogen. Wenn der resultierende X-Wert positiv ist, gibt es eine Ausbreitung nach rechts, wenn er negativ ist hat er sich hauptsächlich nach links ausgebreitet. Selbiges Prinzip ist auch auf die Y-Werte anzuwenden. Am Ende habe ich unterschiedliche Gewichtungen für X- und Y-Richtungen festgelegt. **Anmerkung:** Die Bilder sind auf der einen Seite heller und auf der anderen Seite dunkler. Dies liegt daran, dass die Kristalle am Rand sich in diese Richtung besser ausbreiten können, da keine anderen Kristalle ihre Ausbreitung in diese Richtung stören. Deswegen sind Kristalle am linken Rand eher nach links orientiert und daher dunkler.

6. *„Die Kristalle sollen zu unterschiedlichen Zeitpunkten entstehen“* —> Hierfür habe ich einen neuen Parameter eingeführt, der angibt wie viele Kristallkeime pro Epoche hinzugefügt werden sollen. Am Anfang jeder Epoche wir diese Anzahl an Kristallkeimen platziert.

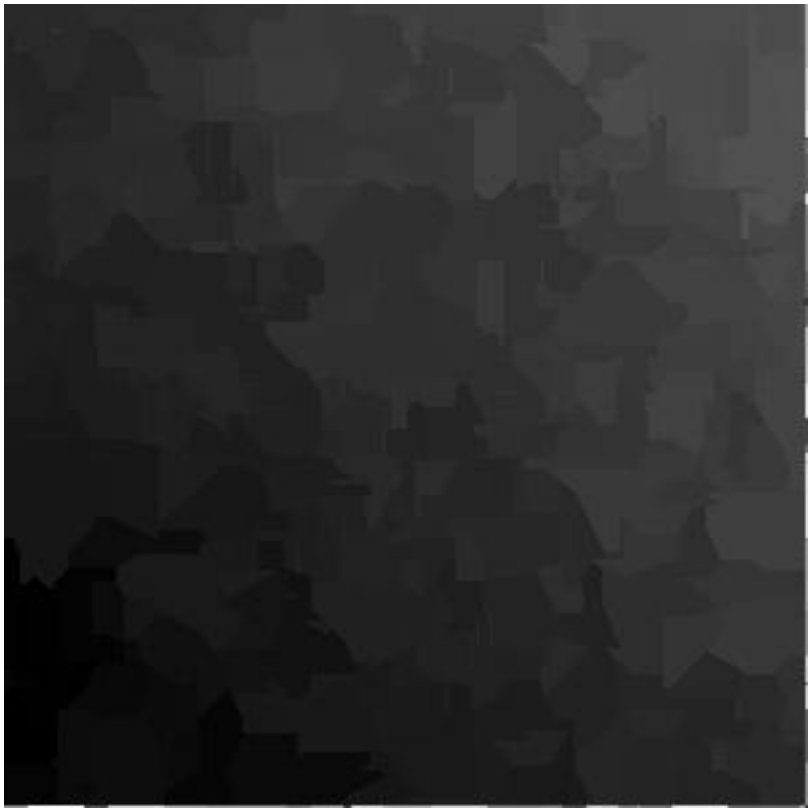
Beispiele:

