27. Bundeswettbewerb Informatik, 2. Runde

Lösungshinweise und Bewertungskriterien

Allgemeines

Es ist immer wieder bewundernswert, wie viele Ideen, wie viel Wissen, Fleiß und Durchhaltevermögen in den Einsendungen zur zweiten Runde eines Bundeswettbewerbs Informatik stecken. Um aber die Allerbesten für die Endrunde zu bestimmen, müssen wir die Arbeiten kritisch begutachten und hohe Anforderungen stellen. Von daher sind Punktabzüge die Regel und Bewertungen über die Erwartungen (5 Punkte) hinaus die Ausnahme. Lassen Sie sich davon nicht entmutigen! Wie auch immer Ihre Einsendung bewertet wurde: Allein durch die Arbeit an den Aufgaben und den Einsendungen hat jede Teilnehmerin und jeder Teilnehmer einiges dazu gelernt; den Wert dieses Effektes sollten Sie nicht unterschätzen.

Bevor Sie sich in die Lösungshinweise vertiefen, lesen Sie doch bitte kurz die folgenden Anmerkungen zu Einsendungen und den beiliegenden Unterlagen durch.

Terminprobleme Einige Einsender gestehen ganz offen, dass Ihnen die Zeit zum Einsendeschluss hin knapp geworden ist, worauf wir bei der Bewertung leider keine Rücksicht nehmen können. Abiturienten macht der Terminkonflikt mit der Abiturvorbereitung Probleme. Der ist für eine erfolgreiche Teilnahme sicher nicht ideal. In der zweiten Jahreshälfte läuft aber die zweite Runde des Mathewettbewerbs, dem wir keine Konkurrenz machen wollen. Also bleibt uns nur die erste Jahreshälfte. Aber: Sie hatten etwa vier Monate Bearbeitungszeit für die zweite BWINF-Runde. Rechtzeitig mit der Bearbeitung der Aufgaben zu beginnen war der beste Weg, Konflikte mit dem Abitur zu vermeiden.

Dokumentation Es ist sehr gut nachvollziehbar, dass Sie Ihre Energie bevorzugt in die Lösung der Aufgaben, die Entwicklung Ihrer Ideen und die Umsetzung in Software fließen lassen. Doch ohne eine gute Beschreibung der Lösungsideen, eine übersichtliche Dokumentation der wichtigsten Komponenten Ihrer Programme, eine gute Kommentierung der Quellcodes und eine ausreichende Zahl sinnvoller Beispiele (die die verschiedenen bei der Lösung des Problems zu berücksichtigenden Fälle abdecken) ist eine Einsendung wenig wert. Bewerterinnen und Bewerter können die Qualität Ihrer Einsendung nur anhand dieser Informationen vernünftig einschätzen. Mängel können nur selten durch gründliches Testen der eingesandten Programme ausgeglichen werden – wenn diese denn überhaupt ausgeführt werden können: Hier gibt es häufig Probleme, die meist vermieden werden könnten, wenn Lösungsprogramme vor der Einsendung nicht nur auf dem eigenen, sondern auch einmal auf einem fremden Rechner getestet würden. Insgesamt sollte die Erstellung des schriftlichen Materials die Programmierarbeit begleiten oder ihr teilweise sogar vorangehen: Wer nicht in der Lage ist, Idee und Modell präzise zu formulieren, bekommt keine saubere Umsetzung in welche Programmiersprache auch immer hin.

Bewertungsbögen Kein Kreuz in einer Zeile bedeutet, dass die genannte Anforderung den Erwartungen entsprechend erfüllt wurde. Vermerkt wird also in der Regel nur, wenn davon abgewichen wurde – nach oben oder nach unten. Ein Kreuz in der Spalte „+“ bedeutet Zusatzpunkte, ein Kreuz unter „–“ bedeutet Minuspunkte für Fehlendes oder Unzulängliches. Dabei haben die Marken nicht immer den gleichen Einfluss auf die Gesamtbewertung, sondern können je nach Einsendung unterschiedlich gewichtig sein. Die Schattierung eines Feldes bedeutet, dass die entsprechenden Zusatz- bzw. Minuspunkte in der Regel nicht vergeben wurden.[1]

Lösungshinweise Bei den folgenden Erläuterungen handelt es sich um Vorschläge, nicht um die einzigen Lösungswege, die wir gelten ließen. Wir akzeptieren in der Regel alle Ansätze, die die gestellte Aufgabe vernünftig lösen und entsprechend dokumentiert sind. Einige Dinge gibt es allerdings, die – unabhängig vom gewählten Lösungsweg – auf jeden Fall diskutiert werden müssen. Zu jeder Aufgabe gibt es deshalb einen Abschnitt, indem gesagt wird, worauf bei der Bewertung letztlich geachtet wurde, zusätzlich zu den grundlegenden Anforderungen an Dokumentation (insbesondere: klare Beschreibung der Lösungsidee, genügend aussagekräftige Beispiele) und Quellcode (insbesondere: Strukturierung und Kommentierung, gute Übersicht durch Programm-Dokumentation).

[1] Ausnahmen bestätigen die Regel: Bei Aufgabe 1 sind beim 5. Bewertungspunkt (‘sinnvolle „triviale“ Fehler …’) beide Spalten fälschlicherweise schattiert; hier konnten Minuspunkte vergeben werden.

Aufgabe 1: k-Sortierer

1.1 Darstellung im Rechner

Bei diesem Punkt geht es nicht um die grafische Darstellung eines Sortierer-Netzwerkes, etwa am Bildschirm, sondern um die Verwendung geeigneter Datenstrukturen zur Repräsentation (und Verarbeitung) der für ein Sortierer-Netzwerk wesentlichen Daten. Allerdings kann eine rechner-interne Darstellung durchaus eine recht direkte grafische Entsprechung haben.

Es ist z.B. nahe liegend, ein Sortierer-Netzwerk als Graph aufzufassen. Dazu betrachtet man jeden Ausgang eines Sortierers als einzelnen Knoten des Graphen und alle Eingänge des Sortierers zusammen ebenfalls als einen Knoten. Die Kanten des Graphen ergeben sich dann zunächst aus der Verkabelung des Sortierers. Außerdem muss jeder Sortierer selbst noch in der Graphenstruktur realisiert werden. Dazu muss der „Eingangsknoten“ mit allen „Ausgangsknoten“ auf besondere Weise verbunden werden, beispielsweise über speziell markierte Kanten. Insgesamt wird die Darstellung als Graph (und die Speicherung des Graphen mit statischen oder dynamischen Datenstrukturen) recht kompliziert.

