26. Bundeswettbewerb Informatik, 1. Runde

Lösungshinweise und Bewertungskriterien

Allgemeines

Zuerst soll an dieser Stelle gesagt sein, dass wir uns sehr darüber gefreut haben, dass einmal mehr so viele Leute sich die Mühe gemacht und die Zeit zur Bearbeitung der Aufgaben genommen haben. Dass das Zeitnehmen nicht immer optimal klappt und es deshalb kurz vor Einsendeschluss etwas brenzlig wird, ist nachvollziehbar und völlig verständlich. Leider dürfen wir darauf keine Rücksicht nehmen. Insbesondere sind vollständige Einsendungen ein Muss. Im Einzelnen:

- Online-Anmelder wurden gebeten, ihre Nummer außen auf den Umschlag zu schreiben. Viele haben das gemacht, aber beinahe ebenso viele leider nicht. Das führt zu Komplikationen und möglicherweise zum Ausbleiben der versprochenen Rückmeldung per E-mail über den Eingang der Einsendung.
- Eine Gruppeneinsendung schicken Sie bitte komplett in einem gemeinsamen Umschlag, wir haben sonst größte Mühe, die Einsendungen richtig zuzuordnen. Wenn mehrere Einsendungen in einen Umschlag gesteckt werden, ist es besonders wichtig, bei der Online-Anmeldung bzw. auf den Anmeldebögen die Zusammensetzung der Gruppe anzugeben. Außerdem: Eine Gruppe muss sich auf eine Lösung pro Aufgabe einigen, und Gruppenmitglieder können nicht gleichzeitig auch eine eigene Einsendung schicken.
- Seien Sie mit Ihrem Namen nicht so geizig! Schreiben Sie ihn ruhig häufiger, z. B. auf das erste Blatt jeder Aufgabe und auf Ihre CD.
- Lösungsidee, Programm-Dokumentation und insbesondere auch Programm-Ablaufprotokolle und ausreichend viele Beispiele müssen ausgedruckt sein. Wir können aus Zeit- und Kostengründen keine Ausdrucke machen und auch nicht jedes eingesandte Programm ausführen.
- Noch schlechter als Einsendungen nur auf Datenträgern wären für uns übrigens Einsendungen via E-Mail oder anderen Internet-Wegen, auch wenn das für die Teilnehmer noch so praktisch wäre. Papiereinsendungen sind (zumindest zur Zeit und sicher auch noch in den nächsten Jahren) einfach unumgänglich.

- Beispiele werden als Teile des Programm-Ablaufprotokolls immer erwartet. Zu wenige Beispiele und erst recht die Nichtbearbeitung vorgegebener Beispiele führen zu Punktabzug. Es ist auch nicht ausreichend, Beispiele nur auf CD abzugeben, ins Programm einzubauen oder den Bewertern das Erfinden und Testen von Beispielen zu überlassen. Ohne abgedruckte Beispiele ist die Bewertung einer Lösung in der knappen vorhandenen Zeit nicht möglich. Leider fehlten in vielen Einsendungen die Beispiele, was oft das Erreichen der zweiten Runde verhindert hat.
- Zu einer Einsendung gehören auch lauffähige Programme. Kompilierung von Quellcode ist während der Bewertung nicht möglich. Für die gängigsten Skript-Sprachen stehen Interpreter zur Verfügung. Nutzen Sie aber bitte dennoch jede Möglichkeit, eigenständig ausführbare Programme abzugeben.

So, vielleicht denken Sie ja an diese Anmerkungen, wenn Sie (hoffentlich) im nächsten Jahr wieder mitmachen.

Auch die folgenden eher inhaltlichen Dinge sollten Sie beachten:

- Lösungsideen sollten Lösungsideen sein und keine Bedienungsanleitungen oder Wiederholungen der Aufgabenstellung. Es soll beschrieben werden, welches Problem hinter der Aufgabe steckt und wie dieses Problem grundsätzlich angegangen wird. Eine einfache Mindestbedingung: Bezeichner von Programmelementen wie Variablen, Prozeduren etc. dürfen nicht verwendet werden – eine Lösungsidee ist nämlich unabhängig von solchen Realisierungsdetails. In diesem Jahr waren die meisten Lösungsideen in dieser Hinsicht recht gut, oft aber ein wenig kurz.
- Auch ein Programmablauf-Protokoll soll keine Bedienungsanleitung sein. Es beschreibt nicht, wie das Programm ablaufen sollte, auch nicht die zum Ablauf nötigen Interaktionen mit dem Programm, sondern protokolliert den tatsächlichen, inneren Ablauf eines Programms. Am besten protokolliert ein Programm seinen Ablauf selbst, z. B. durch Herausschreiben von Eingaben, Zwischenschritten oder -resultaten und Ausgaben.
- Oben wurde schon gesagt, dass Beispiele immer dabei sein sollten, zumindest eines davon in einem Programm-Ablaufprotokoll. Das hat seinen Grund: An den Beispielen ist oft direkt zu sehen, ob bestimmte Punkte korrekt beachtet wurden. Viele meinen nun, wir könnten die Programme ja laufen lassen und selbst auf Beispieldaten ansetzen, und liefern keine Beispiele oder nur Beispieldaten in elektronischer Form. Das können wir aber aus Zeitmangel in der Regel nicht. Außerdem ist nicht immer sicher, dass Programme, die auf dem eigenen PC laufen, auch auf einem anderen Computer ausführbar sind. Generell muss man sich darauf einstellen, dass nur das Papiermaterial angesehen wird!
- Mit den verschiedenen Beispielen sollten Sie wichtige Varianten des Programmablaufs zeigen, also auch Sonderfälle, die Ihre Lösung behandeln kann. Die Konstruktion solcher Testfälle ist eine ganz wesentliche Tätigkeit des Programmentwurfs.

Einige Anmerkungen noch zur Bewertung:

- Pro Aufgabe werden maximal fünf Punkte vergeben, bei Mängeln gibt es entsprechend weniger Punkte. Bei der ersten Aufgabe war es diesmal möglich, einen Bonuspunkt zu

erzielen. Für die Gesamtbewertung sind die drei am besten bewerteten Aufgabenlösungen maßgeblich, es lassen sich also maximal 15 bzw. bei Bearbeitung von Aufgabe 1 maximal 16 Punkte erreichen. Einen 1. Preis erreichen Sie mit 14, 15 oder 16 Punkten, einen 2. Preis mit 12 oder 13 Punkten und eine Anerkennung mit 10 oder 11 Punkten. Die Preisträger sind für die zweite Runde qualifiziert.

- Auf den Bewertungsbögen bedeutet ein Kreuz in einer Zeile, dass die (negative) Aussage in dieser Zeile auf Ihre Einsendung zutrifft. Damit verbunden ist dann in der Regel der Abzug eines oder mehrerer Punkte. Eine Wellenlinie bedeutet „na ja, hätte besser sein können“, führt aber meist nicht zu Punktabzug. Mehrere Wellenlinien können sich aber zu einem Punktabzug addieren. Gelegentlich sind lobende Anmerkungen der Bewerter mit einem ’+’ versehen.
- Wellenlinien wurden übrigens häufig für die Dokumentation (also Lösungsidee, Programm-Dokumentation, Programm-Ablaufprotokoll und kommentierter Programm-Text) verteilt, obwohl Punktabzug auch gerechtfertigt gewesen wäre.
- Aber auch so ließ sich nicht verhindern, dass etliche Teilnehmer nicht weitergekommen sind, die nur drei Aufgaben abgegeben haben in der Hoffnung, dass schon alle richtig sein würden. Das ist ziemlich riskant, da Fehler sich leicht einschleichen.

Zum Schluss:

- Sollte Ihr Name auf der Urkunde falsch geschrieben sein, können Sie gerne eine neue anfordern. Uns passieren durchaus schon mal Tippfehler, und gelegentlich scheitern wir bei Anmeldungen auf Papierformular an der ein oder anderen Handschrift.
- Es ist verständlich, wenn jemand, der nicht weitergekommen ist, über eine Reklamation nachdenkt. Gehen Sie aber bitte davon aus, dass wir kritische Fälle, insbesondere die mit 11 Punkten, schon genau und mit Wohlwollen geprüft haben.

Danksagung

An der Erstellung der Lösungsideen haben mitgewirkt: Felix Arends und Thomas Leineweber (Aufgabe 1), Karl Bringmann (Aufgabe 2), Arno Bücken (Aufgabe 4), Jan Balzer (Aufgabe 5) und Johann Felix von Soden-Fraunhofen (Junioraufgabe).

Die Aufgaben wurden vom Aufgabenausschuss des BWINF entwickelt, aus Vorschlägen von Torben Hagerup (Aufgabe 2 und Aufgabe 5), Wolfgang Pohl (Junioraufgabe), Peter Rossmanith (Aufgabe 1 und Aufgabe 4) und Monika Seiffert (Aufgabe 3).

Aufgabe 1: Prämienjagd

Lösungsidee

Matching

Bei dieser Aufgabe muss der Centbetrag betrachtet werden, der sich ergibt, wenn man die Preise von zwei nebeneinanderliegenden Gegenständen addiert. Die Gegenstände sollen so nebeneinandergelegt werden, dass jeder Gegenstand nur einmal in eine Paarung aufgenommen wird und dass der Centanteil der Summe entweder 11, 33, 55, 77 oder 99 Cent beträgt.

Wenn man genauer hinschaut, merkt man, dass diese Summen nur dann erreicht werden können, wenn der Preis des einen Gegenstandes einen geraden Centanteil, der andere Preis einen ungeraden Centanteil hat. Damit kann ein Gegenstand mit einem geraden Preis nur mit einem Gegenstand mit einem ungeraden Preis zusammen eine Prämie ergeben. Die Menge der Preise besteht also aus zwei Teilen (sie ist bipartit), und es soll eine möglichst große Menge eineindeutiger Zuordnungen (ein Matching) zwischen Elementen dieser Teilmengen gefunden werden.

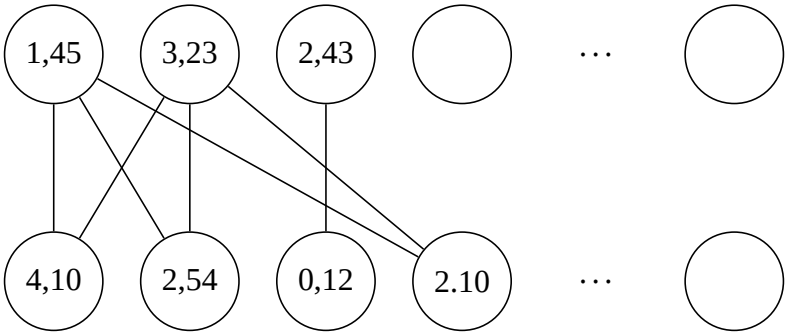


Abbildung 1: Bipartites Matching mit einzelnen Gegenständen

Die Aufgabe kann also auf die Berechnung eines maximalen bipartiten Matchings[1] zurückgeführt werden. Dieses Matching beschreibt, welche Gegenstände miteinander zusammengelegt werden müssen, damit so viele Prämien wie möglich ergattert werden. Dies ist in Abbildung 1 so dargestellt, dass in zwei Reihen die Gegenstände mit geraden bzw. ungeraden Preisen gegenübergestellt werden, wobei jeder Gegenstand einen eigenen Knoten bildet. Anschließend werden die Gegenstände dann miteinander verbunden, wenn die Summe der Preise das Paarungskriterium erfüllt. Damit haben wir einen bipartiten Graphen, auf dem ein maximales Matching berechnet werden muss.