1. Size: 400; Kristallkeime: 500; Zeit: 100; Ausbreitungsgeschwindigkeit:[0,10]; Kristallkeime_Pro_Runde: 20

2. Size: 300: Kristallkeime: 500; Ausbreitungsgeschwindigkeit:[0,10]; Kristallkeime_Pro_Runde: 20

3. Size: 300; Kristallkeime: 200; Zeit: 100; Ausbreitungsgeschwindigkeit:[0,10]; Kristallkeime_Pro_Runde: 3

4. Size: 300; Kristallkeime: 400; Zeit: 100; Ausbreitungsgeschwindigkeit:[0,50]; Kristallkeime_Pro_Runde: 20



Quellcode:

```
import numpy as np
import random as rndm
import imageio

# Eingabe für die Werte:
Size = 300
Kristallkeime = 400
Zeit = 100
Ausbreitungsgeschwindigkeit = [0,50]
Kristallkeime_Pro_Runde = 20


FlächeErsatz = np.zeros((Size+1,Size+1,5))
Fläche = np.zeros((Size+1,Size+1,5))
Ursprung = np.empty((Kristallkeime,2))


#Kristallmuster berechnen

for i in range(Kristallkeime):
    x = rndm.randint(0,Size)
    y = rndm.randint(0,Size)
    Ursprung[i,0] = x
    Ursprung[i,1] = y
    FlächeErsatz[x][y][0] = i
    FlächeErsatz[x][y][1] = rndm.randint(Ausbreitungsgeschwindigkeit[0],Ausbreitungsgeschwindigkeit[1])
    FlächeErsatz[x][y][2] = rndm.randint(Ausbreitungsgeschwindigkeit[0],Ausbreitungsgeschwindigkeit[1])
    FlächeErsatz[x][y][3] = rndm.randint(Ausbreitungsgeschwindigkeit[0],Ausbreitungsgeschwindigkeit[1])
    FlächeErsatz[x][y][4] = rndm.randint(Ausbreitungsgeschwindigkeit[0],Ausbreitungsgeschwindigkeit[1])
a = Ausbreitungsgeschwindigkeit[0]

Kristallkeime_Anfang = 0
Kristallkeime_Ende = Kristallkeime_Pro_Runde
for g in range (Zeit):
    if Kristallkeime_Ende < Kristallkeime:
        for f in range(Kristallkeime_Anfang,Kristallkeime_Ende):
            if Fläche[int(Ursprung[f,0])][int(Ursprung[f,1])][0] == 0:
                Fläche[int(Ursprung[f,0])][int(Ursprung[f,1])][:] = FlächeErsatz[int(Ursprung[f,0])][int(Ursprung[f,1])][:]
        Kristallkeime_Anfang += Kristallkeime_Pro_Runde
        Kristallkeime_Ende += Kristallkeime_Pro_Runde
    Blacklist = np.zeros((Size+1,Size+1))
    print("Berechnung Durchlauf:", g)
    for t in range (Size):
        for i in range (Size):
            if Fläche[t][i][0] != 0 and Blacklist[t][i] == 0:
                if Fläche[t+1][i][0] == 0 and Fläche[t][i][1] <= a:
                    Fläche[t+1][i] = Fläche[t][i]
                    Blacklist[t+1][i] = 1
                if Fläche[t-1][i][0] == 0 and Fläche[t][i][2] <= a:
                    Fläche[t-1][i] = Fläche[t][i]
                    Blacklist[t-1][i] = 1
                if Fläche[t][i+1][0] == 0 and Fläche[t][i][3] <= a:
```

```
                Fläche[t][i+1] = Fläche[t][i]
                Blacklist[t][i+1] = 1
            if Fläche[t][i-1][0] == 0 and Fläche[t][i][4] <= a:
                Fläche[t][i-1] = Fläche[t][i]
                Blacklist[t][i-1] = 1
    a += 1
    if a >= Ausbreitungsgeschwindigkeit[1]:
        a = 0



# Richtung der Kristallle berechnen


Horizontaleauslenkung = np.empty((Kristallkeime))
Vertikaleauslenkung = np.empty((Kristallkeime))
Gessamtauslenkung = np.empty((Kristallkeime))

for i in range(Kristallkeime):
    Kristallnummer = Fläche[int(Ursprung[i,0]),int(Ursprung[i,1]),0]
    Koordinate = np.empty((Size**2,2))
    b = 0
    for t in range (Size):
        for k in range (Size):
            if Fläche[t][k][0] == Kristallnummer:
                Koordinate[b][0] = t
                Koordinate[b][1] = k
                b +=1
    Horizontaleauslenkung[i] = np.average(Koordinate[:][0]) - Ursprung[i][0]
    Vertikaleauslenkung[i] = np.average(Koordinate[:][1]) - Ursprung[i][1]
    Gesamtauslenkung = (Horizontaleauslenkung * 0.75) + (Vertikaleauslenkung * 0.25)


 # Richtung der Kristalle zuweisen

for t in range (Size):
   for i in range (Size):
        Fläche[t][i][0] = Gesamtauslenkung[int(Fläche[t][i][0])]
imageio.imwrite('/Users/Luis/Desktop/Kristallmuster/100.jpeg', Fläche[:,:,0])
```

