24. 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:

- Beinahe das Wichtigste ist, die Teilnahmeformulare vollständig und leserlich auszufüllen und für jedes Gruppenmitglied auch wirklich ein eigenes, vollständig ausgefülltes Formular abzugeben. Zu unnötig langen Suchprozessen führt es, wenn die Formulare zwischen den Aufgaben versteckt sind oder Teilnehmer ihre Namen nicht zumindest auf die erste Seite der Lösung schreiben – und zwar bei jeder Aufgabe!
- Online-Anmelder wurden gebeten, ihre Nummer außen auf den Umschlag zu schreiben. Die meisten haben das gemacht, aber leider nicht alle. Das führt zu Komplikationen und möglicherweise zum Ausbleiben der versprochenen Rückmeldung per E-mail über den Eingang der Einsendung.
- Was uns auch hilft: 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 wenig Beispiele und erst recht die Nichtbearbeitung vorgegebener Beispiele führten zu Punktabzug.
- 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 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.
- 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.

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 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 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.

Wir wollen zusätzlich zu diesen Lösungsideen ausführlichere Beispiellösungen erstellen, die auf den Webseiten des Bundeswettbewerbs Informatik (`www.bwinf.de`) veröffentlicht werden – allerdings wohl erst im ersten Viertel des nächsten Jahres. Bis dahin kann man sich mit den Einsendungen befassen, die unter `www.tobias-thierer.de/bwifiles.html` von einigen Teilnehmern veröffentlicht wurden.

Aufgabe 1: Favorites First

Lösungsidee

Zunächst einmal lässt sich feststellen, dass sich die Aufgabe nur mit der Anordnung und dem Auswählen von Musikstücken beschäftigt. Insbesondere werden keine Funktionen zum Einlesen oder Abspielen realer MP3-Dateien gefordert. Daher kann man sich im Folgenden darauf beschränken, die einzelnen Stücke als abstrakte Datensätze darzustellen, die über eine ID identifiziert werden können.

Teilaufgabe 1

Das Umordnen von Elementen einer Menge zu einer neuen Reihenfolge nennt man in der Mathematik eine Permutation. In dieser Teilaufgabe soll also eine zufällige Permutation der Menge der vorhandenen Stücke erzeugt werden. Dazu muss das Programm „zufällige“ Entscheidungen treffen können. Die Basis dafür bildet ein (Pseudo-) Zufallszahlengenerator, wie ihn etwa die C-Standardbibliothek mit der Funktion rand() zur Verfügung stellt. Mit den erzeugten Zufallszahlen lässt sich für jede Position in der Permutation ein noch nicht verwendetes Element auswählen. Erzeugt $\mathrm{Rand}(a, b)$ eine zufällige Ganzzahl aus dem Intervall $[a..b]$, so kann man die N Elemente eines Feldes S folgendermaßen „in-place“ (d.h. ohne Verwendung zusätzlichen Speicherplatzes) permutieren:

for $i \leftarrow 1$ **to** N
    do VERTAUSCHE$(S[i], S[\mathrm{Rand}(i, N)])$

Da für jede Position mit der gleichen Wahrscheinlichkeit jedes noch verfügbare Stück ausgewählt werden kann, können auf diese Weise alle möglichen Permutationen erzeugt werden und jede Permutation ist genauso wahrscheinlich wie jede andere.

Dieses Verfahren lässt sich verallgemeinern, indem man die Schleife nur bis zu einem gegebenen $M \leq N$ laufen lässt. So kann ein Anfangsstück von M Elementen einer zufälligen Folge generiert werden, und das ist wiederum gleichwertig mit der zufälligen Auswahl von M Elementen aus einer Folge. Mit dem obigen Verfahren lässt sich also eine Funktion $\mathrm{zufallswahl}(M, S)$ realisieren.

Teilaufgabe 2

Um die Favoriten, also die zehn meistgehörten Stücke, jederzeit zu kennen, muss für jedes einzelne Stück gespeichert werden, wie oft es bisher angehört wurde. Praktischerweise wird man dem oben erwähnten Musikstück-Datensatz zusätzlich zu seiner ID noch ein Zählerfeld spendieren und die Datensätze nach diesem Feld sortieren, so dass stets die ersten zehn Einträge dieser Liste auf die Favoriten verweisen. Nun lässt sich eine „Favorites First“-konforme Abspielfolge in vier Schritten generieren:

1. Generiere eine zufällige Folge aus allen Stücken (also Favoriten und Nicht-Favoriten).
2. Wähle aus den (zehn) Favoriten fünf zufällige aus.
3. Wähle aus den ersten 20 Positionen fünf zufällige aus. Ordne jede Position einem der oben gewählten Favoriten zu.
4. Vertausche nacheinander jeden Favoriten mit dem Stück auf der zugehörigen Position. (Auf diese Weise können mehr als fünf Favoriten unter die ersten 20 Stücke gelangen, aber das wird von der Aufgabenstellung nicht verboten.)

Offensichtlich kann man für die Schritte 1 bis 3 unmittelbar auf die oben beschriebene Funktion zufallswahl zurückgreifen:

$folge \leftarrow \text{zufallswahl}(N, Stuecke)$
$favAuswahl \leftarrow \text{zufallswahl}(5, Favoriten)$
$posAuswahl \leftarrow \text{zufallswahl}(5, [1..20])$
for $i \leftarrow 1$ **to** 5
  do $p \leftarrow$ Index von $favAuswahl[i]$ in $folge$
    VERTAUSCHE$(folge[posAuswahl[i]], folge[p])$

Teilaufgabe 3

Für die Simulation einer Player-Benutzung muss zunächst wie oben beschrieben eine Abspielfolge generiert werden. Anschließend lässt man den Zufallsgenerator die Zahl der anzuspielenden Stücke $L \in \{10, \ldots, 20\}$ bestimmen und wählt mit $\text{zufallswahl}(L \operatorname{div} 4, folge[1..L])$ die Stücke aus, die übersprungen werden sollen. Nun kann man die angespielten Stücke ausgeben, für die nicht übersprungenen Stücke den Beliebtheitszähler inkrementieren und ihre Reihenfolge entsprechend anpassen.

Ablaufprotokoll

Als Beispiele werden Ablaufprotokolle eines Programms gezeigt, das über die Kommandozeile aufgerufen wird. Dabei kann die Zahl der zu simulierenden Player-Benutzungen als Argument angegeben werden. Der folgende Befehl erzeugt also 5 Abspielfolgen, die auf die Konsole ausgegeben werden.

```
$ ./favorites-first 5
19: [244] 115 [2] 148 156 [83] 184 163 162 231 [198] 9 113 5 6 118 215 27 51
11: 68 81 5 [108] 113 122 141 11 58 160 [115]
10: [231] 146 158 115 153 186 207 113 65 [5]
13: 22 [244] 113 156 249 192 [155] [115] 9 5 124 60 125
14: 83 115 116 15 [121] 231 230 [167] 27 34 102 234 [96] 211
```

Jede Zeile der Ausgabe entspricht einer Player-Benutzung. Wie man sieht, wird zunächst der aktuelle Wert von L ausgegeben, also die Zahl der angespielten Stücke, gefolgt von deren IDs, wobei übersprungene Stücke mit eckigen Klammern markiert werden. Die in eine Datei geschriebenen Statistiken sehen so aus:

```
115 148 156 184 163 162 231 9 113 5
5 113 115 148 156 184 163 162 231 9
113 5 115 148 156 184 163 162 231 9
113 5 115 156 9 148 184 163 162 231
113 5 115 156 9 231 27 148 184 163

>>> Play statistics <<<
113     4
5       3
115     3
156     2
9       2
...
```

Im ersten Teil stehen die nach jeder Abspielfolge jeweils aktuellen zehn Favoriten. Nach Ende des eigentlichen Programms wird dann für jedes Stück angegeben, wie oft es insgesamt gespielt (und nicht übersprungen) wurde. Die Liste ist nach diesem ’Play Count’ Wert sortiert.

Analyse

Bei Betrachtung der Statistik für 200 Simulationen fällt eine faktische Zweiteilung der ’Play Count’-Zahlen für die einzelnen Stücke auf:

```
>>> Play statistics <<<
133     67
1       64
2       63
7       62
165     62
54      61
78      59
3       56
168     53
100     46
62      16 <-----------
45      14
106     14
...
46      2
249     1
```

Während der ’Play Count’ der übrigen Stücke sich statistisch um etwa 7 verteilt, wurden die obersten zehn Stücke fast zehn mal so oft gespielt. Zunächst entspricht das voll und ganz Bo

Hays Ziel, dass der Nutzer seine Lieblingsstücke öfter zu hören bekommt. Möglicherweise hat er dabei aber einige Implikationen nicht bedacht. Mit der Zeit baut sich nämlich eine sehr statische Struktur auf: Die Menge der Favoriten ist schon nach sieben Abspielfolgen mit der nach 200 Folgen identisch. Tatsächlich werden verglichen mit der Situation nach der allerersten (!) Playerbenutzung nur noch zwei Favoriten ausgetauscht:

```
Z001: 3 133 7 2 78 165 100 1 142 34 sortiert: 1 2 3 7 34 78 100 133 142 165
Z002: 165 3 133 7 2 78 100 1 142 34 sortiert: 1 2 3 7 34 78 100 133 142 165
Z003: 165 3 133 7 2 78 100 1 142 34 sortiert: 1 2 3 7 34 78 100 133 142 165
Z004: 165 54 1 2 3 133 7 78 100 142 sortiert: 1 2 3 7 54 78 100 133 142 165
Z005: 165 2 54 1 78 3 133 7 100 142 sortiert: 1 2 3 7 54 78 100 133 142 165
Z006: 2 165 54 1 78 7 3 133 100 142 sortiert: 1 2 3 7 54 78 100 133 142 165
Z007: 2 165 1 3 54 78 7 133 168 100 sortiert: 1 2 3 7 54 78 100 133 165 168
...
Z200: 133 1 2 7 165 54 78 3 168 100 sortiert: 1 2 3 7 54 78 100 133 165 168
```

In der oben gezeigten Situation müsste ein Stück, das nicht zu den Favoriten gehört, zumindest 30 mal gespielt werden, um in die Favoritenliste kommen zu können. Da pro Player-Benutzung nur etwa 10 Nicht-Favoriten angespielt werden, dauert das im Mittel $30 * (240/10) = 720$ Abspielfolgen! Geht man davon aus, dass sich die Präferenzen eines Nutzers mit der Zeit verändern, weil er sich an oft gespielten Stücken satt hören kann, während er neue Hits gerne von Anfang an öfter hören würde, ist es sehr fraglich ob die „Favorites First"-Funktion den Nutzer auf Dauer so glücklich macht, wie Bo Hay sich das vorstellt. Außerdem ist es recht wahrscheinlich, dass sich längst andere Stücke (die als „ganz nett" empfunden und daher meist nicht übersprungen werden) als Favoriten etabliert haben, bevor der Benutzer sein wahres Lieblingsstück auch nur ein einziges Mal gehört hat.