Ein bipartites Matching kann mit Hilfe des Algorithmus von Hopcroft und Karp (Spezialfall des Flussalgorithmus von Edmonds und Karp) in der Zeit $O(\sqrt{V}E)$ (V ist die Anzahl der Knoten, E die Anzahl der Kanten) berechnet werden. In dem größten der vom BWINF vorgegebenen Beispiele mit 5 Millionen Preisen hat man es aber mit einem Graph zu tun, der ca. 625 Mrd. Kanten hat (zu jedem der 2.5 Mio. ungeraden Knoten passen im Durchschnitt ungefähr

[1] Als Literatur zu Flussalgorithmen und bipartiten Matchings siehe z. B.: Cormen, Leiserson, Rivest, Stein: Algorithmen – eine Einführung. Oldenbourg Wissenschaftsverlag, 2007. ISBN 978-3486582628.

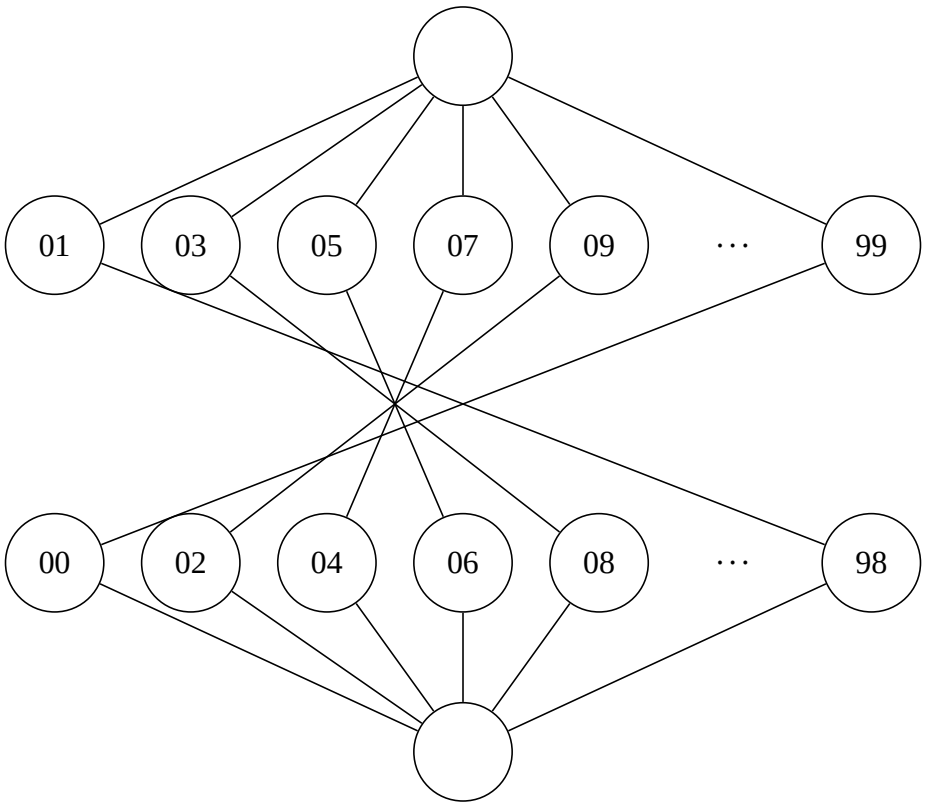


Abbildung 2: Ausschnitt des Flussgraphen mit konstanter Größe (102 Knoten, 350 Kanten), in dem nur für die Centanteile Knoten erzeugt wurden.

10% der 2.5 Mio. geraden Knoten). Aufgrund dieser Zahlen wird ein alternatives Verfahren benötigt, damit die Aufgabe auch für besonders große Eingabemengen lösbar bleibt.

Aufbau eines Flußnetzes / Lösung mit Edmonds Karp

Zunächst lässt sich beobachten, dass für die Bildung der Preispaare nur die Centanteile der Preise eine Rolle spielen. Alle Preise mit gleichem Centanteil sind bei der Bildung von Paaren beliebig austauschbar. Auf der Grundlage dieser Beobachtung kann man das Problem nun ganz unabhängig von der Eingabegröße lösen, denn es gibt immer nur 100 verschiedene Centanteile.

Statt eines bipartiten Graphen lässt sich also ein vereinfachtes Flussnetz erstellen, mit dessen Hilfe die Paarungen berechnet werden können. Es besteht aus 4 Ebenen: einer Quelle, 50 Knoten für ungerade Centanteile, 50 Knoten für gerade Centanteile und die Senke. Von der Quelle geht eine Kante zu jedem der 50 Knoten für die geraden Centanteile. Diese Kanten haben als Kapazität die Anzahl der Gegenstände, die einen Preis haben, der dem jeweiligen Centanteil entspricht. Von jedem der 50 geraden Knoten gibt es 5 Kanten zu jeweils den ungeraden Knoten, mit denen die Summe der Centanteile einen der geforderten Werte liefert. Diese Kanten können unendliche Kapazität haben, es genügt aber auch das Minimum der beiden Gegenstandsanzahlen. Von den ungeraden Knoten geht jeweils eine Kante zur Senke. Diese Kanten haben als Kapazität die Anzahl der Gegenstände mit dem jeweiligen ungeraden Centanteil.

Hier haben wir nun eine konstante Anzahl an Knoten (102) und Kanten (350). Die Berechnung des maximalen Flusses kann mit Hilfe des Algorithmus von Edmonds und Karp unabhängig von der Eingabegröße in konstanter Zeit $O(VE^2)$ (da der Graph konstante Größe hat) berechnet werden. Die Zeit, die für den Aufbau des Flussgraphen benötigt wird, steht in linearem Verhältnis zur Eingabegröße, so dass das Verfahren insgesamt lineare Komplexität hat.

Der berechnete Gesamtfluss sagt aus, wieviele Prämien man insgesamt erhalten kann. Der Fluss auf den Kanten zwischen den geraden und den ungeraden Centanteilen gibt an, wieviele Artikel mit den jeweiligen Centanteilen zusammengelegt werden müssen, um diese Gesamtanzahl an Prämien und damit den Gesamtfluss erreichen zu können.

Für die Ausgabe der zusammengehörigen Artikel muss noch einmal durch den Fluss-Graph gegangen werden, um die Ausgabe aufzusammeln. Dazu merkt man sich zu jedem Centanteil in einer Liste all diejenigen Gegenstände mit ihren Preisen, die diesen Centanteil haben. Damit kann man nun nach Berechnung des Gesamtflusses wie folgt alle Paarungen ausgeben:

- durchlaufe alle ungeraden Knoten V.
- durchlaufe für jeden ungeraden Knoten die ausgehenden Kanten E_i und bestimme deren anderen Knoten V_i und den Fluss $f(E_i)$.
- nehme aus dem Knoten V und dem Knoten V_i entsprechend dem Fluss viele noch nicht vergebene Artikel mit Preis, gebe diese jeweils als Paarung aus und markiere die entnommenen Artikel als schon vergeben.

Greedy-Lösungen

Es hat sich herausgestellt, dass sich das Problem auch mit einem einfacheren Ansatz lösen lässt, der "gierig" vorgeht, also auf dem Greedy-Prinzip basiert. Üblicherweise wurde dieses Prinzip nach der Bestimmung aller Paarungsmöglichkeiten angewandt; es ist aber auch anwendbar, wenn man nur die Centanteile betrachtet. Letzteres ist schon deshalb sinnvoll, um nicht schon bei der Bestimmung aller Paarungsmöglichkeiten jeden Preis mit jedem anderen kombinieren zu müssen.

Die Lösung besteht darin, die Preise (oder Centanteile) herauszusuchen, die die wenigsten Paarungsmöglichkeiten haben. Für diese Preise werden nun Partner gesucht, mit denen sie zusammen eine Prämie bilden. Es wird dabei der Partner genommen, der für sich auch die wenigsten Paarungsmöglichkeiten hat. Nachteil bei diesem Lösungsansatz ist wieder, dass dabei ein sehr großer Graph entstehen kann.

Der Graph muss also auf eine vernünftige Art gespeichert werden, damit die Speicherkomplexität nicht zu gross wird. Hier kann z. B. wieder für jeden möglichen Centanteil eine Liste mit den Artikeln erstellt werden. Nun wird gleichzeitig noch jeweils mit festgehalten, wieviele andere Artikel es als Partner für eine Paarung gibt. Mit diesen Daten kann schnell herausgefunden werden, welche beiden Centanteile miteinander kombiniert werden sollen. Aus diesen Beträgen wird jeweils ein Gegenstand entnommen, und die Buchhaltung, wieviele Paarungen jeweils möglich sind, wird aktualisiert. Danach kann die nächste Paarung gesucht werden. Wenn hierbei Centanteile auftauchen, die keine möglichen Partner mehr haben, fallen die

Beispiel	Anzahl
pj-10	0
pj-20	4
pj-50	21
pj-100	43
pj-200	87
pj-500	249
pj-1000	490
pj-2000	972
pj-5000	2469
pj-10000	4955
pj-20000	9926
pj-50000	24975
pj-100000	49980
pj-200000	99857
pj-500000	249668
pj-1000000	499681
pj-2000000	999587
pj-5000000	2499700

Tabelle 1: Anzahl der Prämien für die vorgegebenen Beispiele

übriggebliebenen Artikel mit diesem Centanteil weg. Mit Hilfe einer geeigneten Prioritätswarteschlange für die Centanteile, die nach der Anzahl der möglichen Paarungen priorisiert, ist auch die Suche nach der jeweils günstigsten Paarung schnell machbar.

Für diesen Algorithmus ist nicht bewiesen, ob er immer korrekte Ergebnisse liefert, aber bei fehlerfreier Umsetzung liefert er für alle bekannten Beispiele korrekte Ergebnisse. Dies scheint wegen der besonderen Struktur des aufgebauten Graphen zu funktionieren.

Falsche Ergebnisse werden aber berechnet, wenn der Greedy-Algorithmus noch weiter vereinfacht wird, z.B. indem in einem Schritt gleich für eine Centanteil-Paarung mehrere Artikel gepaart werden, ohne dass zwischendurch getestet wird, ob jetzt nicht ein anderer Centanteil für eine Paarung vorzuziehen ist. Ebenfalls fehleranfällig ist es, bei der Bestimmung der neuen Paarung nicht beide Partner zu betrachten (erster Partner mit minimalen Paarungsmöglichkeiten, aber zweiter beliebig).

Beispielberechnungen

BWINF-Beispiele

Für die vorgegebenen Beispieleingaben wird in Tabelle 1 die Anzahl der erreichbaren Prämien angegeben. Konkret kann bei den 20 Gegenständen z. B. wie folgt kombiniert werden:

```
(73.82 60.95) (95.84 48.93) (55.88 3.23) (3.98 70.35)
```

Bei den 50 Gegenständen kann z. B. wie folgt kombiniert werden:

```
(74.00 75.33) (17.04 63.07) (86.06 48.93) (87.12 55.99) (98.30 7.69)
(94.32 73.79) (7.38 86.73)  (91.42 82.57) (93.58 20.41) (9.64 70.35)
(7.64 78.35)  (74.80 0.19)  (73.82 60.95) (95.84 63.93) (55.88 0.11)
(35.88 3.23)  (84.92 37.07) (40.92 54.85) (83.96 66.37) (3.98 3.79)
(6.98 42.35)
```

Selbstgewählte Beispiele

Es wurde kein Beweis für die Korrektheit der eigenen Lösung erwartet. Vielen hat es aber genügt, wenn die vom BWINF für drei Beispiele vorab veröffentlichten Ergebnisse auch von ihrer Lösung berechnet wurden. Besser ist es, mit eigenen Eingabedaten, die bestimmte strukturelle Eigenschaften haben, die Richtigkeit einer Lösung zumindest zu untermauern. Solche Eigenschaften können sein:

- Nur ungerade bzw. nur gerade Preise.
- Die Paarung von Gegenständen ist immer eindeutig (es gibt zu einem Gegenstand maximal einen anderen Gegenstand, mit dem er gepaart werden kann).
- Jeder Gegenstand mit geradem Centbetrag kann mit jedem Gegenstand mit ungeradem Centbetrag gepaart werden. Dies jeweils mit gleichen oder unterschiedlichen Centbeträgen.