Aufgabe 5: Hüpfburg

Team-ID: 00878

Team: Advanced Beginners

Bearbeiter/-innen dieser Aufgabe:
Jonas Lawonn

5.11.2022

Inhaltsverzeichnis

Lösungsidee

Bei dieser Aufgabe liegt ein Parkour vor, welcher aus nummerierten Felden besteht. Diese Felder müssen nicht geordnet sein und sind mit Pfeilen, welche nur von einem auf ein anderes Feld zeigen können, verbunden. Sasha und Mika starten auf den Feldern 1 und 2. Bei jeder Runde des Spiels können beide entlang eines Pfeils auf ein anderes Feld gehen. Das Ziel der beiden ist es in einer Runde auf demselben Feld zu landen.

Meine Lösungsidee besteht darin zunächst einmal alle möglichen Wege die Sasha und Mika nehmen können. Um zu überprüfen, ob sie sich irgendwann treffen, müssen sie bei je einem ihrer möglichen Wege nach derselben Anzahl der Schritte auf demselben Feld stehen. Wenn es einen solchen Fall gibt, so ist der Parkour möglich. Da diese möglichen Wege theoretisch unendlich lang sein können, muss man für jeden Weg überprüfen, ob dieser zuvor schonmal denselben Pfeil entlang gegangen ist. Wenn man nun bei zwei Wegen, der beiden, in derselben Runde einen zuvor benutzten Pfeil entlang geht, bedeutet das, dass diese Wege sich nie in derselben Runde treffen werden und es damit keine Lösung für diese zwei Wege gibt. Wenn man nun alle Wege verglichen hat und es keine Lösung gibt, ist der Parkour unmöglich.

Umsetzung

Ich habe die Aufgabe in Javascript gelöst. Man kann das Programm durch öffnen der index.html Datei in einem Browser starten. (Beim Programmieren habe ich zum öffnen Microsoft Edge verwendet)

Um alle möglichen Wege der beiden zu berechnen, lassen wir sie in jeder Runde in alle möglichen Richtungen gehen. Die Felder, auf denen sie dann landen werden in dem Array „gotoSahsa“ / „gotoMika“ gespeichert und als Ausgangspunkte für die nächste Runde benutzt. Auch werden diese Arrays in einem zweidimensionalen Array „pastPosSasha“ / „pastPosMika“ gespeichert.

Dieser Vorgang wiederholt sich so lange bis:

1. Ein Punkt in dem Array gotoSasha einem Punkt in gotoMika entspricht. Dies würde bedeuten, dass es einen Treffpunkt und damit eine Lösung für den Parkour gibt.
2. In einer Runde das Array gotoSasha einem vergangenem gotoSasha in pastPosSasha entspricht und das Array gotoMika einem vergangenem gotoMika in pastPosMika entspricht. Dies würde bedeuten, dass wir uns in einer Dauerschleife befinden und es für diesen Parkour keine Lösung gibt.

Falls es nach Fall 1. Eine Lösung gibt müssen wir nur noch herausfinden wo lang die beiden nun gehen müssen. Ich mache dies mit einer Art Backtracing Algorithmus.

„Ich erkläre unten zur Veranschaulichung den Algorithmus am Beispiel des Weges von Sahsa. Natürlich wird er eigentlich auf beide angewandt.“

Dieser startet zunächst vom gefundenen Treffpunkt und schaut sich alle Pfeile an, welche auf dieses Feld zeigen. Dann überprüft er, ob einer dieser Pfeile als Ausgangspunkt eine vergangene Position von Sahsa in pastPosSasha ist. Dieser Punkt wird dann in dem Array „traceGotoShasha“ gespeichert, welches dann wiederum in dem zweidimensionalen Array „posiblePosSasha“ gespeichert wird. Im nächsten Durchlauf des Algorithmus werden anstatt des Treffpunktes die zuvor gefundenen Punkte in traceGotoSasha untersucht.

Dieser Vorgang wiederholt sich so lange bis man am Startpunkt von Sasha wieder rauskommt.