Bewertungskriterien

Teilaufgabe 1 Die Funktion zur Erzeugung einer zufälligen Permutation muss korrekt beschrieben und implementiert sein.

Teilaufgabe 2 Die Spezifikation von „Favorites First" aus der Aufgabenstellung muss richtig und vollständig umgesetzt sein. Wichtig ist auch, dass die Vorgabe „möglichst einfach" berücksichtigt wurde. Zunächst könnte man bei dieser Aufgabe daran denken, in aufwändiger Weise die Benutzung eines MP3-Players realitätsnah nachzubauen. Wer dies getan hat, sollte erkannt haben, dass das nicht nötig ist.

Teilaufgabe 3 Hier muss die Simulation den Vorgaben entsprechend umgesetzt worden sein. Während und am Ende eines Simulationsdurchganges sollten Daten ausgegeben werden, die die Analyse unterstützen. Die 200 Benutzungen müssen nicht komplett protokolliert werden. Wichtig ist, dass die Angaben über den beobachteten Effekt mit aus der Simulation gewonnenen Daten belegt werden. Die Meinung zur Ursache des Effekts muss schlüssig begründet sein.

Aufgabe 2: Formel-Up

Lösungsidee

Teilaufgabe 1

Es werden alle der gegebenen Form entsprechenden Formeln durchlaufen, bewertet und die besten drei gespeichert. Die Reihenfolge der Formeln beim Durchlaufen muss dabei bestimmte Bedingungen erfüllen, damit jede Formel nach einer endlichen Laufzeit erreicht wird. Es bietet sich an, diese z.B. nach der Summe ihrer Faktoren zu sortieren, um damit die Formeln von den nach dem 3. Schönheitskriterium Schönen hin zu den weniger Schönen zu untersuchen. Es ergibt sich dabei folgende Reihenfolge von Faktorkombinationen (H, A, M):

$$(1,1,1), (2,1,1), (1,2,1), (1,1,2), (2,2,1), (2,1,2), (1,2,2) \ldots$$

Für jedes Faktortripel müssen jeweils die Operatoren $+$ und $-$ berücksichtigt werden. Damit lautet ein mögliches Verfahren:

```
Von SEHR_SCHÖNEN zu WENIGER_SCHÖNEN Faktoren tue
  Für Operator '+' und '-' tue
    Bewerte entstandene Formel
    Sichere die besten drei Formeln
Gib beste drei Formeln aus
```

Da es unendlich viele Faktorkombinationen und damit potenzielle Weltformeln gibt, kann der Algorithmus nicht alle ausprobieren. Wir umgehen dieses Problem, indem wir aus dem vorliegenden Algorithmus einen Anytime-Algorithmus machen, der vom Benutzer an einer beliebigen Stelle abgebrochen werden kann (z.B. nach einer vorher festgelegten Laufzeit). Die Qualität der ausgegebenen Lösungsformeln nimmt dann mit der Laufzeit zu, da mehr potenzielle Formeln untersucht werden können.

Am Ende werden die besten drei gefundenen Lösungen mit Hilfe des folgenden einfachen Textmusters ausgegeben:

Streicht man die Zeilen *bis zu drei Zeilennummern*, dann ist die Formel
H-Faktor * H-Wert $+/-$ *A-Faktor* * A-Wert = *M-Faktor* * M-Wert
eine schöne Formel mit Schönheit *Bewertung*,
weil die Summe ihrer absoluten Abweichungen
gegenüber der Formel dann gleich *Abweichungssumme* ist.

Teilaufgabe 4

Für die richtige Implementierung ist die Wahl der Bewertungsfunktion, die eine gegebene Formel bewertet, von äußerster Bedeutung, da möglicherweise von dieser ausgehend formal festgestellt werden kann, wann eine optimale Lösung gefunden wurde, oder einige Formeln

von vorne herein ausgeschlossen werden können. Eine mögliche Bewertungsfunktion wäre das Produkt aus der Summe der absoluten Abweichungen der M-Werte gegenüber der Formel und der Summe der Faktoren der Formel. Dabei wird die Abweichungssumme über alle Zeilen der Messwerttabelle ohne die drei mit der größten Abweichung gebildet; gibt es weniger als drei Zeilen mit Abweichung, reicht es natürlich, nur diese zu streichen. Diese Funktion beachtet offensichtlich alle Schönheitskriterien.

Allerdings werden z.B. alle Formeln ohne absolute Abweichung gleich behandelt, ohne Rücksicht auf die Größe ihrer Faktoren. Eine Funktion, die dies umgeht, wäre die Summe aus der Summe der absoluten Abweichungen (wieder ohne die größten drei) und des größten Faktors in der Formel. In diesem Fall kommt das Problem ins Spiel, die Kriterien 1 und 3 so zu wichten, dass möglichst ein Kompromiss entsteht, da sich beide ja in gewissem Maße widersprechen – mit größeren Faktoren kann eine komplizierte Formel besser angenähert werden, ihre Faktoren werden aber unschön. Für das Ablaufprotokoll (s.u.) wurde ein Programm mit der ersten Variante realisiert. Abhängig vom zu Teilaufgabe 1 gewählten Verfahren könnte auch die Anzahl der gestrichenen Tabellenzeilen in die Bewertung eingehen.

Teilaufgabe 3

Nun zu den zufälligen Messtabellen: Hier kann es zu dem Fall kommen, dass es eine Formel gibt, die die Messtabelle gut annähert. Dies wird umso unwahrscheinlicher, je mehr zufällige Tabelleneinträge vorhanden sind. Gibt es keine solche Näherungsformel, dann kommt es dazu, dass für jede Formel die absoluten Abweichungen sehr groß sind, so dass diejenigen Formeln mit kleiner Faktorensumme von der hier gewählten Bewertungsfunktion in jedem Fall bevorzugt werden.

Ablaufprotokoll

Hier ein Protokoll für das Pflichtbeispiel:

```
>perl formelUp.pl test.txt
Lese Messwerttabelle ein...
Wie lange soll gesucht werden (in Sekunden)? 3
Beginne Suche...
Die drei besten Loesungen:

Streicht man die Zeilen 1, 4, 7, dann ist die Formel
1 * H-Wert + 1 * A-Wert = 2 * M-Wert
eine schöne Formel mit Schönheit 0.00, weil die Summe ihrer
absoluten Abweichungen gegenüber der Formel dann gleich 0.00 ist.

Streicht man die Zeilen 1, 4, 7, dann ist die Formel
9 * H-Wert + 10 * A-Wert = 19 * M-Wert
eine schöne Formel mit Schönheit 14.00, weil die Summe ihrer
absoluten Abweichungen gegenüber der Formel dann gleich 0.37 ist.
```

```
Streicht man die Zeilen 1, 4, 7, dann ist die Formel
5 * H-Wert + 6 * A-Wert = 11 * M-Wert
eine schöne Formel mit Schönheit 14.00, weil die Summe ihrer
absoluten Abweichungen gegenüber der Formel dann gleich 0.64 ist.
```

Bewertungskriterien

Bei der Lösung der Aufgabe wird man sich nicht unbedingt an die Reihenfolge der Teilaufgaben halten, deshalb sind auch die Bewertungskriterien anders sortiert:

- Entscheidend ist das Abbruchkriterium des Formulierers. Bei etlichen Kompromissen für „schön“ lässt sich formal entscheiden: „Zu den bisher gefundenen drei schönsten Formeln kann keine schönere mehr kommen“. Wir akzeptieren aber grundsätzlich auch Ressourcen-bezogene Abbruchkriterien (Laufzeit, Anzahl bisher untersuchter Formeln, …). Eine Begründung des Abbruchkriteriums ist nicht erforderlich; Hauptsache, es kommen die drei schönsten der bis dahin deutlich mehr als drei untersuchten Formeln heraus.
- Die Gewichtung der Kriterien in einer Bewertungsfunktion sollte sinnvoll sein. Eine zu starke Gewichtung einer möglichst geringen Abweichung kann je nach Formel-Suchverfahren z.B. dazu führen, dass die Faktoren der drei schönsten Formeln nur Vielfache eines Basistripels sind (Beispiel: 1*H+1*A=2*M; 2*H+2*A=4*M; etc.)
- Alternativen zur gewählten Bewertungsfunktion müssen diskutiert werden.
- Die Textbausteine können ruhig sehr knapp ausfallen, aber eine Aussage zur Bewertung der Formel soll enthalten sein.
- Das Pflichtbeispiel und vier weitere Beispiele sind zu bearbeiten. Die Aussage zu Teilaufgabe 3 (Zufallswerte) sollte mit einem Beispiel belegt sein; das kann eines der vier weiteren Beispiele sein. Eigene Beispiele sollten immer mehr als 6 Zeilen lang sein. Für kürzere Tabellen lässt sich, nach Streichung von drei Zeilen, nämlich immer eine abweichungsfreie Lösung berechnen.

Bei den allermeisten Bearbeitungen wurde bei der Berechnung der Abweichungen nicht durch den Faktor des M-Wertes geteilt – das fällt auf, weil die Abweichungswerte dann immer ganzzahlig sind. Dieser Fehler wurde bei der Bewertung ignoriert.

Aufgabe 3: Zaras Zauberfolie

Lösungsidee

Verschlüsselungsalgoritmus

In der Aufgabe geht es darum, geheime Botschaften zu entschlüsseln. Zur Verschlüsselung wird ein Verfahren benutzt, dessen Grundidee unter dem Schlagwort „Visuelle Kryptographie" bekannt ist. Tatsächlich lässt sich ein Schwarz-Weiß-Bild ohne großen Rechenaufwand verschlüsseln. Benötigt wird dafür zusätzlich zum Originalbild eine Transparenz-Folie derselben Größe. Im folgenden wollen wir diese Transparenz-Folie Originalfolie nennen. Diese wird zunächst in Rasterquadrate aufgeteilt. Jedes Rasterquadrat kann auf zwei verschiedene Weisen gefüllt werden, mit Form A: ▚ oder Form B: ▞. Die Rasterquadrate auf der Originalfolie werden zufällig mit Form A oder B gefüllt.