Bewertungskriterien

Bei der Veröffentlichung der BWINF-Beispieleingaben wurde klargestellt, dass die von den Teilnehmern entwickelten Lösungen nicht alle Beispiele schaffen müssten. Damit kann die volle Punktzahl auch von Lösungen erreicht werden, die Eingaben vierstelliger Größe in kurzer Zeit verarbeiten können. Voraussetzung für die volle Punktzahl ist die Erkennntnis, dass nicht jeder Preis mit jedem anderen kombiniert werden muss, sondern die Paarbarkeit der Beträge begrenzt ist. Wichtig ist außerdem die Korrektheit der Ergebnisse, die auch mit dem geschilderten Greedy-Verfahren erzielt werden kann.

- Es soll erkannt worden sein, dass die Menge der möglichen Paare stark begrenzt werden kann. Z. B. sollte erkannt werden, dass nur gerade und ungerade Preise miteinander die jeweiligen Cent-Beträge ergeben. Passende ähnliche Erkenntnisse bzw. Optimierungen werden auch akzeptiert, es darf aber nicht jeder Preis mit jedem verglichen werden. Es ist nicht nötig, dass den Teilnehmern die passenden Algorithmen der Graphentheorie bekannt sind, obwohl es uns sicher freut, wenn dem doch so ist.
- Wer auch noch erkannt hat, dass nur die Centanteile betrachtet werden müssen und damit der Aufwand der Berechnung unabhängig von der Eingabegröße ist, soll – in der ersten Runde sonst nicht üblich – mit einem Pluspunkt belohnt werden.

- Da bei Benutzung von nicht effizienten Algorithmen der Aufwand recht schnell sehr groß wird, genügt es, wenn Beispiele mit vierstelligen Eingabegrößen noch in vernünftiger Zeit lösbar sind. Wenn dies nicht der Fall ist (z.B. beim Ausprobieren aller Kombinationsmöglichkeiten – Brute Force), kann ein Punkt abgezogen werden.
- Punktabzug ist auch möglich, wenn es in der Einsendung keine Aussage zu den Leistungsgrenzen des gewählten Verfahrens gibt – dies ist der zentrale Punkt der Aufgabe. Eine Angabe dazu, welche Fälle das eigene Programm in einigermaßen überschaubarer Zeit bearbeiten kann, genügt; eine Abschätzung der Laufzeitkomplexität wird nicht erwartet. Häufig wurden für einige kleinere Beispiele die Ergebnisse angegeben, aber nicht gesagt, ob und wie die Lösung mit größeren Beispielen zurechtkommt.
- Im Aufgabenblatt steht: „Du solltest aber unbedingt darauf achten, dass Deine Lösung immer so viele Prämien wie möglich erwirkt.“ Bei inkorrekten Paarungen bzw. falscher Anzahl von Paarungen wird mindestens ein Punkt abgezogen. In extremen Fällen, etwa wenn sogar bei den Eingaben, für die die korrekten Ergebnisse bekannt waren (das sind die Fälle mit 20, 200 und 20.000 Preisen) falsche Ergebnisse berechnet wurden und dies unkommentiert bleibt, können auch zwei Punkte abgezogen werden.
- Laut Aufgabenblatt müssen nur die Paarungen ausgegeben werden, nicht die Anzahl der Paarungen. Wenn das Programm keine Paarungen ausgibt, wird ein Punkt abgezogen. Die Ausgabe der Anzahl der Paarungen ist nicht explizit gefordert (aber sehr sinnvoll). Abgedruckt werden müssen die Paarungen natürlich nur bei den kleineren Beispielen.
- Die Beispiele sind bei dieser Aufgabe besonders wichtig. Es müssen auf jeden Fall Ergebnisse für mindestens drei der BWINF-Beispiele mit eingesendet werden, idealerweise nicht nur die, für die vorher schon die Lösung bekannt war.
- Darüberhinaus muss es weitere aussagekräftige Beispiele geben, die ein Indiz für die Korrektheit der Lösung liefern. Wenn ein Beweis für die Korrektheit der eigenen Lösung vorhanden ist, reicht es aus, ein weiteres *interessantes(!)* Beispiel zu bringen.
- Kleine Fehlerquellen:
  - nicht-korrektes Behandeln von kostenlosen Artikeln (0.00 €): Kein Abzug (die BWINF-Beispiele sind nicht kostenlos)
  - Wenn die Paarungen nicht vom Programm, sondern von Hand gezählt wurden, kann es dabei zu Übertragungsfehlern kommen. In solchen Fällen wird kein Punkt abgezogen.

Aufgabe 2: Perlenketten

Lösungsidee

Halsschmuck ist in! Besonders schicke Ketten mit frech aufgereihten Kugeln in einer von sechs fetzigen Farben, die einem möglichst oft um den Hals und bis zum Boden reichen. Groovy, Baby!

Aber wie soll man eine eindeutige Darstellung solcher Ketten finden? Von jeder Kugel in eine Richtung gehend kann die Kette doch anders aussehen!

Normalform

Eine einfache Normalform erhält man, indem man die vorkommenden Farben irgendwie anordnet, also festlegt, dass z.B. rot $>$ gelb $>$ blau. Nun kann man Folgen von Farben gleicher Länge mit der lexikographischen Ordnung vergleichen:

```
vergleicheLexikografisch(Folge A, Folge B)
  für i = 1 bis Länge von A und B
    wenn Farbe A[i] > Farbe B[i] dann gib zurück A > B
    wenn Farbe A[i] < Farbe B[i] dann gib zurück A < B
  gib zurück A = B
```

Damit hat man eine totale Ordnung der Farbfolgen, die man bekommt, wenn man von einer Kugel der Kette in eine Richtung geht, bis man zur Ausgangskugel zurückkommt. Und das gute ist, dass diese Ordnung unabhängig davon ist, wie man die Kette zu Anfang bekommen hat, also in welcher Orientierung und Position. Also ist z.B. die maximale Farbfolge einer Kette nach dieser Ordnung eindeutig bestimmt und wir können diese als Normalform der Kette benutzen. Allerdings sind auch andere Normalformen denkbar, wie die minimale Farbfolge, die k-te Farbfolge oder auch der Median der vorkommenden Farbfolgen. Besonders einfach zu berechnen sind allerdings Maximum bzw. Minimum.

Einfacher, aber nicht mehr so leicht nachvollziehbar, wird es, wenn wir die Farben durch Ziffern darstellen (z.B. 1 bis 6) und die Farbfolgen folglich als Zahlen gleicher Stelligkeit. Damit gibt es automatisch eine Ordnung über den Farben und auch eine Ordnung über den Ketten: die maximale Farbfolge einer Kette ist hier einfach die größte der einer Farbfolge entsprechenden Zahlen. Bei langen Ketten entstehen allerdings große Zahlen. Die müssen geeignet gespeichert und verarbeitet werden; in einigen Programmiersprachen sind Mechanismen zur Verarbeitung solcher „Big Integer“ vorhanden, die selbstverständlich benutzt werden dürfen.

Berechnung der Normalform

Sei N die Länge unserer Kette. Dann haben wir $2 \cdot N$ Kandidaten für die maximale Farbfolge, nämlich N Möglichkeiten für die Wahl der Startkugel und 2 für die Wahl der Richtung. Je nach Annahmen über die Muster unserer Ketten können wir Aussagen über die Laufzeit des Vergleichs zweier Farbfolgen treffen, aber im Allgemeinen wird dies wohl lineare Laufzeit benötigen. Damit ergibt sich ein Algorithmus zur Bestimmung der Normalform mit quadratischer Laufzeit $O(N^2)$:

```
bestimmeNormalform(Kette K)
  Setze F[i,r] = Farbfolge der Kette K startend an
        Kugel i in Richtung r, 1 <= i <= N, r = '<' oder '>'
  Setze maxF = F[1,0]
  für jede Farbfolge F[i,r]:
    wenn F[i,r] > maxF dann setze maxF = F[i,r]
  gib zurück maxF
```

Alternativ, aber etwas umständlicher, lässt sich eine maximale (minimale) Farbfolge direkt ermitteln, indem eine Menge von Kandidaten aufgebaut und sukzessive reduziert wird. Dazu sucht man in der Kette nach allen größten (bzw. kleinsten) Perlen; die Kandidatenmenge sind dann alle Farbfolgen, die in beiden Richtungen von diesen Perlen ausgehen, betrachtet wird aber immer nur ein Anfangsteil. Nun werden die Anfangsteile nach und nach um je eine Perle verlängert (dabei muss in beide Richtungen gegangen werden); die Kandidatenmenge wird dann auf die Folgen reduziert, deren Anfangsteile gleichermaßen maximal sind, bis nur noch ein Kandidat übrig ist – oder die Anfangsteile so lang sind wie die Kette selbst: dann besteht die Kette aus sich wiederholenden Teilstücken; Beispiel: 'rot-gelb-blau-rot-gelb-blau'.

Schnellere Berechnung

Wie uns die Aufgabenstellung ganz dezent mitteilt, kann unser N ziemlich groß werden: Die Ketten sollen mehrere Male um den Hals und bis zum Boden reichen. Damit könnte der einfache quadratische Ansatz zu langsam sein.

Schneller geht es auf den ersten Blick mit der Zahlendarstellung: Hier benötigt man zur Ermittlung der maximalen Kette ein Sortierverfahren, wie zum Beispiel der in einigen Programmiersprachen schon „eingebaute“ Quicksort. Damit erhält man eine Lösung in $O(N \log N)$ sozusagen geschenkt. Bei langen Ketten und großen Zahlen gilt das allerdings nicht, dort kommt man bei der Bestimmung von Maxima doch wieder auf quadratische Aufwände.

Aber auch wenn wir bei der symbolischen Darstellung der Farben bleiben wollen, gibt es eine sehr ästhetische Lösung des Problems in $O(N \log N)$. Hierfür beobachten wir zuerst, dass wir nur eine Richtung betrachten müssen: Haben wir eine Methode zum Finden der maximalen Folge in einer Richtung, so rufen wir sie für die Kette und die umgedrehte Kette auf und geben das größere der beiden Ergebisse zurück. Zum Finden der maximalen Folge der Kette in einer Richtung sortieren wir alle diese Folgen. Dazu bedienen wir uns einiger Ideen von Radix-Sort: Wir sortieren zuerst alle Teilfolgen der Kette der Länge 1. Dies ist einfach in $O(N \log N)$

ausführbar. Alle diese Ketten, die gleich sind, packen wir in einen gemeinsamen *bucket* (dt. Eimer) und diese buckets halten wir sortiert. Nun sortieren wir die Teilfolgen der Kette der Länge 2^k beginnend mit $k = 1$. Da wir bereits alle Teilfogen der Länge 2^{k-1} sortiert haben, müssen wir nur die aufeinanderfolgenden Paare von solchen Folgen $(F_i, F_{i+2^{k-1}})$ betrachten und diese sortieren, was klar in $O(N \log N)$, durch unsere bucket-Repräsentation aber auch in linearer Zeit möglich ist. Dies tun wir, bis $2^k \geq N$, bis wir also alle Teilfolgen der Kette, die länger als die Kette sind, sortiert haben. Nun müssen wir nur das Maximum (beschnitten auf N Zeichen) zurückgeben. Es folgt Pseudocode für diesen Algorithmus:

```
sortiereFarbfolgen(Kette K)
  sortiere Farbfolgen der Länge 1
  packe sie in buckets (für jedes vorkommende Zeichen ein Bucket)
  für k=0 bis floor(log N)
    für jede 2^k lange Folge F_i (aufsteigend sortiert)
      setze F_{i-2^k} an den Anfang seines Buckets
    // nun sind alle 2^{k+1}-Folgen innnerhalb der Buckets sortiert
    // wir müssen aber unter Umständen noch buckets "aufspalten":
    für jeden bucket:
      für jedes Paar aufeinanderfolgender Folgen F_i, F_j im bucket:
        wenn letzterBucket( F_{i+2^k} ) > letzterBucket( F_{j+2^k} )
          spalte den bucket zwischen F_i, F_j in zwei buckets
  gib letzte Folge aus größtem bucket zurück
```

Und es geht noch besser: Man kann sich davon überzeugen, dass es für eine als Farbfolge F gegebene Kette genügt, das lexikographisch größte Suffix (Endstück) der Farbfolge FF (F gefolgt von F) zu bestimmen. Dieses lässt sich in Linearzeit erledigen.