Als Ergebnis bekommt man einen zweidimensionalen Array mit allen möglichen gegangenen Feldern pro Runde. Bei der Ausgabe benutze ich immer nur den ersten möglichen Weg.

Beispiele

1. Für das erste Beispiel habe ich einen eigenen kleinen Parkour gebaut:

   Inhalt der txt-Datei:

   5 5

   1 2

   2 3

   3 4

   4 5

   5 1

   Ausgabe: „Nein, dieser Parkour ist nicht moeglich!“

   Dieser Parkour beschreibt ein Kreislauf und ist unlösbar. Dieses Beispiel soll zeigen, dass das Programm den Kreislauf erkennt.

2. ***Aus der Aufgabenstellung ging nicht klar heraus was passieren soll, wenn es von einem Feld nicht weiter geht. Ich habe dieses Problem so gelöst, dass die Person auf einem solchen Feld einfach stehen bleibt.***

Dieses Beispiel zeigt was dann passiert:

Inhalt der txt-Datei:

5 4

2 3

3 4

4 5

1 5

Ausgabe: „*Ja, der Parkour ist moeglich. Man trifft sich auf dem Feld: 5. Nach 3 Runden.*

Sashas Weg: 1-->5 (Kein Pfeil geht von hier aus weiter!)

Mikas Weg: 2-->3-->4-->5“

3. Ausgabe aus bwinf-Datei: huepfburg0.txt

   „*Ja, der Parkour ist moeglich. Man trifft sich auf dem Feld: 10. Nach 3 Runden.*

   Sashas Weg: 1-->18-->13-->10

   Mikas Weg: 2-->19-->20-->10“

4. Ausgabe aus bwinf-Datei: huepfburg1.txt

   „*Ja, der Parkour ist moeglich. Man trifft sich auf dem Feld: 4. Nach 121 Runden.*

   *Sashas Weg: 1-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-->17-
   ->1-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-->17-->1-->4-->5-
   ->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-->17-->1-->4-->5-->6-->7-->8-
   ->9-->10-->11-->12-->13-->14-->15-->16-->17-->1-->4-->5-->6-->7-->8-->9-->10-
   ->11-->12-->13-->14-->15-->16-->17-->1-->4-->5-->6-->7-->8-->9-->10-->11-->12-
   ->13-->14-->15-->16-->17-->1-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-
   ->15-->16-->17-->1-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-
   ->17-->1-->4*

   *Mikas Weg: 2-->3-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-
   ->17-->1-->2-->3-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-
   ->17-->1-->2-->3-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-
   ->17-->1-->2-->3-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-
   ->17-->1-->2-->3-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-
   ->17-->1-->2-->3-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-
   ->17-->1-->2-->3-->4-->5-->6-->7-->8-->9-->10-->11-->12-->13-->14-->15-->16-
   ->17-->1-->2-->3-->4*“

5. Ausgabe aus bwinf-Datei: huepfburg2.txt

   „*Ja, der Parkour ist moeglich. Man trifft sich auf dem Feld: 27. Nach 8 Runden.*

   Sashas Weg: 1-->51-->76-->59-->42-->65-->54-->92-->27

Mikas Weg: 2-->106-->136-->108-->100-->12-->83-->72-->27“

6. Ausgabe aus bwinf-Datei: huepfburg3.txt

   „*Nein, dieser Parkour ist nicht moeglich!*“

7. Ausgabe aus bwinf-Datei: huepfburg4.txt

   „*Ja, der Parkour ist moeglich. Man trifft sich auf dem Feld: 12. Nach 16 Runden.*

   Sashas Weg: 1-->99-->89-->79-->78-->77-->76-->66-->56-->55-->54-->44-->43-->33-->23-->13-->12

   Mikas Weg: 2-->12-->11-->100-->12-->11-->100-->2-->12-->11-->100-->2-->12-->11-->100-->2-->12“