Abbildung 1: Verschlüsselungsverfahren

Zur Verschlüsselung unseres Bildes stellen wir jedes Pixel ebenfalls durch ein Rasterquadrat dar. Wir wollen eine Faxnachricht gestalten, die die geheime Botschaft enthält. Dazu betrachten wir für jedes Rasterquadrat die Farbe. Ist es weiß, wird das Rasterquadrat auf dem Fax mit der gleichen Form gefüllt wie auf der Folie. Ist es schwarz, füllen wir es auf dem Fax mit der jeweils anderen Form. Dadurch erreichen wir, dass beim Übereinanderlegen der Originalfolie und der Faxnachricht nur die Rasterquadrate ganz schwarz sind, deren zugehörige Pixel in unserem Bild schwarz waren. Die anderen Rasterquadrate sind nur zur Hälfte schwarz und bilden damit den Hintergrund. Man kann also die ursprüngliche Nachricht wieder erkennen, obwohl die Rasterquadrate auf dem Fax „rein zufällig" aussehen.
Dieses Verfahren ist jedoch nur sicher, solange anhand einer geheimen Originalfolie nur eine einzige Botschaft verschlüsselt wird. Im nächsten Abschnitt wird untersucht, wie man mit mehreren verschlüsselten Faxen die einzelnen Botschaften zurückgewinnen kann, ohne die Originalfolie zu kennen.

Grundannahme

Um die Entschlüsselung der abgefangenen Faxe möglich zu machen, müssen wir unser Wissen über den Inhalt möglichst gut ausnutzen. Wären alle Botschaften zufällig generierte Bilder, so wären auch die verschlüsselten Faxe im Prinzip zufällige Folgen von Rasterquadraten. So etwas kann man ohne die Originalfolie nicht entschlüsseln.

Eine recht häufige Entschlüsselungsmethode für viele andere bekannte Verschlüsselungsverfahren ist die Häufigkeitsanalyse. Dabei nutzt man aus, dass die verschlüsselten Inhalte bestimmte Buchstaben oder Ähnliches häufiger enthalten als andere. In unserem Fall können wir eine recht einfache Annahme verwenden. Weil wir wissen, dass die verschlüsselten Botschaften ursprünglich Nachrichten von Zaras Firma sind, also wahrscheinlich Text, können wir vermuten, dass der größte Teil der Bilder die Hintergrundfarbe darstellt und die eigentliche Nachricht nur einen geringen Anteil hat. Wir gehen also davon aus, dass eine der beiden Farben schwarz oder weiß in dem Ursprungsbild viel häufiger ist als die andere Farbe (üblicherweise weiß, bei schwarzem Text auf weißem Hintergrund).

Entschlüsselung per Hand

Mit dieser Idee kann man die Faxe sogar ganz ohne PC entschlüsseln: Man legt einfach zwei verschlüsselte Botschaften übereinander (und hält sie dann vor ein helles Licht). Laut unserer Annahme werden dabei in den meisten Fällen Rasterquadrate übereinanderliegen, die beide die Hintergrundfarbe kodieren (üblicherweise weiß). In diesem Fall sieht das Rasterquadrat beim Übereinanderlegen „weiß" aus. Die beiden Bilder, die in den Faxen verschlüsselt sind, nennen wir Originalbilder. In diesem ersten Fall waren also die Pixel in beiden Originalbildern in der Hintergrundfarbe gefärbt. Hat ein bestimmtes Pixel hingegen in den beiden Originalbilder eine unterschiedliche Farbe, so wird es in den Faxen jeweils durch Rasterquadrate einer anderen Form dargestellt. Durch das Übereinanderlegen der beiden Faxe sieht man dieses Pixel als schwarzes Rasterquadrat. Dadurch sieht man beide Botschaften überlagert.

Schlecht ist nur der Fall, in dem ein bestimmtes Pixel in beiden Originalbildern mit der Farbe gefüllt war, die nicht die Hintergrundfarbe ist (also üblicherweise mit schwarz). In diesem Fall ist es in beiden Faxen durch die gleiche Form dargestellt. Dadurch erscheint das zugehörige Rasterquadrat beim Übereinanderlegen fälschlicherweise „weiß" zu sein, obwohl das Pixel in beiden Originalbildern zur Nachricht und nicht zum Hintergrund gehörte. Wir nehmen jedoch an, dass dieser Fall nur selten auftritt. An den Bildern in Abbildung 2 sieht man, dass man zwei übereinandergelegten Beispielfaxe aus der Aufgabe bereits gut lesen kann.

Abbildung 2: Überlagerungen der Folien 4 und 6 bzw 5 und 6. Man erkennt, dass die Botschaft auf Folie 6 „Preis ist zu hoch." lauten muss.

Entschlüsselungsalgorithmus

Natürlich kann man auch einen Algorithmus beschreiben, der unsere Grundannahme umsetzt. Im letzten Abschnitt haben wir zum „Entschlüsseln" jeweils nur zwei Faxe benutzt, da beim

Übereinanderlegen von noch mehr Nachrichten die Lesbarkeit eher abnimmt. Bei einem automatisierten Vorgehen können wir ohne Probleme alle uns zur Verfügung stehenden Faxe auswerten. Dazu lesen wir die abgefangenen Faxe ein und speichern, welche Rasterquadrate dort mit welcher Form belegt sind. Anschließend zählen wir für jedes Rasterquadrat, in wie vielen Faxen an dieser Stelle Form A, und wie oft Form B vorkommt. Den häufigeren der beiden Zustände interpretieren wir dann als Hintergrundfarbe (bzw. ,,weiß"), d.h. wir gehen davon aus, dass diese Form auf der Originalfolie in dem Rasterquadrat steht. So erzeugen wir eine nachgeahmte Originalfolie. Mit dieser entschlüsseln wir dann die einzelnen Faxe.

Bisher haben wir immer ganze Formen (A oder B) und nicht einzelne Pixel betrachtet. Man kann sich überlegen, dass es auch möglich ist, einen einfachen Entschlüsselungsalgorithmus zu entwerfen, der für einzelne Pixel statt ganzer Quadrate die Farben festlegt. Dazu zählt man für jedes Pixel einzeln, wie oft dieses schwarz oder weiß ist. Ist ein Pixel in der Mehrzahl der Faxe schwarz, geht man davon aus, dass dieses Pixel in der Originalfolie auch schwarz ist, analog für weiß. Dadurch erhält man ebenfalls eine wahrscheinlich gut rekonstruierte Originalfolie. Ein einfaches Programm dieser Bauart hat dann z.B. folgende Struktur:

```
for jedes Pixel
    do Zähle, in wie vielen Faxen dieses Pixel schwarz (= 1) ist
       for jedes Fax                    ▷ vermutete Folie und Fax über einander legen
           do if Pixel in der Mehrzahl der Faxe schwarz
                 ODER Pixel im Fax schwarz
                 then Pixel im Ergebnis schwarz
                 else  Pixel im Ergebnis weiß
```

Grafische Darstellung der Ergebnisse

Für die Darstellung gibt es mehrere Möglichkeiten. Zunächst einmal kann man ein Bild erzeugen, dass tatsächlich dem Übereinanderlegen von nachgestellter Transparenz-Folie und Fax entspricht, d.h. es gibt ,,halbschwarze" und "schwarze" Rasterquadrate. Dieses erreicht man, indem man die Rasterquadrate auf Bild und rekonstruierter Originalfolie mit einem logischen Oder verknüpft. Etwas übersichtlicher ist es, die ,,weißen" Rasterquadrate tatsächlich weiß darzustellen, so dass der Kontrast der eigentlichen Botschaft zum Hintergrund größer ist. Hierfür benutzt man die logische Operation XOR, d.h. im Ergebnis wird eine 1 (bzw. schwarz) angezeigt, wenn die beiden Rasterquadrate auf Originalfolie und Bild unterschiedlich sind, und eine 0, wenn beide gleich sind.

Es gibt noch eine dritte, fehlertolerantere Darstellungsform. Bei unserem Entschlüsselungsverfahren kommt es natürlich auch zu Fehlern, d.h. Rasterquadrate auf der Transparenz-Folie werden falsch belegt. Kommt ein Rasterquadrat auf den Faxen ungefähr gleich häufig mit Form A und Form B vor, können wir kaum sicher sein, dass wir die richtige Belegung wählen. Um dieser Problematik Rechnung zu tragen, kann man das Ausgabebild auch in Graustufen anzeigen. Dabei gibt man einem Rasterquadrat eine Farbe im Bereich von weiß bis schwarz, die der Wahrscheinlichkeit entspricht, dass wir das Rasterquadrat auf der Transparenzfolie

richtig gesetzt haben. Dadurch erhält man eine Darstellung, in der falsch gesetzte Rasterquadrate häufig weniger auffallen.

Alle drei Möglichkeiten lassen sich auch auf Pixelebene realisieren.

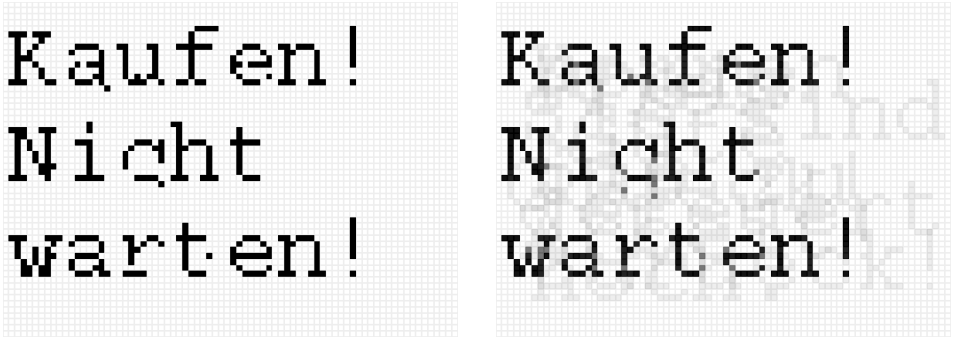

Abbildung 3: Rekonstruktion von Nachricht 1 in zwei Darstellungsformen

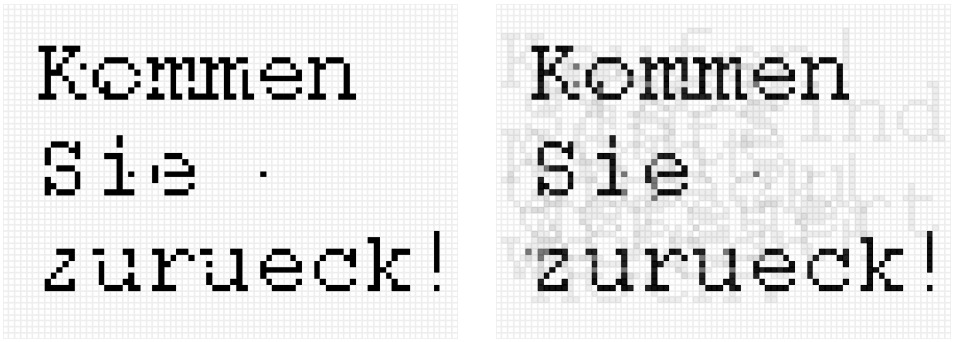

Abbildung 4: Rekonstruktion von Nachricht 2 in zwei Darstellungsformen

Grenzen dieses Ansatzes

Wir haben gesehen, dass sich die verschlüsselten Bilder mit unserer Annahme gut entschlüsseln lassen. Trotzdem handelt es sich natürlich um eine Annahme. Zaras Firma könnte es uns deutlich schwieriger machen, indem sie z.B. die Hälfte der Originalbilder als schwarze Bilder mit weißer Farbe erstellt. Eine andere Möglichkeit, die Entschlüsselung zu erschweren, besteht darin, den Text immer an der gleichen Stelle zu platzieren, im Gegensatz zu den wechselnden Positionen in unseren Faxen. Dadurch verschiebt sich in dem Textbereich die Häufigkeit von schwarzen und weißen Pixeln im Vergleich zum restlichen Bild. Allgemein kann man die Entschlüsselung erschweren, indem man die Originalbilder so verändert, dass alle vorkommenden Farben mit gleicher Häufigkeit auftreten, so dass die Grundannahme falsch wird.

