23. 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 verzögert insbesondere die versprochene 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. In diesem Jahr war bei fast allen Aufgaben mindestens ein Beispiel vorgegeben. 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. 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.

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 war dann in der Regel der Abzug eines oder mehrerer Punkte. Eine Wellenlinie bedeutet „na ja, hätte besser sein können“, führte aber meist nicht zu Punktabzug. Mehrere Wellenlinien konnten 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 in allen kritischen Fällen, insbesondere denen mit 11 Punkten, genau nachgeprüft haben, ob nicht doch irgendwoher die Punkte zu holen gewesen wären, die das Weiterkommen ermöglicht hätten.

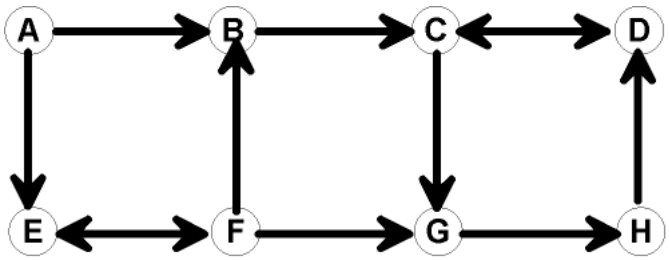
Wir wollen zusätzlich zu diesen Lösungsideen ausführlichere Beispiellösungen erstellen, die auf den Webseiten des Bundeswettbewerbs Informatik (`www.bwinf.de`) veröffentlicht werden – allerdings wohl erst im ersten Viertel des nächsten Jahres. Bis dahin kann man sich mit den Einsendungen befassen, die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

Aufgabe 1: tratsCH trATsch

Lösungsidee

Chatonen sind merkwürdige Personen, lästern gerne über Gott und die Welt (natürlich auch über die eigenen Freunde), verstehen aber keinen Spaß, wenn über sie gelästert wird. Nebenbei sind sie derartig zerstreut, dass sie nicht mitbekommen, wem sie jetzt was erzählen dürfen. Das in dieser Aufgabe zu entwickelnde Programm soll da Abhilfe schaffen.

Betrachtet man die Sympathiebeziehungen der Chatonen, so bietet sich als Datenstruktur ein gerichteter Graph an, in dem die Chatonen die Knoten bilden und die Sympathiebeziehunngen durch gerichtete Kanten ausgedrückt werden. Ein Graph für das geforderte Beispiel ist in der folgenden Abbildung zu sehen:



In dieser Struktur muss für jeden Chatonen ermittelt werden, mit wem er kommunizieren kann – direkt oder indirekt über andere Chatonen. Über diese „Freunde“ des Chatonen und über ihn selbst sollte ein Tratscher dann nichts Schlechtes erzählen. Es sei denn, der Chatone kann das Objekt der Lästerei nur indirekt über den Absender des Tratsches erreichen, denn der würde die selbst produzierte üble Nachrede natürlich nicht durchreichen.

Etwas formaler: Es muss also für einen Sender-Chatonen s und einen Empfänger-Chatonen e (der s sympathisch sein muss) ermittelt werden, welche direkten und indirekten Adressaten e erreichen kann, ohne dass s die Nachricht weiterleiten muss. Ein mögliches Verfahren hierzu ist die Breitensuche, bei der die in einer Bearbeitungsliste enthaltenen Knoten nacheinander wie folgt behandelt werden: Die direkten Nachfolger des betrachteten Knotens werden an die Liste angehängt, der Knoten selbst als bearbeitet markiert (damit er nicht später noch einmal bearbeitet wird – so führen Zyklen im Graph nicht zu Problemen). Wenn zu Beginn e als einziger Knoten in der Bearbeitungsliste enthalten und s als schon bearbeitet markiert ist, werden durch die Breitensuche genau alle die Chatonen besucht, die von e aus erreichbar sind, ohne dass der Weg über s führt. Die Menge der bearbeiteten Knoten entspricht also denjenigen Chatonen, über die s dem e nichts Negatives erzählen sollte; diese „Tabu-Menge für s bezüglich e“ wollen wir kurz mit tabu(s, e) bezeichnen. Diese Menge muss für jedes Sender/Empfänger-Paar s und e eigens berechnet werden.

Nun verlangt die Aufgabenstellung aber eine positive Aussage. Die Menge der Chatonen, über die s gegenüber e lästern darf, entspricht der Gesamtmenge der Chatonen ohne e (und auch ohne s, wie in der Aufgabenstellung gesagt) und ohne die Chatonen, die von e aus über andere

Chatonen als s erreichbar sind, also ohne tabu(s, e). Diese Positiv-Menge für Lästereien wird für jeden Chatonen s und alle ihm sympathischen Chatonen e ausgegeben.

Beispiel

Für das geforderte Beispiel lautet das Ergebnis der Überprüfung wie folgt:

```
Chatone A kann Chatone B Gerüchte erzählen über:  EF
Chatone A kann Chatone E Gerüchte erzählen über:
Chatone B kann Chatone C Gerüchte erzählen über:  AEF
Chatone C kann Chatone D Gerüchte erzählen über:  ABEFGH
Chatone C kann Chatone G Gerüchte erzählen über:  ABEF
Chatone D kann Chatone C Gerüchte erzählen über:  ABEF
Chatone E kann Chatone F Gerüchte erzählen über:  A
Chatone F kann Chatone B Gerüchte erzählen über:  AE
Chatone F kann Chatone E Gerüchte erzählen über:  ABCDGH
Chatone F kann Chatone G Gerüchte erzählen über:  ABE
Chatone G kann Chatone H Gerüchte erzählen über:  ABEF
Chatone H kann Chatone D Gerüchte erzählen über:  ABEF
```

Bewertungskriterien

- Wichtig ist, dass der eingereichte Charminator auch die Charmefehler findet, die auf indirekter Kommunikation beruhen.
- Nichtbeachtung der Bedingung, dass ein Absender einer Nachricht diese nicht selbst weiterleitet, führt zu falscher Berechnung der Tabu-Mengen. Im Pflichtbeispiel darf C dem D durchaus Gerüchte über G und H erzählen, und F darf dem E praktisch alles erzählen, weil E nur via F kommuniziert.
- Das vorgegebene Beispiel betrifft acht Chatonen, in der Aufgabenstellung ist von bis zu 10 Chatonen die Rede. Die Anzahl der Chatonen oder auch deren Maximalzahl dürfen im Programm nicht fest verdrahtet sein, ebensowenig die Namen der Chatonen.
- Wenn die Bedingung, dass ein Sender seine eigenen Nachrichten nicht weiterleitet, nicht gelten würde, könnten Tabu-Mengen bzw. Positiv-Listen nur abhängig vom Empfänger berechnet werden, also nur einmal pro Chatone und nicht einmal pro Paar. Eine solch vereinfachte Berechnung erfüllt die Aufgabenstellung aber nicht.
- Nicht nur für die Weiterleitung erhaltener Nachrichten gilt die Beschränkung auf sympathische Chatonen, sondern auch für das Verschicken eigener Nachrichten.
- Die Ausgabe sollte nach Aufgabenstellung kompakt sein. Akzeptabel ist hier z.B. eine Tabelle oder auch eine Form wie die oben angegebene. Nicht akzeptabel ist eine Ausgabe, in der jedes Trio mit Sender, Empfänger und möglichem Tratschobjekt einzeln aufgelistet wird.

Aufgabe 2: Kosmische Schaltjahre

Lösungsidee

Einteilung in X-Monate

Die Anzahl der X-Monate wird mit $m = \frac{a}{30}$ berechnet, wobei kaufmännisch gerundet wird. Damit ergibt sich die Länge der Monate zu ungefähr 30. Wenn man annimmt, dass $a > 100$, dann liegen die schlimmsten Fälle bei $a = 104, ...$ (Länge 34/35) und $a = 105, ...$ (Länge 26/27), was akzeptabel ist. Die Länge der kurzen Monate ist dann $l_k = \left\lfloor \frac{a}{m} \right\rfloor$, es wird also abgerundet, die langen Monate sind einen X-Tag länger, $l_l = l_k + 1$. Somit gibt es $m_l = \lfloor a \rfloor - m l_k$ lange Monate, und dementsprechend $m_k = m - m_l$ kurze Monate.