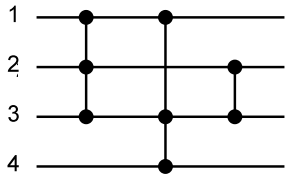


Abbildung 1.1: Ein gültiges (also korrekt sortierendes) 4-Elemente-Sortiernetzwerk.

Zu einer anderen Möglichkeit führt die Beobachtung, dass in der grafischen Darstellung eines Sortierer-Netzwerkes immer genau k Leitungen übereinander laufen. (Jede Senkrechte durch eine Netzwerkzeichnung, die weder einen Sortierer noch einen Kreuzungspunkt schneidet, schneidet genau k Leitungen.) Das heißt, man kann die Leitungen von oben nach unten durchnummerieren. Dann speichert man für jeden Sortierer, welche Leitungen verändert bzw. verglichen werden. An überkreuzten Leitungen speichert man entsprechend, welche Leitungen dort vertauscht werden. Kreuzungen kann man sich sogar sparen, wenn erlaubt wird, dass die von einem Sortierer „bearbeiteten“ Leitungen nicht alle unmittelbar aufeinander folgen müssen. Diese Repräsentation hat eine einfache grafische Entsprechung[2]; das Sortiernetzwerk in Abbildung 1.1 entspricht genau dem Sortierer-Netzwerk der Aufgabenstellung. Zur Auswertung bzw. Berechnung der Funktion des Netzwerks läuft man dann von links nach rechts und führt die durch die einzelnen Sortierer gegebenen Operationen auf einem Array von Eingabewerten durch.

[2]Diese wird etwa von Donald E. Knuth in Band 3 von „The Art of Computer Programming“ verwendet.

Eine Variante dieser Idee ist, die Schaltung (also das Sortierer-Netzwerk) als ein Programm zu betrachten. Dazu definiert man drei Prozeduren, die einer Leitungskreuzung und den Sortierern entsprechen. Sie modifizieren die globale Variable a: ein Array, in dem zunächst der Zustand der Eingangsspannungen an den Positionen 1 bis k gespeichert ist.

```
procedure intersection(i,j:longint);
var t:longint;
begin
  t:=a[i];
  a[i]:=a[j];
  a[j]:=t;
end;

procedure twoComparer(i,j:longint);
begin
  if i>j then twoComparer(j,i)
  else if a[i]>a[j] then intersection(i,j);
end;

procedure threeComparer(i,j,k:longint);
begin
  twoComparer(i,j);
  twoComparer(j,k);
  twoComparer(i,j);
end;
```

Dann kann man jedes Netzwerk als eine Liste von Aufrufen dieser Prozeduren betrachten. Das Beispiel der Aufgabenstellung etwa entspricht folgenden Aufrufen:

```
threeComparer(1,2,3);
threeComparer(1,3,4);
twoComparer(2,3);
```

Das Ausführen dieser Anweisungen berechnet nun automatisch das Ergebnis des Netzwerks für die Anfangswerte von a.

Bequeme Eingabe von Netzwerken

Realisiert man die eben beschriebene Darstellungsvariante in einer interpretierten Sprache, die dynamisches Codeladen unterstützt (z.B.: JavaScript, Ruby, Scheme, ...) kann man die das Netzwerk beschreibenden Programmaufrufe gleich als Eingabeformat für Netzwerke nehmen. Zur textuellen Eingabe eines Netzwerks kann auch eine selbst definierte kleine Beschreibungssprache verwendet werden, deren Ausdrücke dann analysiert und in die oben beschriebenen Prozeduraufrufe umgesetzt werden müssen. Je nach Gewohnheit oder Geschmack kann die textuelle Eingabe eines Netzwerks durchaus als „bequem“ bezeichnet werden.

Die Alternative ist natürlich eine grafische Oberfläche zur Eingabe von Netzwerken. Hier sollte aber darauf geachtet werden, dass der Vorteil, den man vom intuitiven Platzieren der Sortierer hat, nicht durch allzu mühsames Kabelziehen wieder verloren geht. Schön ist, wenn nach dem Platzieren automatisch Kabelverbindungen gezogen werden, z.B. abhängig von der Position der Sortierer. Diese Ausgangssituation könnte dann vom Benutzer so weit wie nötig verändert werden.

1.2 Prüfverfahren

„Triviale Gründe“

Wie sich herausstellen wird, ist die komplette Prüfung eines k-Sortierers auf korrektes Funktionieren aufwändig. Die Aufgabenstellung weist explizit darauf hin, dass „triviale“ Fehler-Fälle schnell entdeckt werden sollen. Bei der Überprüfung eines Sortierer-Netzwerkes können deshalb zunächst einige definierende Eigenschaften überprüft werden wie die Zyklenfreiheit der Vernetzung oder die 1:1-Zuordnung zwischen Ein- und Ausgängen. Auch sollte es in einem Netzwerk keine direkte Verbindung von einem Eingang zu einem Ausgang geben. Solche oder gar noch grundlegendere Eigenschaften (alle Aus-/Eingänge sind verkabelt, etc.) können schon bei der Eingabe bzw. bei der Generierung der Netzwerke überprüft, ihre Verletzung also von vornherein verhindert werden.

Aber auch etwas weniger offensichtliche Mängel können ohne großen Aufwand erkannt werden. Beispiel: Von jedem Eingang eines Netzwerkes muss jeder Ausgang erreichbar sein, da sonst nicht alle möglichen Eingabetupel sortierbar wären. Mit einer Breitensuche durch das Netzwerk lässt sich diese Bedingung sicher stellen.

Funktionstest

Ein vollständiger Funktionstest erfordert im schlechtesten Fall die Anwendung eines Netzwerkes auf alle unterschiedlichen Anordnungen der Zahlen 1 bis k, das sind $k!$ viele. Dies ist natürlich zu vermeiden. Zunächst sind einige Ansätze möglich, mit denen Sortierfehler schnell gefunden werden können, die aber die Korrektheit nicht garantieren.