Bewertungskriterien

Teilaufgabe 1

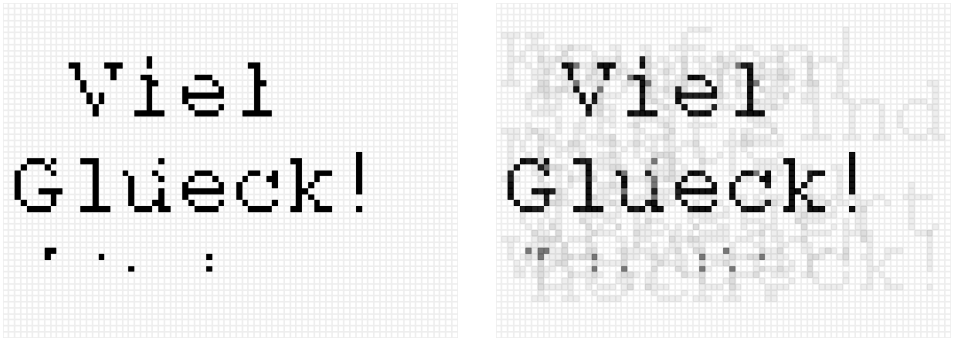

Abbildung 5: Rekonstruktion von Nachricht 3 in zwei Darstellungsformen

Abbildung 6: Rekonstruktion von Nachricht 4 in zwei Darstellungsformen

- Es muss erkannt werden, dass das Verfahren auf einer Annahme beruht. Dabei ist es nicht so wichtig, dass lang und breit erklärt wird, in welchen Fällen der Algorithmus nicht funktioniert. Das Wort „Annahme" oder etwas Ähnliches sollte aber doch auftauchen.
- Es muss erklärt werden, warum das Verfahren funktioniert.

Teilaufgabe 2

- Wenn nicht alle Faxe zur Entschlüsselung eines Faxes benutzt werden, sollte das als Nachteil erkannt werden.
- Das Programm muss die Originaldateien einlesen können.
- Es muss eine ansehbare Ausgabe (auf dem Bildschirm oder in einer Bilddatei) produzieren.
- In der Ausgabe sollte der Faxinhalt gut lesbar sein. Dies kann evtl. nicht festgestellt werden, nämlich wenn gar kein Ergebnisfax abgedruckt ist.

Teilaufgabe 3

- Es müssen alle Beispielfaxe entschlüsselt bzw. deren Inhaltstexte angegeben werden.

Abbildung 7: Rekonstruktion von Nachricht 5 in zwei Darstellungsformen

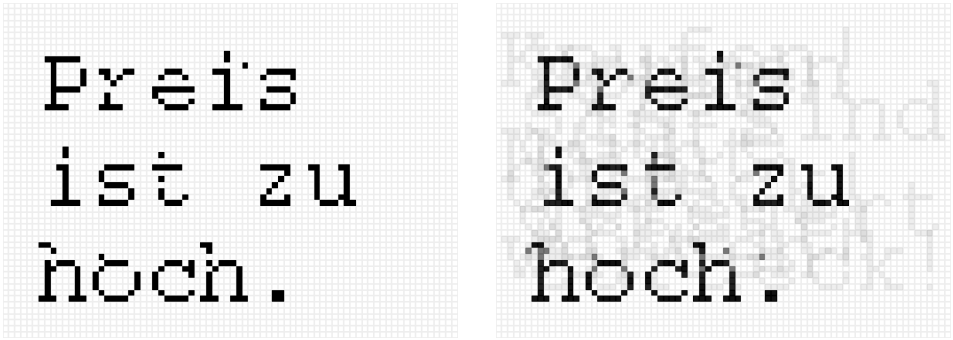

Abbildung 8: Rekonstruktion von Nachricht 6 in zwei Darstellungsformen

Aufgabe 4: Fehlzeitendatenbank

Teilaufgabe 1

Es soll ein „sinnvolles Verfahren für die Abwicklung von Entschuldigungen für Fehlzeiten“ beschrieben und dessen Praxistauglichkeit beurteilt werden.

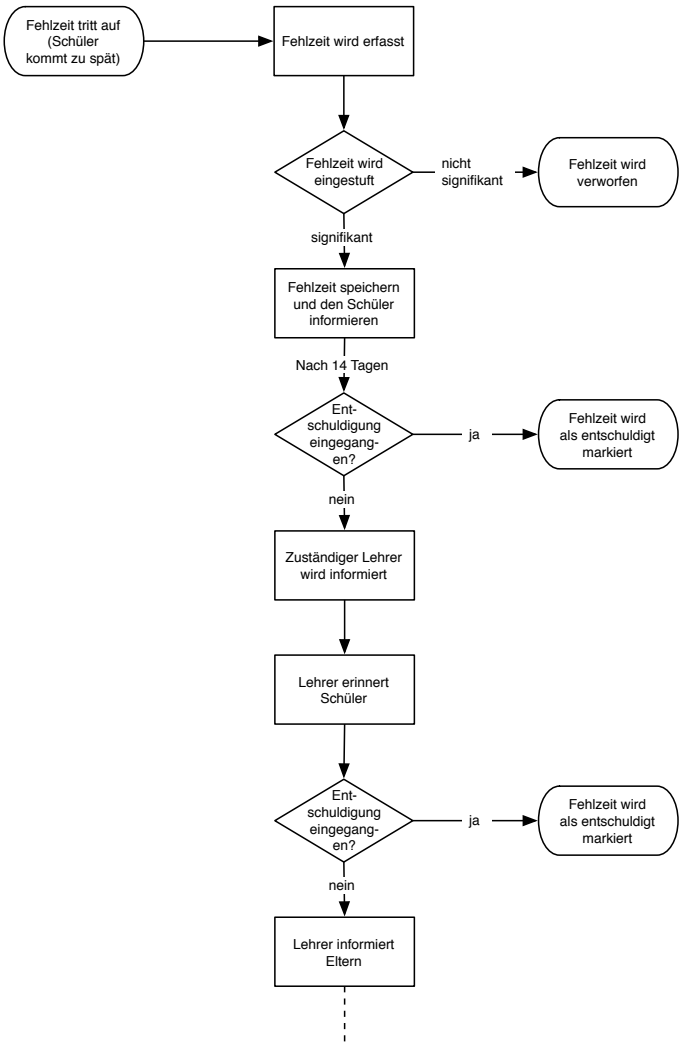

Abbildung 9: Ein möglicher Prozess zur Verwaltung von Fehlzeiten

Dieses Verfahren ist im Prinzip ein Prozess, der beim Auftreten der Fehlzeit, also beim Zuspätkommen des Schülers beginnt und (hoffentlich) mit dem Eingehen der Entschuldigung endet. Ein möglicher Prozess wird in Abbildung 9 in groben Zügen durch ein einfaches Flussdiagramm beschrieben.

Nachdem ein Schüler nicht oder zu spät zum Unterricht erschienen ist, muss diese Fehlzeit zunächst automatisiert vom Fehlzeitensystem erfasst und in geeigneter Form abgespeichert

werden (siehe Teilaufgabe 2). Um das System in der Praxis verwendbar zu machen, sollte eine erfasste Fehlzeit danach erst einmal als signifikant bzw. nicht signifikant eingestuft werden. Nicht signifikant wäre beispielsweise ein Zuspätkommen von 7 Sekunden oder ein zu frühes Verlassen des Klassenraums, falls der Lehrer die Stunde eine Minute früher beendet hat.

Signifikante Fehlzeiten müssen dann den Schülern zur Kenntnis gebracht werden. In der Praxis wäre es etwa vorstellbar, dass die Schüler Emails erhalten oder über ein WWW-Portal informiert werden, falls man von ihnen erwarten kann, dass sie Zugriff aufs Internet haben. Alternativ könnten zum Beispiel Terminals im Foyer der Schule aufgestellt werden, an denen sich Schüler über ihre Fehlzeiten informieren können. Oder man könnte einen klassischeren Weg gehen und allmorgendlich die Klassenlehrer mit Fehlstunden-Listen versorgen, so dass sie ihre Schüler direkt über die noch zu entschuldigenden Fehlstunden informieren können.

Weiterhin sollen die Eltern der Schüler die Fehlzeiten ihrer Kinder per Internet einsehen können. Dazu müsste ein WWW-Portal geschaffen werden, das (wie auch die Schülerterminals oder das Schülerportal) an die Fehlzeitendatenbank angebunden ist, um die Fehlzeiten zu präsentieren.

Nach einer angemessenen Zeit, etwa nach 14 Tagen, sollte dann überprüft werden, ob die Fehlzeit entschuldigt wurde. Das kann in verschiedenen Automatisierungsgraden geschehen. Das Überwachungssystem könnte beispielsweise nicht nur Fehlzeiten, sondern auch Entschuldigungen verwalten. Dann müssten die Lehrer bei Erhalt einer Entschuldigung diese in das System einpflegen. Oder das System könnte nach 14 Tagen den zuständigen Lehrer dazu auffordern, zu überprüfen, ob die Entschuldigung inzwischen eingegangen ist.

Ist eine Entschuldigung fristgerecht eingegangen, ist der Prozess abgeschlossen. Falls die Entschuldigungen automatisiert verwaltet werden, wird die Fehlzeit dann im System gestrichen oder als entschuldigt markiert. Ist die Entschuldigung jedoch nicht eingegangen, muss spätestens jetzt (falls das System bis hier hin in der Lage war, vollautomatisch zu agieren) ein Lehrer informiert werden, der den Schüler ermahnen oder die Eltern des Schülers informieren kann.