Schaltjahr-Regelung

Die hier vorgeschlagene Schaltjahr-Regelung ist der tatsächlich auf der Erde verwendeten angelehnt. Dabei gibt es alle t_0 X-Jahre ein Schaltjahr, alle t_1 X-Jahre wieder keines, alle t_2 X-Jahre wiederum doch eines, alle t_3 ist wieder kein Schaltjahr usw. Dabei müssen alle t_i natürliche Zahlen sein und außerdem ist t_{i+1} immer durch t_i teilbar.

Die t_i werden folgendermaßen bestimmt: Sei a^* der Nachkommateil von a. In der ersten Stufe ist dann dieser Rest $r_0 = a^*$. In jeder Stufe i (beginnend mit $i = 0$) wird nun die größtmögliche natürliche Zahl t_i gewählt, so dass $\frac{1}{t_i} \geq r_i$ – der Rest wird also „von oben“ möglichst genau angenähert. Für die nächste Stufe ist der Rest dann jeweils $r_{i+1} = -\left(r_i - \frac{1}{t_i}\right)$. Mit diesem neuen Wert wird weiter iteriert. r_i bleibt also immer positiv, aber eigentlich erfolgt das Ausgleichen des Rests immer alternierend, also abwechselnd durch Adddition und Subtraktion. Ein Beispiel: Für $a = 365{,}24219$ ist $r_0 = 0{,}24219$ und $t_0 = 4$ ($1/4 \geq 0{,}24219$, aber $1/5 < 0{,}24219$). Für die nächste Stufe ergibt sich $r_1 = -(0{,}24219 - 0{,}25) = 0{,}00781$.

Die Genauigkeit nimmt natürlich mit der Anzahl der Stufen zu, die Komplexität der Regel jedoch auch. Daher ist die maximale Anzahl der Stufen im Programm einstellbar, Voreinstellung ist 5. Ist die Länge des X-Jahres schon mit weniger Stufen bis auf eine konfigurierbare Restfehlerrate erreicht, so wird die Approximation schon vorher abgebrochen. Die Voreinstellung hier ist eine Abweichung von einem X-Tag in einer Milliarde X-Jahren. Diese Genauigkeit wird aber eventuell mit der maximalen Anzahl Stufen nicht erreicht.

Die Zeitspanne, nach der der Kalender einen X-Tag „falsch geht“, ist durch $T = \frac{1}{r_{j-1}}$ gegeben, wobei j die Anzahl der Stufen bis zum Abbruch ist.

Eine kleine Unschönheit hätte dieses Verfahren, wenn es immer genau so durchgeführt würde. Ist a^* nämlich größer als $0,5$, so wäre zunächst einmal jedes X-Jahr ein Schaltjahr ($t_0 = 1$). Damit ist aber quasi eine Stufe im Kriterium „verschwendet“. Deshalb wird in diesem Fall ein Monat im Schaltjahr verkürzt anstatt verlängert. Der Algorithmus startet dann dafür mit $1 - a^*$ und es wird ein kurzer Monat in einen langen umgewandelt.

Die beschriebene Lösung ist sinnvoll, da sie einen Kompromiss zwischen effizienter Berechnung und einfacher Anwendung einerseits und hoher Genauigkeit andererseits darstellt. Die in der Aufgabenstellung vorgeschlagene Regelung für $a = 250,26192$ kann mit dieser Lösungsidee nicht erreicht werden, da sie obige Teilbarkeitsbedingung nicht erfüllt. Mann müsste wohl die natürlichen Zahlen von 1 an aufwärts durchgehen und für jede mittels ggT-Berechnungen entscheiden, ob ihre Aufnahme in die Regel die Genauigkeit „lohnenswert“ verbessern. Regeln dieser Form dürften zwar im Allgemeinen genauer, aber für Menschen sehr schwer anwendbar sein. Außerdem würde sich das Programm dadurch stark verkomplizieren und ineffizient werden.

Andersherum wäre es eine Verbesserungsmöglichkeit der hier präsentierten Regelung, die bestimmten Teiler auf „glatte“ Zahlen zu runden, bei denen die Teilbarkeit der Jahreszahl auch einfach im Kopf auszurechnen ist. Man könnte z. B. die Anzahl der von 0 verschiedenen Ziffern auf eine beschränken, analog zum Gregorianischen Kalender. Dieser würde sich dann für $a = 365{,}24219$ tatsächlich ergeben.

Beispiele

Vorgegebenes Beispiel 1 (416,78132)

```
Bitte geben sie die Länge eines X-Jahres in X-Tagen ein: 416.78132

Vorgeschlagener X-Kalender
--------------------------
Das X-Jahr hat 14 X-Monat(e), davon
3 kurze(r) X-Monat(e) mit 29 X-Tagen und
11 lange(r) X-Monat(e) mit 30 X-Tagen.
Einer der langen Monate hat in einem Schaltjahr 29 X-Tage statt 30.
Schaltjahr, wenn die Jahreszahl
durch 4 teilbar ist
und nicht durch 28 teilbar ist
oder wiederum durch 224 teilbar ist
und nicht durch 14112 teilbar ist
oder wiederum durch 1157184 teilbar ist
Nach diesem Kalender weicht das Datum erst nach ca. 401801001 X-Jahren um einen X-Tag ab.
```

In diesem Fall ist das Schaltjahr einen X-Tag kürzer als ein Nicht-Schaltjahr. 5 Stufen reichen nicht aus, um eine Genauigkeit von nur einem X-Tag Abweichung in einer Milliarde X-Jahren zu erreichen.

Vorgegebenes Beispiel 2 (365,24219)

```
Bitte geben sie die Länge eines X-Jahres in X-Tagen ein: 365.24219

Vorgeschlagener X-Kalender
--------------------------
Das X-Jahr hat 12 X-Monat(e), davon
7 kurze(r) X-Monat(e) mit 30 X-Tagen und
5 lange(r) X-Monat(e) mit 31 X-Tagen.
Einer der kurzen Monate hat in einem Schaltjahr 31 X-Tage statt 30.
Schaltjahr, wenn die Jahreszahl
durch 4 teilbar ist
```

```
und nicht durch 128 teilbar ist
oder wiederum durch 400000 teilbar ist
Nach diesem Kalender weicht das Datum erst nach ca. 158431065151502 X-Jahren um einen X-Tag ab.
```

Das Ergebnis für das Erdenjahr weicht von der tatsächlichen Regelung in der Monatseinteilung ab sowie auch in der Schaltjahr-Regelung. In der Realität wurde wohl mehr Wert auf glatte Zahlen als auf Genauigkeit gelegt. Dafür geht der hier vorgeschlagene Kalender erst nach ca. 10^{15} X-Jahren um einen X-Tag falsch, im Gegensatz zum Gregorianischen Kalender, der schon nach ca. 3100 X-Jahren einen ganzen X-Tag verliert.

Bewertungskriterien

- Die Länge der Monate sollte nah bei 30 liegen. Mögliche Längen kleiner 25 oder größer 35 können für $a > 100$ eigentlich immer vermieden werden.
- Die Formeln zur Berechnung sind nicht gerade einfach, daher können sich hier leicht Fehler einschleichen, was natürlich nicht passieren soll.
- Ein Schaltjahr sollte immer einen X-Tag kürzer oder länger als ein Nicht-Schaltjahr sein. Ein größerer Abstand ist nie notwendig. Ein verkürztes Schaltjahr macht die Lösung eleganter, wenn der Nachkommateil von a größer 0.5 ist. Das ist nicht verlangt, aber eine Regelung, in der zunächst jedes Jahr Schaltjahr ist, ist nicht akzeptabel.
- Der schwierigste Teil dieser Aufgabe ist das Finden einer Schaltjahr-Regelung. Hier ist eigentlich (fast) alles erlaubt. Üblicherweise wird man die Teilbarkeit der Jahreszahl zu Grunde legen, andere Ideen sind jedoch auch zulässig. Wichtig ist, dass gute Gründe für die gewählte Regelung angegeben werden und dargelegt wird, warum die gewählte Regelung für sinnvoll gehalten wird.
- Die Schaltjahr-Regelung sollte immer näher an die wirkliche Jahreslänge kommen, je mehr Teil-Regeln sie verwendet, sie muss also gegen a „konvergieren“. Das Programm muss aber natürlich immer nach einer endlichen Zahl von Teil-Regeln abbrechen.
- Trifft die Regel exakt die Jahreslänge, so darf die Regel auch nicht länger sein als nötig. Ausnahme sind hier Fehler auf Grund der Fließkomma-Arithmetik, allerdings sollten diese bemerkt worden sein.
- Nicht zu vergessen ist die Angabe der Abweichung der berechneten Regelung von der Realität.

