25. 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:

- 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 oder Diskette(n).
- Beispiele werden als Teile des Programm-Ablaufprotokolls immer erwartet. Zu wenige Beispiele und erst recht die Nichtbearbeitung vorgegebener Beispiele führen zu Punktabzug. Leider war das sehr häufig der Fall.
- Beispiele, aber auch Programmdokumentation und Programmtext müssen ausgedruckt sein. Wir können aus Zeit- und Kostengründen keine Ausdrucke machen, so dass es prinzipiell nichts nützt, Beispiele nur auf CD/Diskette abzugeben oder gar nur ins Programm einzubauen. In solchen Fällen gab es meist Punktabzug.
- 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.
- 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.
- 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.

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 vom Programm berücksichtigt werden. 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. Für die Gesamtbewertung sind die drei am besten bewerteten Aufgabenlösungen maßgeblich, es lassen sich also maximal 15 Punkte erreichen. Einen 1. Preis erreichen Sie mit 14 oder 15 Punkten, einen 2. Preis mit 12 oder 13 Punkten und eine Anerkennung mit 9, 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.

- Wellenlinien wurden übrigens häufig für die Dokumentation (also Lösungsidee, Programm-Dokumentation, Programmablauf-Protokoll 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.

Aufgabe 1: Närrische Wirtschaft

Problemstellung

Geschichte

In Aufgabe 1 werden wir Schatzmeister eines Karnevalvereins und haben Probleme mit den leidigen Finanzen. Unser Verein gibt uns zwar in jedem Monat einen festen Geldbetrag, hat dafür aber auch eine lange Wunschliste mit verschiedenen Gegenständen. Leider sind manche Wünsche so groß, dass das Geld in einem Monat nicht reicht – wir müssen über mehrere Monate sparen. Um unsere Aufgabe noch schwieriger zu machen, haben unsere Vereinsmitglieder einen unstillbaren Hunger auf Eis und lassen sich nur bis zum Monatsende zurückhalten, dann wird für das verbliebene Geld Eis eingekauft. Deshalb müssen wir Vermögen in Form von gekauften Wertgegenständen ansammeln. Unsere Einkäufe können wir in einem anderen Monat wieder in Geld umtauschen und davon neu einkaufen. So können wir Geld in den nächsten Monat „hinüberretten". Wir können allerdings nur Gegenstände kaufen, die auch auf unserer Wunschliste stehen (und nicht zwischendurch noch andere Sachen einkaufen, um unser Budget möglichst gut auszuschöpfen). Unser Ziel ist es, alle Gegenstände einzukaufen, die sich unser Verein wünscht.

Problemverständnis, Schwierigkeiten

So setzen wir uns also eines schönen Tages an unseren Schreibtisch und versuchen mit Feuereifer, dem Problem mit Rechenleistung beizukommen. Doch schnell stroßen wir auf drei Schwierigkeiten:

1. Wir müssen für jeden Monat entscheiden, welche Gegenstände wir einkaufen möchten. Dafür gibt es sehr viele verschiedene Möglichkeiten, und selbst unserem Computer trauen wir so viel Rechnerei nicht zu.
2. Zusätzlich haben wir die Freiheit, Gegenstände wieder zu verkaufen. Dadurch gibt es noch mehr Möglichkeiten.
3. Wir sorgen uns, dass sich vorschnelle Entscheidungen später rächen: Was wir später einkaufen können, hängt von unseren vorherigen Entscheidungen ab. Die Entscheidungen in den einzelnen Monaten sind *abhängig* von einander.

Wir werden in den nächsten beiden Abschnitten sehen, dass wir 2. und 3. relativ leicht bewältigen können. Für Schwierigkeit 1 hingegen finden wir erst später eine Lösung.

Gesamtvermögen

Zunächst einmal sieht es so aus, als müssten wir nicht nur jeden Monat entscheiden, welche Gegenstände wir neu *einkaufen*, sondern zusätzlich vorher darüber nachdenken, welche

Gegenstände wir *verkaufen*. Das möchten wir gerne vermeiden und tun das so: Zu Beginn jedes Monats verkaufen wir alle Gegenstände. Anschließend kaufen wir von unserem *Gesamtvermögen* – also dem Erlös dieses Verkaufs plus dem neu hinzugekommenen Monatsbudget – ein. Wenn wir auf diese Weise berechnet haben, welche Gegenstände wir neu einkaufen wollen, können wir leicht vergleichen, welche wir gar nicht erst hätten verkaufen sollen.

Abhängigkeit zwischen den Monaten

Diese Frage macht uns ein wenig mehr Probleme. Wir betrachten ein Beispiel:

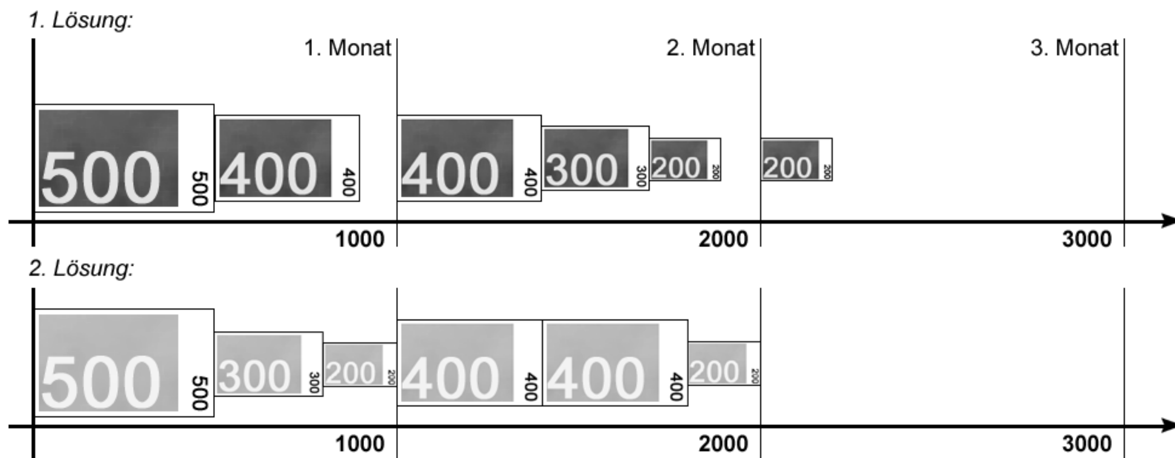


Abbildung 1: Oben sehen wir die kurzentschlossene Lösung. Unten lösen wir das Problem optimal.

Unser Karnevalverein hat ein monatliches Budget von 1000 Euro und möchte Gegenstände im Wert von 500, 400, 400, 300, 200 und 200 Euro kaufen. Schnellentschieden kaufen wir nacheinander immer den teuersten Gegenstand, den wir uns noch leisten können, d.h.: im ersten Monat geben wir erst 500 Euro und dann 400 Euro aus; anschließend können wir keinen weiteren Gegenstand mehr kaufen. Zu Beginn des zweiten Monats steht uns dann ein Gesamtvermögen von 1900 Euro zur Verfügung. Wir kaufen davon die Gegenstände für 500 und 400 Euro (bzw. verkaufen sie nicht), und kaufen dann anschließend die Gegenstände für 400, 300 und 200 Euro. Jetzt besitzen wir Gegenstände im Wert von 1800 Euro und können das letzte Objekt leider nicht mehr einkaufen. Das schaffen wir erst im dritten Monat, denn zu Beginn des dritten Monats stehen uns 2800 Euro Gesamtvermögen zur Verfügung, mehr als genug, um alle Gegenstände zu kaufen. Ein solches Vorgehen (höchster Wert zuerst) nennt sich übrigens „greedy“, auf deutsch also „gierig“.

Bekanntermaßen ist Gier keine positive Eigenschaft. Besser wäre es gewesen, zu Beginn die Gegenstände für 500, 300 und 200 Euro zu kaufen und damit 1000 Euro in den zweiten Monat hineinzuretten. Das Gesamtvermögen zu Beginn des zweiten Monats hätte dann 2000 Euro betragen und wir hätten einen Monat früher alle Gegenstände einkaufen können. Wir müssen also aufpassen, dass wir in jedem Monat genug einkaufen. Selbst wenn wir nur einen kleinen Betrag zu wenig anlegen, kann uns das später einen oder mehrere zusätzliche Monate bescheren. Wenn wir unseren Einkauf so auswählen, dass es mit Sicherheit keine andere Einkaufsmöglichkeit mit höherem Wert gibt, nennen wir den Einkauf *optimal*. Dann stellen wir

folgendes fest: Kaufen wir in jedem Monat optimal ein, besitzen wir zu jedem Zeitpunkt so viel Vermögen wie möglich. Dann enden wir auf jeden Fall mit einer minimalen Anzahl an Monaten. Durch optimale Einkäufe sind wir auf der sicheren Seite und können die Monate *unabhängig* voneinander betrachten.

Später werden wir feststellen, dass das optimale Einkaufen ein schwieriges Problem ist. Es sei daher an dieser Stelle noch gesagt, dass es theoretisch nicht immer notwendig ist, das Budget in jedem Monat optimal auszunutzen. Hätten wir z.B. im vorhergehenden Beispiel nur die Objekte mit den Preisen 500, 400, 400, 300 und 200, so wären wir auch mit gierigem Vorgehen nach zwei Monaten fertig (und schneller geht es nicht). Leider wissen wir nicht, wie wir erkennen können, dass im vorliegenden Fall eine „gute", aber nicht optimale Lösung ausreicht. Deshalb müssen wir den Aufwand für die optimale Lösung in Kauf nehmen.