Egal wie umfangreich der Prozess wird, kann man sich in jedem Teil des Prozesses die Frage nach dem Automatisierungsgrad stellen. Dass das Fehlzeitensystem eigenständig die Polizei ruft, wenn eine gewisse Toleranzschwelle überschritten ist, ist beispielsweise eher nicht wünschenswert, dass das System wiederholtes Zuspätkommen aufzeigt oder den ersten Teil (in Abbildung 9 zum Beispiel bis zur „Entschuldigung eingegangen?“ Entscheidung) des Entschuldigungsprozesses vollautomatisch abwickelt, hingegen schon.

Gewisse Dinge kann das Fehlzeitensystem den Lehrern natürlich nicht abnehmen, z.B. das Einpflegen von eingegangenen Entschuldigungen. Auch ist es bei einem hohen Automatisierungsgrad sehr wichtig, dass das System ständig „up to date“ ist. Entscheidet sich ein Lehrer etwa für eine Raumänderung („Heute brauche ich den Beamer in Raum 101.“), ist es wichtig, dass er das dem System mitteilt, damit nicht fälschlicher Weise der gesamten Klasse Nichtanwesenheit im Unterricht vorgeworfen wird, da sie sich nicht im vom System erwarteten Raum aufgehalten hat.

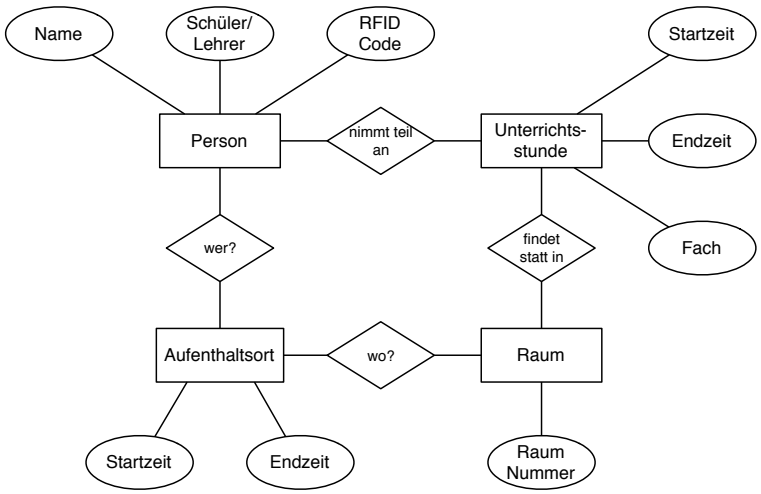

Abbildung 10: Ein einfaches ERM

Für die Praxistauglichkeit des Systems ist sicher die Akzeptanz ein kritischer Punkt. Wird das System von den Lehrern nicht akzeptiert, kann es nicht funktionieren. Denn die Lehrer müssen stets fleißig die eingegangenen Entschuldigungen verarbeiten und den Stundenplan im System auf dem neuesten Stand halten. Aus diesem Grund, aber auch auf Grund der „Gründlichkeit“ des Systems stellt es eine Mehrbelastung für die Lehrer dar. Konnten die Lehrer bisher Zuspätkommen und Fehlen von Schülern nach eigenem Ermessen behandeln, müssen sie jetzt nach einem genauen Schema vorgehen, um solche Vorfälle abzuwickeln. Da die Lehrer zudem noch selbst durch das System überwacht werden sollen (und das ziemlich unbequem für sie ist), ist in der Tat fraglich, ob sie es unterstützen werden.

Der letzte Punkt wirft weitere Fragen auf: Auch Fehlzeiten von Lehrern sollen behandelt werden. Es ist nicht sinnvoll, dass diese ihre eigenen Entschuldigen komplett selbst bearbeiten. Hier ist die Schulleitung oder zumindest das Schulsekretariat einzubinden; dies dürfte in der Praxis zu weiteren Problemen führen, auch zwischenmenschlicher Art.

Teilaufgabe 2

Es soll ein Entity-Relationship-Modell (ERM) für eine Datenbank, die die Abfrage der Fehlzeiten ermöglicht, erstellt werden. In dieser Teilaufgabe gibt es ein sehr breites Spektrum von „richtigen“ Lösungen. Wichtig ist jedoch, dass das Modell die Speicherung der Informationen vorsieht, die für das Abfragen von Fehlzeiten benötigt werden. Außerdem sollte beschrieben werden, wie das Abfragen von Fehlzeiten vor sich geht.

Denkbar sind zum Beispiel relativ einfache Modelle, wie das in Abbildung 10. Die überwachten Personen, die Räumlichkeiten, in denen sie überwacht werden und die Unterrichtsstunden (deretwegen sie überwacht werden) sind hier abgebildet, der eigentliche Überwachungsprozess ist allerdings stark vereinfacht. Das RFID-Überwachungssystem liefert nämlich nicht fertige Daten nach dem Schema *Person X war von 12.04 Uhr bis 13.15 Uhr in Raum 101*.

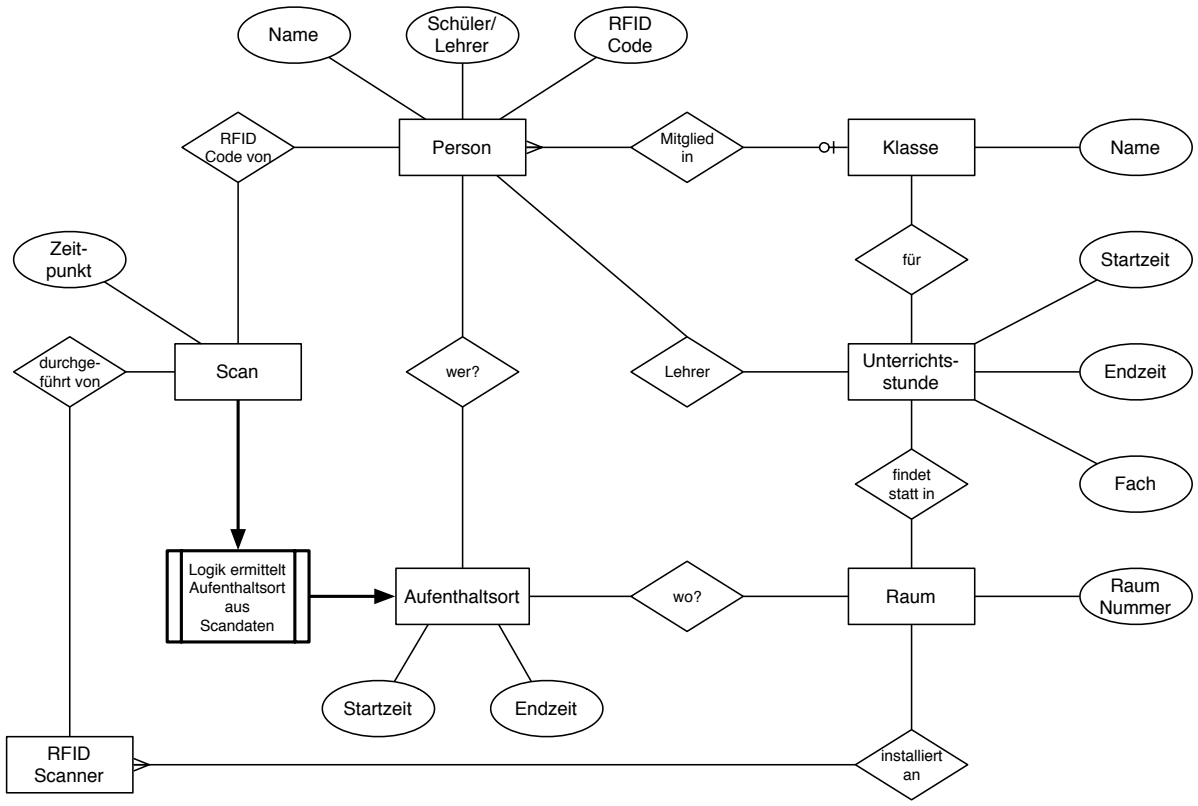

Abbildung 11: Ein komplizierteres ERM

Vielmehr erfassen die RFID-Scanner in jedem Türrahmen RFID-Chips während sie sich durch ihn hindurchbewegen. Dabei entstehen Daten der Form *Der RFID-Chip mit dem Code A war zur Zeit B im Empfangsbereich von Scanner C*. Diese nur eingeschränkt aussagekräftigen Datensätze müssen erst aufbereitet werden. Es muss etwa herausgefunden werden, welcher Person Code A zugeordnet ist, wo Scanner C steht und ob die Person den Raum betreten oder verlassen hat, oder ob sie vielleicht nur zu nah am Türrahmen vorbeigelaufen ist. Diese Aspekte müssen nicht in das Modell einfließen; letztlich ist ja nur ein ERM für die zur Abfrage von Fehlzeiten nötige Datenbank gefordert. Ein stark vereinfachtes Modell ist also akzeptabel, wenn dessen Lücken erkannt sind.

Ein Beispiel für ein komplizierteres ERM ist Abbildung 11. Hier wird das Überwachungssystem detaillierter abgebildet.

Beide Systeme enthalten die zur Fehlzeiten-/Entschuldigungsabwicklung benötigten Daten, nämlich die Unterrichtsstunden (oder auch „Soll-Ort“) für jeden Schüler und den Ort, an dem sich ein Schüler tatsächlich zu einer gegebenen Zeit aufgehalten hat. Wird jetzt (etwa von der Eltern-WWW-Plattform) die Anfrage gestellt, welche Fehlzeiten für Schüler x eingetragen sind, so müssen zunächst alle Unterrichtsstunden, an denen ein Schüler teilnehmen sollte, abgefragt werden. Im Modell in Abbildung 10 müssen dazu alle „nimmt teil an“ Beziehungen für diesen Schüler ausgewertet werden, im Modell in Abbildung 11 muss erst abgefragt werden, in welcher Klasse der Schüler Mitglied ist und dann, welche Unterrichtsstunden dieser

Klasse zugeordnet sind. Als nächstes muss für jede in der Abfrageantwort enthaltenen Unterrichtsstunde mittels „Aufenthaltsort“ bestimmt werden, ob sich der Schüler zur Zeit der Unterrichtsstunde am richtigen Ort befunden hat. Gibt es eine (signifikante) Diskrepanz zwischen den Daten der Unterrichtsstunde und dem Aufenthaltsort des Schülers, wurde eine Fehlzeit gefunden.