Aufgabe 3: Wie wasserdicht ist Schweizer Käse?

Lösungsidee

Wieviel Luft darf in den Schweizer Käse, so dass der Käse noch wasserdicht ist? Zuviel Luft eröffnet Verbindungen von der Oberseite quer durch den Käseblock, durch die das Wasser fließen und an der Unterseite des Käsewürfels austreten kann. In dieser Aufgabe soll ein Zusammenhang zwischen dem Anteil der Löcher und der Wasserdichtigkeit gefunden werden.

Dazu wird der Käse als dreidimensionales Array betrachtet, bei dem jedes Feld entweder Käse oder Luft sein kann. Die aktuelle Belegung des Arrays wird zu Beginn des Durchlaufs mit dem Zufallsgenerator bestimmt. Der Zufallsgenerator liefert Werte, die zufällig, linear verteilt in einem festgelegten Intervall liegen. Je nach Programmiersprache wird dieses Intervall vom Programmierer vorgegeben (zum Beispiel in Pascal: Random(n) liefert Integer-Werte aus dem Interval [0;n[) oder ist festgelegt (meistens auf den Bereich [0;1]). Ob ein Feld des Arrays aus Käse besteht oder nicht, wird durch Bilden einer Zufallszahl und Vergleich mit dem vorgegebenen p bestimmt. Beispiel in Pascal:

```
if random(101) < p*100 then {Käse} else {Luft}
```

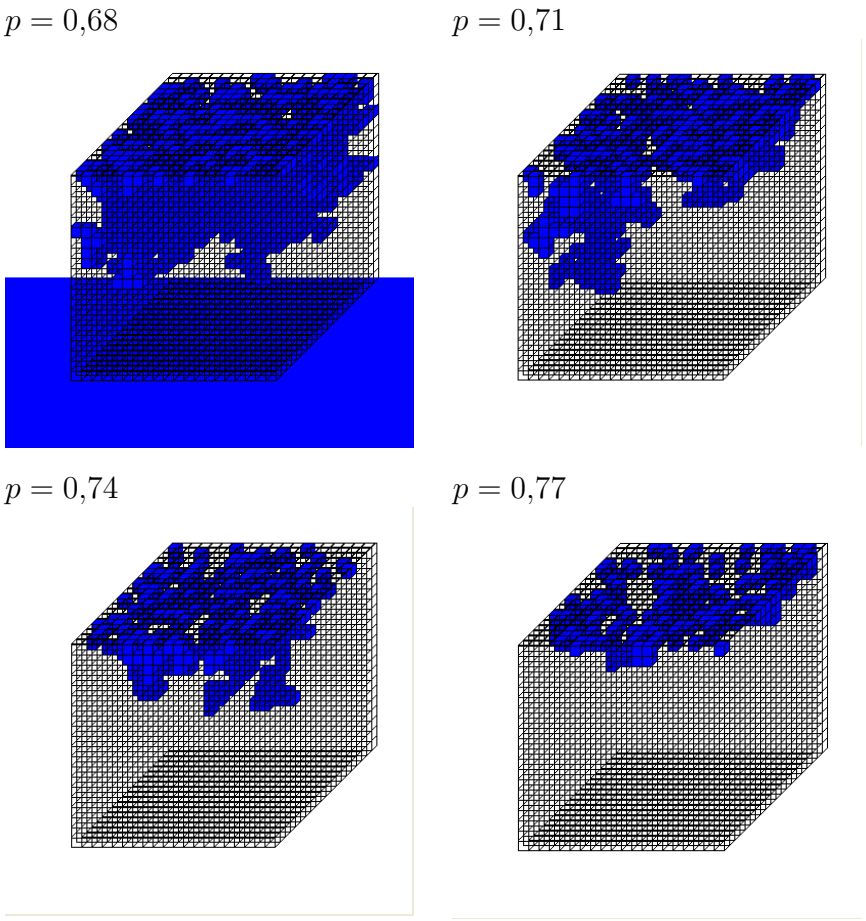
Random(101) liefert dabei eine zufällige ganze Zahl im Intervall [0;100], $p * 100$ gibt den Prozentwert der vorgegebenen Wahrscheinlichkeit ans.

Zur Simulation des Wasserflusses eignet sich zum Beispiel das folgende Vorgehen: Die Luftlöcher in der obersten Schicht des Käseblocks werden in einem Startschritt auf „Wasser“ gesetzt. Jeder Simulationsschritt läuft wie folgt ab: Der Käseblock wird vollständig durchlaufen. Für jede Arrayzelle, die derzeit „Luft“ enthält, wird geprüft, ob es eine benachbarte Arrayzelle (mit gemeinsamer Seitenfläche) gibt, die derzeit den Zustand „Wasser“ hat. Falls ja, wird der Status der Luft-Zelle ebenfalls auf Wasser gesetzt. Der Simulationsschritt wird wiederholt bis entweder keine weitere Veränderung der Zustände im Array erfolgt oder eine Zelle der untersten Ebene den Zustand auf Wasser wechselt. In letzterem Fall ist der Käse undicht.

Eine Alternative wäre jeweils eine Breitensuche von jedem mit Luft gefüllten Element der oberen Ebene des Arrays. Dabei wird zunächst das Feld selbst betrachtet. Für jedes betrachtete Feld, das Luft enthält, werden die Nachbarfelder in die Suchliste aufgenommen und im Anschluss betrachtet. Enthält ein bei der Suche gefundenes Element des Arrays Luft und liegt in der untersten Ebene, so ist der Käse undicht.

Programm-Ablaufprotokoll

Die folgenden Grafiken zeigen Wege des Wassers für verschiedene Werte von p. Sie verdeutlichen, wie schnell sich die Eindringtiefe des Wassers bei relativ kleinen Änderungen in der Käsedichte ändert.



Um den Zusammenhang zwischen p und der Durchflusswahrscheinlichkeit zu ermitteln, werden für verschiedene Werte von p jeweils mehrere (ca. 1000) Simulationen gestartet. Der Käse wird dabei für jeden Durchlauf neu per Zufallsgenerator erzeugt.

Trägt man die Werte für p und die Häufigkeit (in Prozent), dass der Käse undicht ist, gegeneinander auf, erhält man ungefähr die Kurve aus Abbildung 1.

Ein steilflankiger Übergang erfolgt im Intervall [0,65;0,75]. Für kleinere Käsewahrscheinlichkeiten gibt es praktisch immer einen Durchfluss, für größere nie.

Bewertungskriterien

- Der Käse muss zufällig erzeugt werden.
- Der Zufall muss für jeden Rasterwürfel neu herangezogen werden, so dass bei vorgegebenem p die Anzahl der Käsewürfel leicht variieren kann. Immer eine feste Anzahl Käsewürfel zu nehmen (für $p = 0{,}4$ also 3200) und nur für deren Platzierung im Käse den Zufall zu verwenden, ist weniger geeignet.
- Das Wasser darf nur an Seitenflächen von einer Array-Zelle in die nächste fließen. Kanten- oder Eckenübergänge werden ausdrücklich ausgeschlossen.