Bewertungskriterien

Natürlich muss die vorgeschlagene Normalform eindeutig sein. Dazu gehört, dass wirklich alle Kettenvarianten berücksichtigt worden sind: Beginn ab jeder Perle (bei der Suche nach einer „größten“ oder „kleinsten“ Kette genügt es, bei jeder größten bzw. kleinsten Perle zu beginnen), Verlauf in beiden Richtungen. Neben einer ausreichenden Beschreibung der Normalform sollte auch eine gute Begründung der Eindeutigkeit der Normalform vorhanden sein. Weiterhin sollte die Normalform korrekt und möglichst einfach berechnet werden, mehr als quadratische Laufzeit sollte nicht nötig sein.

Wichtig ist auch für diese Aufgabe die Qualität der Beispiele. Es sollten genügend aussagekräftige Beispiele vorhanden sein, also etwa symmetrische Ketten und Ketten, die von zwei Kugeln in dieselbe Richtung gehend gleich aussehen. Wenn die Normalform schon anhand des Vergleichs der ersten Kugel jeder Folge ablesbar ist, so kann wohl kaum von einem guten Beispiel die Rede sein. Insgesamt sollten mindestens drei Beispiele vorhanden sein.

Aufgabe 3: Winddiagramme

Lösungsidee

Es waren Fragestellungen zu formulieren, die mit Hilfe der Winddaten aus der vorgegebenen Wettertabelle zu beantworten waren. Dabei hat die Aufgabenstellung recht klar gesagt, was unter „Winddaten“ zu verstehen ist, nämlich im Wesentlichen die Angaben zu Windrichtung und -geschwindigkeit, die in Wind(richtungs)diagrammen veranschaulicht werden sollten.

Es ist aber nicht ausgeschlossen, sondern sogar interessant, die Winddaten mit anderen Inhalten der Tabelle in Beziehung zu setzen, z.B. Zeitangaben oder auch Temperaturbereichen. Fragen, die am besten mit einer in der Aufgabenstellung geforderten „Windrosen-Grafik“ beantwortet werden können, beziehen sich naturgemäß besonders auf die Windrichtung. Die folgenden beiden Fragen liegen dazu besonders nahe:

- Wie ist die durchschnittliche Windstärke in den verschiedenen Windrichtungen?
- Wie häufig weht der Wind aus den verschiedenen Windrichtungen?

Eher Kandidaten für die Beantwortung mit Hilfe von Balkendiagrammen bzw. Kurven sind:

- Wie ist die durchschnittliche Windstärke in den verschiedenen Monaten? (Bzw.: In welchem Monat weht der Wind durchschnittlich am stärksten?)
- Wie ist der Zusammenhang zwischen Luftdruck und Windgeschwindigkeit?

In den Windrichtungsdiagrammen sollen die Windrichtungen als Strahl einer Windrose dargestellt werden. Sinnvoll ist die Zusammenfassung mehrerer Richtungen zu einer Gruppe (z.B. in 5-Grad-Schritten), für die dann ein Strahl angezeigt wird. Es liegt nahe, die bei einer Windrichtung auftretenden Windgeschwindigkeiten als Länge des Strahls darzustellen. Alternativ kann die Häufigkeit des Auftretens einer Windrichtung (bzw. einer Richtungsgruppe), evtl. auf Zeitpunkte oder Zeiträume eingeschränkt, als Länge eines Strahls umgesetzt werden. Statt der Länge könnte auch die Stärke des Strahls variiert werden, aber das kann besonders im Zentrum des Diagramms zu informationsfreien, komplett eingefärbten Bereichen führen.

In einem guten Winddiagramm ist die Richtung eines Strahls und die Bedeutung seiner Länge gut abzulesen. Es sind also die Richtungen zu markieren, mindestens die vier Himmelsrichtungen, besser ist eine detailliertere Einteilung. Wie in allen Diagrammen helfen Raster (die aber nicht zu dicht sein sollten), die Größen einzelner Einträge zu vergleichen. Bei einem Winddiagramm besteht ein solches Raster aus konzentrischen Kreisen. Auch solche Raster sollen beschriftet sein, zumindest mit den Werten, idealerweise auch mit der Größe, die hier abgetragen wird. Die Beschriftung darf nur dann fehlen, wenn entsprechende klare Erläuterungen im Text vorhanden sind.

Es gibt durchaus interessante Alternativen zu der Verwendung von Strahlen. So kann man auch den Wertebereich der Größe, die durch die Entfernung vom Mittelpunkt ausgedrückt wird, in Bereiche einteilen. Zusammen mit den Richtungsgruppen entstehen so Sektoren, die markiert oder eingefärbt werden können. Solche Diagramme sind recht anschaulich, besser jedenfalls als Diagramme mit einzelnen Strahlen für jeden ganzzahligen Richtungswert. Auch wenn sie die Vorgaben der Aufgabenstellungen nicht ganz erfüllen, sind sie eine akzeptable Alternative.

Ebenfalls geeignet kann es sein, nur die Endpunkte der Strahlen zu visualisieren und diese zu einer Linie zu verbinden. So wird die Farbhäufung im Diagrammzentrum vermieden.

Es ist nicht nur in Ordnung, sondern sogar sinnvoll, einfache Diagramme mit Hilfe von Standardsoftware zu erstellen. Windrichtungsdiagramme sind in den üblichen Tabellenkalkulations-Programmen aber nicht vorgesehen. Sollte jemand ein Werkzeug gefunden haben, das dies doch erlaubt, ist dies – quasi als Finderlohn – positiv zu bewerten. Ansonsten müssen die Winddiagramme mit Hilfe eines eigenen Programms erstellt werden. Es genügt also nicht, irgendwie ermittelte Werte von Hand in eine Zeichnung umzusetzen.

Bewertungskriterien

- Es müssen mindestens zwei Fragestellungen formuliert sein.
- Die Fragestellungen müssen sich zumindest auf eine der Größen Windrichtung und Windgeschwindigkeit beziehen. Ein Missverständnis der Aufgabenstellung liegt vor, wenn dem Benutzer ein Diagramm vorgelegt wird und er eine Frage zu dem Diagramm durch einen Blick in die Tabelle beantworten soll.
- Damit für eine Frage ein Windrichtungsdiagramm vernünftig verwendet werden kann, muss sie sich auf die Windrichtung beziehen und eine andere Größe, deren Intensität pro Richtung gefragt ist; das ist natürlicherweise die Windstärke, kann aber auch eine Häufigkeitsangabe sein. Windrichtungsdiagramme, in denen die Strahlen alle gleich lang sind, haben ihren Zweck verfehlt.
- Alle verwendeten Diagramme müssen übersichtlich und gut ablesbar sein. Dazu gehören geeignete Beschriftungen, die teilweise durch Erläuterungen im Text ersetzt werden können. In den Windrichtungsdiagrammen sollten zumindest die vier Haupt- (N, O, S, W) und die vier Neben-Himmelsrichtungen (NO, SO, SW, NW) eingetragen sein. Sollte ein ansonsten überzeugendes Diagramm nur mit den vier Hauptrichtungen aufwarten, kann dies mit $\sim$ bemängelt werden und muss nicht zu Punktabzug führen.
- Die Diagramme sollen nicht von Hand erstellt worden sein (weder per Zeichensoftware noch per klassischer Zeichen-Hardware).
- Selbstverständlich dürfen die Diagramme nicht fehlerhaft sein. Falsche Werte können durch Programmfehler oder Übertragungsfehler entstehen. Aus der Tabelle lässt sich ermitteln, dass der Wind häufiger und durchschnittlich stärker aus westlichen Richtungen weht. Selbstverständlich müssen Diagramme (mindestens eines zu jeder Frage) auch abgedruckt vorhanden sein; sie sind wesentlicher Bestandteil der Lösung.

Aufgabe 4: Pass-Algorithmen

Lösungsidee

Die Bearbeitung der Aufgabe wird wesentlich davon beeinflusst, wie das Konzept der Pass-Algorithmen verstanden wird. Bei Authentifizierungsverfahren gibt es zwei Klassen von Beteiligten: Diejenigen, die das Prinzip des Verfahrens kennen und evtl. auch nutzen, und diejenigen, die es nicht kennen. Das Prinzip der PIN-Authentifizierung am Geldautomaten kennen sehr viele Leute, deshalb genügt einem Angreifer (diesen Begriff wollen wir hier für jemand verwenden, der sich in betrügerischer Weise als jemand anders authentifizieren möchte), Kenntnis der PIN zu erlangen. Das Prinzip der RSA-Verschlüsselung ist auch weithin bekannt, so dass es vielen Angreifern genügen würde, an die beteiligten Schlüssel zu gelangen (weshalb ja auch ab und zu die Schlüssellänge erhöht wird, um steigenden Rechenkapazitäten einige Schritte voraus zu bleiben).

Es kann aber auch Authentifizierungsmethoden geben, die als solche nur einem kleineren Kreis bekannt sind. Dann genügt es einem eingeweihten Angreifer immer noch, Passwörter, Kennnummern oder Schlüssel zu ermitteln. Ein Angreifer, der nicht aus diesem Kreis stammt, müsste erst etwas über die Methode selbst herausfinden.

Bei der Aufgabenstellung war vorausgesetzt worden, dass das beschriebene Prinzip (Addition der Zahlen an zwei Positionen einer Matrix und einer weiteren Zahl) allgemein bekannt ist, so dass ein Angreifer versuchen müsste, die drei Parameter eines individuellen Algorithmus herauszufinden. Wir wollen dies als Standpunkt A bezeichnen. Einige Formulierungen der Aufgabenstellung legen aber nahe, dass auch das Prinzip (der Algorithmus bzw. die funktionale Beziehung f) noch zu ermitteln ist; das ist Standpunkt B. Deshalb soll die Bewertung nicht davon beeinflusst werden, welchen dieser beiden Standpunkte die Teilnehmer eingenommen haben.

Teilaufgabe 1

Überlege, warum die genannten Eigenschaften wünschenswert sind. Welche weiteren Eigenschaften sollte ein solches System haben?

a) *Die Anzahl möglicher Pass-Algorithmen ist groß.*
Dieser Punkt tritt gerade bei Systemen mit einem großen Benutzerkreis in den Vordergrund. Gibt es wenige Algorithmen in der Klasse, so kann der korrekte für einen fremden Zugang eventuell durch vollständiges Ausprobieren gefunden werden.

b) *Ein Pass-Algorithmus kann mit wenig Mühe in kurzer Zeit im Kopf ausgerechnet werden.*
Die Benutzer des Systems sind nicht notwendigerweise Mathe-Profis oder Zahlen-Jongleure, die Sinus-Funktionen mit 15 Nachkommastellen im Kopf berechnen. Diese Forderung ergibt die Handhabbarkeit des Algorithmus'. Auch Ungeübte sollen den Algorithmus im Kopf ausführen und dann das Ergebnis korrekt eintippen können. Ein komplexer Algorithmus würde auch zu mehr Fehleingaben führen, wodurch die Verlässlichkeit des Logins nicht mehr gegeben wäre.

c) *Die Beobachtung weniger Paare $(x, f(x))$ erlaubt kaum Rückschlüsse auf den benutzten Pass-Algorithmus f.*
Die klassische PIN, die bei der EC-Karte und bei vielen anderen Systemen eingesetzt wird hat einen großen Nachteil: Wenn jemand anderes die Eingabe einmal beobachtet, hat er sämtliche Informationen, um selbst die Passworteingabe zu überwinden. Der Vorteil eines Pass-Algorithmus soll ja gerade hier liegen: Trotz Kenntnis der Eingabe hat der Beobachter keine (oder nur eine sehr geringe) Möglichkeit, sich ins System einzuloggen. Auch wenn das Prinzip f schon bekannt ist, sind die Hürden für den Angreifer hoch: Um die individuellen Parameter eines Benutzers ermitteln zu können, muss nicht nur die Eingabe des Benutzers, sondern müssen auch alle Werte der dargestellten Matrix beobachtet werden; und gerade durch die zusätzliche Zahl können sehr viele Beobachtungen zur Ermittlung der individuellen Werte nötig sein.