Man kann leicht überprüfen, ob der höchste und der niedrigste Wert korrekt verarbeitet werden, da man weiß, dass diese in jedem Sortierer durch den obersten bzw. untersten Ausgang geleitet werden. Somit muss man nur sicherstellen, dass sie für jeden Eingang auf die korrekte Weise durchgeleitet werden. Ebenso kann man für alle $k*(k-1)$ Anordnungsmöglichkeiten der beiden niedrigsten oder beiden höchsten Werte das Ergebnis berechnen, da man dann weiß, dass alle anderen Werte größer/kleiner sind. Macht man dies für alle $k/2$ kleinsten/höchsten, muss man $2*\frac{k!}{k/2!}$ Möglichkeiten durchprobieren, was $\frac{k/2!}{2}$-mal so schnell ist wie das Durchprobieren aller Werte.

Bei größeren Netzwerken könnte eine probabilistische Herangehensweise sinnvoll sein, bei der mehrere zufällige Eingangskombinationen durchprobiert werden. Hier müsste man dann eine sinnvolle Strategie zur Wahl der Zufallsverteilungen angeben und begründen, warum die Wahrscheinlichkeit für das Übersehen eines Fehlers gering ist.

Auch eine Kombination der beiden vorherigen Verfahren wäre denkbar: Zuerst beweist man, dass die niedrigsten und höchsten Werte korrekt sind, und testet dann einige Zufallskombinationen, um manche Fehler in der Mitte aufspüren zu können.

Sichere Prüfung der Korrektheit

Für kleine k (also auch für $k = 4$) ist ein vollständiger Korrektheitstest durchaus machbar, für größere k aber nicht. Ein wenig mehr Spielraum ist drin, wenn man zur sicheren Überprüfung der Korrektheit das 0-1-Prinzip[3] heranzieht. Dieses besagt, dass ein Sortierer-Netzwerk eine beliebige Folge (in $\mathbb{N}$ oder $\mathbb{R}$) genau dann korrekt sortiert, wenn jede Folge im $K_2 = \{0,1\}$ korrekt sortiert wird. Das bedeutet: man kann den 2-Sortierer als Funktion $f : K_2^2 \to K_2^2$ betrachten mit $f_2(a,b) = (a \vee b, a \wedge b)$. (Also: der erste Ausgang ist wahr, wenn mindestens ein Eingang wahr ist, der zweite, wenn es beide sind.) Entsprechend ist der 3-Sortierer:

$$f_3(a,b,c) \;=\; (a \vee b \vee c, (a \wedge b) \vee (b \wedge c) \vee (a \wedge b), a \wedge b \wedge c)$$

Und der 4-Sortierer:

$$\begin{aligned} f_4(a,b,c,d) \;=\; & (a \vee b \vee c \vee d, \\ & (a \wedge b) \vee (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d) \vee (c \wedge d) \vee (c \wedge d), \\ & (a \wedge b \wedge c) \vee (a \wedge b \wedge d) \vee (a \wedge c \wedge d) \vee (b \wedge c \wedge d), \\ & a \wedge b \wedge c \wedge d) \end{aligned}$$

Für den k-Sortierer testet man entsprechend im ersten Ausgang, ob ein Eingangswert vorhanden ist, im zweiten, ob es mindestens zwei gibt, im dritten drei usw. Damit kann man jedes Netzwerk als booleschen Ausdruck schreiben, indem man für jeden Eingang den Ausdruck für den Ausgang des angeschlossenen Sortierers einsetzt (rekursiv, wenn es eine längere Kette gibt). Auf diese Weise braucht man für einen k-Sortierer $\sum_{i=0}^{k} \binom{k}{i} = 2^k$ Terme, was im Vergleich zu $k!$ eine deutliche Verbesserung darstellt, sofern man in der Lage ist, die beiden booleschen Ausdrücke einigermaßen schnell zu vergleichen. (Da nur 'und' und 'oder' vorkommen, kann man es eventuell über Normalformen leicht ausrechnen, bei vielen Sortierern im Netzwerk könnten die Ausdrücke aber auch sehr kompliziert werden.)

[3] http://theorie.informatik.uni-ulm.de/Lehre/SS3/PA/sort.pdf

1.3 Billigste 4-Sortierer

Hier muss man für jede Zahl von n 2-Sortierern und m 3-Sortierern überprüfen, welche am billigsten sind. Dazu kann man systematisch „von unten“ alle (n,m)-Kombinationen durchgehen, dafür Netzwerke bauen (falls möglich) und prüfen. Eine Obergrenze für die Anzahl der Sortierer im Netzwerk liefert das korrekt funktionierende Netzwerk, das aus möglichst wenigen 2-Sortierern gebaut werden kann. Von daher ist es geschickt, zuerst $(n,0)$-Kombinationen zu probieren, bis die Obergrenze gefunden ist. Auch eine Untergrenze lässt sich bestimmen: Für den vollständigen Vergleich von vier Elementen (jedes mit jedem) benötigt man $3+2+1=6$ paarweise Vergleiche. Ein 3-Sortierer erledigt drei Vergleiche, aber bei Hintereinanderschaltung (in einem 4-Sortierer) wird mindestens ein Vergleich zweimal ausgeführt. Zwei 3-Sortierer genügen also nicht, so dass ein 4-Sortierer aus mindestens drei Teilsortierern bestehen muss. Die Überlegung der maximal benötigten paarweisen Vergleiche liefert gleichzeitig eine Obergrenze von sechs 2-Sortierern; allerdings genügen auch schon fünf 2-Sortierer für einen gültigen 4-Sortierer.

Da nur die Anzahl der Sortierer zählt und die Verkabelung irrelevant ist (so lange das Netz funktioniert), bleiben genau drei relevante Netzwerke übrig: eines mit fünf 2-Sortierern, eines mit einem 3-Sortierer und drei 2-Sortierern, sowie eines mit zwei 3-Sortierern und einem 2-Sortierer (vgl. Abbildung 1.2). Drei 3-Sortierer gingen auch, sind aber wegen der Einschränkung in der Aufgabenstellung immer teurer als der vorherige Fall.

Der Einfluss des Preisverhältnisses ist nun recht einfach zu beschreiben. Bei teuren 3-Sortierern werden die Schaltungen ohne diese günstiger, bei billigen die Schaltungen mit wenigen 2-Sortierern. Wenn 3-Sortierer mindest doppelt so teuer sind wie 2-Sortierer, so ist das Netzwerk mit fünf 2-Sortierern am billigsten. Sind sie genau doppelt so teuer, sind alle drei Netzwerke gleich teuer. Sind sie weniger als doppelt so teuer, so ist das Netzwerk aus zwei 3-Sortierern und einem 2-Sortierer am billigsten.