Optimales Einkaufen pro Monat

Der Weg zum Ziel

Wir wollen (in einem bestimmten Monat) für ein vorgegebenes Gesamtvermögen ausrechnen, wie man es am besten ausnutzen kann, d.h. die Menge von Gegenständen mit dem höchsten Gesamtwert finden, die das Gesamtvermögen nicht überschreitet. Da derartige Budgetplanungen auch in vielen anderen Zusammenhängen auftreten, sind wir zum Glück mit unserem Problem nicht alleine: Es ist in der Informatik unter dem Namen „Subset-Sum-Problem" (SSS) bekannt.

Subset-Sum-Problem

Für das Subset-Sum-Problem ist klassischerweise eine Menge von Zahlen $A := \{s_1, \ldots, s_n\}$ sowie eine weitere Zahl S als Eingabe gegeben. Gesucht wird eine Auswahl (d.h. eine Teilmenge $A' \subset A$) der Zahlen, deren Summe möglichst nahe an S liegt, aber S nicht überschreitet. Formal heißt das:

$$\text{maximiere } S' := \sum_{s \in A'} s \text{ unter der Bedingung } S' \leq S$$

Im Folgenden meinen wir mit *Summe* oder *Wert* einer Auswahl von Zahlen die *Summe der Zahlen, die in der Auswahl enthalten sind.*

Komplexität und ein naiver Ansatz

Das SSS-Problem zählt zu den *NP-vollständig*en Problemen. Das bedeutet, dass es zu einer sehr großen Menge von 'gleich schweren' Problemen gehört, zu denen bisher niemand einen effizienten Algorithmus gefunden hat. NP-vollständige Probleme haben gemeinsam, dass man sie entweder *alle* oder aber *keines* von ihnen effizient lösen kann. In der Vergangenheit haben deshalb viele Informatiker versucht, für eines dieser vielen Probleme eine effiziente Lösung zu finden. Da sie alle gescheitert sind, erwarten wir nicht, eine effiziente Lösung für das SSS

zu finden. Wie bei den meisten NP-vollständigen Problemen kann man natürlich alle möglichen Lösungskandidaten bilden und dann am Ende den besten (hier: Auswahl A' mit Wert möglichst nahe, aber kleiner S) auswählen. Leider gibt es 2^n viele Auswahlen der n Objekte. Ein Algorithmus, der nach diesem Prinzip arbeitet, hat also eine Laufzeit von $\Omega(2^n)$. Deshalb verwerfen wir diese Idee wieder.

Lösung mit dynamischer Programmierung

Idee Das Problem ist zwar NP-vollständig, aber folgende Beobachtung lässt uns doch auf eine brauchbare Lösung hoffen: Es gibt zwar 2^n Möglichkeiten, Zahlen auszuwählen, aber viele Auswahlen haben genau den gleichen Wert. Als Beispiel betrachten wir Tabelle 1. Hier

Auswahl	Wert	Auswahl	Wert
	0	3,5	8
2	2	3,8	11
3	3	5,8	13
5	5	2,3,5	10
8	8	2,3,8	13
2,3	5	2,5,8	15
2,5	7	3,5,8	16
2,8	10	2,3,5,8	18

Tabelle 1: Alle möglichen Auswahlen der Zahlen 2, 3, 5, 8 und die entstehenden Werte

haben wir die Zahlen 2, 3, 5, 8. Für diese Zahlen gibt es zwar 16 Auswahlmöglichkeiten, aber nur 12 verschiedene Ergebnisse. Dementsprechend werden manche Werte gar nicht erreicht. Woran liegt das?

Bestimmte Werte kann man generell durch verschiedene Auswahlen erreichen, z.B. hier die 5 durch die Auswahlen $\{5\}$ oder $\{2, 3\}$. Zusätzlich können wir beobachten, dass eine einzelne Summe, die zweifach erreichbar ist, weitere mehrfach erreichbare Werte mit sich bringt. In unserem Beispiel kann man die 5 zweimal erreichen und hat dadurch sofort zwei Möglichkeiten, die 13 zu erreichen: $5+8$ oder $2+3+8$. Wir möchten eigentlich nicht beide Auswahlen berechnen. Es würde uns reichen, wenn wir wüssten, dass man die Zahl 5 mit irgendeiner Auswahl der Zahlen 2,3,5 erreichen kann: dann können wir ausrechnen, dass man mit der 8 zusätzlich auch die 13 erreichen kann.

Solche Beobachtungen sind charakteristisch für Probleme, die sich mit *dynamischer Progammierung* lösen lassen. Mit diesem Algorithmenkonzept kann man redundante Berechnungen verhindern. Es funktioniert für ein Problem, wenn man folgendes finden kann:

- eine Zerteilung des Problems in kleine Teilprobleme und
- einen Zusammenhang zwischen den Teilproblemen, mit dem größere Teilprobleme mit Hilfe von bereits berechneten kleineren Teilproblemen gelöst werden können.

Dann versucht man, die Teilprobleme geordnet zu durchlaufen, so dass kleinere Teilprobleme zuerst und nur einmal gelöst werden.

Wir kehren zurück zum SSS. Unsere Überlegungen am Anfang dieses Abschnitts lassen uns vermuten, dass es gut ist, das Problem in folgende kleinere Teilprobleme zu zerlegen:

> *Kann der Wert x (für x zwischen 0 und S) mit einer Auswahl der Zahlen $s_1, s_2, \ldots, s_i$ (für i zwischen 0 und n) dargestellt werden?*

Wir finden auch einen Zusammenhang zwischen diesen Teilproblemen: Wir haben alle Teilprobleme für ein bestimmtes i und alle x gelöst, wenn wir für alle Werte zwischen 0 und S wissen, ob sie mit einer Auswahl der Zahlen $s_1, \ldots, s_i$ dargestellt werden können. Jetzt nehmen wir die Zahl s_{i+1} hinzu. Dadurch erhalten wir S neue, größere Teilprobleme:

> *Kann man x ($0 \leq x \leq S$) mit einer Auswahl der Zahlen $s_1, \ldots, s_{i+1}$ darstellen?*

Wir können diese Probleme schnell lösen: Alle Werte, die wir vorher erreichen konnten, können wir weiterhin erreichen. Zusätzlich können wir zu jedem bisher erreichbaren Wert die neue Zahl hinzuaddieren. Die Summen sind dann neu erreichbar. Anschließend kennen wir die erreichbaren Werte für Auswahlen aus $s_1, \ldots, s_{i+1}$. Wir haben jeden möglichen Wert nur einmal betrachtet.

Umsetzung Diese Idee lässt sich nun umsetzen, indem wir in einem Array an der Position x abspeichern, ob (und wie) wir den Wert x erreichen können. Da wir maximal den Wert S erreichen wollen, genügt ein Array der Größe S. Für alle Positionen wird zunächst gespeichert, dass sie mit 0 (also der leeren Auswahl) erreichbar sind. Dann wird der Array für jede Zahl s_i aus der Einkaufsliste einmal durchlaufen: An jeder Stelle, deren (Positions-)Wert als erreichbar markiert ist, addieren wir s_i zum eingetragenen Wert s. Sofern die Summe den Zielwert S überschreitet, können wir den Durchlauf für s_i schon beenden. Ansonsten tragen wir an Position $s + s_i$ den Wert $s + s_i$ ein und merken uns noch, dass (a) s_i hier als Summand verwendet wurde und (b) s der Ausgangswert der Summenbildung war. Da für s ähnliche Informationen vorliegen, können später rekursiv alle einzelnen Summanden rekonstruiert werden. Wenn die gesamte Einkaufsliste verarbeitet wurde, wird diese Rekonstruktion auf den größten Wert angewandt, der im Array als erreichbar markiert ist. Auf diese Weise haben wir den optimalen Einkauf für das Gesamtvermögen S berechnet.

Großeinkauf

Jetzt können wir unser Vorgehen insgesamt beschreiben.

Zunächst lesen wir unser monatliches Budget und die Preise der Gegenstände ein. Dann sortieren wir die Gegenstände aufsteigend nach dem Wert, um die Objekte herauszufiltern, die wir niemals kaufen können. Wir können einen Gegenstand nicht erwerben, wenn die Summe aller kleineren Gegenstände plus einem Monatsbudget nicht ausreicht, um ihn zu bezahlen. In diesem Fall können wir das nötige Geld nämlich nicht ansparen. Um festzustellen, ob die Summe der kleineren Objekte groß genug ist, führen wir die Summe der bisherigen Objekte immer mit und addieren den Wert eines Gegenstandes hinzu, sobald wir diesen als kaufbar eingestuft haben.

Anschließend geht es auf zur eigentlichen Lösung. In jedem Monat besitzen wir bestimmte Objekte und es steht uns ein bestimmtes Gesamtvermögen zur Verfügung. Wir geben dieses als

Zielwert in unseren SSS-Algorithmus und übergeben als Zahlen die Preise aller Gegenstände auf unserer Wunschliste. Unser SSS-Algorithmus liefert uns dann den Wert, den wir in den nächsten Monat „hinüberretten” können und eine passende Auswahl der Objekte. Durch Vergleich können wir sehen, welche Gegenstände wir ver- und welche Gegenstände wir einkaufen müssen. Dann gehen wir zum nächsten Monat, addieren das monatliche Budget zu unserem geretteten Geld und fahren analog fort.