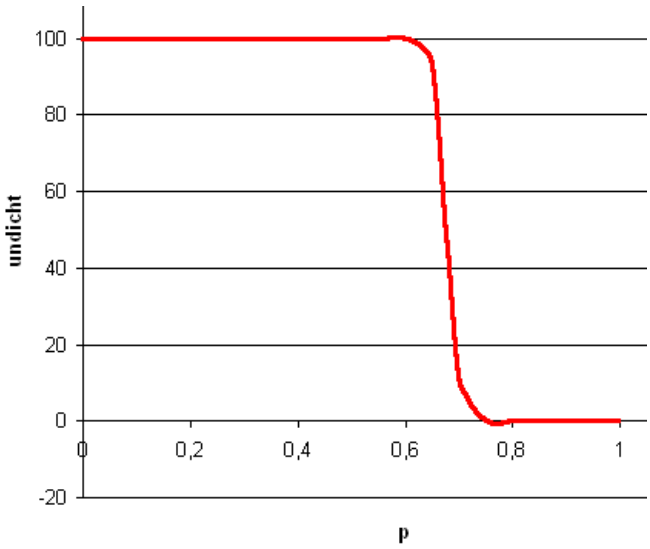


Abbildung 1: Schweizer-Käse-Kurve

- Für jeden Wert von p müssen mehrere Versuche gemacht werden, um Ausreißer aus den Werten zu eliminieren. Eine sinnvolle Anzahl der Versuche liegt in der Größenordnung 100 bis 1000 und ist selbst auf langsamen Rechnern noch problemlos auszuführen. Außerdem müssen ausreichend viele unterschiedliche Werte für p gewählt werden (z.B. mit Abstand 1/100), damit der geschilderte Zusammenhang gut erkennbar wird.
- Der Zusammenhang zwischen p und der Dichtigkeit sollte erkannt sein. Am besten ist es, wenn ein entsprechendes Diagramm vorhanden ist. Aussagen wie „Für $p < 0{,}71$ kommt da meistens Wasser durch.“ reichen in diesem Fall nicht.
- Der steilflankige Übergang sollte ungefähr in dem Bereich zwischen $p = 0{,}65$ und $p = 0{,}75$ liegen, sonst steckt irgendwo ein Fehler.

Aufgabe 4: Klasse Arbeit

Lösungsidee

Die Aufgabe besteht darin, aus den vorhandenen Büchern möglichst wenig Bücher auszuwählen (zu selektieren), so dass jede Aufgabe in mindestens einem der ausgewählten Bücher zu finden ist. Eine bestimmte Menge von Aufgaben (nämlich die, die von älteren Mitschülern gesammelt werden konnte), muss dabei nicht berücksichtigt werden. Sowohl die Information, welche Aufgaben in welchem Buch zu finden sind, als auch die, welche Aufgaben von älteren Mitschülern gesammelt werden konnten, nimmt der Algorithmus als Eingabe vom Benutzer entgegen.

Etwas formeller ausgedrückt ist eine Anzahl von Büchern gegeben, die jeweils eine bestimmte Aufgabenmenge enthalten, und ein Aufgabenarchiv, das die Menge der Aufgaben ist, die von älteren Schülern zusammengesammelt werden konnten. Gefordert ist es, so wenige der gegebenen Bücher wie möglich zu selektieren, dass die Vereinigungsmenge der Aufgaben aller selektierten Bücher gleich der Vereinigungsmenge der Aufgaben aller gegebenen Bücher abzüglich der Aufgaben im Aufgabenarchiv ist.

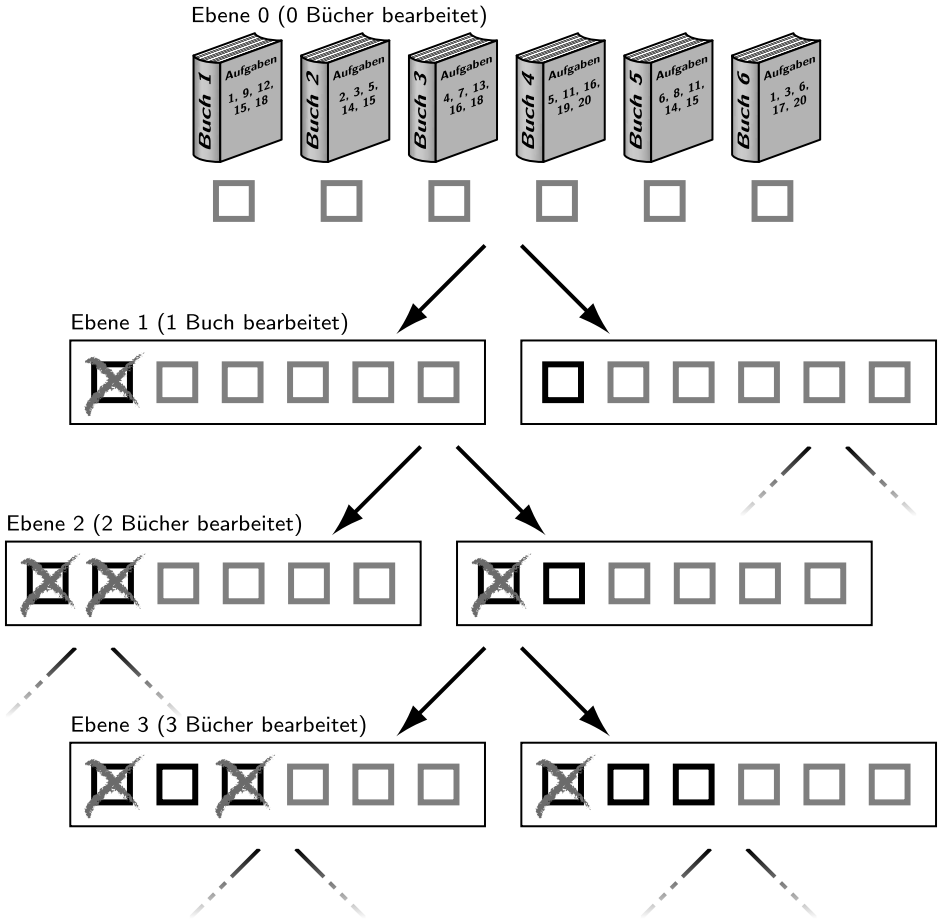
Eine Heuristik (also eine Faustregel zur einfachen Lösung von Problemen, wie sie auch von Menschen gerne verwendet wird) kann diese Aufgabe nicht in allen Fällen optimal lösen. Für einfache Heuristiken lässt sich das leicht an einem Gegenbeispiel zeigen: Wenn zum Beispiel immer das Buch hinzugenommen wird, das die meisten noch nicht in der Selektion enthaltenen Aufgaben abdeckt (eine „Greedy“, also gierige, Heuristik), würde der Algorithmus an den folgenden drei Büchern scheitern: Buch 1 (Aufgaben 3, 4, 5, 6), Buch 2 (Aufgaben 1, 3, 4), Buch 3 (Aufgaben 2, 5, 6), bereits gesammelte Aufgaben gibt es nicht. Offensichtlich besteht die optimale Selektion aus den Büchern 2 und 3, doch die gierige Heuristik würde zunächst Buch 1 selektieren, weil es vier Aufgaben enthält und müsste dann Buch 2 und 3 zusätzlich selektieren, um die Aufgaben 1 und 2 abzudecken.