1.4 Mögliche Erweiterungen

Die Aufgabenstellung beschränkt sich auf 4-Sortierer, die aus 2- bzw. 3-Sortierern zu bauen sind. Das lässt offensichtlich Spielraum für die Beschäftigung mit Zahlen $k > 4$. So kann man aus den 2- und 3-Sortierern auch 5-, 6-, 7- …Sortierer bauen und deren Preiskonstellationen studieren. Richtig verallgemeinert wird die Aufgabe, wenn man sich bei der Konstruktion nicht auf 2- oder 3-Sortierer beschränkt, sondern einen n-Sortierer aus allen k-Sortierern mit $2 \leq k < n$ baut.

Außerdem können andere Kosten berücksichtigt werden, wie die maximale Anzahl von in Reihe geschalteten Sortierern, Drahtkosten, Kreuzungskosten (der aus fünf 2-Sortierern gebaute 4-Sortierer benötigt eine Kreuzung, der aus sechs gebaute nicht).

Theoretisch wäre es auch möglich, einen Tintenstrahldrucker so zu modifizieren, dass er eine (elektrisch leitende) Schaltung ausdruckt, die genau einem solchen Sortiernetzwerk entspricht,

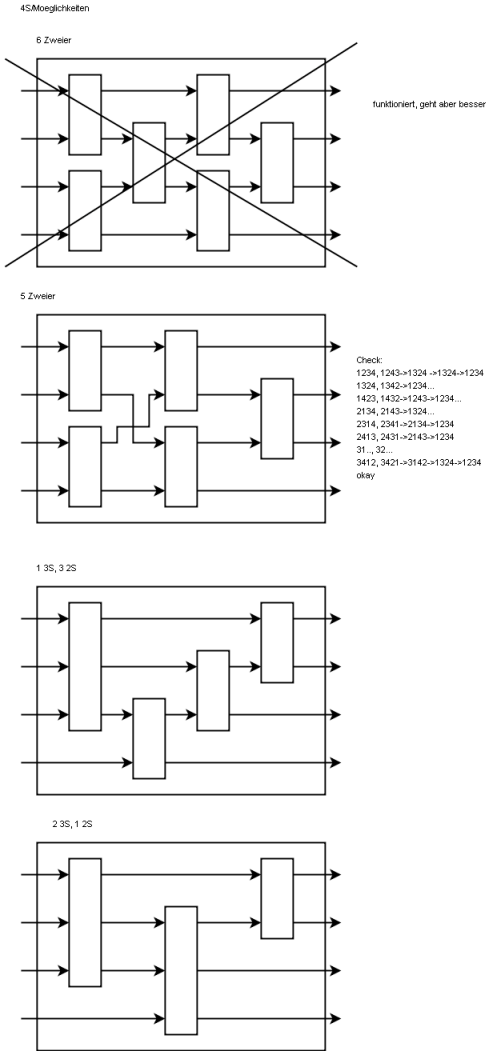


Abbildung 1.2: Billigste 4-Sortierer

und den Korrektheitstest dann auf dieser echten Schaltung durchzuführen (außerdem muss man dann keine Sortierer kaufen ...).

1.5 Bewertungskriterien

- Darstellung: Verschiedene Arten von (programm-internen!) Netzwerkdarstellungen werden akzeptiert. Entscheidend ist, dass die gewählte Darstellung die Bearbeitung der weiteren Aufgabenteile nicht unnötig erschwert. Die alleinige Angabe einer grafischen Darstellung zeigt, dass die Aufgabenstellung in diesem Punkt missverstanden wurde.
- Bequeme Eingabe von Netzwerken: Eine textuelle Eingabe ist grundsätzlich akzeptabel. Generell sollte gelten, dass die Verkabelung einfach angegeben werden kann. Bei den Prozeduraufrufen (s.o.) etwa ergibt sich die Verkabelung aus den Prozedur-Parametern. Manuelles Ziehen aller Kabel ist eher lästig.
- Prüfverfahren: Zunächst sollten die Grundeigenschaften von k-Sortierern gesichert sein (insbesondere die Zyklenfreiheit), idealerweise schon bei deren Generierung bzw. Eingabe, spätestens aber bei der Funktionsprüfung. Außerdem sollten einige sinnvolle „triviale“ Funktions-Fehler-Fälle beschrieben und in entsprechende Methoden zu ihrer Entdeckung umgesetzt sein.

  Es dürfen auf keinen Fall korrekte Netzwerke für fehlerhaft erklärt werden. Fehler beim Maximum bzw. Minimum müssen erkannt werden, was auch durch eine vollständige Prüfung sichergestellt wird. Eine sichere Korrektheitsprüfung sollte zumindest beschrieben, ein Verzicht auf deren Umsetzung (etwa wegen zu hoher Laufzeit) begründet sein.
- Finden der billigsten Sortierer: Die drei möglichen billigsten Netzwerktypen müssen erkannt sein. Es muss außerdem begründet sein, dass dies alle 4-Sortierer sind, „die die billigsten sein können“. Wer auf ein programmiertes Verfahren zur Ermittlung der billigsten Sortierer verzichtet, sollte eine mathematisch hieb- und stichfeste Argumentation liefern; reine Plausibilitätsüberlegungen genügen nicht. Die Auswirkungen des Preisverhältnisses müssen diskutiert und angegeben werden, insbesondere die Kostengleichheit für alle drei Typen bei einem Verhältnis von 1:2.
- Ein einzelner Bonuspunkt hätte vergeben werden können, falls bemerkt worden wäre, dass „Ida Osten“ ein Anagramm für „T. A. Edison“ ist, den berühmten amerikanischen Bastler und Erfinder.

Aufgabe 2: Kunst mit drei Ecken

2.1 Kriterien

Wir formulieren einige Kriterien für schöne Dreiecke, die Möglichkeit der Umsetzung wird zunächst außer Acht gelassen:

1. Jeder Punkt auf dem Bild gehört nur zu einem Dreieck. Dieses Kriterium muss bei der Lösung unbedingt erfüllt sein, da laut Aufgabenstellung aus $3n$ Punkten genau n Dreiecke entstehen sollen.
2. Die Dreiecke überlappen sich nicht.
3. Jedes Dreieck soll eine Fläche haben, d.h. seine drei Punkte liegen nicht auf einer Linie.
4. Die Dreiecke sollten möglichst gleichmäßig im Kreis verteilt sein.
5. Die Seiten innerhalb eines Dreiecks sind etwa gleich lang.
6. Alle Dreiecke sind ähnlich groß.
7. Bei symmetrisch angeordneten Punkten ist das Ergebnis ebenfalls symmetrisch.
8. Ineinander geschachtelte Dreiecke können als besonders schön angesehen werden.
9. als Erweiterung: Dreiecke sollen ein Bild annähernd darstellen.