Wir können stoppen, sobald uns in einem Monat genug Geld zur Verfügung steht, um alle Gegenstände einzukaufen.

Verbesserungsidee Wir haben also eine Lösung! Doch für manche Testfälle dauert die Berechnung entschieden zu lange. Auf unserem Schreibtisch liegen eine Menge bekritzelter Zettel mit verschiedenen Beispielen, die wir beim Entwurf unseres bisherigen Algorithmus' vollgeschrieben haben. Ein Blick darauf bringt uns den rettenden Gedanken. Die SSS-Probleme, die wir für jeden Monat lösen, unterscheiden sich nur in dem jeweiligen Zielwert! Wir berechnen immer wieder fast das gleiche Problem – tatsächlich sind die früheren SSS-Probleme in Wirklichkeit Teilprobleme der späteren Probleme. Genauer berechnen wir jeden Monat alle möglichen Summen der immer gleichen Objekte neu, brechen nur zu unterschiedlichen Zeitpunkten die Berechnung ab. Wenn wir nun stattdessen *einmal* das größte SSS-Problem (also das, mit dem größten Zielwert) im voraus lösen, können wir in den anderen Problemen sofort auf diese Lösung zurückgreifen.

Umsetzung Zur Umsetzung dieser Idee berechnen wir zunächst nach dem Aussortieren der nicht erreichbaren Gegenstände ein großes SSS-Problem. Dieses erhält als Eingabe die Preise aller Objekte sowie die Summe dieser Preise. Nach dem Aufruf unseres SSS-Algorithmus sind wir nicht an der Auswahl der Objekte interessiert, die zum optimalen Wert führt (man wähle alle Objekte), sondern an dem Array, das unsere SSS-Prozedur berechnet. Dieses enthält nämlich jetzt alle überhaupt mit unseren Gegenständen möglichen Summenwerte. Suchen wir dann in einem Monat die Objektauswahl für ein momentanes Gesamtvermögen S, so durchlaufen wir unser vorberechnetes Feld von der Position S aus in Richtung Anfang. Sobald wir auf den ersten positiven Eintrag treffen, rekonstruieren wir die Lösung wie vorher. Damit müssen wir jetzt nicht mehr in jedem Monat das Array neu aufbauen!

Eine letzte, kleine Verbesserung Eine weitere kleine Verbesserung erreichen wir, wenn wir das Summen-Array mit einem Indexarray indizieren: Im Indexarray speichern wir an Position i einen Zeiger auf den ersten positiven Eintrag im Summenarray, der dort links von Position i steht (also die größte Summe, die gebildet werden kann und die noch kleiner als i ist). Jetzt finden wir die Lösung für ein gegebenes Budget sofort. Eine Veranschaulichung dieser Idee findet sich in Abbildung 2

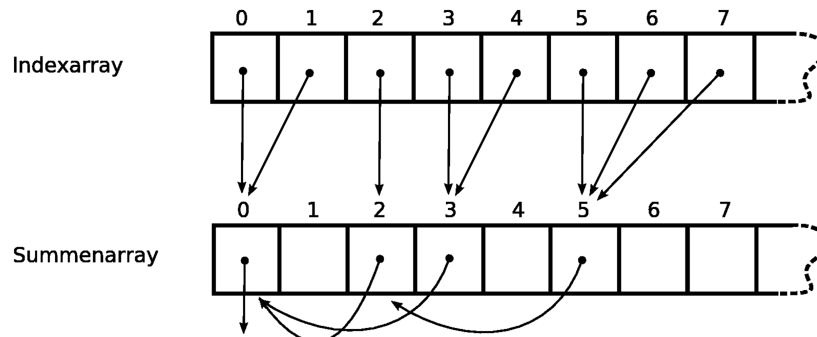


Abbildung 2: Indexarray zur schnellen Suche der Schranken.

Ergebnisse

Laufzeiten

Erfreulicherweise macht sich unsere theoretische Verbesserung auch in der Praxis deutlich bemerkbar. Die für Beispiel 1 (das ist das zeitaufwändigste) benötigte Zeit sinkt von anderthalb Stunden auf anderthalb Minuten. Noch ein wenig Zeit lässt sich dann einsparen, wenn man die Ausgabe abschaltet. Diese ist nämlich so umfangreich, dass alleine die Ausgabe des Einkaufsplans in diesem Fall die anderthalb Minuten in Anspruch nimmt. Ohne Ausgabe berechnet unser verbesserter Algorithmus die Lösung für alle fünf Probleme in insgesamt etwa drei Sekunden.

Dokumentation der Ausgabe

Aufgrund der erwähnten Größe der Ausgabe können wir sie hier nur für das Beispiel aus dem Aufgabentext ungekürzt wiedergeben. Für die anderen Beispiele haben wir jeweils die Ergebnisse angegeben.

Das Beispiel aus der Aufgabenstellung – Budget 1000 Euro

```
1. Monat. Unser Gesamtvermögen beträgt 1000 EUR
Folgende Gegenstände besitzen wir:
Davon verkaufen wir:
Stattdessen kaufen wir: 790

2. Monat. Unser Gesamtvermögen beträgt 1790 EUR
Folgende Gegenstände besitzen wir: 790
Davon verkaufen wir: 790
Stattdessen kaufen wir: 1320 340

3. Monat. Unser Gesamtvermögen beträgt 2660 EUR
Folgende Gegenstände besitzen wir: 1320 340
Davon verkaufen wir:
Stattdessen kaufen wir: 790

4. Monat. Unser Gesamtvermögen beträgt 3450 EUR
```

```
Folgende Gegenstände besitzen wir: 1320 790 340
Davon verkaufen wir: 790 340
Stattdessen kaufen wir: 2100

5. Monat. Unser Gesamtvermögen beträgt 4420 EUR
Folgende Gegenstände besitzen wir: 2100 1320
Davon verkaufen wir:
Stattdessen kaufen wir: 790

6. Monat. Unser Gesamtvermögen beträgt 5210 EUR
Folgende Gegenstände besitzen wir: 2100 1320 790
Davon verkaufen wir: 2100 1320 790
Stattdessen kaufen wir: 5200

7. Monat. Unser Gesamtvermögen beträgt 6200 EUR
Folgende Gegenstände besitzen wir: 5200
Davon verkaufen wir:
Stattdessen kaufen wir: 790

8. Monat. Unser Gesamtvermögen beträgt 6990 EUR
Folgende Gegenstände besitzen wir: 5200 790
Davon verkaufen wir: 790
Stattdessen kaufen wir: 1320 340

9. Monat. Unser Gesamtvermögen beträgt 7860 EUR
Folgende Gegenstände besitzen wir: 5200 1320 340
Davon verkaufen wir:
Stattdessen kaufen wir: 790

10. Monat. Unser Gesamtvermögen beträgt 8650 EUR
Folgende Gegenstände besitzen wir: 5200 1320 790 340
Davon verkaufen wir: 790 340
Stattdessen kaufen wir: 2100

11. Monat. Unser Gesamtvermögen beträgt 9620 EUR
Folgende Gegenstände besitzen wir: 5200 2100 1320
Davon verkaufen wir:
Stattdessen kaufen wir: 790

12. Monat. Unser Gesamtvermögen beträgt 10410 EUR
Folgende Gegenstände besitzen wir: 5200 2100 1320 790
Davon verkaufen wir:
Stattdessen kaufen wir: 670

13. Monat. Unser Gesamtvermögen beträgt 11080 EUR
Folgende Gegenstände besitzen wir: 5200 2100 1320 790 670
Davon verkaufen wir:
Stattdessen kaufen wir: 340

Wir haben in 13 Monaten alle Gegenstände eingekauft!
```

Und hier die Ergebnisse für die weiteren Pflichtbeispiele:

Beispiel 1: 131071 Monate

Beispiel 2: 172 Monate

Beispiel 3: 1026 Monate

Beispiel 4: Wir können kein Objekt kaufen!

Im vierten Beispiel kann man keinen Gegenstand kaufen, da schon das billigste Objekt mehr kostet, als unser Budget erlaubt.

Bewertungskriterien

Diese Aufgabe ist wirklich nicht einfach. Eine Lösung und Überlegungen der obigen Qualität werden in der ersten Runde nicht erwartet. Entscheidend ist aber die Einsicht, dass einmal berechnete Teillösungen wieder verwendet werden können; dies ist auch bei einem rekursiven Verfahren möglich. Auf der nächsten Stufe liegen Lösungen mit (a) einem Verfahren, das optimale Werte berechnet und dabei nicht völlig brute force vorgeht, oder (b) einem gierigen Verfahren bei gleichzeitiger Einsicht, dass ein solches nicht immer optimale Werte liefert (für das Pflichtbeispiel 2 berechnet ein gieriges Verfahren den Wert 199 statt 172). Reine Rekursion (mit der manche Beispiele nicht zu lösen sind) oder gedankenlose Gier kommen erst an dritter Stelle und sind schon keine akzeptablen Lösungen mehr.