Der einfachste (und in allen Fällen richtige) Weg, die kleinste Buchselektion zu finden, die die geforderte Bedingung erfüllt, ist es, alle möglichen Selektionen auszuprobieren und sie auf ihre Gültigkeit zu testen. Die kleinste gültige Selektion muss folglich die optimale Lösung für die Aufgabe sein. Das Finden aller möglichen Selektionen wird mittels eines Backtracking-Algorithmus realisiert:

```
bearbeite_buch (buch_nr)
  wenn buch_nr = anzahl_bücher + 1
    überprüfe_selektion
  sonst
    markiere buch buch_nr als selektiert
    bearbeite_buch (buch_nr + 1)
    markiere buch buch_nr als deselektiert
    bearbeite_buch (buch_nr + 1)
```

Listing 1: Pseudocode der Rekursion

Wie man sieht, ruft sich die Selektionsfunktion des Algorithmus pro Durchlauf zwei Mal selbst auf (sie rekurriert), um alle möglichen Buchselektionen zu erzeugen. Wie die Rekursionsebenen mit den Teilselektionen korrespondieren, ist im folgenden Diagramm verdeutlicht:



In jeder Rekursionsebene wird ein Buch betrachtet und entweder zur Buchselektion hinzugefügt oder nicht. Es fällt auf, dass der Algorithmus nach jeder möglichen Selektion aus $i-1$ Büchern genau zwei Mal verzweigt (ein Mal mit Buch i selektiert, ein Mal mit Buch i deselektiert). Erst wenn eine Selektion aus allen zur Verfügung stehenden Büchern vorliegt, wird die Selektion überprüft. Das bedeutet, dass `überprüfe_selektion` (die Komponente des Algorithmus, die eine Selektion auf ihre Gültigkeit überprüft) genau zwei Mal so oft abgearbeitet wird, wenn ein Buch mehr zur Verfügung steht. Folglich wird sie bei 8 zur Verfügung stehenden Büchern 256 Mal, bei 10 zur Verfügung stehenden Büchern 1024 Mal und bei 20 Büchern bereits über eine Million mal abgearbeitet. Die Laufzeit des Algorithmus steigt mit linear zunehmender Buchanzahl also exponentiell an: die Komplexität des Algorithmus' ist $O(2^n)$. Dies hat zur Folge, dass der Algorithmus ab einer gewissen Anzahl an Büchern nicht mehr in einer akzeptablen Zeit terminiert. (Bereits bei 20 Büchern braucht er schon je nach Implementation viele Sekunden.) Selbst wenn die Implementation oder der Computer, auf dem sie läuft, doppelt so schnell wird, kann nur ein weiteres Buch in akzeptabler Zeit bearbeitet werden.

Indem man verhindert, dass offensichtlich zum Scheitern verurteilte Selektionen (solche, die auf keinen Fall Bestandteil der optimalen Selektion sein können) die nächste Rekursionsebene erreichen, kann man die Laufzeit ein wenig verbessern, auch wenn sie prinzipiell exponentiell bleibt. Offensichtlich zum Scheitern verurteilte Selektionen sind zum Beispiel solche, die größer sind, als die beste bereits gefundene gültige. Weiterhin ist es überflüssig, einer Selektion ein Buch hinzuzufügen, das keine neuen Aufgaben enthält, also die Menge der gelösten Aufgaben nicht vergrößert. Die zweite Verbesserung erzielt einen noch höheren Wirkungsgrad, wenn man die Aufgaben, die von älteren Mitschülern gesammelt wurden, in den Büchern ignoriert; denn wenn ein Buch die Selektion nur um eine Aufgabe bereichert, die durch ältere Mitschüler ohnehin zur Verfügung steht, ist das genau so nutzlos, als wenn es gar keine neue Aufgabe enthält.

Zusätzlich kann man versuchen, noch vor Beginn des Backtracking-Verfahrens die Ausgangssituation zu verbessern. Wenn es unter den noch benötigten Aufgaben welche gibt, die nur einmal in einem der Bücher vorhanden sind, muss das betreffende Buch auf jeden Fall in die Selektion aufgenommen werden. Im Backtracking-Prozess spielt es dann keine Rolle mehr, dessen Laufzeit wird halbiert.

Beispiele

Beispiel aus der Aufgabenstellung

```
Bitte geben Sie die Aufgaben per Nummer, separiert durch Leerzeichen an.
Beenden Sie die Aufgabenliste mit einem newline (<return>).
Schließen Sie die Buchspezifikation mit einer leeren Aufgabenliste ab.
Aufgaben in Buch 1: 1 9 12 15 18
Aufgaben in Buch 2: 2 3 5 14 15
Aufgaben in Buch 3: 4 7 13 16 18
Aufgaben in Buch 4: 5 11 16 19 20
Aufgaben in Buch 5: 6 8 11 14 15
Aufgaben in Buch 6: 1 3 6 17 20
Aufgaben in Buch 7: 2 8 10 13 19
Aufgaben in Buch 8: 4 9 10 12 16
Aufgaben in Buch 9:

Bitte geben Sie nun die von älteren Mitschülern gesammelten Aufgaben an:
1 2 4 6 7 8 10 12 14 16 18 20

Starte Backtracking...Fertig
Die beste Selektion besteht aus folgenden Büchern: 1 4 6 7
```

Listing 2: Beispiel 1

Der Bedarf an Aufgaben kann genauso günstig durch den Kauf der Bände 1, 3, 4 und 6 gedeckt werden.

Beispiel 2

```
Bitte geben Sie die Aufgaben per Nummer, separiert durch Leerzeichen an.
Beenden Sie die Aufgabenliste mit einem newline (<return>).
Schließen Sie die Buchspezifikation mit einer leeren Aufgabenliste ab.
Aufgaben in Buch 1: 1 2 3 6 7
Aufgaben in Buch 2: 1 2 3 4
Aufgaben in Buch 3: 5 6 7
Aufgaben in Buch 4:

Bitte geben Sie nun die von älteren Mitschülern gesammelten Aufgaben an:

Starte Backtracking...Fertig
Folgende Bücher werden benötigt: 2 3
```

Listing 3: Beispiel 2

Algorithmen, die immer das Buch auswählen, was am meisten neue Probleme löst, schlagen bei diesem Beispiel fehl.

Bewertungskriterien

- Dieses Problem ist greedy nicht optimal lösbar, deshalb ist eine prinzipiell erschöpfende Tiefen- oder Breitensuche mit exponentieller Laufzeit klar bevorzugt. Eine (allzu einfach zu realisierende) Implementation der Greedy-Heuristik ist nicht akzeptabel, insbesondere wenn nicht erkannt wird, dass optimale Ergebnisse nicht erreicht werden.
- Bei erschöpfender Suche sollte aber eine Abbruchbedingung oder anderweitige Verbesserung implementiert oder zumindest in der Dokumentation angesprochen sein.
- Die Tatsache, dass Aufgaben durch alte Mitschüler zur Verfügung stehen, sollte unabhängig vom generellen Ansatz ausgenutzt werden.
- Die Probleminstanzen sollen, genausowenig wie die Bezeichnungen der Bände und die Maximalwerte von Büchern und/oder Aufgaben, nicht fest verdrahtet sein. Die Aufgabennummern müssen aber nicht unbedingt beliebig wählbar (1 bis 2^{16}) oder unzusammenhängend sein dürfen. Es ist also z.B. in Ordnung, wenn für Aufgaben und/oder Bände nur (zusammenhängende) Nummern von 1 bis zu einem vernünftig gewählten N möglich sind.

Aufgabe 5: Zappen und zählen

Lösungsidee

Modellierung

In der Aufgabe sollen Datensätze von drei bis fünf Rechtecken in ein Raster gezeichnet und alle dabei entstehenden Rechtecke gezählt werden. Allerdings gibt es zunächst zwei grundsätzliche Möglichkeiten, Raster und Rechtecke programmintern darzustellen. Dabei geht es nicht darum, wie das Programm die Aufgabe grafisch aufbereitet (die Abbildungen sind nur zur Veranschaulichung), sondern um die Datenstruktur.

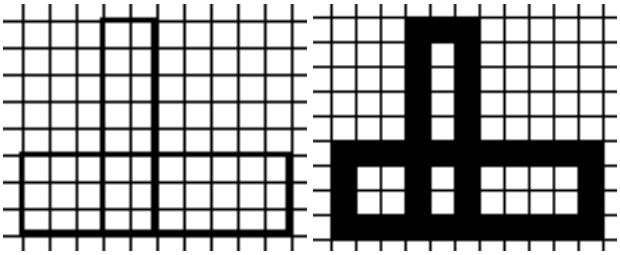


Abbildung 2: Darstellungsmöglichkeit 1 und 2

Gitter (Siehe Abbildung 2, links.) Man stellt sich das Raster als Koordinatensystem vor, in dem die Rechtecke auf Gitterschnittpunkten beginnen und enden. Dazu speichert man, welche der Gitterpunkte verbunden sind; es entsteht also ein Graph aus einer Teilmenge der Gitterschnittpunkte. Für jede Koordinate speichert man, ob sie mit der darüber und der rechts daneben verbunden ist. Um zu testen, ob zwei Koordinaten verbunden sind, muss man bei der linken bzw. unteren Koordinate beginnen und für alle Koordinaten dazwischen testen, ob sie eine Verbindung nach rechts haben.