Da die Anzahl der Punkte immer durch drei teilbar ist, ist das erste Kriterium in allen Fällen erfüllbar. Es ist ein Ausschlusskriterium, das von jedem Vorschlag erfüllt werden muss. Die anderen Kriterien sind Optimierungskriterien, die möglichst gut erfüllt werden sollen. Dies kann insbesondere bei ungünstiger Anordnung der Eckpunkte nur unzureichend möglich sein. Wie wir noch sehen werden, ergeben sich bei bestimmten regelmäßigen Mustern Probleme.

Schon drei Punkte auf einer Linie als Eingabe sorgen dafür, dass das dritte Kriterium nicht erfüllt werden kann; wir müssen also darauf achten, ob bestimmte Kriterien oder deren Kombinationen überhaupt erfüllbar sind. Weiterhin hängen manche der Kriterien sehr stark zusammen, so dass eine Lösung, die ein bestimmtes Kriterium vollständig umsetzt, oft auch gleichzeitig andere Kriterien erfüllt. Dieser Zusammenhang lässt sich vor allem bei den Kriterien vier bis sechs beobachten.

Um den Rahmen dieser Musterlösung nicht zu sprengen, beschränken wir uns auf die Betrachtung des ersten bis sechsten Kriteriums.

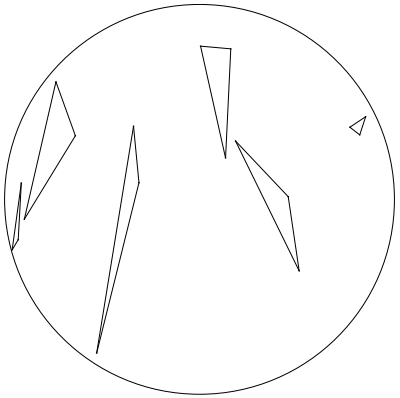


Abbildung 2.1: Beispiel 1 des BWINF-Materials mit einer Scanline gelöst.

2.2 Scanline

Ein einfacher, aber durchaus praktikabler Ansatz ist ein Scanline-Algorithmus. Dabei läuft eine Linie (die so genannte Scanline) von links nach rechts (bzw. oben nach unten) durch das Bild und fügt nach und nach die Punkte zu einer Liste hinzu (man sortiert die Punkte also nach X- bzw. Y-Koordinaten). Sobald drei Punkte „gesammelt“ wurden, wird aus diesen ein Dreieck gebildet und die Scanline fährt fort.

Dieser Algorithmus ist sehr einfach zu implementieren – ein Sortierverfahren, ein Schleife und eine grafische Ausgabe reichen aus – und hat eine sehr angemessene Laufzeit: $O(n \log n)$. Der Hauptaufwand des Verfahrens besteht lediglich im Sortieren der Punkte.

Welche Kriterien erfüllt dieser Ansatz gut? Das zwingende erste Kriterium wird offensichtlich nie verletzt. Auch das zweite Kriterium wird immer erfüllt, da die Punkte in n Bereiche aufgeteilt werden, wobei die Linien nur innerhalb dieser Bereiche verlaufen und dort drei Punkte verbinden. Doch schon das dritte Kriterium bereitet Probleme, z.B. falls die ersten drei Punkte (der sortierten Liste) auf einer Linie liegen.

Das Verfahren erweist sich jedoch als ungeeignet, um auch das vierte bis sechste Kriterium einzuhalten, da die Dreiecke nur nebeneinander und nicht übereinander positioniert sein können; das Ergebnis sieht für uns deswegen nicht sehr schön aus, da im Allgemeinen sehr schmale, lange Dreiecke entstehen.

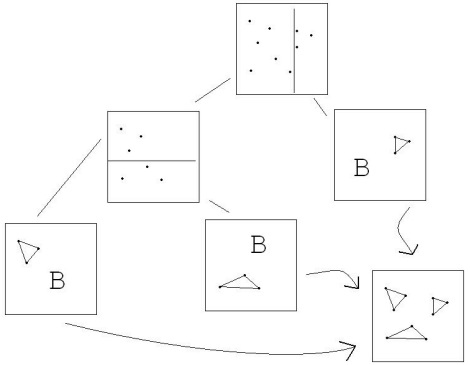


Abbildung 2.2: Ein Rekursionsbaum für ein Beispiel mit neun Punkten

2.3 Divide and Conquer

Der Grundgedanke dieses Verfahrens besteht darin, unsere Punktmenge rekursiv in immer kleinere Teilmengen aufzuteilen, bis eine Teilmenge nur noch drei Punkte enthält und zu einem Dreieck wird. Werden die Punkte dabei immer möglichst gleichmäßig aufgeteilt, so sollte auch eine möglichst gleichmäßige Verteilung von Dreiecken entstehen.

Zunächst werden die Punkte nach X-Koordinaten sortiert. Falls zwei Punkte auf gleicher Höhe liegen, entscheidet die Y-Koordinate. Die Anzahl der Punkte ist immer durch drei teilbar. Bei einer geraden Anzahl wird die sortierte Liste einfach halbiert. Bei einer ungeraden Anzahl enthält die erste Teilliste drei Punkte mehr als die zweite. Man kann sich leicht davon überzeugen, dass die Anzahl der Punkte so in jedem Rekursionsschritt durch drei teilbar ist. Es bietet sich an, die Punkte abwechselnd waagerecht und senkrecht zu partitionieren, also abwechselnd nach X- und Y-Koordinaten zu sortieren. Die Rekursion terminiert bei einer Anzahl von drei Punkten, dieser Fall ist der so genannte Basisfall der Rekursion.

Dieses Verfahren kann durch seine Art der Partionierung, die berücksichtigt, dass die Teilprobleme auch eine durch drei teilbare Anzahl an Punkten erhalten, das erste Kriterium immer erfüllen. Vom Prinzip her ähnlich und wie im Bild 2.2 veranschaulicht, teilt der Algorithmus den Kreis in Teilbereiche mit jeweils drei Punkten auf, sodass nur innerhalb dieser Teilbereiche Linien gezogen werden. Damit ist auch das zweite Kriterium erfüllt.