In einer Lösung kann auch noch dokumentiert werden, wie die Logik funktionieren könnte, die aus den rohen Scandaten Aufenthaltszeiten in einzelnen Räumen ermittelt. Denkbar wäre zum Beispiel, dass jeweils zwei aufeinanderfolgende Scans des selben RFID-Codes betrachtet werden. Fanden sie am selben Scanner/Türrahmen statt, kann angenommen werden, dass die Person beim ersten Scan den Raum betreten und ihn beim zweiten Scan verlassen hat. Fand der zweite Scan woanders statt, ist anzunehmen, dass die Person den Raum gar nicht betreten hat, sondern nur zu nah am Türrahmen entlang gelaufen ist. Natürlich ist auch denkbar, dass die Person im zweiten Fall den Raum über einen unüberwachten Zugang betreten/verlassen hat oder der Scanner beim Durchschreiten des Türrahmens versagt hat, etc. Diese Teilaufgabe ist die richtige Stelle, um sich über derlei Mehrdeutigkeiten und Unzulänglichkeiten im Modell Gedanken zu machen.

Teilaufgabe 3

In dieser Teilaufgabe sollen (Datenbank-)Tabellen angegeben werden, die dafür geeignet sind, die im ERM aus Teilaufgabe 2 abgebildeten Daten aufzunehmen. Toll ist, wenn erkannt wird, dass das Entity-Relationship-Modell prinzipbedingt nicht unbedingt direkt in einen Tabellensatz umgewandelt werden kann. Beziehungen können nämlich unterschiedlich interpretiert werden. Eine Beziehung kann an sich ein Datensatz (in einer eigenen Tabelle) sein, dessen Felder die zu verknüpfenden Datensätze bezeichnen. Eine Beziehung kann auch in Form eines Tabellenfeldes, das ein Schlüssel in eine andere Tabelle ist, umgesetzt werden.

Beispielsweise müssen Unterrichtsstunden nicht unbedingt (wie in den beiden Beispielen) als Entities umgesetzt werden. Man kann sie auch als Beziehung zwischen einem Raum, einem Lehrer und einer Schulklasse mit den Attributen „Startzeit“, „Endzeit“ und „Fach“ auffassen. Implementieren würde man Unterrichtsstunden aber schon als Datensätze in einer eigenen Tabelle. Die Beziehung zwischen einer Person und einer Klasse als eigenständige Datensätze umzusetzen ist hingegen unpraktisch. Hier tut es auch ein Schlüssel in die Klassentabelle in jedem Schüler-Datensatz. Die „nimmt teil an“ Beziehung (aus Abbildung 10) wiederum kann man nicht als Attribut in der Personen- oder Unterrichtsstundentabelle umsetzen, weil es sich um eine $n : m$ Beziehung handelt. Die „Mitglied in“ Beziehung (aus Abbildung 11) hingegen kann man als Attribut in der Personentabelle unsetzen.

Das ERM aus Abbildung 10 könnte man folgender Weise in einen Tabellensatz umsetzen:

Person (ID, Name, Schüler/Lehrer?, RFID Code)
Unterrichtsstunde (ID, Startzeit, Endzeit, Fach, Ort[Raum→ID])
Raum (ID, Raumnummer)
Aufenthaltsort (ID, Person[Person→ID], Raum[Raum→ID], Startzeit, Endzeit)
Unterrichtsteilnahme (ID, Person[Person→ID], Unterrichtsstunde[Unterrichtsstunde→ID])

Das ERM aus Abbildung 11 kann in Tabellenform so aussehen:

Person (ID, Name, Schüler/Lehrer?, RFID Code, Klasse[Klasse→ID])
Klasse (ID, Name)
Unterrichtsstunde (ID, Startzeit, Endzeit, Fach, Ort[Raum→ID],
Klasse[Klasse→ID], Lehrer[Person→ID])
Raum (ID, Raumnummer)
Aufenthaltsort (ID, Person[Person→ID], Raum[Raum→ID], Startzeit, Endzeit)
Scan (ID, Wen[Person→ID], Von[Scanner→ID], Zeitpunkt)
Scanner (ID, installiert[Raum→ID])

Teilaufgabe 4

In dieser Teilaufgabe ist nach einer persönlichen Meinung gefragt. Hier ist natürlich alles richtig, was gut begründet ist. Einige der Aspekte Datenschutz, Einschränkung der Freiheit, Verletzung von Persönlichkeitsrechten und Kosten-Nutzen Verhältnis sollten allerdings schon betrachtet werden.

Darüber hinaus stellt sich noch eine ganz andere Frage: Ist eine technische Einrichtung wie die in der Aufgabenstellung beschriebene hilfreich, wenn eine Schule ein Fehlzeitenproblem hat und dieses bewältigen will? Fehlverhalten kann man generell mit Kontrolle und Sanktionen begegnen – besser aber noch mit Erziehung. Sinnvollerweise sollten zusätzlich oder vielleicht besser noch an Stelle eines Kontrollsystems pädagogische Maßnahmen ergriffen werden; insbesondere muss sich eine Schule mit gravierender Fehlzeitenproblematik über ihr Schulkonzept und ihre Unterrichtskultur Gedanken machen.

Bewertungskriterien

- Wird in Teilfaufgabe 1 ein Verfahren vorgeschlagen, dass den *gesamten* Entschuldigungsprozess abwickelt (nicht nur das Erfassen von Fehlzeiten)? Auf jeden Fall muss die Phase ab Eingang einer Entschuldigung für einen bestimmten Zeitraum beschrieben sein.
- Wird auf die Praxistauglichkeit eingegangen? Für die Beschreibung des Verfahrens genügt die natürliche Sprache, auch wenn Veranschaulichungen wie das obige Flussdiagramm zu begrüßen sind. Wenn Lehrerfehlzeiten nicht berücksichtigt wurden, ist das zwar ein Mangel, der aber für sich alleine nicht zu einer Abwertung führt.
- Das Entity-Relationship-Modell soll wenigstens so detailliert sein, dass es „Soll-“ und „Ist-Aufenthaltsort“ von einzelnen Personen zu bestimmten Zeiten abbilden kann. Auf den „Soll-Ort“ kann verzichtet werden, wenn davon ausgegangen wird, dass sich dieser aus schon vorhandenen Systemen (z.B. Stundenplansystem) ergibt. Der „Ist-Ort“ kann durchaus unbekannt sein, wenn der oder die Betroffene erst gar nicht in den Einzugsbereich des Erfassungssystems gelangt ist.

- Vereinfachungen im ERM gegenüber der Realität müssen erkannt und begründet sein. Hier genügen relativ einfache Aussagen.
- In den Tabellenspezifikationen soll das ERM aus Teilaufgabe 2, insbesondere die Beziehungen wiedererkennbar sein. Es wird nicht erwartet, dass die Mehrdeutigkeiten und Unzulänglichkeiten des ERM beim Implementieren einer Datenbank erkannt werden.
- Fremdschlüssel müssen in der Tabellenspezifikation in „geeigneter Weise“ markiert sein. Man soll also leicht erkennen können, auf welche Tabelle sie sich beziehen. Minimalforderung ist, dass in Ausgangs- und Zieltabelle des Schlüsselattributs der gleiche Bezeichner verwendet wird und die Markierung der Fremdschlüssel von der der Primärschlüssel zu unterscheiden ist.
- Die Meinung in Teilaufgabe 4 soll schlüssig begründet sein. Es sollte wenigstens auf einen Teil der Aspekte Datenschutz, Freiheit, Persönlichkeitsrechte und Verhältnismäßigkeit eingegangen werden.

Aufgabe 5: Kleingeld

Dünne Portemonnaies sind in! Clara schafft es tatsächlich, zu keinem Zeitpunkt mehr als 6 Münzen mit einem Wert unter €1 dabei zu haben. Doch der Reihe nach. Wie schafft Clara das eigentlich – oder anders: Wie kann man das berechnen?

Man kann selbstverständlich für jede Kombination von Münzen in Claras Portemonnaie einzeln berechnen, wie viele Münzen sie bei einem gegebenen Preis zurückbekommt und dann das Optimum auswählen (Brute Force). Alternativ gibt es die triviale Lösung, alles Kleingeld bei jedem Einkauf auf den Tisch zu legen. Sie bekommt ja immer die ideale Menge Kleingeld zurück, also liefert dieses Vorgehen immer das ideale Ergebnis. Es bleibt nur zu befürchten, dass das Kassenpersonal bei diesem Vorgehen irgendwann einmal nicht mehr freundlich bleibt. Will Clara die ideale Kombination von Münzen hinlegen, so kann sie den Bezahlvorgang mit allen Münzen einmal im Kopf ausführen und sich überlegen, was sie zurückbekommt. Jede Münze, die sie abgibt und später wieder zurückbekommt, gehört nicht zur minimalen Menge an Kleingeld, das sie für ein optimales Ergebnis dem Verkäufer geben muss.

Damit Clara eine solche Berechnung mal eben im Kopf ausführen kann, müsste es einen einfachen Algorithmus geben, mit dem man die optimale Geldrückgabe berechnen kann. Gott sei Dank gibt es (für die beim Euro vorkommenden Cent-Stücke) eine entsprechende Vorschrift: Man nehme immer die größte Münze, die noch ins Rückgeld passt und gebe diese zurück. Das Rückgeld verringert sich um diesen Betrag. Das wird so lange wiederholt, bis das Rückgeld Null ist. Bei diesem Vorgehen handelt es sich um einen so genannten Greedy-Algorithmus (von greedy = gierig). Diese Vorschrift gilt aber nur bei den Münzabstufungen, wie sie zum Beispiel beim Euro vorhanden sind. Würde es zum Beispiel nur 50 Cent-, 30 Cent- und 5-Cent-Münzen geben, so wären 60 Cent ideal als zwei Münzen zu 30 Cent zurück zu geben. Der Algorithmus würde aber eine Münze zu 50 Cent und zwei Münzen zu 5 Cent nehmen.

Damit ist das Vorgehen der beiden verwendeten Algorithmen beschrieben. Zurückgeben funktioniert nach dem Greedy-Prinzip. Beim Bezahlen überlegt sich Clara, was sie zurückbekommt, wenn sie alle Münzen abgibt, und gibt die Münzen ab, die sie nicht zurückbekommen würde.

Doch warum kann Clara niemals mehr als sechs Münzen mit einem Wert unter €1 haben? Wenn sie einen Euro oder mehr an Kleingeld im Portemonnaie hat, hat sie sicherlich etwas falsch gemacht (Umtausch in ein 1-Euro-Stück möglich!). Und alle Beträge zwischen €0,00 und €0,99 lassen sich mit maximal sechs Münzen zusammenstellen.