Weitere Eigenschaften:

d) *Der Algorithmus muss eindeutig sein (eine Funktion sein).*
Diese Eigenschaft wird implizit schon in der Aufgabenstellung gefordert. Sind auf eine Vorgabe mehrere Antworten möglich, erhöht dies zum einen die Chance, den Algorithmus zu knacken und erschwert auf der anderen Seite die Kontrollmöglichkeit durch die Software.

e) *Der Algorithmus sollte „beweglich“ sein, also möglichst wenig davon abhängig, wo sich der Benutzer befindet.*
Es sind Algorithmen denkbar, die benutzerspezifische Wertetabellen verwenden, die jeder Benutzer mit sich führen muss (wie etwa im TAN-System des Online-Banking). Wenn diese umfangreich sind, ist das eine Belastung für den Benutzer, außerdem wird die Verwendung der Wertetabelle allzu offensichtlich. Beispiel: Ein Tripel (x,y,z) verweist auf das x. Wort in der y. Zeile auf Seite z in einem Codebuch. Nun heißt die Anfrage (132,13,2) * (13,12,1), dass man die Länge des 2. Wortes in der 13. Zeile auf Seite 132 mit der des 1. Wortes der 12. Zeile auf Seite 13 multiplizieren und das Ergebnis eingeben soll. Nachteil dieser Prozedur: Der Anwender muss sein persönliches Codebuch jederzeit mitführen, wenn er sich in das System einloggen will.

f) *Der Algorithmus sollte nicht zu einfach sein.*
Eine falsche Authentifizierung darf einem Dritten nicht schon dann möglich sein, wenn er z. B. lediglich eine Codetabelle kennt. Diese könnte man ja verlieren und dann einem Angreifer Tür und Tor öffnen. Bei einem Pass-Algorithmus in der Art des Beispiels aus der Aufgabenstellung wäre es schlecht, wenn nur eine Zahl aus der Matrix eingegeben werden müsste, denn dann würde einem Angreifer eine Beobachtung genügen.

g) *Die Anzahl, wie oft ein Algorithmus benutzt werden kann, sollte nicht limitiert sein.*
Gibt es nur eine Tabelle, in der jedes Element nur einmal verwendet werden darf, kommt man schnell zum Ende der Nutzungsdauer und muss ein neues Codewerk anfordern.

h) *Die Anzahl der möglichen Benutzerantworten sollte ausreichend groß sein.*
Bei einer kleinen Antwortmenge kann die Antwort zu leicht erraten bzw. durch Probieren ermittelt werden.

Teilaufgabe 2

Wie bewertest Du das Beispielsystem?

Hierzu werden die genannten Punkte herangezogen.

a) Es gibt $25 \times 25 \times 99 = 61.875$ verschiedene Pass-Algorithmen. Das ist ausreichend und deutlich mehr als z.B. die Anzahl vierstelliger PINs – aber deutlich weniger z.B. als Passwörter mit einer Länge von 8 Buchstaben.

b) Die Addition dreier zweistelliger Zahlen ist beliebigen Benutzerkreisen vermutlich nicht zuzumuten, bei einer gewissen Eingrenzung des Benutzerkreises aber durchaus anwendbar.

c) Bei diesem Kriterium spielt der Standpunkt eine große Rolle. Zunächst einmal Standpunkt B (das Prinzip ist unbekannt): Die Frage ist hier, wie leicht sich die funktionale Beziehung zwischen Tabelle und Benutzerantwort ermitteln lässt. Auch ein Angreifer kann davon ausgehen, dass die Berechnungsfunktion nicht allzu komplex ist. Würde die Antwort sich allein aus der angezeigten Tabelle berechnen, wäre es recht leicht, eine Beziehung herzustellen. Die hinzuzuaddierende Zahl ist hingegen nicht zu beobachten, das macht das System schon sicherer. Allerdings liegt die Verwendung von Rechenoperationen im Zusammenhang mit Zahlen recht nahe, und die Addition ist davon auch noch die einfachste. Sicherer wäre die Verwendung unterschiedlicher Rechenoperationen bei der Berechnung der Antwort oder – noch besser – ganz anderer Arten von Operationen.

   Standpunkt A (das Prinzip ist bekannt): Wie oben schon berechnet, gibt es 61.875 mögliche individuelle Parameterkombinationen. Die Beobachtung eines Paares $(x, f(x))$ lässt noch alle Matrixpositionen i und j zu (die Positionen sind eigentlich durch Paare gegeben, aber das spielt keine Rolle), so dass $x_i + x_j < f(x)$. Für jede dieser Paare (i, j) ist die Zahl $n = f(x) - x_i - x_j$ festgelegt. Es gibt also höchstens noch 625 mögliche Parameterkombinationen. Mit einer nächsten Beobachtung lässt sich die Menge der noch möglichen Tripel (i, j, n) weiter einschränken; es sind aber sicher mehrere Beobachtungen nötig.

d) Ist hier gegeben.

e) Der Algorithmus ist beweglich, da man sich nur die beiden Tabellenpositionen und die zusätzliche Zahl merken muss. Mit ein wenig Übung wird man die Tabellenpositionen visuell erinnern und muss sich dann deren Koordinaten nicht mehr merken.

f) Die Durchführung zweier Additionen ist nicht gerade eine komplexe Berechnungsvorschrift, aber auch nicht trivial (wie etwa: gib immer die Zahl oben links in der angezeigten Matrix ein).

g) Die Häufigkeit der Verwendung der Pass-Algorithmen ist nicht eingeschränkt.

h) Es gibt weniger als 300 verschiedene mögliche Benutzerantworten, nämlich die Zahlen von 21 bis 297.

Ergebnis: Die vorgestellte Klasse von Pass-Algorithmen ist nicht schlecht, aber nicht gerade für die höchste Sicherheitsstufe geeignet.

Teilaufgabe 3

Entwirf selbst zwei solche Systeme für zwei unterschiedliche Szenarien, implementieren eines davon und beurteile kritisch die Vor- und Nachteile beider. Erläutere auch, in welchen Situationen deine Systeme gut eingesetzt werden können.

Es gibt eigentlich keine bekannte Authentifizierungsmethode, die auf der Idee der Pass-Algorithmen aufbaut. Ansatzweise ist die iTAN vergleichbar: Hier präsentiert das System eine Zahl, und der Benutzer muss mit dem Code antworten, der an der durch die Zahl gegebenen Position einer Tabelle steht. Dabei werden allerdings die Tabelle und damit auch die Positionswerte nach und nach verbraucht, womit keine gleichbleibende funktionale Abhängigkeit zwischen Systemvorgabe und Benutzerantwort mehr gegeben ist. Nachteile der iTAN-Methode: Punkt e): Man muss den TAN-Bogen immer mitnehmen. f) Hat jemand den Bogen, ist der Rest selbsterklärend. g) Man muss sich regelmäßig eine neue TAN-Liste geben lassen.

Auch nur ansatzweise vergleichbar ist die Methode Smart-TAN plus. Hier wird eine TAN aus der Zielkontonummer, einer Codenummer und den eigenen Kontodaten (die durch einen Kartenleser aus der Bankkarte des Benutzers ausgelesen werden) von einer Software generiert, der Benutzer muss diese TAN nur eingeben. Nachteile: b) Die Rechenvorschrift ist zu komplex, um sie im Kopf zu rechnen. e) Man muss einen entsprechenden kleinen Rechner bzw. ein Kartenlesegerät zur Verfügung haben.

Erstes eigenes Beispiel

Anonymes Remote-Login, wechselseitige Authentifizierung nicht erforderlich

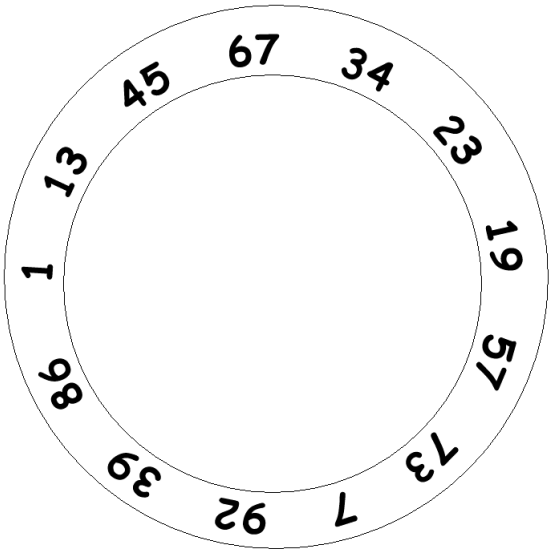


Abbildung 3: Codescheibe

Die Codezahlen stehen auf einer Scheibe (Abbildung 3). Als Anforderung kommt beispielsweise $(39, 10, 3)$, was bedeutet: Nimm die 39, gehe 10 Zahlen im Uhrzeigersinn weiter und addiere die entsprechende Zahl $(39 + 73)$ und gehe nochmals 3 Ziffern weiter und addiere die entsprechende Zahl $(39 + 73 + 39 = 151)$. Verbesserung gegenüber dem Matrix-Algorithmus

vom Aufgabenblatt ist: a) Die Ziffern sind auf verschiedene Weise erreichbar, der Algorithmus ist also nicht injektiv. Beispiel: $(73, 3, 13)$ erreicht dieselben Komponenten. Dadurch wird es noch schwieriger, den Code zu knacken. Auch die Bedeutung der Zahlen ist dem Uneingeweihten nicht direkt offensichtlich (Standpunkt B). Kennt jedoch jemand das Prinzip (Standpunkt A) und erlangt eine Code-Scheibe, kommt er sehr einfach ins System.

Dieses System kann als einfaches Zugriffskontrollsystem verwendet werden. Die Sicherheit, die hier geboten wird, liegt zwar über einem normalen Passwort, ist allerdings maximal einseitig – das System authentifiziert den Benutzer. Nachteil im Vergleich zum Beispiel der Aufgabenstellung: es gibt genau genommen nur eine geheime Komponente, nämlich die Code-Scheibe.

Zweites eigenes Beispiel

Zugangskontrollsystem mit wechselseitiger Authentifizierung

Zum Teil kann es auch gewünscht sein, dass nicht nur das System den Benutzer identifiziert, sondern auch der Benutzer das System. Beispiel ist hier wiederum das Online-Banking. Als Benutzer möchte man natürlich sicherstellen, dass der Betrag, den man überweisen möchte, auch wirklich beim Empfänger ankommt. Kommt man versehentlich auf eine Hackerseite, die der eigentlichen Bankseite ähnlich sieht, kann die eingegebene Information manipuliert werden und in diesem Zustand an die Bank übertragen werden (Pharming). Dieses Risiko kann durch eine wechselseitige Authentifizierung verringert werden.

Bei diesem Design erfolgt die Authentifizierung des Systems gegenüber dem Benutzer erst nach erfolgreicher Eingabe des korrekten Schlüssels, da andersherum ein Angreifer sukzessive alle möglichen Authentifizierungen ausspähen könnte. Nachteil dieses Systems ist, dass Benutzer die schadhafte gegenüberliegende Seite erst erkennen, wenn sie sich bereits eingeloggt haben.