Durch die abwechselnd waagerechte und senkrechte Partitionierung der Punkte verteilen sich die Dreiecke gleichmäßig im Kreis, da sie geeignet neben- und übereinander positioniert werden. Damit einher geht eine allgemeine Verringerung der Seitenlängenunterschiede innerhalb der Dreiecke, während auch etwa gleich große Dreiecke entstehen. Damit werden die Kriterien 4 bis 6 erfüllt, vorausgesetzt die Punkte sind gleichmäßig verteilt.

Das Problem des dritten Kriteriums: Linien aus drei Punkten

In den meisten Fällen funktioniert das Divide-and-Conquer-Verfahren problemlos. Allerdings kann es passieren, dass in einem Basisfall drei Punkte auf einer Linie liegen, was natürlich kein schönes Dreieck ergibt; das dritte Kriterium kann verletzt werden.

Im Folgenden wollen wir kurz Möglichkeiten andeuten, auch dieses Kriterium einzuhalten. Die Punkte p_1, p_2 und p_3 daraufhin zu überprüfen, ob sie auf einer Linie liegen, ist simpel:

- berechne die Vektoren $\vec{p_1 p_2}$ und $\vec{p_2 p_3}$;
- falls die beiden Vektoren in die selbe (oder genau entgegengesetzte) Richtung zeigen, dann bilden die Punkte eine Linie;
- ansonsten bilden die Punkte ein Dreieck.

Auf ähnliche Art und Weise lässt sich auch überprüfen, ob mehr als drei Punkte auf einer Linie liegen. Dies könnte man anwenden, um schon früh festzustellen, dass die Punkte eines Teilbereiches auf einer Linie liegen. In diesem Fall kann man (statt senkrecht zur X- oder Y-Achse) senkrecht zu dieser Linie partitionieren. Um dies zu erreichen, könnten die Punkte nach veränderten Koordinaten sortiert werden, d.h. man dreht die Punkte entsprechend des Neigungswinkels der Partitionierungslinie. Für das endgültige Bild werden wieder die ursprünglichen Koordinaten der Punkten verwendet.

Es gibt aber durchaus Beispiele, bei denen ein Basisfall mit einer Linie nicht zu vermeiden ist. Man kann sich dazu zum Beispiel ein 3×3 Raster vorstellen – es gibt keine Aufteilung in Dreiecke, die nach unseren Kriterien schön ist[4].

Zur Lösung eines solchen Problems wäre es nützlich, bei der Erfüllung des Kriteriums, dass sich die Kanten nicht überschneiden, etwas Spielraum zu lassen. In solchen Fällen könnte es erlaubt sein, minimale Überschneidungen zuzulassen oder Dreiecke innerhalb von Dreiecken zu erlauben.

Laufzeit

Die Laufzeit $T(n)$ des Algorithmus' für n Punkte lässt sich sehr gut durch eine Rekursionsgleichung beschreiben:

$$\begin{aligned} T(n) &\approx 2 \cdot T\left(\frac{n}{2}\right) + O(n \log n) \\ T(3) &= O(1) \end{aligned}$$

Wir teilen die Punktmenge in jedem Rekursionsschritt in etwa zwei gleich große Mengen auf und müssen das Problem dafür lösen. Weiterhin müssen wir in jedem Rekursionsschritt die

[4] Wenn man verbietet, dass Dreiecke innerhalb von Dreiecken liegen.

Punktmenge sortieren, was in der Zeit $O(n \log n)$ möglich ist. Der Basisfall von drei Punkten ist in konstanter Zeit lösbar.

Man kann diese Rekursionsgleichung in eine geschlossene Form bringen und erhält $T(n) = O(n \log^2 n)$. Der Divide-and-Conquer Algorithmus ist also etwas langsamer als die Scanline, aber immer noch sehr angemessen in Anbetracht der stark verbesserten Ergebnisse.

2.4 Triangulation

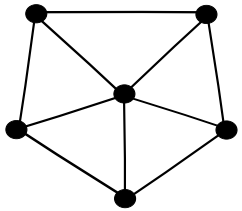


Abbildung 2.3: Die Triangulation eines Pentagramms. Eine Aufteilung in zwei Dreiecke ist nicht allein durch Löschen von Kanten möglich.

Ein weiterer Lösungsansatz liegt nahe und soll deswegen, auch wenn er nicht ohne Weiteres generell funktioniert, kurz vorgestellt werden: die Benutzung einer Triangulation. Es gibt viele Triangulations-Algorithmen, die eine Menge von Punkten vollständig in nicht-überlappende Dreiecke aufteilen. Besonders mit der Delaunay-Triangulation erhält man schöne Dreiecke. Damit wir jeden Punkt für nur genau ein Dreieck verwenden, müssen aber nachträglich einige Kanten aus der Triangulation entfernt werden. Genau hier liegt das Problem: Nicht jede Triangulation kann auf eine Menge von einzelnen Dreiecken reduziert werden. Ein Beispiel ist ein Pentagramm mit einem Punkt in der Mitte, das in Abbildung 2.3 angedeutet ist. Mit geeigneten Verfahren, um eine ungünstige Triangulation zu reparieren, ist eine Lösung möglich.

2.5 Erweiterungen

Farben Wozu sollte Maler Trilikos sich noch die Arbeit des Ausmalens machen? Auch das kann ein Programm leicht erledigen. Besonders gut sind natürlich sinnvolle Kriterien zur Auswahl von schönen Farben und Farbkombinationen.

Muster Zusätzlich zu den Koordinaten der Punkte wird auch noch ein gewünschtes Muster an das Programm übergeben. Die Dreiecke sollen dabei möglichst gemäß des Musters angeordnet sein.

Animationen Anstatt eines langweiligen Standbildes könnte auch eine Animation mit sich bewegenden Dreiecken ausgegeben werden. Hier könnten physikalische Gesetze zur An- und Abstoßung sowie Attraction-Points[5] ins Spiel kommen. Eine Kombination mit einer Muster-Erweiterung wäre interessant.

Interaktive Variante Zusätzlich zu einem automatischen Vorschlagsverfahren, das einen fertigen Vorschlag liefert, kann eine Methode realisiert werden, die Teilvorschläge liefert bzw. dem Künstler Modifikationen eines kompletten Vorschlags erlaubt. Der künstlerischen Kreativität und Individualität von Herrn Trilikos käme das sicher entgegen.