- In der Regel werden optimale Ergebnisse erwartet. Wird in einer Einsendung eine Lösung mit nicht immer optimalen Ergebnissen verwendet (z.B. mit einem gierigen Verfahren), soll Einsicht in dieses Problem dokumentiert sein.
- Eine wesentliche Erkenntnis ist, dass die Lösung dieses Problems deutlich beschleunigt werden kann, wenn Teilergebnisse wieder verwendet werden. Falls die Laufzeiten sehr lange sind, weil das gewählte Verfahren diese Erkenntnis nicht umsetzt oder generell zu einfach gestrickt ist (z.B. eine einfache Rekursion für das Subset-Sum-Problem), bedeutet das einen Mangel. Aber auch gierige Verfahren kommen hier nicht ungeschoren davon, denn auch sie verwenden keine Teilergebnisse.
- Der Speicherbedarf des Programms darf nicht so hoch sein, dass das Programm nicht in der Lage ist, alle lösbaren Beispiele zu lösen (das größte Beispiel ist nicht wichtig, da es sowieso nicht lösbar ist). Zu hoher Speicherbedarf kann z.B. auftreten, wenn man das Problem nicht als Subset-Sum-, sondern als Rucksackproblem auffasst und den dafür bekannten dynamischen Algorithmus verwendet. In diesem Fall wird die Matrix bei den größeren Beispielen zu groß.
- Laut Aufgabenstellung muss berechnet werden, welche Gegenstände sich der Verein kaufen kann. Das müssen nicht immer alle sein. Im Extremfall kann, wie im 4. vorgegebenen Beispiel, gar kein Gegenstand erworben werden. Dies muss berücksichtigt werden.
- Die Aufgabe fordert explizit eine Aufstellung dessen, was in jedem Monat an- und verkauft wird. Die Ausgabe sollte übersichtlich und kompakt sein, aber alle Angaben zu Verkäufen und Einkäufen machen sowie am Ende die Anzahl der benötigten Monate ausgeben. Natürlich ist es nicht sinnvoll und auch nicht gefordert, alle Einkaufspläne komplett abzudrucken.

Aufgabe 2: Robot Dressing

Teilaufgabe 1

Zuerst müssen wir die Eingabedaten einlesen. Diese bestehen aus einer Menge von Kleidungsstücken und Angaben zu deren Reihenfolge. Letztere bereiten Schwierigkeiten, wenn man die deutsche Grammatik korrekt beachten will, weshalb wir die Reihenfolgeangaben in einer vereinfachten Form erwarten, in der die Kleidungsstücke immer genau bleich bezeichnet werden; Beispiel: „Jacke vor Schuhe“. Aussagen wie „keine Angaben für die Mütze“ wie im gegebenen Beispiel lassen wir wegfallen. Dann bietet sich folgendes Beschreibungsformat an:

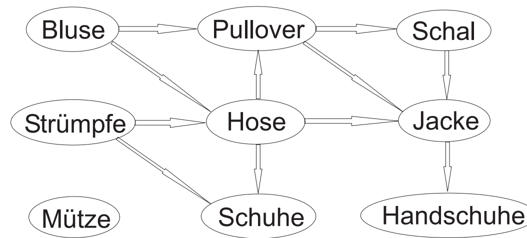
```
Zeile 1: Anzahl n an Kleidungsstücken
Zeile 2..n+1: jeweils ein Kleidungsstück mit vollständigem Namen
Zeile n+2: Anzahl m an Angaben zur Reihenfolge
Zeile n+3..n+m+2: Angabe zur Reihenfolge in folgender Form:
             <Kleidungsstück1> vor <Kleidungsstück2>
```

Das gegebene Beispiel hat dann die folgende Form:

```
9
Bluse
Handschuhe
Hose
Jacke
Muetze
Pullover
Schal
Schuhe
Struempfe
11
Struempfe vor Schuhe
Struempfe vor Hose
Hose vor Schuhe
Hose vor Pullover
Bluse vor Hose
Bluse vor Pullover
Pullover vor Jacke
Hose vor Jacke
Schal vor Jacke
Jacke vor Handschuhe
Pullover vor Schal
```

Teilaufgabe 2

Nehmen wir an, wir schrieben alle Namen der gegebenen Kleidungsstücken auf ein großes Blatt Papier. Was uns fehlt, ist eine kompakte Darstellungsform der vielen Angaben zu deren Reihenfolge. Eine Lösung ist, für eine Angabe „A vor B“ einen Pfeil von A nach B zu zeichnen. Wenn wir dann alle Eingabedaten dargestellt haben, könnte das gezeichnete Objekt für das Beispiel wie im folgenden Bild aussehen.



Ein solches Objekt kann man nun als ein in der Informatik gebräuchliches Modell namens *Graph* auffassen: als eine Menge von *Knoten*, die in unserem Fall die Kleidungsstücke sind, und einer Menge von *Kanten* zwischen diesen Knoten. Über dieses Modell ist viel bekannt, so z.B. dass man einen Graph leicht repräsentieren kann, indem man zu jedem Knoten v eine Liste mit allen Nachbarknoten u speichert, zu denen eine Kante von v nach u existiert. Auch ist bekannt, wie man das Problem der *topologischen Sortierung* löst, wie das in dieser Aufgabe gestellte Problem in der Fachliteratur auch genannt wird.

Aber zurück zur Frage: Welche Objekte können wir denn nun anziehen? Offensichtlich genau solche, zu denen keine Kante hinführt, da wir sonst ein anderes Kleidungsstück vorher anziehen müssten. Also suchen wir uns einen Knoten A ohne solche eingehenden Kanten aus und ziehen das zugehörige Kleidungsstück an. Um ein weiteres heraussuchen zu können, müssen wir nun den gewählten Knoten mitsamt aller ausgehenden Kanten löschen, da ja alle Angaben zur Reihenfolge der Form „A vor B“ für das angezogene Kleidungsstück A erfüllt sind. Algorithmisch könnte das folgendermaßen aussehen:

```
Initialisiere den Graph G mit den eingelesenen Werten
Solange nicht alle Kleidungsstücke angezogen sind:
  Suche einen Knoten v ohne eingehende Kante
  Zieh zugehöriges Kleidungsstück an
  Lösche v und alle von v ausgehenen Kante aus G
```

Kann es passieren, dass (irgendwann) kein Knoten ohne eingehende Kanten existiert? Ja, das kann passieren, aber nur dann, wenn es keine Ankleidereihenfolge gibt, die alle gegebenen Angaben erfüllt. Der Graph enthält dann einen Kreis (auch Zyklus genannt), und in diesem Fall brechen wir mit einer Fehlermeldung ab.

Teilaufgabe 3

Soll man alle möglichen Ankleidereihenfolgen bestimmen, kann man ähnlich wie oben vorgehen. Der Unterschied ist, dass man jede Möglichkeit, einen Knoten ohne eingehende Kanten zu wählen, durchgehen muss. Folgende rekursive Prozedur macht dies deutlich:

```
ZiehWasAn( Graph G ):
  Wenn kein Kleidungsstück mehr übrig
    dann gibt aktuelle Reihenfolge aus
  sonst
    für alle Knoten v ohne eingehende Kanten:
      ziehe zugehöriges Kleidungsstück V an
      berechne rekursiv:
        ZiehWasAn( G ohne v und ausgehende Kanten von v )
      ziehe zugehöriges Kleidungsstück V aus
```

Wiederum muss man darauf achten, dass es keine gültige Ankleidereihenfolge geben kann. Außerdem sollte man beachten, dass Ausgabe sowie Laufzeit schnell astronomisch groß werden können, da es bei n Kleidungsstücken ohne Einschränkungen zu deren Reihenfolge so viele Ankleidereihenfolgen wie Permutationen der ersten n natürlichen Zahlen gibt, und das sind $n!$ und somit sehr viele. Im obigen Verfahren werden die Reihenfolgebedingungen direkt berücksichtigt, so dass nicht alle Permutationen durchgespielt werden müssen. Ein Verfahren, das die Kleidungsstücke auf jeden Fall in alle möglichen Reihenfolgen bringt und erst nachträglich prüft, ob auch die Bedingungen erfüllt sind, ist deutlich im Nachteil.

Bewertungskriterien

- In der ersten Teilaufgabe sollte man die grammatikalischen Schwierigkeiten des gegebenen Eingabeformates erkennen und ein genügend vereinfachtes Format angeben.
- Die zweite Teilaufgabe verlangt eine geschickte Modellierung der Problemstellung. Die Umsetzung in einen Graphen wird nicht verlangt, aber manche Modellierungen erschweren die Lösung unnötig oder verhindern sogar die Berechnung aller möglichen Reihenfolgen und sind damit nicht ausreichend.
- Dann sollte ein möglichst fehlerfreies Programm zur Berechnung einer Reihenfolge angegeben werden, dass auch den Spezialfall abfängt, dass keine gültige Reihenfolge existiert.
- Das Programm sollte nicht grundsätzlich alle Reihenfolgen der Kleidungsstücke durchspielen (und die Reihenfolgebedingungen erst nachträglich anwenden).
- Der dritte Aufgabenteil muss natürlich bearbeitet sein.
- Es müssen so viele Beispiele wie verlangt angegeben werden. Im Beispiel der Aufgabenstellung gibt es 90 verschiedene den Bedingungen entsprechende Ankleidereihenfolgen.