Es wird eine Matrix als benutzerspezifische Code-Tabelle verwendet. Um das System im Vergleich zum Beispiel auf dem Aufgabenblatt (wenn dort die Matrix auch für den Benutzer gleichbleibend wäre) sicherer und praktikabler zu machen, gibt es folgende Möglichkeiten:

- Anzahl der Ziffern je Zahl erhöhen. Drei- oder vierstellige Zahlen bedeuten deutlich mehr mögliche Matrizen. Sie bedeuten aber auch einen höheren Rechenaufwand für den Benutzer; Rechungen mit Zahlen über 100 können viele Menschen bereits nicht mehr im Kopf rechnen.
- Anzahl der Operatoren in der Rechnung erhöhen. Auch hierdurch wird die Anzahl der möglichen Matrizen erhöht. Nimmt man z.B. vier Operatoren, so ergeben sich im Beispiel vom Blatt bereits 100*100*100*100=100.000.000 mögliche Matrizen, die ausgegeben werden können. Gleichzeitig wird natürlich auch bei jedem Zugriff ein größerer Teil der Matrix benutzt, so dass die Chance wächst, irgendwann die gesamte Matrix berechnen zu können. Auf der anderen Seite wächst auch die Zahl der möglichen Kombinationen, so dass es unwahrscheinlicher wird, dass ein Beobachter eine abgefragte Zahlenfolge bereits kennt.

- Die Größe der Matrix verändern. Die Anzahl der möglichen Matrizen wird dadurch nicht erhöht. Jedoch wird die Auswahlmöglichkeit größer, so dass die Chance für einen Angreifer sinkt, eine Kombination bereits erspäht zu haben.
- Operationen variieren. Hierdurch kommt es zu mehr verschiedenen Abfragen, so dass die Chance für den Angreifer, eine Kombination bereits zu kennen, sinkt.

Gewählt wird eine 6x6 Matrix mit zweistelligen Zahlen und vier Operatoren sowie den Operationen + und -. Im ersten Teil bekommt der Benutzer eine Rechenaufgabe mit vier Werten aus der Matrix gestellt. Nach erfolgreicher Eingabe des Ergebnisses stellt der Computer sich selbst eine Aufgabe aus der zweiten Matrix und übermittelt diese samt Ergebnis an den Benutzer, so dass dieser die Authentizität kontrollieren kann. Bei diesem Verfahren wird eine deutlich höhere Sicherheit als im ersten Fall erreicht. Insbesondere werden auch Pharming-Attacken abgewehrt.

Bewertungskriterien

Diese Aufgabe wird wahrscheinlich mit viel Text und wenig Programm gelöst werden, es ist nur die Implementation eines der vorgeschlagenen eigenen Systeme nötig. Generell wird von allen schriftlichen Ausführungen erwartet, dass sie sachlich richtig sind.

- Teil 1: Zunächst ist es wichtig, dass die Bedeutung der drei auf dem Aufgabenblatt genannten Eigenschaften gut begründet wird.
- Teil 1: Zusätzlich sollten mindestens zwei weitere sinnvolle Eigenschaften selbst gefunden und begründet werden.
- Teil 2: Bei der Bewertung des Beispielsystems sollte auf die in der Aufgabenstellung und vom Teilnehmer selbst genannten Kriterien – soweit möglich und sinnvoll – eingegangen werden.
- Teil 3: Die eigenen Ansätze sollten sich vom vorgestellten Ansatz nicht nur in der Spaltenzahl oder der Anzahl der Operatoren unterscheiden und sollten auch voneinander deutlich verschieden sein.
- Teil 3: Es muss angegeben werden, in welchen Zusammenhängen die Systeme gut einsetzbar sind. Die Systeme sollten zu den vorgeschlagenen Szenarien und Situationen natürlich auch passen.
- Teil 3: Auch die selbst entwickelten Systeme sollen selbstkritisch betrachtet werden, wobei wieder die Kriterien aus Teil 1 nach Möglichkeit anzuwenden sind. Vorteile und Nachteile müssen beschrieben sein – es gibt kein System ohne zumindest kleine Nachteile.
- Die Funktionalität und Funktionsfähigkeit der eigenen Implementation sollte wie üblich mit ausreichend vielen Beispielen belegt werden.

Aufgabe 5: Kosmischer Tanz

Lösungsidee

Gegeben sind die Position von K, dessen Bewegung b und Position p im Raum, die Massen M und m (wobei m für die Simulation irrelevant ist), die Gravitationskonstante G. S liegt der Einfachheit halber immer auf $S(0|0|0)$. $p(t)$ sei eine Funktion, welche die Position von K abhängig vom Zeitpunkt t (in Sekunden) angibt. $\Delta p(t)$ ist entsprechend die Geschwindigkeit von K. Es gilt $\Delta p(0) = b$. Würde die Gravitation von S nicht wirken, würde sich der kleinere Himmelskörper K geradlinig weiterbewegen. K wird jedoch in Richtung S (= der Vektor $-p(t)$) beschleunigt. Die Zusammenhänge von $\Delta p(t)$, $p(t)$ und der Gravitationskraft lassen sich durch folgenden Gleichungen darstellen:

$$\begin{aligned} \text{Beschleunigung} &= \text{Kraft}/\text{Masse} \\ \text{Kraft} &= GMm/r^2 \\ \text{Masse} &= m \\ \Rightarrow \text{Beschleunigung} &= GM/r^2 \end{aligned}$$

$$\begin{aligned} \Delta p(0) &= b \\ \Delta p(t) &= \Delta p(t-1) - (GM/r^2) * n, n \text{ ist der Einheitsvektor von } p \\ p(t) &= p(t-1) + \Delta p(t) \end{aligned}$$

$p(t)$ und b sind Vektoren im Raum. $p(0)$ entspricht der Startposition von k. Gerechnet wird in den Standardeinheiten m, s, N und kg.

Das Simulationssystem soll schrittweise die Bewegung von k simulieren und in drei Koordinatensystemen - bestehend aus den Raumachsen xy, yz und xz ? darstellen. Das Zeitintervall Δt sollte so gewählt werden, dass sich K pro Zeiteinheit ungefähr einen Pixel weiterbewegt.

Programm-Dokumentation

Die geschilderten Ideen wurden als Perl-Skript implementiert. Das Programm erhält zunächst die Anfangsposition und -geschwindigkeit von K sowie die Masse von S und die Gravitationskonstante als Kommandozeilenargumente. Die Skalierung der Koordinatensysteme wird anhand der Anfangsposition von K berechnet, sodass K höchstens 125 Pixel von jeder Achse entfernt ist. Die Bewegung des Himmelskörpers wird schrittweise in einer while-Schleife (Abbruchbedingung: Zeit überschreitet den Wert von 100000 Zeitintervallen, entspricht 100000 Schritten) simuliert:

1. Geschwindigkeit von K ändern
2. beim ersten Durchlauf der while-Schleife das Zeitintervall so setzen, dass K auf der grafischen Darstellung genau einen Pixel zurückgelegt hat
3. Position von K ändern, es gilt: $p(t) = \Delta p(t-1)\Delta t$

4. ggf. Himmelskörper als schwarzen Punkt zeichnen, sofern er sich innerhalb der Ränder des jeweiligen Koordinatensystems befindet

Es genügt, wenn *K* in jedem zehnten Simulationsschritt gezeichnet wird, dies spart Speicherplatz und tut der Bildqualität keinen Abbruch. Je nach Geschwindigkeit und Zeitintervall ergibt sich eine durchgezogene bzw. gepunktete Linie, welche die Bahn von K beschreibt. Das Programmfenster wird nun angezeigt, der Benutzer kann das `Tk::Canvas-Objekt` als Postscript-Datei speichern (und auf Papier ausdrucken) und ggf. weitere 10000 Zeitintervalle simulieren.

Programm-Ablaufprotokolle

In den folgenden Programmabläufen entspricht *G* ungefähr der Gravitationskonstante unseres Universums. Der Himmelskörper *S* ist etwas leichter als die Erde.

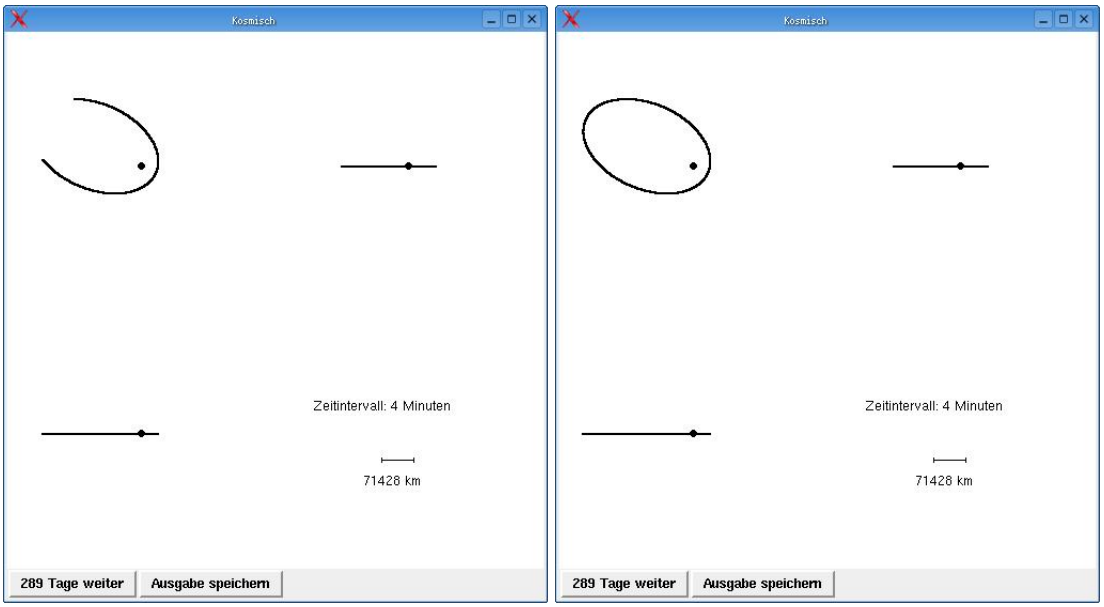
Das Perl-Script kosmisch.pl wird wie folgt aufgerufen:

`kosmisch.pl` [X-Position von K] [Y-Position] [Z-Position]
[X-Bewegung von K] [Y-Bewegung] [Z-Bewegung]
[Masse von S] [Gravitationskonstante]

Im Grafikfenster werden die beiden Himmelkörper und die Bahn von *K* jeweils auf drei Projektionsebenen dargestellt. Die X-Y-Ebene befindet sich links oben, darunter ist die Y-Z-Ebene und rechts oben wird die X-Z-Ebene gezeichnet. Um eine ungefähre Vorstellung von den Größenordnungen der Entfernungen und Geschwindigkeiten auf dem Bild zu geben, werden ein Maßstab und das Zeitintervall angezeigt.