2.6 Beispiele

Das vierte Beispiel enthält die gleichen Punkte wie das dritte, allerdings in anderer Reihenfolge. Die Ausgabe ändert sich dadurch nicht, die Punkte werden ohnehin in jedem Rekursionsschritt nach X- oder Y-Koordinaten sortiert.

In Abbildung 2.4(d) wird deutlich, wie beim Scanline-Ansatz das vierte Kriterium nicht berücksichtigt wird.

In Beispiel 5 tritt das diskutierte Problem der Linien auf. Es gibt mehrere Lösungen, welche die ersten vier Kriterien erfüllen, allerdings ist die Berechnung mit unserem einfachen Divide-and-Conquer-Algorithmus nicht möglich.

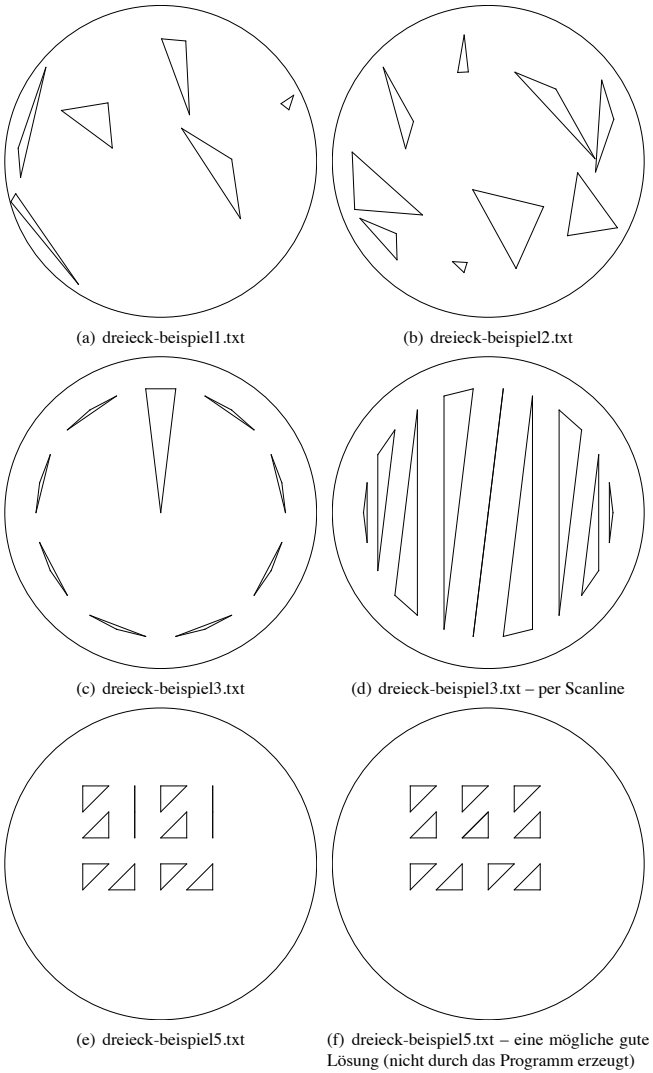
Ein 3×3 Raster, wie in Abbildung 2.5(b) illustriert, lässt tatsächlich keine Aufteilung in Dreiecke zu, die alle vier Kriterien erfüllen, wenn keine Dreiecke innerhalb von Dreiecken erlaubt sind.

Auch große Eingabedateien (dargestellt in Abbildung 2.6 mit 3000 Punkten) sind kein Problem.

2.7 Bewertungskriterien

- Es werden mindestens drei sinnvolle Kriterien für schöne Dreiecke genannt.
- Mindestens ein Kriterium (wünschenswert sind natürlich mehrere) wird vollständig (d.h. in dem Maße, in dem es erfüllbar ist) umgesetzt. Die erfolgreiche Umsetzung besonders schwieriger Kriterien wird belohnt. Ausschließlich triviale Kriterien umzusetzen reicht nicht aus.
- Die $3n$ Punkte einer Eingabedatei dürfen nicht verschoben werden und müssen n Dreiecke ergeben.

[5] Punkte, die alle Dreiecke anziehen.



(a) dreieck-beispiel1.txt

(b) dreieck-beispiel2.txt

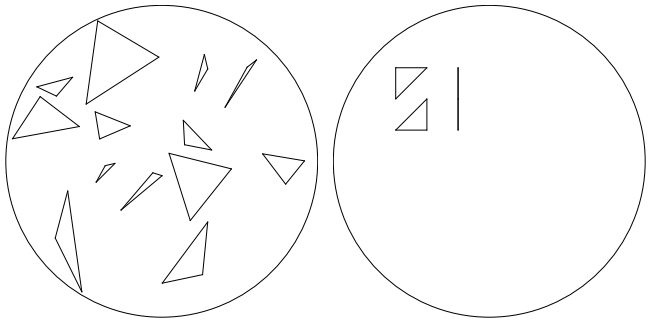
(c) dreieck-beispiel3.txt

(d) dreieck-beispiel3.txt – per Scanline

(e) dreieck-beispiel5.txt

(f) dreieck-beispiel5.txt – eine mögliche gute Lösung (nicht durch das Programm erzeugt)

Abbildung 2.4: BWINF-Beispiele (1)



(a) dreieck-beispiel6.txt (b) ein problematisches Raster aus neun Punkten

Abbildung 2.5: BWINF-Beispiele (2)