Aufgabe 3: HTML-Mobiles

Lösungsidee

In diese Aufgabe ist viel hineinzudeuten, gedacht ist sie jedoch recht einfach. Im wesentlichen sollte eine Lösung den Unterschied zwischen inhaltlichen und strukturellen Änderungen offensichtlich machen. Dabei greifen „inhaltliche" (dieses Wort wurde vielleicht nicht ganz ideal gewählt) Änderungen auf der – um Begriffe aus dem Bereich der formalen Sprachen zu verwenden – lexikalischen Ebene, strukturelle Änderungen eher auf der syntaktischen Ebene ein. Insgesamt sollen drei inhaltliche und drei strukturelle Änderungsarten realisiert werden, z.B.:

inhaltlich Ändern eines Farbwerts (s. Aufgabenstellung), Verschieben einer xy-Position (s. Aufgabenstellung), Ändern von Zeichensätzen, Ändern von Schriftgrößen, ...

strukturell Reihenfolge von Seitenelementen ändern, Vertauschen von Tabellenelemten (beides s. Aufgabenstellung, wobei Letzteres als Spezialfall des Ersteren angesehen werden kann), alle Überschriften eine Ebene tiefer setzen und eine neue Überschrift der ersten Ebene einfügen (wobei die Wahl des Textes der ersten Überschrift einen inhaltlichen Aspekt hat; um den zu minimieren, könnte man einen Standardtext wie „Überschrift 1" verwenden), ...

Auf der technischen Seite ist vieles möglich. Inhaltliche Änderungen können per Eingriff in CSS-Definitionen geschehen, mit z.B. JavaScript lässt sich gleich das Domain Object Model des HTML-Dokuments manipulieren, was insbesondere strukturelle Änderungen leicht macht. Es ist beinahe alles erlaubt; allerdings sollten alle programmierten Änderungen zumindest grundsätzlich auf beliebige HTML-Dokumente anwendbar sein können, also nicht speziell auf eines der sechs zu schreibenden HTML-Dokumente ausgerichtet sein.

Auch wenn für die Dokumentation nur sechs Bildpaare gefordert sind, mit denen jeweils eine Änderungsart zu demonstrieren ist, soll das Programm dennoch auch zur geforderten Erzeugung von Bildfolgen in der Lage sein. Dabei soll das Programm auf ein HTML-Dokument zufällig eine der Änderungsarten anwenden (und, wenn die Änderungsart Parameter hat, diese zufällig auswählen) und sich praktisch mit dem Ergebnis als Eingabe wieder selbst aufrufen.

Zu beachten ist noch die Größenangabe des Handybildschirms. Abgesehen davon, dass dieser ein wirklich ungewöhnliches Format hat, ist es nicht nötig, dass die HTML-Dokumente wirklich nur 225x100 Pixel groß sind. Aber die wesentlichen Elemente und die Effekte der Änderungsarten sollten innerhalb dieses Rahmens sichtbar sein.

Bewertungskriterien

- Dder Unterschied zwischen inhaltlichen und strukturellen Änderungsarten muss verstanden und richtig umgesetzt sein.
- Es müssen mindestens drei inhaltliche und mindestens drei strukturelle Änderungsarten implementiert sein. Mehr ist nicht gefordert, aber auch nicht verboten.

- Das Programm muss in der Lage sein, Bildfolgen wie beschrieben zu generieren: sukzessive Anwendung der Änderungsarten auf ein Dokument; dabei wird bei jeder Anwendung die Änderungsart zufällig ausgewählt und ggf. deren Parameter zufällig bestimmt.
- Es sind sechs HTML-Dokumente gefordert. Diese sollten so angelegt sein, dass die Effekte der Änderungsarten im Größenbereich 225x100 Pixel (hochkant oder quer) klar zu erkennen sind. Außerdem sollen sie „optisch interessant“ sein. Der BWINF ist kein Gestaltungswettbewerb, aber allzu große Einfallslosigkeit ist nicht gerade zu belohnen.
- Jede Änderungsart muss durch mindestens ein Bildpaar dokumentiert sein.

Aufgabe 4: Supermarkt

Algorithmen und Datenstrukturen diese beiden informatischen Antagonisten können recht unterschiedlich gewichtet sein. Während zur Lösung mancher Probleme der richtige Algorithmus entscheidend ist, kommt es in anderen Fällen auf die richtige Organisation der Daten an. Typisch für den letzten Fall sind Anwendungen auf Basis von Datenbanksystemen – bei denen die schwierigen algorithmischen Probleme in den Datenbankfunktionen schon gelöst sind.

In dieser Aufgabe steht die Datenorganisation im Vordergrund. Orientieren muss sie sich an vier „Geschäftsfällen“, also Angaben zur Nutzung der Daten. Welche Anforderungen stellen die Nutzungsfälle, d.h.: welche Informationen werden jeweils benötigt? In der Aufgabenstellung werden wir darauf aufmerksam gemacht, dass zwischen abgepackter Ware (deren Menge in Stückzahlen angegeben werden kann) und nicht abgepackter Ware (deren Menge in einer geeigneten Maßeinheit anzugeben ist, z.B. kg oder l). Wir wollen im folgenden kurz von Stückware bzw. Mengenware sprechen und bei Mengenware als Maßeinheit prinzipiell kg annehmen – immerhin ist in der Aufgabenstellung von wiegen die Rede.

Im folgenden werden für die verschiedenen Nutzungsfälle angegeben, welche Einzelinformationen benötigt werden, in welcher Beziehung sie zueinander stehen sollten und wie auf dieser Grundlage die benötigten Ausdrucke erstellt werden. Eine konkrete Realisierung der Informationsstrukturierung geben wir nicht an. Eine Datenbank wäre wohl das ideale Werkzeug, aber es kann auch mit Hashtabellen oder (weniger geschickt) Arrays gearbeitet werden.

1. Kassenbon

Ein Kassenbon ist eine Liste der eingekauften Waren und ihrer Preise. Beim Scannen und Wiegen wird die Produktnummer erfasst, so dass über diese Angabe alle weiteren Informationen erreichbar sein müssen. Zum Erstellen des Bons muss beim Scannen einer Stückware auf deren Stückpreis zurückgegriffen werden, bei Mengenware muss der Preis pro kg abgefragt werden können. In beiden Fällen möchten wir auf dem Kassenbon auch eine Bezeichnung der Ware lesen, um Kassenbon und Einkauf abgleichen zu können. Wir haben also folgenden Informationsbedarf:

Produktnummer → Bezeichnung, Preis

Diese Notation besagt, dass sich aus der Produktnummer Bezeichnung und Preis ableiten lassen müssen. Eine Angabe über die Warenart (Stück oder Menge) ist nicht nötig. Wir gehen hier – durchaus wirklichkeitsnah – übrigens von einem einfachen Kassenbon aus, bei dem jede Ware einzeln aufgelistet ist, auch wenn mehrere identische Exemplare eines Produktes (z.B. Milchtüten) eingekauft wurden.

Der Kassenbon wird also erstellt, indem für jede erfasste Produktnummer deren Bezeichnung ausgedruckt wird und dazu nach einem Scan der Preis bzw. nach einem Wiegevorgang der mit dem Wiegeergebnis verrechnete Preis.

2. Mangelliste

In diesem Fall benötigen wir Informationen über den Warenbestand. Der kann bei Stückware in Stück und bei Mengenware in kg angegeben werden. Um dies auch bei der Ausgabe der Liste berücksichtigen zu können, sollten wir nun wirklich für jede Produktnummer wissen, ob es sich um Stück- oder Mengenware handelt:

Produktnummer → Produktart

Nun benötigen wir noch Informationen über den Bestand

Produktnummer → Bestandswert

und über den Wert, unter den der Bestand fallen muss, um auf die Mangelliste zu kommen:

Produktnummer → Mindestwert

Bestandswert und Mindestwert sind in der Maßeinheit der jeweiligen Ware angegeben, können also direkt miteinander abgeglichen werden. Der Mindestwert könnte auch bei Abfrage der Mangelliste vom zuständigen Mitarbeiter (z.B. der Filialleiterin) eingegeben werden.

Für eine lesbare Ausgabe der Mangelliste greifen wir natürlich auch wieder auf die Produktbezeichnung zu. Beim Bestandswert stellt sich noch die Frage, wie er berechnet wird. Wir gehen davon aus, dass die Verkaufsdaten der Kasse (einschließlich des Wiegeergebnisses) direkt zur Veränderung des Bestandswertes führen;

Die Mangelliste wird nun ausgegeben, indem für jede Produktnummer Bestandswert und Mindestwert miteinander verglichen werden und ggf. Produktnummer, Bezeichnung, Bestandswert und das Bestandsmaß (Stück oder kg) gemäß der Produktart ausgegeben werden.