Kästchen (Siehe Abbildung 2, rechts.) Alternativ teilt man das Raster in Kästchen ein, so dass jedes Koordinatenpaar zu einem Kästchen gehört. Dieses Raster speichert man in einem boolschen Array, das für alle Kästchen eine 1 enthält, die auf einer Rechteckseite liegen. Diese Darstellung ist einfach und kompakt, birgt aber eine Falle: Wird für zwei nebeneinanderliegende Kästchen je eine 1 notiert, wobei sie aber auf unterschiedlichen Rechteckseiten liegen, so erscheinen diese beiden Kästchen verbunden, obwohl sie es eigentlich nicht sein sollten. Das Problem wird in Abbildung 3 veranschaulicht.

Die Lösung dieses Problems ist aber einfach: Beim Malen der Rechtecke verdoppelt man alle Koordinaten und füllt die Kästchen zwischen den Eckpunkten aus (s. auch Abbildung 4.

Für die tatsächliche grafische Darstellung, falls vorhanden, ist die Gitterdarstellung vorzuziehen (wie auch auf der Abbildung auf dem Aufgabenblatt zu sehen). Bei der Datenstruktur ist die Wahl der Modellierung egal, solange man diese korrekt umsetzt (siehe Problem bei der Kästchendarstellung).

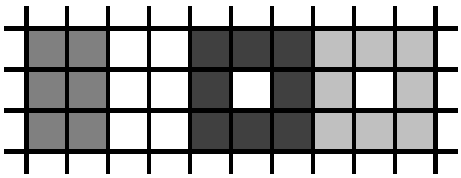


Abbildung 3: Das linke Rechteck sollte als Rechteck erkannt werden, die nebeneinander liegenden Linien rechts aber nicht.

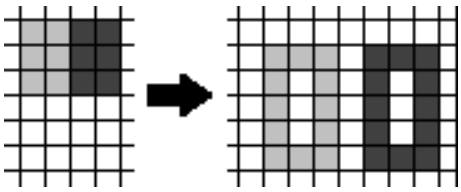


Abbildung 4: Durch die Vergrößerung entsteht ein Abstand zwischen eigentlich nicht verbundenen Kästchen.

Rechtecke zählen – die einfache Lösung

Für die einfache Lösung male man alle Rechtecke in einem Raster (in einer der beiden Darstellungsformen) übereinander. Dann gehe man alle Koordinaten durch und betrachte die, die durch ein Rechteckseitenlinie überdeckt sind (d.h. auf einer Seitenlinie liegen). Für je zwei solcher Koordinatenpaare teste man, falls das erste Koordinatenpaar links und unterhalb des zweiten liegt, ob das durch sie beschriebene Rechteck existiert. Dazu läuft man die vier Seiten des Rechtecks im Raster durch und testet, ob alle Punkte, an denen man vorbeikommt, bedeckt bzw. verbunden sind. Dabei muss darauf geachtet werden, dass von einem Rasterpunkt zum nächsten auch wirklich eine Verbindung besteht. Sonst kann man leicht nebeneinander liegende Rechtecke falsch zählen und z.B. bei R={((1;1),(2;2)),((3;1),(4;2))} auf sechs Rechtecke kommen statt auf zwei.

Die beiden großen Laufzeitfaktoren dieser Lösung sind zum einen, dass man alle Koordinatenpaare durchgeht und zum anderen, dass man für je zwei Punkte alle Seiten entlangläuft - somit ist die Laufzeit von der Größe der Rechtecke abhängig. Allerdings sollte erwähnt werden, dass auch diese Lösung für die in der Aufgabe angegebenen Werte durchaus angemessen und schnell ist!

Verbesserungen

Schnittpunkte berechnen Es ist natürlich unnötig, für alle möglichen Punktpaare durchzuprobieren, ob sie ein Rechteck bilden. Stattdessen kann man auch alle Schnittpunkte von Rechtecken berechnen, die, zusammen mit den Eckpunkten der Eingabe-Rechtecke, die möglichen Eckpunkte der zu zählenden Rechtecke sind.

Für die Berechnung der Schnittpunkte zweier Rechtecke kann man sich entweder alle Möglichkeiten überlegen, wie sich zwei Rechtecke schneiden können, welche Schnittpunkte dabei ent-

stehen und wie man den vorliegenden Fall erkennt, oder man testet für jede Seite eines Rechtecks, ob sie eine Seite des anderen Rechtecks schneidet (16 Tests). Man erhält dann zusammen mit den Eckpunkten der ursprünglichen Rechtecke die Menge aller möglichen Eckpunkte neuer Rechtecke. Daher untersucht man dann für je zwei dieser Punkte, ob sie ein Rechteck aufspannen. Jedes Rechteck kann mit jedem Rechteck maximal vier Schnittpunkte erzeugen, somit erhält man bei n Rechtecken maximal $O(4n^2)$ Schnittpunkte, plus $4n$ ursprüngliche Eckpunkte. Man muss also $O(4n^4)$ Punktpaare durchprobieren.

Komprimieren Wenn man einen Algorithmus benutzt, der von der Größe der Rechtecke abhängig ist (wie die einfache Lösung), kann man die Rechtecke erst komprimieren. Gemeint ist, dass man nicht benötigte Abschnitte der Rechtecke wegnimmt und die Größe des betrachteten Rasters anpasst, Beispiel siehe Abbildung 5. Dadurch erreicht man, dass die Laufzeit des Programms im Weiteren direkt von der Rechteckanzahl abhängig ist. Die Rastergröße ist dann nämlich nur noch davon abhängig, wie viele unterschiedliche x- und y-Koordinaten es gibt.

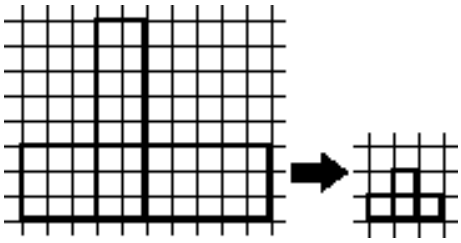


Abbildung 5: Komprimierung von Vierecken

Die Komprimierung bedeutet, dass man alle Zeilen und Spalten entfernt, die keine Eck- oder Schnittpunkte enthalten. Da Schnittpunkte aber nur an Koordinaten auftreten, wo bereits eine Linie und somit ein Eckpunkt war, muss man nur die Eckpunkte berücksichtigen.

Zur Realisierung kann man zum Beispiel so vorgehen: Alle x-Koordinaten in ein Array eintragen, sortieren und gleiche Zahlen entfernen. Anschließend in allen Koordinaten die x-Koordinaten durch die Nummer der Stelle ersetzen, an der sie im Array gespeichert wurden. Werden z.B. die Koordinaten 3, 6, 8 und 13 verwendet (und nur diese), kann man sie durch die Werte 0, 1, 2 und 3 ersetzen.

Verbindungen mitberechnen Wenn man die Schnittpunkte berechnet, ist es auch möglich, direkt mitzuberechnen, welche Verbindungen zwischen den Schnittpunkten und Eckpunkten bestehen, so dass die Eingabe direkt in einen Graphen mit Kanten zwischen Eck- bzw. Schnittpunkten umgerechnet wird. Dabei sind am Anfang alle Eckpunkte ursprünglicher Rechtecke verbunden. Neue Schnittpunkte sind auf jeden Fall mit den Eckpunkten der beiden beteiligten Seiten verbunden. Außerdem sind sie verbunden mit allen Schnittpunkten, die bisher auf diesen Seiten gefunden sind (das muss man also irgendwie speichern). Man muss ebenfalls berücksichtigen, dass sich Rechtecke in einem Eckpunkt schneiden können und dann die anderen Eckpunkte der beteiligten Seiten verbunden werden.