Quellcode

```
let input = document.querySelector("input");
let textarea = document.querySelector("textarea");
input, addEventListener("change", () => {
    let files = input.files;

    if (files.length == 0) {
        return;
    }
    const file = files[0];
    let reader = new FileReader();

    reader.onload = (e) => {
        console.log("start");
        textarea.value = null;
        const file = e.target.result; //Inhalt der txt in file speichern
        const lines = file.split(/\r\n|\n/); //Inhalt in Zeilen unterteilen
        const numbOf = lines[0].split(" "); // Anzahl der Felder / Pfeile

        //Speicherung der Pfeile in einer Array
        var arrows = [];
        for (var k = 1; k <= parseInt(numbOf[1]); k++) {
            var arrow = lines[k].split(" ");
            arrows.push(arrow);
        }

        //Startpositionen und Variabeln zur speicherung der vergangenen Positionen
        var pastPosSasha = [["1"]];
        var pastPosMika = [["2"]];

        var treffpunkt;

        //Laufen

        var onSame = 0; //Variabel zur Überprüfung ob sie sich auf dem selben Feld befinden
        var curPosIsSame = 0; // Variabel zur Überprüfung ob sie die selben Positionen haben
wie zuvor
        var round = 0;
        while (onSame == 0 && curPosIsSame == 0) {
            //Sasha
            var gotoSasha = []; //Array für die Positionen wo sie hin kann
```

```
            var curPosSasha = []; //Array der derzeitigen Positionen
            curPosSasha = pastPosSasha[round];

            // Berechnung der Möglichen Schritte und Speicherung dieser in gotoSasha
            for (var k = 0; k < curPosSasha.length; k++) {
                var countWA = 0;
                for (var i = 0; i < arrows.length; i++) {
                    if (curPosSasha[k] == arrows[i][0] && gotoSasha.indexOf(arrows[i][1]) ===
-1) {
                        gotoSasha.push(arrows[i][1]);
                    }
                    else if (gotoSasha.indexOf(arrows[i][1]) === -1) { //zähle unzutreffende
Pfeile
                        countWA++;
                    }

                }
                if (countWA == arrows.length) { //wenn es keinen Pfeil gibt dann bleibt sie
stehen
                    gotoSasha.push(curPosSasha[k]);
                }
            }


            //Mika
            var gotoMika = []; //Array für die Positionen wo sie hin kann
            var curPosMika = []; //Array der derzeitigen Positionen
            curPosMika = pastPosMika[round];

            // Berechnung der Möglichen Schritte und Speicherung dieser in gotoMika
            for (var k = 0; k < curPosMika.length; k++) {
                var countWA = 0;
                for (var i = 0; i < arrows.length; i++) {
                    if (curPosMika[k] == arrows[i][0] && gotoMika.indexOf(arrows[i][1]) ===
-1) {
                        gotoMika.push(arrows[i][1]);
                    }
                    else if (gotoMika.indexOf(arrows[i][1]) === -1) { //zähle unzutreffende
Pfeile
                        countWA++;
                    }

                }
                if (countWA == arrows.length) { //wenn es keinen Pfeil gibt dann bleibt sie
stehen
                    gotoMika.push(curPosMika[k]);
                }
            }


            //Check if onSame/Sind auf dem selben Feld zur selben Zeit
            for (var k = 0; k < gotoSasha.length; k++) {
                for (var i = 0; i < gotoMika.length; i++) {
                    if (gotoSasha[k] == gotoMika[i]) {
                        onSame = 1;
                        treffpunkt = gotoSasha[k]
                    }
                }
            }

            //Check if curPosIsSame/Sind auf den selben Feldern zur selben Zeit wie schonein-
mal => Loop
```

```
            var mika = 0;
            var sasha = 0;
            for (var k = 0; k < pastPosSasha.length; k++) {
                if (pastPosSasha[k].toString() == gotoSasha.toString()) {
                    sasha = 1;
                }
            }


            for (var k = 0; k < pastPosMika.length; k++) {
                if (pastPosMika[k].toString() == gotoMika.toString()) {
                    mika = 1;
                }
            }

            if (mika == 1 && sasha == 1) {
                curPosIsSame = 1;
            }

            //set new pastPoses
            pastPosMika.push(gotoMika);