3. Hitliste

Jetzt wird es komplizierter, denn die Hitliste soll nach Produktgruppen sortiert sein und außerdem Verkäufe eines bestimmten Zeitraums berücksichtigen. Beispiele für Produktgruppen: Obst und Gemüse, Fleisch- und Wurstwaren, Getränke. Man kann aber auch feiner gliedern, z.B. die Getränke in Wasser/Limonaden, Säfte, Bier, Wein und sonstige unterteilen. Auf jeden Fall wird folgende Angabe benötigt (der Doppelpfeil besagt, dass man auch von der Produktgruppe auf die Produktnummer schließen können möchte):

Produktnummer ↔ Produktgruppe

Bisher sind wir damit ausgekommen, Informationen über Produkte und ihren Bestand zu verwalten. Um in der Hitliste einen Verkaufszeitraum berücksichtigen zu können, müssen wir nun Informationen über Verkäufe speichern. Wir können dies aber immer noch unabhängig von einzelnen Kunden bzw. von individuellen Einkäufen tun, wenn wir die Informationen für eine vorher festzulegende Zeiteinheit akkumulieren, z.B. für einen Tag (ein Monat würde für diesen Nutzungsfall auch ausreichen). Am Ende hätten wir dann für jeden Tag und jedes Produkt eine Angabe zu den verkauften Mengen:

(Produktnummer, Tag) → Verkaufsmenge

Zur Ausgabe der Hitliste einer Produktgruppe müssen nun zunächst alle Produktnummern dieser Gruppe herausgesucht werden. Für jede dieser Nummern werden (obige tageweise Speicherung von Verkaufsmengen vorausgesetzt) die Verkaufsmengen aller Tage des gesuchten Monats addiert und zu einer Monatsmenge zwischengespeichert: Nach Sortierung der Produktnummern nach der Monatsmenge wird, beginnend bei der größten Menge, jede Produktnummer, die zugehörige Produktbezeichnung und die Monatsmenge ausgegeben.

4. Werbebrief

Hier spielen nun zum ersten Mal individuelle Kundendaten eine Rolle. Für jeden Kunden soll es eine Kundennummer geben, die an der Kasse über das Einlesen einer Kundenkarte erfasst werden kann. Zu einer Kundennummer sind Kontaktdaten gespeichert:

Kundennummer → Vorname, Name, Straße, PLZ, Ort

Für einen Kunden, der beim Einkauf die Kundenkarte vorlegt, werden seine Einkäufe aufgezeichnet. Hier könnte man die kompletten Kassenbons speichern (in diesem Fall gemeinsam mit der Kundennummer, die dann die Verkaufsdaten in Relation mit den Kundendaten setzen würde). Das würde die Menge der gespeicherten Daten deutlich aufblähen. Es reicht aber auch eine akkumulative Speicherung: Pro Kunde und Produkt wird festgehalten, welche Menge des Produkts bisher gekauft wurde.

(Kundennummer, Produktnummer) → Kaufmenge

Über Kundennummer und Produktnummer werden die Kaufmengen mit Produktdaten und Kundendaten in Beziehung gesetzt. Außerdem legen wir für jede Produktgruppe fest, ab welcher Menge ein Werbebrief geschrieben werden soll:

Produktgruppe → Werbemindestmenge

Zur Ausgabe der Adressetiketten werden für jede Kundennummer und jede Produktnummer, die der Produktgruppe Wein zuzuordnen ist, die Kaufmenge herausgesucht und alle Mengenwerte zu einer Gesamtmenge addiert. Falls diese Gesamtmenge größer ist als die Werbemindestmenge der Produktgruppe, werden für die aktuelle Kundennummer die Kontaktdaten für den Etikettausdruck bereitgestellt. Außerdem werden für den Kunden alle Kaufmengen auf 0 gesetzt.

Bewertung des Geschäftsfalls „Werbebrief“ „Bewertung“ hat etwas mit „Werten“ zu tun; hier spielt das Thema Datenschutz eine wichtige Rolle. Für eine solche Werbeaktion sollte auf jeden Fall das Einverständnis des Kunden vorliegen. Eine rein ökonomische Bewertung ist zwar interessant, aber nicht ausreichend.

Bewertungskriterien

- Der Unterschied zwischen Stück- und Mengenware muss berücksichtigt werden.

- Die Strukturierung der Daten soll durch eine zumindest halbformale Notation beschrieben werden. Insgesamt soll aus der Beschreibung hervorgehen, wie (mit welchen Schlüsseldaten) auf die gespeicherten Informationen zugegriffen werden kann, und wie die verschiedenen Informationsmengen (z.B. Tabellen) miteinander verknüpft werden.
- Das Vorgehen zum Erstellen der Ausdrucke muss klar beschrieben sein.
- Die Strukturierung der Daten muss natürlich zu den Geschäftsfällen passen: die beschriebenen Zugriffs- und Verknüpfungsmöglichkeiten müssen auf jeden Fall die Erstellung des jeweils geforderten Ausdrucks ermöglichen.
- Bei der Bewertung von Geschäftsfall 4 wird erwartet, dass das Thema Datenschutz angesprochen und das Einverständnis des Kunden mit einer solchen Werbemaßnahme vorausgesetzt wird.

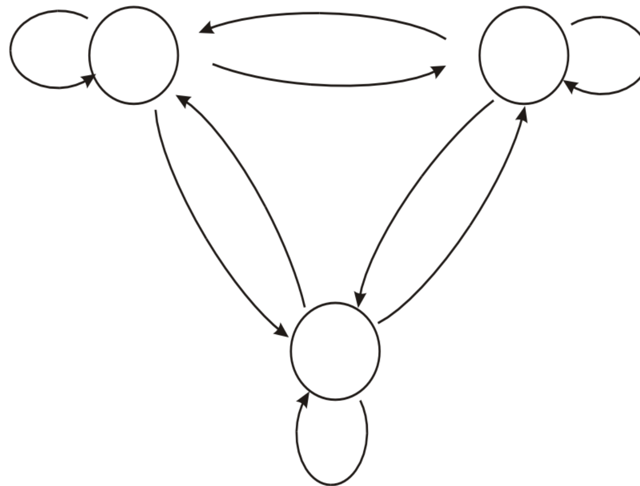
Aufgabe 5: Paderbox

Lösungsidee

Paderbox formal

Zu programmieren ist ein einfaches Rätselspiel, bei dem es darum geht, die interne Verschaltung eines geheimnisvollen Kastens zu entschlüsseln: der Paderbox. Zunächst wollen wir die Paderbox etwas formaler beschreiben: Intern besteht die Paderbox aus einem einfachen Automaten mit einer Menge von drei Zuständen $Z = Z_0, Z_1, Z_2$. Nach außen zeigt die Paderbox drei Lämpchen ($L = L_0, L_1, L_2$) und zwei Tasten ($T = T_0, T_1$). Jederzeit befindet sich die Paderbox in einem Zustand $z \in Z$. Bei Druck der Taste $t \in T$ geht die PB in den Zustand $z' \in Z$ über. Welcher Zustand dies ist, wird durch die Übergangsfunktion f bestimmt. Die Übergangsfunktion f liefert also für einen Zustand und eine Taste (bzw. einen Tastendruck) den Folgezustand: $f : (Z \times T) \rightarrow Z$. Diese Funktion lässt sich am besten als Tabelle ablegen.

Die Grundstruktur des „PB-Automaten“ mit allen seinen Verbindungsmöglichkeiten lässt sich wie folgt darstellen:

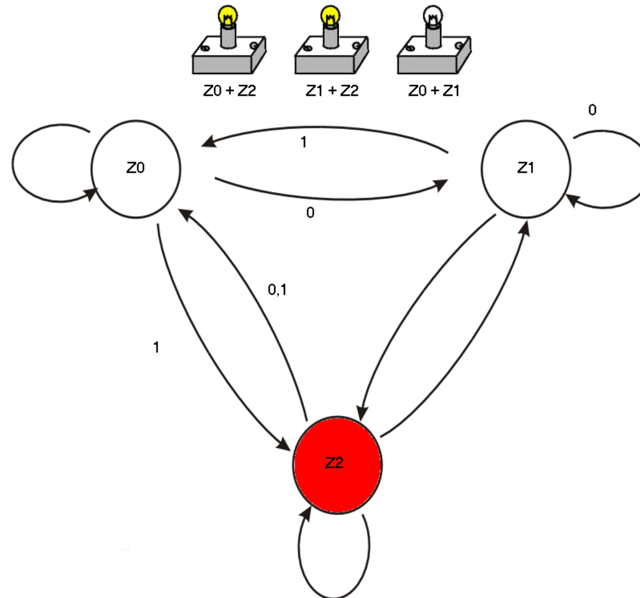


PB-Spiel

Das PB-Spiel soll eine Paderbox zufällig erzeugen können. Dazu wird zunächst die Übergangsfunktion festgelegt. In der Aufgabenstellung ist es zwar nicht explizit gefordert, aber im Sinne der Lösbarkeit sollte man hier von einem deterministischen Automaten ausgehen. Das Ergebnis der Übergangsfunktion ist daher immer eindeutig, wobei die Ergebniswerte hier zufällig gewählt werden können.

Weiterhin werden die Schaltung der Lampen (welche an bzw. aus sind) und der zu Beginn aktuelle Zustand des Automaten in der Paderbox zufällig gewählt. Anschließend müssen noch die Lampenverbindungen generiert werden. Die Verbindungen kann man einfach generieren, indem man zum Beispiel den Zustand Z_0 mit allen Lampen außer einer zufälligen Lampe

L_i verbindet und dann die Zustände Z_1 und Z_2 jeweils mit dieser zufälligen Lampe L_i und (zufällig) einer der beiden Z_0 zugeordneten Lampen verbindet. Die Zuordnung ist auf diese Weise zufällig, jedoch immer den Vorschriften entsprechend. Eine vollständig definierte Konfiguration einer Paderbox sieht zum Beispiel folgendermaßen aus:



Aus dieser zufällig generierten (Anfangs-)Konfiguration muss der Lampenstatus sowie der aktuelle Zustand gesichert werden, um beim Rücksetzen der Paderbox wieder verwendet werden zu können. Das Verhalten der Paderbox lässt sich einfach simulieren. Nach jedem Tastendruck wird zunächst der neue Zustand der Paderbox mit Hilfe der Übergangsfunktion ermittelt. Anschließend werden die beiden mit dem neuen Zustand verbundenen Lämpchen umgeschaltet, also von an nach aus bzw. aus nach an.