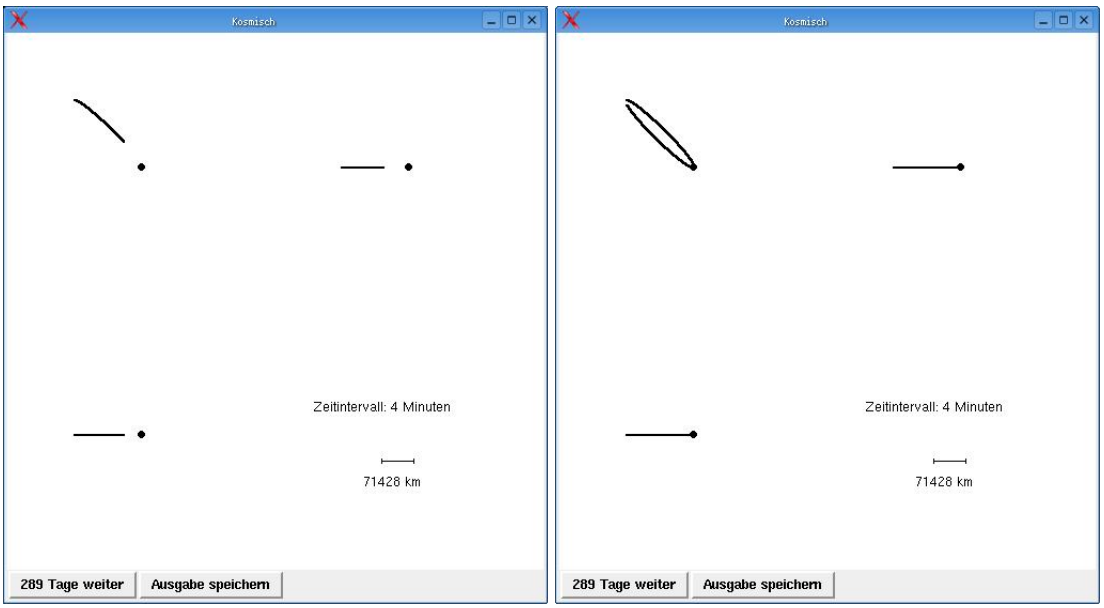
```
./kosmisch.pl -150000000 -150000000 0 6 0 0 5e22 6.67428e-11
```

K umrundet *S* in einer Elliptischen Umlaufbahn:



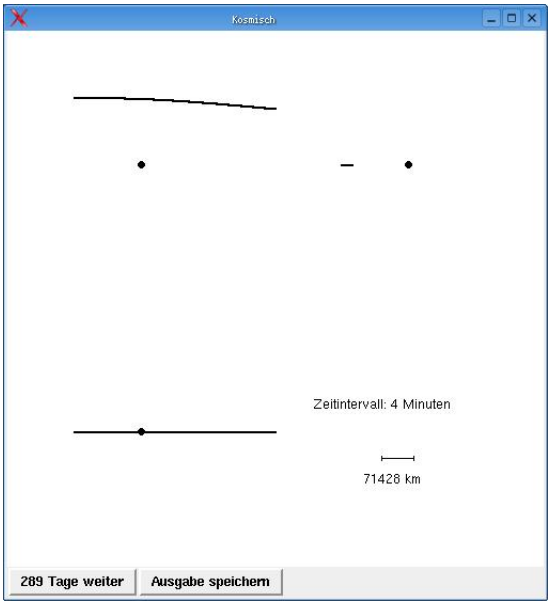

```
./kosmisch.pl -150000000 -150000000 0 1 0 0 5e22 6.67428e-11
```

K stürzt auf *S*; da es sich jedoch um punktförmige Himmelskörper handelt, entsteht eine sehr flache elliptische Bahn:



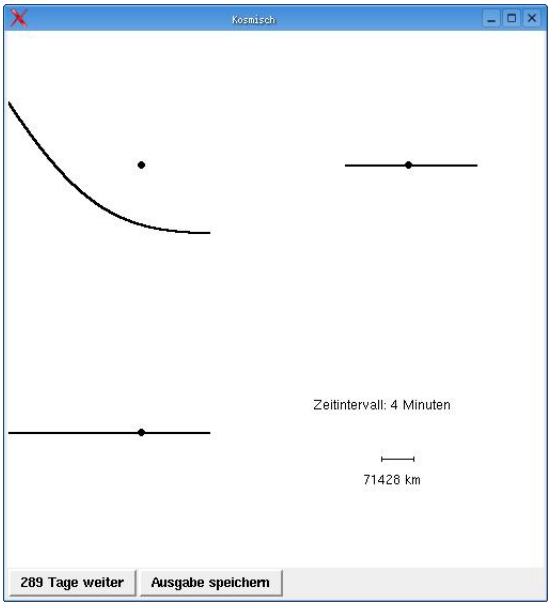
```
./kosmisch.pl -150000000 -150000000 0 40 0 0 5e22 6.67428e-11
```

K ist sehr schnell und wird daher kaum von der Gravitation beeinflusst:




```
./kosmisch.pl 150000000 150000000 0 -14 0 0 5e22 6.67428e-11
```

Eine geringere Startgeschwindigkeit führt zu einer stärkeren Ablenkung der Bahn:



Bewertungskriterien

- Die Formeln müssen korrekt umgesetzt sein.
- Die Genauigkeit der Simulation ist sinnvoll zu wählen (die Flugbahnen sollten z.B. nicht allzu eckig verlaufen, die Simulation sollte wiederum nicht allzu viel Zeit in Anspruch nehmen); die Wahl der Genauigkeit sollte begründet sein.
- Die Anfangsgeschwindigkeit und -position von *K* muss variierbar sein; die Veränderung der Masse von *K* ist zwar in der Aufgabenstellung gefordert, aber nicht sinnvoll, und kann deshalb entfallen. Die Variation der Masse von *S* ist zwar sinnvoll, aber nicht gefordert, und muss deshalb nicht implementiert sein.
- Die Nutzungsgrenzen des eigenen Systems sollten erkannt und dann auch beschrieben worden sein. Mögliche Grenzen sind:
  - Bei einer diskreten Lösung (wovon auszugehen ist) können Ungenauigkeiten auftreten.
  - Punktförmige Himmelskörper machen eine Kollision unmöglich, was teilweise zu unrealistischen Flugbahnen führt
  - Die Masse von S ist wie bei einem schwarzen Loch auf einen Punkt konzentriert.
- Die grafische Darstellung ist korrekt, übersichtlich und nachvollziehbar und auch als Ausdruck auf einem Blatt Papier brauchbar – dies ist in der Aufgabenstellung explizit gefordert.
- Die Ergebnisse von mindestens zwei interessanten und unterschiedlichen Simulationen müssen dokumentiert sein.

Junioraufgabe: Canonicus

Lösungsidee

Das Programm liest den Kanon taktweise und (dabei) die einzelnen Stimmeinsätze ein. Danach berechnet es die Gesamtlänge des Kanons etwa mit Hilfe der Formel:

$$\text{Gesamtlänge} = \text{Taktzahl des letzten Stimmeinsatzes} + \text{Länge der Melodie}$$

Für die erste Stimme wird nun die Melodie solange wiederholt ausgegeben, bis die Gesamtlänge des Kanons erreicht ist. Für jede weitere Stimme werden mit Pausen gefüllte Takte ausgegeben, bis der jeweilige Stimmeinsatz erfolgt, dann die Melodie wiederum so oft wiederholt, bis die Gesamtlänge erreicht ist.

Der Kanon muss nicht die oben berechnete Gesamtlänge haben. Ein Beispiel für eine andere Länge ist das Beispiel in der Aufgabenstellung. Jedoch ist es allgemein üblich, einen Kanon mindestens so lange zu singen, bis alle Stimmen den Kanon (mindestens) einmal durchgesungen haben. Mit dieser Bedingung gibt die Formel die Länge des so kürzest möglichen Kanons an.

Da in ABC eine einfache Möglichkeit fehlt, ganztaktige Pausen anzugeben, muss das Programm je nach Taktart (Header `M:`) und Standardnotenlänge (Header `L:`) verschiedene Pausen ausgeben: z. B. ist bei Standardnotenlänge $\frac{1}{8}$ und $\frac{4}{4}$-Takt die Taktpause `z8`, bei gleicher Standardnotenlänge $\frac{1}{8}$ und $\frac{2}{4}$ Takt dagegen `z4`. Die Formel für den nötigen Faktor bei der Pausenlänge beträgt:

$$\text{Pausen-Faktor} = \frac{\text{Taktart}}{\text{Standardnotenlänge}}$$

Also für das erste Beispiel $\frac{4/4}{1/8} = 8$, für das zweite $\frac{2/4}{1/8} = 4$.

Zu Teilaufgabe 1

Da das Musik-Eingabeformat mit ABC in der Aufgabenstellung schon so gut wie festgelegt ist, braucht man nur ABC für die Stimmeinsätze zu erweitern. Dies ist auf verschiedene Weisen möglich:

Markierungen innerhalb der Melodie Ideal hierfür eignen sich die Akkordbezeichnungen, welche automatisch über den Notentext gesetzt werden. Die Akkordbezeichnungen sind in Anführungszeichen. Die einzelnen Einsätze können so durch `"1."`, `"2."` usw. erfolgen. Sollte man den Eingabetext auf diese Weise modifizieren, erhält man ein ähnliches Druckbild wie in dem Beispiel in der Aufgabenstellung. Zusätzlich wäre die Notation ABC-freundlich, d. h. sie hindert andere ABC-Programme nicht daran, die Eingabe einzulesen.

Weiteres Header-Feld Da viele Buchstaben schon benutzt sind, eignen sich `U:` und `Y:`, welche in ABC (noch) nicht verwendet werden. Eine Möglichkeit, die Stimmeinsätze zu definieren, wäre nach der jeweiligen Header-Markierung mehrere Nummern zu schreiben,

welche nacheinander die jeweilige Taktzahl angeben, ab welcher die jeweilige Stimme ihren Einsatz hat.
(Standard: `http://www.walshaw.plus.com/abc/abc2mtex/abc.txt`).

über Kommentare Laut dem ABC-Standard wird ab einem Prozentzeichen der Rest einer Zeile ignoriert. Nach einem Prozentzeichen würden sich also auch Markierungen für Stimmeinsätze einfügen lassen. Hier muss man aber aufpassen, dass sonstige Kommentare im Stück nicht zu Problemen führen.

In einem Kanon müssen Stimmeinsätze nicht unbedingt regelmäßig sein. So kann es durchaus vorkommen, dass die 2. Stimme beginnt, wenn die 1. Stimme im 2. Takt ist, die 3. Stimme aber erst beginnt, wenn die 1. Stimme den 10. Takt erreicht. Dies kann man berücksichtigen, wird aber von den Einsendungen nicht erwartet.

Besondere Schwierigkeiten

In der Aufgabenstellung ist vorgegeben, dass es reicht, Stimmeinsätze am Taktanfang verarbeiten zu können. Bei vielen Kanons aber hört das Lied nicht für jede Stimme ganztaktig auf: Nach ihrem eigentlichen Endton, der mit den anderen Stimmen den Endakkord erzeugt, steht in den Noten für diesen Takt noch ein Auftakt, der aber dann meist nicht mehr harmonisch in den Endakkord passt. Würde man diesen Auftakt mit in die Ausgabe des letzten Taktes übernehmen, würde der Kanon dann für unsere Ohren abrupt aufhören. Der PC kann aber schlecht wissen, welcher Ton nun der wirkliche Endton ist. Daher reicht es eigentlich nicht aus, nur die Stimmeinsätze zu markieren, man müsste auch die Endtöne markieren. Diese könnte man, wie allgemein verbreitet, mit dem Fermaten-Zeichen tun, welches in ABC per `!fermata!` vor der jeweiligen Note bewerkstelligen lässt. Da dieses aber für die interne Logik der ABC-Programm die jeweilige Note auf das $1\frac{1}{2}$-fache vergrößert, muss man die Fermaten während des Stückes entfernen. Eine weitere Folge ist, dass dann der letzte Takt unvollständig sein kann, falls man ihn nicht mit Pausen füllt. Dies wird aber bei der Verarbeitung durch andere ABC-Programme meistens ignoriert oder führt zu harmlosen Warnmeldungen.

Beispiel

Als Beispiel für die Funktionsweise des Programmes wird hier der Kanon *Bona Nox* mehrstimmig „gespielt“. Es soll angemerkt sein, dass der ABC-Standard für die Notation mehrerer Stimmen in der Ausgabe mehrere Möglichkeiten vorsieht. Die in einer Einsendung dokumentierten Ausgaben müssen also nicht ganz genau so aussehen; entscheidend ist die korrekte Berechnung des Einsatzabstandes und die entsprechende Berechnung und Einfügung der Pausen.