Alternative Lösungsansätze

„Greedy“ Eine denkbare Alternative ist, die Rechtecke nacheinander im Raster zu platzieren und nur Vergleiche von je zwei Rechtecken durchzuführen. Man verwaltet dazu in einer Datenstruktur die aktuell im Raster vorhandenen Rechtecke. Sobald man ein neues hinzufügt, testet man für jedes vorhandene Rechteck, ob es sich mit dem neuen schneidet, und fügt alle neu entstandenen Rechtecke in die Datenstruktur hinzu. Hierfür muss man sich im Voraus überlegen, wie viele Möglichkeiten es gibt, dass sich zwei Rechtecke schneiden, und wie viele neue Rechtecke dabei jeweils entstehen. Natürlich können Rechtecke als Schnitt mehrerer Rechtecke entstehen, und man muss darauf achten, dass man neue Rechtecke nur hinzufügt, wenn sie „echt“ neu sind (siehe Abb. 6).

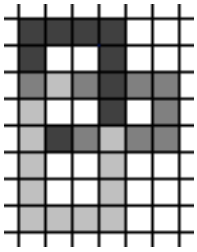


Abbildung 6: Das „bunte“ Rechteck ist der Schnitt zweier unterschiedlicher Rechteckpaare.

Das größere Problem ist aber, dass nicht alle Rechtecke wirklich als Schnitt zweier Rechtecke entstehen. Man betrachte dazu Abbildung 7 links. Rechts ist dargestellt, wie man das Prinzip auf beliebig viele Rechtecke erweitern kann. Ein neues Rechteck kann also aus beliebig vielen anderen entstehen. Dieser Sonderfall verhindert, dass ein Algorithmus, der immer nur Rechteckpaare vergleicht, funktionieren kann!

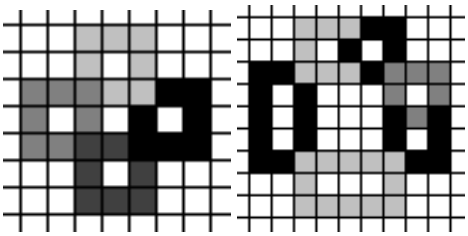


Abbildung 7: Links: Vier Rechtecke erzeugen ein neues.
Rechts: Sechs Rechtecke erzeugen ein neues in der Mitte.

Menschlicher Algorithmus Wie würde man vorgehen, wenn man die Rechtecke selbst zählen sollte? Ein intuitiver Ansatz ist es, zunächst alle kleinsten Rechtecke zu zählen, also alle Rechtecke, die nicht aus mehreren zusammengesetzt sind. Dann zählt man alle Rechtecke, die aus zwei solchen Rechtecken zusammengesetzt sind, dann die, die aus drei zusammengesetzt sind usw. Dieses Verfahren kann man auch programmieren.

Zuerst muss man die kleinsten Vierecke finden. Dazu startet man jeweils an einem Eck-/ oder Schnittpunkt und sucht das kleinste Rechteck, das an dieser Stelle z.B. nach oben rechts aufgespannt wird. Hierzu läuft man nach rechts, bis man eine Verbindung nach oben findet. In

gleicher Weise sucht man von dem Punkt aus nach oben bis zur ersten Abzweigung nach rechts (siehe Abbildung 8). Abschließend muss man testen, ob die beiden erreichten Punkte ein Rechteck aufspannen. Wenn nicht, muss man entweder nach oben oder nach rechts weitersuchen, dort, wo es keinen vollständigen Weg gab (siehe Abbildung 8).

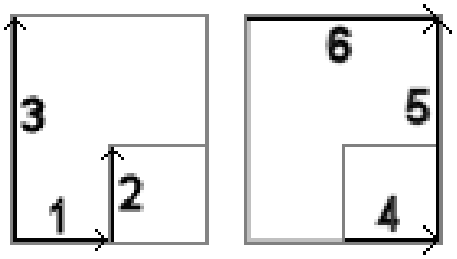


Abbildung 8: Der menschliche Algorithmus, der die kleinsten Rechtecke sucht.

Wenn man ein Rechteck gefunden hat, kann man aufhören. Es gibt jedoch eine Falle: in manchen Fällen beginnen mehrere kleinste Rechtecke im gleichen Punkt (siehe Abbildung 9), deshalb muss man genau das Gleiche aus einer anderen Richtung machen, damit man alle findet. Am Schluss fügt man alle gefundenen Rechtecke zusammen, wobei man doppelte natürlich aussortiert.

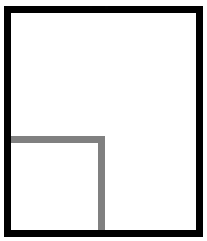


Abbildung 9: Das große Rechteck wird nicht gefunden, wenn man von unten links ausgeht, daher muss man aus verschiedenen Richtungen suchen.

Im zweiten Schritt muss man die Rechtecke zusammenlegen. Das macht man am einfachsten, indem man alle Rechtecke durchguckt und für jedes Rechteckpaar testet, ob sie nebeneinander liegen. Neu gefundene Rechtecke muss man seiner Rechteckdatenstruktur hinzufügen, so dass sie im Weiteren berücksichtigt werden. Man erhält auf diese Weise eine Liste, die alle Rechtecke beinhaltet.

Beispiele

Beispiel 1 aus der Aufgabenstellung (Abbildung 10 links).
R={((3;2),(13;5)),((6;2),(8;10))}
Lösung: 8
(((3;2),(6;5)), ((6;2),(8;5)), ((6;5),(8;10)), ((8;2),(13;5)),
((3;2),(8;5)), ((6;2),(8;10)), ((6;2),(13;5)), ((3;2),(13;5)))

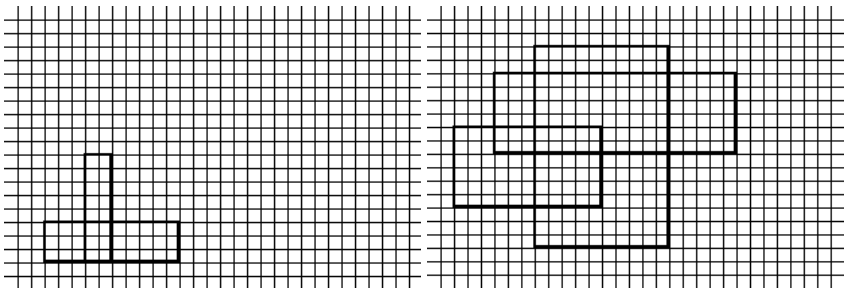


Abbildung 10: Beispiele 1 und 2

Beispiel 2 aus der Aufgabenstellung (Abbildung 10 rechts).
R={((2;6),(13;12)),((8;3),(18;18)),((5;10),(23;16))}
Lösung: 19
(((2;6),(8;12)), ((5;10),(8;12)), ((5;12),(8;16)), ((8;3),(18;10)), ((8;6),(13;10)), ((8;10),(13;12)), ((8;16),(18;18)), ((18;10),(23;16)), ((8;10),(18;16)), ((5;10),(8;16)), ((5;10),(13;12)), ((8;3),(18;16)), ((8;6),(13;12)), ((8;10),(18;18)), ((8;3),(18;18)), ((8;10),(23;16)), ((5;10),(18;16)), ((5;10),(23;16)), ((2;6),(13;12)))

Bewertungskriterien

- Das Lösungsprogramm sollte keine Probleme mit mehrfach vorkommenden Schnitt- oder Eckpunkten haben.
- Rechtecke, die Schnitt unterschiedlicher Rechteckpaare sind, dürfen nicht mehrfach gezählt werden.
- Wer einen „greedy“ oder ähnlichen Ansatz realisiert, sollte den Sonderfall berücksichtigen oder zumindest erkennen, dass Rechtecke von mehr als zwei Rechtecken erzeugt werden können.
- Auch aus anderen Gründen darf das Programm nicht falsch zählen.
- Insbesondere müssen die Kanten mitverfolgt werden, um bei zwei nebeneinander liegenden Eingabe-Rechtecken nicht auf sechs Rechtecke zu kommen.
- In der Aufgabenstellung war gefordert, für alle Beispiele sowohl ein Raster mit den Eingabe-Rechtecken zu zeichnen (ruhig per Hand) als auch die sich ergebenden Rechtecklisten auszugeben.