            pastPosSasha.push(gotoSasha);

            round++;
        }

        //Backtrace/ Zurückverfolgung der gegeangenen Schritte
        var roundBack = round;
        var traceStep = 0;
        var posiblePosSasha = [[treffpunkt, treffpunkt]];
        var posiblePosMika = [[treffpunkt, treffpunkt]];
        while (roundBack > 0 && curPosIsSame == 0) {
            roundBack--;

            //Sasha
            var traceGotoSasha = [];
            var curPosiPosSasha = [];
            curPosiPosSasha = posiblePosSasha[traceStep];
            //console.log(curPosiPosSasha);
            for (var k = 0; k < curPosiPosSasha.length; k++) {
                for (var i = 0; i < arrows.length; i++) {
                    if (curPosiPosSasha[k] == arrows[i][1]) { //Überprüfe ob ein Pfeil auf
das Feld zeigt auf dem sie sich befindet
                        for (var l = 0; l < pastPosSasha[roundBack].length; l++) { // Über-
prüfe ob der Oben erkannte Pfeil von einer Position ausgeht auf dem sie war
                            if (arrows[i][0] == pastPosSasha[roundBack][l]) {
                                traceGotoSasha.push(pastPosSasha[roundBack][l]); //Speichern
des zurückverfolgten Schrittes in traceGotoSasha
                            }
                        }
                    }
                }
            }
            posiblePosSasha.push(traceGotoSasha);

            //Mika
            var traceGotoMika = [];
            var curPosiPosMika = [];
            curPosiPosMika = posiblePosMika[traceStep];
            for (var k = 0; k < curPosiPosMika.length; k++) {
                for (var i = 0; i < arrows.length; i++) {
```

```
                    if (curPosiPosMika[k] == arrows[i][1]) { //Überprüfe ob ein Pfeil auf das
Feld zeigt auf dem sie sich befindet
                        for (var l = 0; l < pastPosMika[roundBack].length; l++) { // Überprüfe
ob der Oben erkannte Pfeil von einer Position ausgeht auf dem sie war
                            if (arrows[i][0] == pastPosMika[roundBack][l]) {
                                traceGotoMika.push(pastPosMika[roundBack][l]);   //Speichern
des zurückverfolgten Schrittes in traceGotoMika
                            }
                        }
                    }
                }
            }
            posiblePosMika.push(traceGotoMika);

            traceStep++;
        }
        if (curPosIsSame == 0) {
            //Output Weg von Sasha
            var sWege = "Sashas Weg: 1-->";
            var sWasNull = 0; //Überprüfe ob posiblePosSasha mal null war
            for (var k = posiblePosSasha.length - 2; 0 <= k; k--) {
                if (posiblePosSasha[k][0] != null) {
                    sWege += posiblePosSasha[k][0];
                    if (k > 0) {
                        sWege += "-->";
                    }
                }
                else {
                    sWasNull = 1;
                }
            }
            if (sWasNull == 1) {
                sWege += " (Kein Pfeil geht von hier aus weiter!)";
            }

            //Output Weg von Mika
            var mWege = "Mikas Weg: 2-->";
            var mWasNull = 0;//Überprüfe ob posiblePosMika mal null war
            for (var k = posiblePosMika.length - 2; 0 <= k; k--) {
                if (posiblePosMika[k][0] != null) {
                    mWege += posiblePosMika[k][0];
                    if (k > 0) {
                        mWege += "-->";
                    }
                }
                else {
                    mWasNull = 1;
                }
            }
            if (mWasNull == 1) {
                mWege += " (Kein Pfeil geht von hier aus weiter!)";
            }
        }

        //output
        if (onSame == 1 && curPosIsSame == 0) {
            var s = "Ja, der Parkour ist moeglich. Man trifft sich auf dem Feld: " + treffpunkt
+ ". Nach " + (round) + " Runden.";
            s += "\n" + sWege + "\n" + mWege;
            textarea.value += s;
        }
        else if (onSame == 0 && curPosIsSame == 1) {
```

```
            var s = "Nein, dieser Parkour ist nicht moeglich!"
            textarea.value += s;
        }

        //debug
        console.log(pastPosSasha);
        console.log(pastPosMika);
        console.log(posiblePosSasha);
        console.log(posiblePosMika);


    }
    reader.readAsText(file);
});
```