Zu den Teilaufgaben:

1. Der Bestand an Centmünzen im Portemonnaie zu Beginn der Simulation kann zum Beispiel festgelegt werden, indem das Rückgeld auf einen beliebigen Preis berechnet wird. Auf diese Art und Weise ist sichergestellt, dass Clara auch zu Beginn keine Kombination von Kleingeld hat, die nicht aus minimal vielen Münzen besteht. Alternativ ist auch eine Eingabe der Münzen mit anschließender Plausibilitätskontrolle möglich.
2. Clara hat im Schnitt 3,4 Münzen (mit einem Wert unter €1) im Portemonnaie. (Durchschnitt über 10000 Verkaufvorgänge).

3. Hier ist es sinnvoll, alle möglichen Münzkombinationen zu betrachten. Dazu berechnet man das Rückgeld für alle Beträge zwischen €0,00 und €0,99 und betrachtet die Anteile der einzelnen Münzsorten. Die Münzen sind mit folgenden (relativen) Häufigkeiten verwendet:

50 Cent:	14,71%
20 Cent:	23,53%
10 Cent:	11,76%
5 Cent:	14,71%
2 Cent:	23,53%
1 Cent:	11,76%

   Während das 1-Cent-Stück beispielweise benötigt wird um auf ungerade Beträge herauszugeben, sind alle anderen Münzen zumindestens theoretisch ersetzbar. Am ehesten könnte man die 10-Cent-Münze weglassen. Im Portemonnaie kann zu keinem Zeitpunkt mehr als eine 10-Cent-Münze sein (vorausgesetzt, Clara hat keine Fehler gemacht!) Eine 10-Cent-Münze lässt sich durch zwei 5-Cent-Münzen ersetzen, Clara müsste hier also im schlechtesten Fall mit 7 Münzen leben.

Ablaufprotokoll

Ausschnitt aus einem Protokoll (aufgeführt sind jeweils nur die Münzen mit einem Wert kleiner als 1€: 50 Cent/20 Cent/10 Cent/5 Cent/2 Cent/1 Cent):

```
Aktueller Inhalt von Claras Portemonnaie: 0/0/0/0/0/0
------------------------------------------------------
Zu bezahlen: 7.22; Clara gibt: 0/0/0/0/0/0; Rückgeld: 1/1/0/1/1/1
Aktueller Inhalt von Claras Portemonnaie: 1/1/0/1/1/1
------------------------------------------------------
Zu bezahlen: 6.05; Clara gibt: 0/0/0/1/0/0; Rückgeld: 0/0/0/0/0/0
Aktueller Inhalt von Claras Portemonnaie: 1/1/0/0/1/1
------------------------------------------------------
Zu bezahlen: 0.71; Clara gibt: 1/1/0/0/0/1; Rückgeld: 0/0/0/0/0/0
Aktueller Inhalt von Claras Portemonnaie: 0/0/0/0/1/0
------------------------------------------------------
Zu bezahlen: 0.34; Clara gibt: 0/0/0/0/0/0; Rückgeld: 1/0/1/1/0/1
Aktueller Inhalt von Claras Portemonnaie: 1/0/1/1/1/1
------------------------------------------------------
Zu bezahlen: 3.20; Clara gibt: 1/0/1/0/0/0; Rückgeld: 0/2/0/0/0/0
Aktueller Inhalt von Claras Portemonnaie: 0/2/0/1/1/1
------------------------------------------------------
Zu bezahlen: 8.91; Clara gibt: 0/2/0/0/0/1; Rückgeld: 1/0/0/0/0/0
Aktueller Inhalt von Claras Portemonnaie: 1/0/0/1/1/0
------------------------------------------------------
Zu bezahlen: 1.09; Clara gibt: 1/0/0/0/0/0; Rückgeld: 0/2/0/0/0/1
Aktueller Inhalt von Claras Portemonnaie: 0/2/0/1/1/1
------------------------------------------------------
```

Bewertungskriterien

Teilaufgabe 1 Das Programm funktioniert korrekt, wenn niemals (bis vielleicht auf eine Anfangsphase) mehr als 6 Kleingeld-Münzen im Portemonnaie sind und wenn Preis, gegebenes Geld und Rückgeld zusammen passen. Bei Fehlern ist zu unterscheiden, ob diese der Berechnung der herzugebenden oder der rückzugebenden Münzen zuzuordnen sind. In beiden Fällen kann die Münzmenge unnötig groß werden, aber im zweiten Fall wurde „nur“ die Vorschrift der Aufgabenstellung verletzt, während im ersten Fall ein Denkfehler bzgl. der Optimalität des Verfahrens vorliegen kann. Damit eine Überprüfung gut möglich ist, muss das Programm sowohl die Münzen, die Clara dem Verkäufer gibt, als auch das Rückgeld und den anschließenden Stand im Portemonnaie ausgeben.

Teilaufgabe 2 Das Programm soll die durchschnittliche Kleingeldmenge im Portemonnaie bestimmen. Sinnvolle statistische Aussagen benötigen hier eine ausreichend große Stichprobe. Optimalerweise werden für die Durchschnittsberechnung deutlich über 1000 Versuche gemacht, 100 sollten es aber mindestens sein. Im Gegensatz zur Forderung in der Aufgabenstellung wurde großzügig darüber hinweg gesehen, wenn weniger als 100 Einkäufe dokumentiert sind. Es sollten aber so viele Beispiele dokumentiert sein (Richtwert: etwa 10), dass sich die Korrektheit der Programmfunktion daran gut nachvollziehen lässt. Der Durchschnittswert muss nicht nur berechnet, sondern auch angegeben werden; er sollte irgendwo zwischen 3,2 und 3,6 liegen.

Teilaufgabe 3 Um die hier indirekt geforderte Statistik angeben zu können ist es (eine Gleichverteilung der möglichen Preise vorausgesetzt) sinnvoll, über sämtliche möglichen Kombinationen zu mitteln, sofern der Suchraum nicht zu groß ist. In diesem Fall ist der Suchraum mit 100 Möglichkeiten sehr überschaubar. Alternativ kann natürlich auch über alle simulierten Einkaufsvorgänge gemittelt werden. Die Aussage, ob eine Münze weggelassen werden kann und wenn ja welche, muss sinnvoll sein und mit den Messwerten des Einzelnen begründet werden. Überlegungen über die Konsequenzen eines Weglassens sind auf jeden Fall wünschenswert.

Junioraufgabe: Der Segelflug der Solinge

Lösungsidee zu Teilaufgabe 1

“segeln”

Die Aufgabenstellung geht nicht auf die genaue Bedeutung des Wortes “segeln” ein. Das Segeln ist allerdings entscheidend für das Ziel des Spiels, denn wenn segeln bedeutet, dass der Soling mit konstanter Geschwindigkeit nach unten fällt, dann brauchen alle Solinge die gleiche Zeit, bis sie unten angekommen sind (zumindest dann, wenn die Zeit für die horizontale Bewegung keine Rolle spielt). Natürlich gibt es Möglichkeiten, das Spiel trotzdem interessant zu machen, z.B. indem man die Regel einführt, dass die Solinge keine Barriere berühren dürfen.

Anstelle der konstanten Geschwindigkeit kann natürlich jede beliebige Art von Beschleunigung verwendet werden. Diese Entscheidung gehört in jedem Fall in diesen Abschnitt, und es ist ebenfalls wichtig, dass die Folgen für den Spielablauf erläutert werden.

Horizontale Bewegung

Es ist auch unklar, inwiefern der Spieler die horizontale Bewegung kontrollieren kann. Eine einfache Methode wäre, dass der Soling durch jeden Tastendruck nach links bzw. rechts verschoben wird. Man könnte auch die Geschwindigkeit oder sogar die Beschleunigung durch Tastendrücke verändern lassen. Damit würde das Spiel dann wahrscheinlich zu einem Geschicklichkeitsspiel und wesentlich interessanter als die vorliegende Version.

Barrieren

Die Aufgabenstellung fordert die Veränderung der Barrieren mit dem Auftreten eines neuen Solings. Dazu kann man Barrieren dynamisch erzeugen, aber auch aus einer Reihe von vorgefertigten Spielfeldern wählen. Es ist also zu beschreiben, wie Barrieren verändert werden.

Bei dynamischer Erzeugung ist eine Ober- und Untergrenze für die Anzahl der Barrieren erforderlich. Es ist außerdem wichtig, wo und wieviele Löcher in den Barrieren erscheinen (die Aufgabenstellung gibt nur “bis zu drei” vor). Eine Barriere direkt unter dem Startpunkt des Solings hat sicherlich keinen Sinn. Festlegungen zu Position und Abstand der Barrieren müssen getroffen werden.

Start- und Zielposition

Die Startposition des Solings ist nicht spezifiziert. Es ist eine gute Idee, den Soling zufällig am oberen Rand des Spielfeldes starten zu lassen. Es ist natürlich auch erlaubt, den Soling von einer festen Stelle starten zu lassen, wenn das dokumentiert wird.

Ziel des Spiels

Das Ziel des Spiels ist in der Aufgabenstellung vorgegeben. Allerdings sollte darauf eingegangen werden, wie sinnvoll das Ziel in Kombination mit den zuvor getroffenen Entscheidungen ist. Z.B. ist das Ziel des Spiels dann sinnvoll, wenn man durch Drücken einer Taste die Fallgeschwindigkeit variieren kann und evtl. bei komplizierten Konstellationen auch durch das "Löschen" eines Solings dafür sorgen kann, dass neue Barrieren generiert werden. Das Zeitlimit muss übrigens auch noch festgelegt werden, etwa auf 90 Sekunden.

Das Spielfeld

Schließlich muss noch über die Spielfeldgröße entschieden werden. Die klassische Größe einer Textkonsole (80 x 24 Zeichen) ist durchaus akzeptabel.

Lösungsidee zu Teilaufgabe 2

Zunächst ist zu klären, wie der „gerasterte Bildschirm" realisiert wird. Eine Textkonsole ist hier durchaus das Richtige, Umsetzungen in einem GUI sind natürlich genauso möglich. Darüber hinaus sind wesentliche Punkte der Lösungsidee die Erzeugung der Barrieren und die Eingabeverarbeitung. Letztere muss parallel zur Ausgabe erfolgen und kann entweder in Form einer Game-Loop, d.h. durch ständigen Wechsel von Ausgabe des Spielfeldes und Abfrage der Eingabe, oder durch ereignisorientierte Programmierung realisiert werden.

Wenn ereignisorientierte Programmierung verwendet wird, werden Benutzereingaben also nicht ständig abgefragt, sondern erzeugen nur einen Ereignisaufruf im Programm, wenn tatsächlich eine Taste gedrückt wird. Genauso kann es Timer-gesteuerte Ereignisaufrufe geben. Das hat den Vorteil, dass der Prozessor nicht die ganze Zeit vollständig ausgelastet ist. Die Benutzeroberfläche wird dann neu gezeichnet, wenn es zu Änderungen gekommen ist.