Bewertungskriterien

Zu dieser Aufgabe wurden im Vorfeld viele Fragen gestellt. Wie befürchtet, wurde das Automatenmodell häufig grundsätzlich oder teilweise falsch verstanden. Für die Bewertung im Detail sind die Anforderungen an das PB-Spiel in der Aufgabenstellung klar formuliert. Zunächst müssen alle geforderten Funktionen korrekt realisiert sein:

Zufällige Initialisierung Werden beim Erzeugen der Paderbox nicht alle Parameter zufällig initialisiert, kann ein Punkt abgezogen werden. Wird nur die Lampenzuordnung nicht zufällig berechnet, muss nicht abgezogen werden, wenn (zu Recht) begründet wird, dass sich die verschiedenen Zuordnungen nur in der Permutation der Zustände unterscheiden. Erraten werden kann die Programmierung der PB übrigens nur, wenn bei der Initialisierung dafür gesorgt wird, dass alle Zustände auch erreichbar sind. Allerdings schränkt dies die geforderte Zufälligkeit ein; deshalb wird eine entsprechend kontrollierte Initialisierung nicht verlangt.

Zustandsübergang Die Übergangsfunktion muss eindeutig sein und in der Simulation korrekt befolgt werden, einschließlich der Ansteuerung der Lampen.

Rücksetzen Um eine Chance zu haben, die Paderbox zu entschlüsseln, muss man unter Umständen mehrfach nacheinander von einem definierten Zustand ausgehen. Dazu ist es unerlässlich, einen Reset ausführen zu können. Sollte diese Funktion vergessen worden sein, muss daher ein Punkt abgezogen werden. Das in der Aufgabenstellung geforderte „Rücksetzen in den Anfangszustand" kann unterschiedlich verstanden werden: nur den Automatenzustand auf den initialen Wert setzen oder zusätzlich die Lampen wie zu Beginn schalten. Letzteres ist im Sinne des PB-Spiels sinnvoller, akzeptiert werden aber beide Varianten.

Verraten Hier soll die „geheime Programmierung" verraten werden; das ist genau genommen nur die Übergangsfunktion. Die zu verraten ist aber nur sinnvoll, wenn auch die Zuordnung zwischen Zuständen und Lampen verraten wird. Nett, aber nicht gefordert ist, wenn auch Anfangszustand und anfängliche Lampenschaltung preisgegeben werden.

Gefordert wird, die Lampen auf dem Bildschirm darzustellen. Natürlich ist hier eine grafische Darstellung sinnvoll, immerhin sollte ein Spiel programmiert werden. Ähnlich wie in der Junioraufgabe werden aber Konsolenprogramme mit „ASCII-Grafik" akzeptiert – zähneknirschend.

Gänzlich offen lässt die Aufgabenstellung, wie das Funktionieren des PB-Spiels dokumentiert werden soll. Insbesondere mit Blick auf die Bewertbarkeit genügt es nicht, nur die äußere Erscheinung des Spiels zu dokumentieren - dann sieht auch der Bewerter nur Lampen, aber kein Licht. In Ablaufprotokollen sollte auch die Konfiguration, also die Übergangsfunktion, die initiale Schaltung der Lampen und der Anfangszustand, angegeben werden. Außerdem sollten mehrere Zustandsübergänge mit Taste und aktuellem Zustand gezeigt werden.

Junioraufgabe: Maya-Zahlen

Lösungsidee

Die zwei wesentlichen Probleme bei der Aufgabe sind zum einen die Umrechnung von Dezimalzahlen in Maya-Zahlen und zum anderen die Darstellung dieser auf dem Bildschirm. Die Umrechnung von Dezimalzahlen in Maya-Zahlen erfolgt durch die Methode der sukzessiven Division. Das heißt, die Dezimalzahl wird durch 20 (Basis für Maya-Zahlen) dividiert. Der sich ergebende Rest ist genau der Wert der niedrigsten Stelle der gesuchten Maya-Zahl. Anschließend wird der ganzzahlige Quotient der Division erneut durch 20 dividiert. Der sich nun ergebende Rest ist der Wert der zweitniedrigsten Stelle der Maya-Zahl. Mit dem Quotienten der zweiten Division wird dann wieder analog verfahren, usw. Dies wird solange fortgesetzt, bis sich als Quotient 0 ergibt.

Beispiel (Dezimalzahl 44039):

$$\begin{array}{rcrcrcr} 44039 & = & 20 & * & 2201 & + & 19 \\ 2201 & = & 20 & * & 110 & + & 1 \\ 110 & = & 20 & * & 5 & + & 10 \\ 5 & = & 20 & * & 0 & + & 5 \end{array}$$

Für die Ziffern der Maya-Zahl, beginnend bei der niedrigsten Stelle, ergeben sich in diesem Fall also die Werte 19, 1, 10 und 5 (s. auch unter Beispiele).

Die Ausgabe einer Maya-Zahl auf dem Bildschirm ist, wie in der Aufgabenstellung gefordert, so strukturiert, dass sich die Darstellung übergeordneter Objekte aus kleineren Teilausgaben zusammensetzt. Für die Ausgabe einer Ziffer gibt es zwei Fälle. Ist die Ziffer größer als 0, setzt sie sich aus einer oder mehreren Zeilen zusammen, die die Maya-Symbole Punkt und Strich enthalten. Die Anzahl der darzustellenden Punkte ist dabei der Rest bei Division der Ziffer durch 5 (Ziffer mod 5) und die Anzahl der darzustellenden Striche der ganzzahlige Quotient bei Division der Ziffer durch 5 (Ziffer div 5). Die Punkte werden, falls überhaupt welche benötigt werden, in der obersten Zeile nebeneinander dargestellt, die Striche untereinander in den folgenden Zeilen (einer pro Zeile).

Umsetzung

Die komplette Maya-Zahl wird ausgegeben, indem nacheinander von oben nach unten die einzelnen Ziffern der Maya-Zahl mit festgelegtem y-Offset nach jeder Ziffer ausgegeben werden. Die Zeilen werden, analog zum Vorgehen bei den Ziffern, nacheinander von oben nach unten mit festgelegtem y-Offset nach jeder Zeile ausgegeben. Ist die Ziffer gleich 0, dann wird einfach nur einmal das Maya-Symbol Muschel ausgegeben, mit gleich großem anschließendem y-Offset wie bei der Ausgabe einer Zeile, um die Abstände konsistent zu halten.

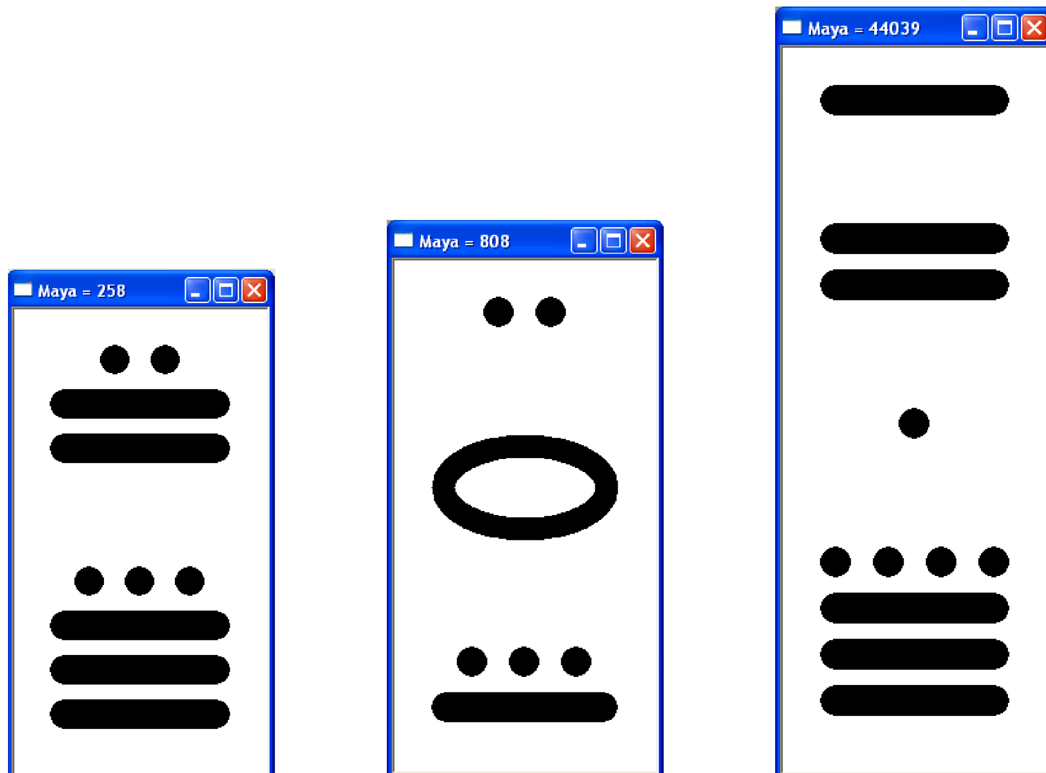
Für die Ausgabe der elementaren Maya-Symbole Punkt (als gefüllte, kreisförmige Ellipse), Strich (als gefülltes Rechteck mit abgerundeten Ecken) und Muschel (als nicht gefüllte Ellipse mit dickem Rand) werden elementare Zeichenoperationen z.B. des Windows-GDI verwendet. Es wäre jedoch auch beispielsweise eine Ausgabe als ASCII-Grafik denkbar, die grundsätzlich

genauso strukturiert sein könnte wie die oben beschriebene. Eine ASCII-Ausgabe ist zwar deutlich einfacher zu realisieren, ist aber akzeptabel, solange sich um eine gewisse Ästhetik bemüht wird, insbesondere um geeignete Abstände zwischen den Ziffern und die Zentrierung der Punktzeilen.