Junioraufgabe: Wörter-Ketten

Lösungsidee

Aus einer Datei müssen die zur Verfügung stehenden Wörter eingelesen werden. Anschließend werden rekursiv alle möglichen Wörter-Ketten gebildet, die den Regeln der Aufgabenstellung entsprechen und zum Schluss diejenigen Lösungen ausgewählt, die möglichst lang sind. Ein Wort kann an eine bestehende Wörter-Kette angefügt werden, wenn seine ersten drei Buchstaben mit den letzten drei Buchstaben der bestehenden Kette übereinstimmen. Das Beispiel, das in der Aufgabenstellung zur Erklärung des Begriffs der Wörter-Kette angegeben ist, würde übrigens von einer korrekten Lösung nicht gefunden, da es auch Vier-Buchstaben-Überlappungen enthält.

Drei wichtige Punkte sind dabei zu beachten:

- Das Einlesen der Wortliste muss auch aus einer Datei funktionieren, die so beschaffen ist wie die vorgegebene Datei faust.txt. Das heißt, die Liste der zu verkettenden Wörter muss auch aus einem „normalen“ Text extrahiert werden können. Dabei ist einiges zu bedenken: Wörter sollten nur einmal in die Liste aufgenommen werden. Das gerade im vorgegebenen Faust-Text häufig vorkommende Apostroph kann aus den Wörtern entfernt werden. Kleine und große Buchstaben werden nicht unterschieden. Damit Umlaute und andere besondere Zeichen richtig behandelt werden, muss darauf geachtet werden, dass die Zeichenkodierung des Textes so vorliegt, wie das Programm es erwartet.
- Man muss sich für ein Kriterium entscheiden, nach dem die Länge einer Wörter-Kette zu messen ist: denkbar ist entweder die Zahl der Buchstaben der Wörter-Kette oder aber die Zahl der Wörter, aus denen die Kette zusammengesetzt ist. Beides sind sinnvolle Kriterien, man muss sich aber natürlich für eines entscheiden – oder beide Varianten umsetzen und dem Benutzer die Wahl lassen.
- Wörter mit weniger als drei Buchstaben können ignoriert werden, da sie nicht zur Bildung von Wörter-Ketten beitragen können. Wörter mit drei Buchstaben sind dann interessant, wenn die Zahl der Wörter entscheidend ist. Sie können aber ebenfalls ignoriert werden, wenn die Zahl der Buchstaben maßgeblich ist, da diese durch das Hinzufügen eines Wortes mit drei Buchstaben nicht erhöht wird.

Beispiele

Wir haben die Lösung als Java-Programm implementiert. Dem Programm wird beim Start als Kommandozeilenargument der Name einer Datei übergeben, in der die zu verwendenden Wörter enthalten sind, also z.B.: `java WoerterKetten faust.txt`

Beispiel 1 (Pflichtbeispiel aus der Aufgabenstellung mit maximaler Buchstabenzahl):

```
> java WoerterKetten faust.txt
1 Kettenwort(e) mit Länge 29 gefunden:
befriedigungestümenschenliebe
```

Man sieht, dass das Wort aus den Wörtern Befriedigung, ungestümen und Menschenliebe entstanden ist.

Beispiel 2 (Pflichtbeispiel aus der Aufgabenstellung mit maximaler Wortanzahl):

```
> java WoerterKetten faust.txt
1 Kettenwort(e) bestehend aus 5 Wort(en) gefunden:
mangeliebtestamentschlafen
```

Die Wörter-Kette besteht also aus man, Mangel, geliebtes, Testament und entschlafen.

Bewertungskriterien

- Natürlich dürfen weder der Text noch einzelne Wörter noch andere Dinge im Quelltext fest verdrahtet sein.
- Das Programm muss seine Wortliste aus einem normalen Text einlesen können.
- Die Entscheidung für oder gegen Wörter mit drei Buchstaben sollte begründet sein; wie sie ausfällt, ist gleich.
- Entsprechend der Aufgabenstellung darf ein Wort nicht mehrfach in einer Wörter-Kette vorkommen.
- Auch wenn nur die Ausgabe für faust.txt gefordert ist, sollte das Funktionieren des Programms durch weitere Beispiele verdeutlicht werden, wie im allgemeinen Teil des Aufgabenblatts gesagt wird. Da viele dies übersehen haben, haben wir darauf geachtet, dass niemand allein wegen dieses Fehlers die zweite Runde verpasst hat.

Aus den Einsendungen: Perlen der Informatik

Allgemeines

w00t! w00t! Erste Runde 23. BWINF! w00t! Gut, dass es diesen Wettbewerb gibt. Ich liebe ihn und alle, die damit zu tun haben, gibt er einem doch die einzigartige Möglichkeit, das Zeug, was die Freunde nicht verstehen und die Lehrer nicht hören wollen und was selbst meinen kleinen Kuschelbären regelmäßig in den Wahnsinn treibt, aufzuschreiben.

Da lacht das Herz! Gleich nochmal, aber vielleicht etwas seriöser:
Übrigens finde ich die Aufgaben sehr schön, und es macht Spaß, sich damit zu beschäftigen.

Das erste Ziel war es, den ersten Teil der Lösungsidee umzusetzen.

Da unsere Lösungsidee sehr tiefgehend war, gab es keine Umsetzungsschwierigkeiten.

Wir bitten um Verständnis und wir entschuldigen uns für unsere Unfähigkeit, aber wir haben alles gegeben.

Athlon 2k, 128 MB RAM, ewig nicht mehr defragmentiert.

Es erfolgte eine ausführliche Dokumentation [...] mit taktischen Kommentaren.

Man bemerke, wie unwichtige Kommentare den Leser verwirren, während wichtige Kommentare ausgelassen werden.

Der gesamte Code ist eine einzige Faselei.

tratsCH trATsch

Zusätzlich wäre ein Chatone ziemlich blöd, wenn er jemanden direkt anmacht.

Da jeder Chatone von jedem erreichbar ist, kann keiner über jemanden lästern. Demzufolge muss man sich andere Gesprächsthemen suchen.

Ator kann Vtor schreiben, dass Etor und Ftor laut pupsen.

`m4$terX` kann `^h00r4xx` Tratsch über `!V@#`, `!33700r`, `@3}{t`, `-]MDS[-`, `m1k@` und `*/-+` erzählen.

Kosmische Schaltjahre

... und wahrscheinlich hat Papst Gregor auch nur gebruteforced!

Eines der Hauptprobleme dieses Systems ist es zweifellos, die erstellten Regeln in Sätze zu verpacken, ohne der verwendeten Sprache bleibende Schäden zuzufügen.

Wie wasserdicht ist Schweizer Käse?

Wie wasserdicht ist Schweißer Käse?

Seit dieser Runde des Wettbewerbs Informatik bleiben im Supermarkt scheinbar immer die nicht würfelförmigen Käse liegen; die würfelförmigen sind kaum noch zu bekommen.

. . . eine gute Nährung für die Durchflusswahrscheinlichkeit.

. . . Subroutine, welche den Käse rekursiv flutet.

Um den Zusammenhang zwischen p und der Durchflusswahrscheinlichkeit herauszufinden, habe ich für verschiedene Werte von p durch Festhalten der Enter-Taste auf dem OK-Button Versuchsreihen mit ca. 1000 Versuchen gemacht.

Mein Programm rechnet diesen Graphen, indem es für jeden ganzzahligen Prozentsatz p 50 zufällige *hier das Plural von Käse einsetzen* generiert und sie auf ihre Durchfließbarkeit prüft.

Der Würfel wird um 2 Kästchen in jeder Dimension vergrößert, so dass der eigentliche Käse noch eine Rinde erhält.

Der Betrag des Anstiegs im eigentlichen Abfall kann teilweise sogar mit dem uneigentlichen Grenzwert von unendlich bezeichnet werden.

Klasse Arbeit

Bänder

`fuergaby.pas`

Jeder Band muss mindestens eine Aufgabe beherbergen.

Somit wird am Ende das niedrigste Minimum ermittelt.

Zappen und zählen

. . . wobei das Koordinatensystem umgedreht werden muss, da in der Informatik der Nullpunkt links oben und nicht links unten, wie in der Mathematik, liegt.

Die Konstante „maxmax“ enthält die maximale Anzahl Rechtecke. Da sich „Fernsehen“ und „anspruchsvoll“ ausschließen, muss man sie sicher nicht erhöhen.