Auf den Druck von Tasten zur Bewegungssteuerung muss das Programm reagieren, indem es den Soling entsprechend verschiebt. Wenn der Soling verschoben wird, wird sichergestellt, dass er nicht links, rechts oder unten aus dem Bildschirm geschoben wird (und je nach Spezifikation noch, dass er keine Barriere berührt). Ist er unten angekommen, gilt er als "gerettet", wenn er in der Zielregion ist, und als nicht gerettet, wenn er außerhalb dieser Region ist. Anschließend werden neue Barrieren erstellt (oder eine andere vordefinierte Barrierenkonstellation gewählt) und ein neuer Soling segelt von oben herab. Dieser Vorgang wiederholt sich, bis die Zeit abgelaufen ist. Die geretteten Solinge werden gezählt und nach Ablaufen der vorher festgelegten Zeit wird die Anzahl der geretteten Solinge als Spielergebnis angezeigt.

Es stellt sich heraus, dass das Erstellen der Barrieren ziemlich trickreich sein kann. Es muss z.B. sichergestellt werden, dass es immer einen Weg von oben nach unten gibt. Außerdem sollte es keinen durch den Zufallsgenerator bedingten Fall geben, in dem es zu einer Endlosschleife kommt. Dies kann beispielsweise passieren, wenn man immer wieder versucht zufällige Parameter für die Barrieren zu erzeugen, bis es keine Konflikte mehr gibt. Dieses Problem sollte ggf. erkannt und dokumentiert werden. Die Generatorfunktionen müssen *nicht*

anspruchsvoll sein. Es soll nur sichergestellt werden, dass das Erzeugen der Barrieren terminiert und dass es immer einen Weg nach unten gibt.

Beispiele

Beispiele sind hier am sinnvollsten mit Screenshots zu dokumentieren, aber auch andere nachvollziehbare Varianten sind denkbar. Es sollten mindestens drei Beispiele präsentiert werden; es muss ersichtlich werden, dass das eingesandte Programm funktioniert.

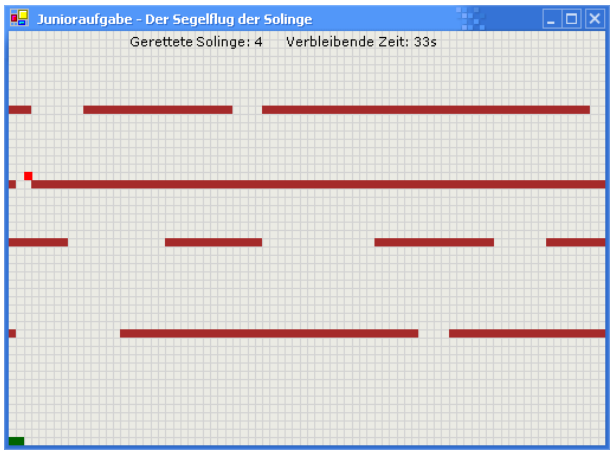

Abbildung 12: Während des Spiels, mehrere zufällige Barrieren

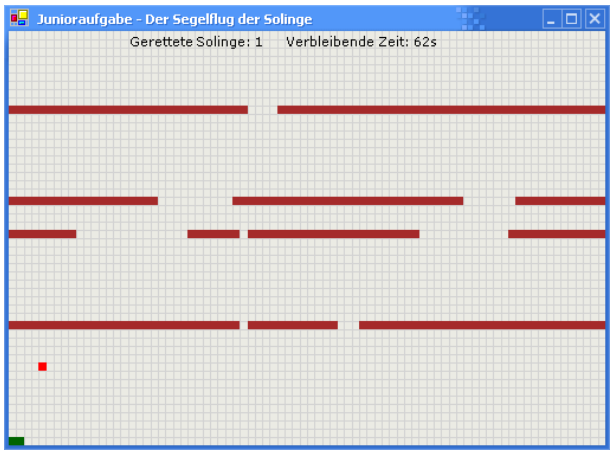

Abbildung 13: Während des Spiels, zufällige Barrieren in anderer Konfiguration

Die Abbildungen 12 bis 14 zeigen Screenshots aus einer Beispielimplementation.

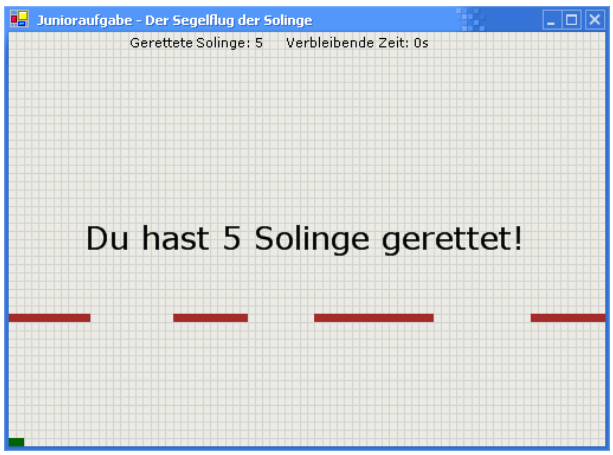

Abbildung 14: Ergebnis einer gespielten Runde

Bewertungskriterien

Teilaufgabe 1

- Die Spielregeln müssen klar und umfassend genug präzisiert sein; wichtig ist, dass zu Bewegung und Geschwindigkeit sowie zu den Barrieren sinnvolle Festlegungen getroffen werden.
- Die Spielregeln bzw. die Entscheidungen dazu müssen begründet werden.
- Das letztlich beschriebene Spiel muss sinnvoll sein bzgl. des vorgegebenen Ziels (möglichst viele Solinge sollen in einer bestimmten Zeit das Ziel erreichen). Konstante Geschwindigkeit in beiden Achsen (vertikal *und* horizontal) sind akzeptabel, weil dann immerhin noch der kürzeste Weg durch die Barrieren gefunden werden muss.

Teilaufgabe 2

- Die Implementation muss mit den vereinbarten Spielregeln aus Aufgabenteil 1 übereinstimmen.
- Die Barrieren oder Löcher müssen im oben beschriebenen Sinne zufriedenstellend erzeugt oder verändert werden.
- Und natürlich soll die Implementation auch funktionieren.
- Wie in den allgemeinen Regeln des Wettbewerbs festgehalten, sollte das Funktionieren des Programms auch bei dieser Aufgabe durch ausreichend Programmablaufprotokolle bzw. Beispiele belegt sein; drei Screenshots sind eher eine Minimalforderung.

Aus den Einsendungen: Perlen der Informatik

Allgemeines

Worte des Wettbewerbs: Algorhythmus, Bruthforce

Dieses System besteht aus vier Units: unit1, unit2, unit3, unit4 und unit5

. . . , vermeide ich durch eine Verdreifachung des Codes Redundanz.

Danach bleibt sie für 1,9 Sekunden in einer Endlos-Schleife.

Ja, ich weiß, dass das irgendwo schwachsinnig ist.

Im Voraus schon mal Entschuldigung für den Stack-Overflow in Aufgabe 1.

Man muss in der Eingabeschleife eine Zahl von 1 bis 4 eingeben, ansonsten wird die Eingabe wiederholt oder, falls man einen Buchstaben eingegeben hat, dreht das Programm durch.

Favorites First

Rheinfolge

Anzahl der Lider

ESTHER - die Elektronische Schnelltesthörerin mit erweiterter Rückmeldefunktion

K.O.M.P.L.E.X.T. - Modus: kompliziert-organisch multidynamische Problembetrachtung durch Langzeitsimulation mit extra Titelanalyse

Dauerdudeleigenervekorrektur

Nach dem Sortieren splittert man den Stapel in zwei Stapel auf.

Die Simulation scheint einbahnfrei zu funktionieren.

Die zufällige Reihenfolge der Stücke wird durch die Prozedur Sort erstellt.

Dadurch tritt noch ein nicht gewünschter Effekt auf: Bo wird entlassen, da die MP3-Sparte seiner Firma wegen mangelndem Gewinn aufgelöst wird.

Formel-Up

. . . , dass ich mir sowieso überhaupt nur deswegen Gedanken über so etwas machen muss, weil es ein durchgeknallter Freizeitforscher nicht geschafft hat, sich Assistenten anzulachen, die wenigstens ein Maßband gerade halten können.

Da klein aber ein relativer Begriff ist und sich das Programm vor dem Anwender keine Blöße geben will, kann man diese Begründung dennoch so stehen lassen.

```
Ich schlage unverbindlich die Formel 1*h+1*a=2*m vor.
```

Zaras Zauberfolie

Die Funktion ’encode’ leistet das Dekodieren des Bildes.

Das Programm kann nur mit 2.147.483.646 Faxen umgehen.

Jedoch nach kurzer Bearbeitung mit einem Grafikprogramm kann man erkennen, dass es sich um ein verschlüsseltes Webmuster handelt . . . oder so . . . wie auch immer . . . gute Nacht.

Die Schwachstelle dieses stenographischen Algorithmus . . .

Fehlzeitendatenbank

Theoretisch wäre dieses System praxistauglich . . .

Einige Fremdschlüssel zeigen auf Tabellen, die nicht existieren.

Nach der Formel für Zeitdilatation könnte ein Schüler, der sich mit einer Geschwindigkeit von 294792417 m/s bewegt, innerhalb nur einer Sekunde die kompletten 45 Minuten einer Schulstunde bewältigen.

Angenommen, ein Schüler baut eine Zeitmaschine und betritt einen Raum ganz normal. Anschließend reist er in dem Raum in der Zeit zurück und verlässt ihn dann, ohne ihn vorher betreten zu haben. Dieses könnte Fehler im Zentralcomputer hervorrufen und ist deshalb unbedingt bei der Programmierung zu berücksichtigen.

Ich schlage die Lagerung der Daten in einem zeitunabhängigen Raum vor, z.B. in einer Disco im 60er-Jahre-Stil. Oder man isoliert den Server mit zeitabsorbierenden Materialien, etwa mit 100 Jahre altem Wein oder Frauen.

Kleingeld

$51 * 50$ Cent $\approx 51 * 5$ Cent

Posttransaktionalergeldstückanzahl

Bei den folgenden Ausdrucken sind die Farben umgekehrt, um meinen Drucker zu schonen. Die weiße Säule ist hier schwarz, die gelbe dunkelblau und die rote hellblau.

„Na toll“, wird die virtuelle Clara sich jetzt sagen, „und um das herauszufinden, musste ich 251,46 Euro ins Daten-Nirvana schicken?“

Der Segelflug der Solinge

Das Wichtigste im Spiel ist, wann man gewinnt, und ob.