Die Darstellung einer Maya-Zahl kann somit durch folgende Größenattribute vollständig charakterisiert werden: Breite der Zahl, Höhe einer Zeile, Höhe einer Muschel, Dicke des Randes einer Muschel und y-Offsets nach Ausgabe von Zeilen/Muscheln und Ziffern. Von diesen Werten wird nur die Breite direkt angegeben, alle anderen Werte werden durch Multiplikation der Breite mit konstanten Faktoren ermittelt, um eine proportional einheitliche Darstellung zu erreichen. Der Durchmesser der Punkte sowie die Höhe der Striche sind stets so groß wie die Höhe einer Zeile. Die Breite der Striche und Muscheln stimmt mit der Breite der Maya-Zahl überein.

Beispiele

Hier drei grafische Darstellungen der Maya-Zahlen für 258, 808 und 44039:



Bewertungskriterien

- Die umzurechnende Dezimalzahl muss in geeigneter Weise vom Benutzer eingegeben werden können. Eventuelle (durchaus akzeptable) Einschränkungen, z.B. bzgl. der Größe der verarbeitbaren Zahl, müssen bei der Eingabe geprüft werden. Das Programm muss Eingabewerte verarbeiten können, die zu dreistelligen Maya-Zahlen führen.

- Die Umrechnung von Dezimalzahlen in Maya-Zahlen (Basis 20) muss korrekt sein.
- Auch muss korrekt berechnet werden, mit welchen Symbolen und in welcher Anordnung die Darstellung der einzelnen Maya-Ziffern erfolgen muss.
- Die Maya-Zahlen müssen erkennbar und korrekt gemäß den in der Aufgabenstellung angegebenen Symbolen grafisch dargestellt werden. Eine ASCII-Darstellung ist akzeptabel, sollte aber die oben genannten Kriterien erfüllen.
- Die Darstellung muss so erfolgen, dass die Grundsymbole Punkt, Strich und Muschel dynamisch zu Maya-Zahlen zusammengesetzt werden. Eine Lösung, die fertige Bilder für alle 19 möglichen Ziffern verwendet, ist zu einfach. Akzeptabel ist die Verwendung vorbereiteter Graphiken für die vier möglichen Punktzeilen; bei einer ASCII-Darstellung sollten aber auch die Punktzeilen aus einer Darstellung des Punktes dynamisch zusammengesetzt werden.

Aus den Einsendungen: Perlen der Informatik

Allgemeines

Worte des Wettbewerbs: back dragging, if-Anweißung

Dank der klar formulierten Aufgabenstellung ist das Programm auch ohne besondere Ideen zu realisieren.

Dieses Problem kann auf eine eher „russische Methode“ gelöst werden.

Mit einfacher Implantierung einer grafischen Benutzeroberfläche …

Anschließend beginnt eine Endlosschleife, die erst dann wieder abbricht, wenn …

No animals were harmed in the making of this program.

Die Laufzeit des Algorithmus liegt im selben Kalkül.

Aus unauffindbaren Gründen war mir allerdings dieser allgemeine Ansatz entfallen.

Bei manchem weiß ich ehrlich gesagt selber nicht mehr auf den ersten Blick, was ich mir dabei gedacht habe.

Also blieb mir nichts anderes übrig, als die traditionelle Informatik-kritische Aufgabe und die Aufgabe (Paderbox), die Spaß macht, zu lösen.

… für all die Rhectscrheifbehler, den Grammatik und und eventuelle Wiederholungen möchten wir uns […] kräftig entschuldigen!

Technisches

Die Umsetzung des Ganzen erfolgte in Perl. Dem sich gerade erbrechenden Korrektor sei versichert, dass es so schlimm nicht ist.

Wenn sich der Benutzer sehnlichst wünscht, den Anblick dieser hässlichen Benutzeroberfläche nicht länger ertragen zu müssen, hat der die Möglichkeit, ganz getarnt auf jenen gewissen Button zu klicken, und schon verschwindet das Fenster wie von der Luft verschluckt.

Das Programm wurde mit Turbo Pascal realisiert, die gruselige Grafik stellt somit ein Feature, keinen Mangel dar.

Nun erscheint der Schriftzug „Das Spiel wird geladen…“, der auch für ziemlich lange bleibt. Dies ist der Fall, da … die von mir verwendete Funktion zur Zufallszahlengeneration etwas Zeit braucht, bis sie einen neuen Wert ausspuckt.
(Im Quelltext: `int zufall(int max) { sleep(2000); ... }`)

Närrische Wirtschaft

Die Lösung des P=NP Problems ist bestimmt nicht Aufgabe in der ersten Runde des BWINF, deshalb habe ich mich für die grobe Näherung entschieden.

Selbstverständlich müssen nicht alle Möglichkeiten ausprobiert werden (Brute Force), weil das unnötig lange dauert und einen bekanntlich bei den Korrektoren unbeliebt macht.

Man müsste sich nach einem zeitlich effizienteren Programm umsehen!!!

In der Klasse 'Verein' werden alle Klassen vereint.

Dieser Algorithmus könnte in bestimmten Fällen von dynamischer Programmierung profitieren, das habe ich der Einfachheit halber aber nicht implementiert. *Schade auch!*

Theoretisch steht zwar für jeden Schritt ein Monat Berechnungszeit zur Verfügung ...

Zu dieser Lösung ist noch zu sagen, dass der Konsum derart großer Mengen Eis zu dauerhaften Gesundheitsschäden führen kann und daher nicht empfohlen wird.

Robot Dressing

Gegenständer

Zwei Gegenstände: Huhn, Ei. Reihenfolgebedingungen: Ei vor Huhn, Huhn vor Ei. Ergebnis: nicht möglich.

Es ist ein schwerer Bekleidungsfehler aufgetreten.

Die Beziehungen zwischen den Kleidungsstücken sind relativ arg ineinander verstrickt.

Nun stellt man die Strümpfe vor die Leggins und stellt fest, so klappt das nicht.

Außerdem hätte ich es viel sinnvoller gefunden, wenn ich ein Programm schreiben sollte, das Elba hilft, sich auszuziehen statt anzuziehen. *So spricht ein echter Technikliebhaber.*

Achtung: Das Programm wurde für einen fortschrittlichen Roboter mit einem fortschrittlichen Prozessor geschrieben.

... zwar ist es für Zuschauer sicherlich interessant, dabei zuzusehen, wie sich der Roboter in allen möglichen Kombinationen an- und wieder auszieht, dennoch ist es für ihn selbst weitaus praktischer, die erstbeste mögliche Kombination zu wählen.

Sollen also als einzige Regeln „Hose vor Schuhe" und „Bluse vor Schuhe" gelten, lautet die Eingabe: `((tshirt pullover) (pullover jacke))`

HTML-Mobiles

Da das Programm nur nach kurzen Syntaxen sucht, ...

Supermarkt

Kundenkartenbesitzerlistenklasse

’name’ ist, wie der Name schon sagt, der Name.

Es veranlasst zudem denkende Menschen dazu, Kundenfreund zu meiden. Aber zum Glück gibt es davon nicht so viele.

Meiner Meinung nach ist dieses Vorgehen jedoch aus Sicht eines datenschutzsensibilisierten Menschen höchst verachtenswert.

… wird der Säuferalarm ausgelöst, und der Kunde erhält ein nettes Werbeprospekt der anonymen Alkoholiker.

zu Geschäftsfall 4: Ganz einfach bewertet: Ich finde ihn toll!

Die Paderbox

Idee war bei dieser Aufgabe keine erforderlich.

Weil Zustand so ein abstraktes Wort ist, habe ich es durch „Taster“ ersetzt.

Jeder Zustand kann zwei Zustände haben. *Seufz!*

Die Zustände stellen Knotenpunkte der enthaltenen Informationen dar, die das Programm leiten.

Mein Programm soll die geheime Programmierung verdeutlichen.

Sobald der erste Zustand ’1’ ist und genau ein weiterer Zustand auch ’1’ ist, dann wird der eine Zustand mit ’0’ auch eins und genau einer der Zustände ’1’ wird ’0’, der andere bleibt ’1’.

Die Zustände werden nach den Daten der geheimen Programmierung moduliert.

Maya-Zahlen

In Echtzeit wird an der rechten Fensterseite die Maya-Zahl ausgegeben. *Wow!*