Die Eingabedatei mit eingefügten Markierungen für die Stimmeinsätze und (freiwilligen) Fermaten für die Endtakte (die Stimmangabe `V:1` ist nicht notwendig):

```
X:1     %1 Stueck
V:1     %1 Stimme
M:4/4   %Takt
```

```
L:1/8   %Standardlaenge der Noten
K:A     %Tonart
"1." A4 c4 |F4 z4 |B2 c2 d2 B2 |!fermata!c4 z2 e d |
"2." c2 A2 z2 A c |d2 f2 z2 d2 |E4 z2 e2 |!fermata!e2 e d c2 c B |
"3." A2 c2 A2 F A |B2 B c d2 B A |G2 A2 B2 G2 |!fermata!A2 c B A2 z2 |
"4." A4 F4 |D4 B,4 |E2 C2 B,2 E2 |!fermata!A,4 z4 |]
%Ende
```

Der von dem Programm 4-stimmig ausgeschriebene Kanon dazu lautet (die Stimmangabe `V:1` könnte auch unmittelbar vor der Stimme stehen, also unter der Angabe der Tonart):

```
X:1     %1 Stueck
V:1     %1 Stimme
M:4/4   %Takt
L:1/8   %Standardlaenge der Noten
K:A     %Tonart
A4 c4|F4 z4|B2 c2 d2 B2|c4 z2 e d|c2 A2 z2 A c|d2 f2 z2 d2|E4 z2 e2
|e2 e d c2 c B|A2 c2 A2 F A|B2 B c d2 B A|G2 A2 B2 G2|A2 c B A2 z2
|A4 F4|D4 B,4|E2 C2 B,2 E2|A,4 z4|A4 c4|F4 z4|B2 c2 d2 B2|c4 z2 e d
|c2 A2 z2 A c|d2 f2 z2 d2|E4 z2 e2|e2 e d c2 c B|A2 c2 A2 F A
|B2 B c d2 B A|G2 A2 B2 G2|!fermata!A2|]
V:2
z8|z8|z8|z8|A4 c4|F4 z4|B2 c2 d2 B2|c4 z2 e d|c2 A2 z2 A c|d2 f2 z2 d2
|E4 z2 e2|e2 e d c2 c B|A2 c2 A2 F A|B2 B c d2 B A|G2 A2 B2 G2
|A2 c B A2 z2|A4 F4|D4 B,4|E2 C2 B,2 E2|A,4 z4|A4 c4|F4 z4|B2 c2 d2 B2
|c4 z2 e d|c2 A2 z2 A c|d2 f2 z2 d2|E4 z2 e2|!fermata!e2|]
V:3
z8|z8|z8|z8|z8|z8|z8|z8|A4 c4|F4 z4|B2 c2 d2 B2|c4 z2 e d|c2 A2 z2 A c
|d2 f2 z2 d2|E4 z2 e2|e2 e d c2 c B|A2 c2 A2 F A|B2 B c d2 B A
|G2 A2 B2 G2|A2 c B A2 z2|A4 F4|D4 B,4|E2 C2 B,2 E2|A,4 z4
|A4 c4|F4 z4|B2 c2 d2 B2|!fermata!c4|]
V:4
z8|z8|z8|z8|z8|z8|z8|z8|z8|z8|z8|A4 c4|F4 z4|B2 c2 d2 B2|c4 z2 e d
|c2 A2 z2 A c|d2 f2 z2 d2|E4 z2 e2|e2 e d c2 c B|A2 c2 A2 F A
|B2 B c d2 B A|G2 A2 B2 G2|A2 c B A2 z2|A4 F4|D4 B,4
|E2 C2 B,2 E2|!fermata!A,4|]
```

Bewertungskriterien

- Die von dem Programm einlesbare Musiknotation sollte auf ABC basieren. Die Kennzeichnung der Stimmeinsätze sollte dokumentiert sein. Das Programm muss die Stimmeinsätze allein der angepassten Eingabedatei entnehmen können. Nicht im Sinne der Aufgabenstellung sind Lösungen, die nur Teile der Melodie wirklich im Textformat einlesen, andere Informationen wie Taktbeginn, Stimmeinsatz, etc. als explizite Benutzereingabe einfordern.
- Die Gesamtlänge des ausgesetzten Kanons sollte so sein, dass in jeder Stimme die Melodie mindestens einmal vollständig vorkommt.

- Da die vorgegebenen Beispiele verschiedene Taktarten ($\frac{2}{4}$ und $\frac{4}{4}$) haben, sollten mindestens diese beiden bei der Pausenerzeugung berücksichtigt werden. Einschränkungen bzgl. der Standardnotenlänge (Header `H:`) oder der zulässigen Taktarten sollten angesprochen werden. Die Anzahl der Pausen und deren Länge muss korrekt sein.
- Das Programm muss eine ABC-konforme Ausgabe erzeugen, die in allen Stimmen gleich viele Takte enthält.
- Die in der Aufgabenstellung verlangten Kanons (also die drei vorgegebenen und mindestens ein selbst gewählter) sollten jeweils durch Ein- und Ausgabe dokumentiert werden, und zwar mindestens in der (angepassten) ABC-Notation. Eine Darstellung in Notenschrift ist nett, aber nicht gefordert. Wenn gar Musikdateien erzeugt werden (z.B. MIDI), kann das auf dem Bewertungsbogen positiv vermerkt sein, führt aber nicht zu einem Bonuspunkt.

Das Problem, dass der Endtakt einen Auftakt enthalten kann, der am Schluss des Kanons unterdrückt werden sollte, tritt im vorgegebenen Kanon *Bona Nox* auf. Da jedoch praktisch niemand dieses Problem erkannt oder angesprochen hat, wurde darauf verzichtet, deswegen einen Punkt abzuziehen.

Aus den Einsendungen: Perlen der Informatik

Allgemeines

Wort des Wettbewerbs: Algrythmus

Dokumentation 27. BWINF

In dubio pro algorithmo.

Genau das, was ich in meinem Kopf gemacht habe, sollte nun auch das Programm machen. *Ja ja, die Hoffnung stirbt zuletzt.*

Einfach drauflos zu programmieren führt selten zum Erfolg.

. . . was ~~ziemlich blöd ist~~ noch verbessert werden sollte.

Vorbemerkung eines Lehrers: Die Modellierung der Lösungen ist daher etwas, nun, sagen wir: umständlich.

Technisches

Die Wahl der Programmiersprache fiel, wie wir später bereuten, auf PHP/HTML.

Das Programm hat mehr als 100 Mal funktioniert. Aus diesem Grund führe ich hier nur einen Testlauf vor.

Wegen des bewusst durchsichtigen objektorientierten Designs wurde auf eine Optimierung der Laufzeit verzichtet.

. . . Programmiersprache C++. Dabei wurde (als Hommage an die guten alten Zeiten, als Pascal noch hip war) auf neumodischen Schnickschnack wie objektorientierte Programmierung, eine durchdachte Lösungsidee oder übersichtlichen Quelltext so gut es geht verzichtet.

Prämienjagd

Das Schöne an dieser Aufgabe ist, dass man fast freie Auswahl der Laufzeitkomplexität hat.

Nimm die Partner und erniedrige den Count.

. . . und von den hinterbliebenen Nachbarn . . .

Aber wenn ich 5 Mio. Waren im Wert von 250 Mio €kaufe, kommt mich ein Übernahmeangebot an die Supermarktkette vielleicht günstiger.

Wer jedoch über 4 Milliarden Artikel kauft, ist entweder nicht ganz dicht, hat zu viel Geld oder sollte wenigstens ganz andere Prämien fordern.

Wenn ich 20.000 Tüten Chips im Aldi aufs Fließband lege, werden ich sowieso goldener Kunde, ohne auf irgendeine Reihenfolge zu achten.

Daher wurde die maximale Skriptlaufzeit auf 12 Stunden begrenzt.

Programmausgabe nach dem Einlesen der Daten: `Finisched reading`

Der Graf am Anfang ...

Damit dürfte mein Algorithmus maximal 17h brauchen, ist also relativ rasant.

Durch das Speichern von Preisindizien ...

Perlenketten

Perlensorten: weise Perlen, Präferenzperlen, Masterperlen

Man muss die Kette, die eigentlich rund ist, drehen.

Die Konsolenausgaben sind absichtlich als Negative ausgedruckt – Druckertinte ist teuer.

Winddiagramme

In Hamburg soll eine Skischanze gebaut werden.

Wir gehen also davon aus, dass uns eine deklarative Sprache die entsprechenden Datensätze aus der Tabelle liefert, die wir brauchen, und eine Black Box diese Daten in eine Grafik umwandelt.

1 Radioaktivität entspricht 2 Pixel.

Ich appelliere aber an Ihr Abstraktions- und Vorstellungsvermögen als Informatiker und beharre darauf, dass ich doch eine kreativere Lösung gefunden habe, als einfach nur die Daten als Punktwolke in einem Koordinatensystem darzustellen.

Pass-Algorithmen

Als faule Informatiker besinnen wir uns des Monte-Carlo-Versuchs.

Also bringt dem Räuber das Erraten des Systems nichts. Er ist dann im Treppenflur und kann das Geländer hinunterrutschen, sonst nichts.

Dieser Algorithmus ist recht schwer zu berechnen. Daher könnte er z.B. in Forschungslaboren an Großrechnern und Ähnlichem angewendet werden, da dort intelligente Menschen arbeiten, denen solche Rechnungen keine Probleme bereiten, und ein Rechenvorgang auch etwas länger dauern kann.

Dadurch haben wir einen verdammt sicheren Algorithmus, an dem sich jeder Dieb die Zähne ausbeißen wird.

Der erst Logarithmus könnte $y = 2x^2 + 3$ sein.

Als zusätzliche Sicherheitsmaßnahme sollte man den Algorithmus nicht an Personen weitergeben, die beim Rechnen Selbstgespräche führen.

Der Anwender kennt nur seine Form des Quadrates, die er vorher erfahren hat.

Als ich das Beispielsystem zum ersten Mal ausprobiert habe, stellte ich fest, dass es mich verwirrt hat.

... da die Welt technologisiert wird und neue Klassen von Pass-Algorithmen gebraucht werden.

Wenn der Vergleich wahr ist, dann wäre der Benutzer eingelegt.

Mit einem wie oben beschriebenen Verfahren aber würde ein solch tödliches Verhalten der Vergangenheit angehören, da selbst ein als Wissenschaftler verkleideter Tom Cruise meine Matrix nicht knacken kann.

Außerdem sind die Gehirnflüsse des Mensch z.Zt. nicht ohne Entdeckung lesbar, wodurch die Berechnung im Kopf eine Abtrennung von allen anderen Netzwerken bewirkt und somit eine höhere Sicherheit folgt.

Als Beispiel dient mir ein Laborkomplex, zu dem sich wenige zerstreute Wissenschaftler durch eine Passworteingabe Zutritt verschaffen können.

Einigen Personen würde dies sicherlich stark fordern; erst recht, wenn sie an einem stressigen Tag in einer sowieso viel zu kurzen Mittagspause zu lange in einer Schlange gestanden haben, um dann von einer solchen Maschine malträtiert zu werden.

... und merken wir uns zu Demonstrationszwecken die PIN 2847 meines Kontos.

... wird 10 mit 44 addiert, und 44 wird mit 10 addiert. Ergebnis ist jeweils 55.

Kosmischer Tanz

... dass der Planet auf ewig und einer Ellipse um den Stern kreisen wird.

Der Stern verschwindet in den ewigen Weiten des Fensters.

Wenn man die Physik vernachlässigt und künstlerisch aktiv wird, kann man die Schrittweite erhöhen. Dies führt zu stärkeren Berechnungsfehlern, aber auch zu schöneren Grafiken.

Wenn die kinetische Energie des Himmelskörpers zu gering ist, macht er unliebsame Bekanntschaft mit dem Zentralkörper.

Nur ein kleiner Schritt für den Planeten, doch ein großer Schritt für den Computer.

Es ist zu erwarten, dass der Körper K sich in einer Spirale S annähert, bis er auf S prallt.

Durch die Angabe von negativen Massen kann sogar Antigravitation simuliert werden.

Hierbei sieht man, dass der kleine Planet zunächst etwas um den großen Planeten eiert, sich dann aber in eine feste Bahn einordnet.