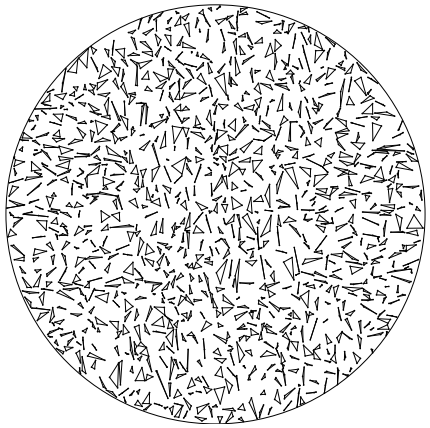


Abbildung 2.6: 1000 Dreiecke

- Bei einem schwierigen Kriterium darf eine angemessene Umsetzung auch rechenaufwändiger sein als bei einem einfachen Kriterium.
- Der Sonderfall eines Dreiecks ohne Fläche sollte zumindest erkannt und diskutiert werden. Es sollte zumindest der Versuch unternommen werden, diesen Fall auch zu behandeln. Es genügt nicht, flächenlose Dreiecke einfach zu akzeptieren. Eine gute Lösung des Problems ist ein großer Pluspunkt. Ähnlich positiv ist, wenn das Programm sicher erkennt, dass eine problematische Linie von Punkten nicht zu vermeiden ist.
- Problematische Fälle, in denen ein Kriterium schlecht oder nur bedingt erfüllt werden kann, sollten diskutiert werden. Die wenigen Einsendungen, in denen dies gemacht wurde, erhielten Pluspunkte.
- Eine zumindest grobe Laufzeitbetrachtung (polynomielles oder exponentielles Wachstum) ist mit Blick auf Beispiele mit vielen Punkten sinnvoll, wird aber nicht erwartet. Eine genaue Betrachtung wird belohnt.
- Eine grafische Ausgabe der Lösungen ist natürlich Pflicht. Nützliche Bibliotheken dürfen dazu verwendet werden.
- In der Dokumentation wird beschrieben, wie vollständig die einzelnen Kriterien umgesetzt sind.
- Alle Beispiele aus dem Material tauchen in Ablaufprotokollen auf.

Aufgabe 3: Content Provider Unit

3.1 Lösungsidee

Das gewünschte Programm soll ausgehend von Eingabedaten, die scheinbar wenig Informationen enthalten, in der Lage sein, einem beliebigen Nutzer gute Vorschläge zu unterbreiten. Dabei ist nur bekannt, welcher Benutzer welche Filme zu Ende gesehen hat.

Die erste Hürde ist das Einlesen und Speichern der Daten. Dazu eignen sich am besten zwei Hashtabellen für Filme und Nutzer, mit denen gespeichert wird, welche Filme und Nutzer es gibt. Für jeden Eintrag ist somit ein schneller Zugriff, d.h. in konstanter Zeit, gewährleistet.

Das Hauptaugenmerk sollte jedoch auf die Bestimmung der Empfehlungswürdigkeit eines Films für einen bestimmten Nutzer gelegt werden. Folgende Kriterien sind dabei sinnvoll:

- *globales Kriterium:* die Anzahl, wie oft ein Film überhaupt gesehen wurde. Wir nehmen an, dass besonders häufig gesehene Filme auch besonders gut sind und schlagen diese häufiger vor.
- *„spirituelles" Kriterium:* Hier versuchen wir z.B. Independent-Film-Liebhaber und Hollywood-Romantiker voneinander zu trennen und Filme nur der passenden Gruppe vorzuschlagen. Dabei gibt es viel Spielraum in der Umsetzung.
- *soziales Kriterium:* Das ist nur als Erweiterung sinnvoll: Filmfavoriten von Freunden sind zu bevorzugen. Wir gehen davon aus, dass Freunde auch einen ähnlichen Filmgeschmack haben (darüber lässt sich natürlich streiten).

Das Programm soll zu einem gegebenen Benutzer die Empfehlungswerte aller Filme im System berechnen und danach die fünf besten auswählen. Mit der Zeit sollten sich damit bei der „spirituellen" Komponente bestimmte Geschmäcker herauskristallisieren. So mag zum Beispiel eine Benutzergruppe die Filme A, B und C. Hat ein Benutzer A und C bereits gesehen, B aber noch nicht, so sollte ihm nach Möglichkeit Film B vorgeschlagen werden. Anmerkung: Das Modell geht davon aus, dass es so etwas wie einen Filmgeschmack gibt. Das muss theoretisch nicht der Fall sein, allerdings hat sich vor allem im Bereich der Musik (vgl. Audioscrobbler/last.fm) gezeigt, dass solch ein statistisches System sehr gut funktioniert.

Mit Blick auf Teil 3 muss das Programm in der Lage sein, ganze Reihen von Vorschlägen zu liefern und dabei auf die Auswahl des Nutzers zu reagieren. Dabei muss banalerweise beachtet werden, dass nicht einfach die Filme vorgeschlagen werden, die den Geschmack des Nutzers am besten treffen; das könnten sonst immer die gleichen sein. Eine Wiederholung in den Vorschlägen ist akzeptabel, wenn sie nicht zu häufig ist.

Umsetzung

Es liegt nahe, zunächst ein Maß für die Empfehlungswürdigkeit eines Films zu definieren. Wir benötigen also eine Funktion e : Benutzer $\times$ Film $\to \mathbb{R}$, die zu einem Benutzer b und einem Film f angibt, wie sehr der Film f dem Benutzer b zu empfehlen ist. Je höher $e(b,f)$, desto besser. Diese Funktion sollte durch die Teilfunktionen, die die verschiedenen Kriterien darstellen, kombinierbar sein, also:

$$e(b,f) = \alpha \cdot e_{\text{global}}(b,f) + \beta \cdot e_{\text{spirituell}}(b,f)$$

Dies erlaubt uns, durch die Wahl geeigneter Parameter α und β den Einfluss des globalen und des spirituellen Kriteriums zu gewichten. Unser Ziel ist es, um intuitiv verständliche Werte zu erhalten, den Bereich der Empfehlungswerte auf Zahlen zwischen 0 und 1 einzuschränken, wobei 1 den Film im höchsten Maße anrät und 0 den Film gar nicht empfiehlt.

Global Die Wertbestimmung für das globale Kriterium liegt auf der Hand: Man iteriere über die Filmliste für jedes Mitglied und inkrementiere in der Filmtabelle die Häufigkeit für den jeweiligen Film. Zur Normierung teilt man diese Zahl nun durch die Gesamtanzahl der Views (= Anzahl der Zeilen in der Eingabedatei).

Spirituell Wir definieren eine Funktion g : Benutzer $\times$ Benutzer $\to \mathbb{R}$, die zu jedem Benutzerpaar angibt, wie verwandt sie im Geiste sind. Wir benutzen bei der Herleitung folgende Notation:

A – Menge der Filme, die b und b' beide gesehen haben
n – Anzahl der Filme, die b gesehen hat
m – Anzahl der Filme, die b' gesehen hat

Zwei Benutzer sind sich ähnlich, wenn es viele Filme gibt, die beide gesehen haben. Dies entspricht der Anzahl der Elemente in der Menge A, also $|A|$. Zur Normierung teilen wir das Doppelte dieses Wertes durch $n+m$, also die Gesamtanzahl der gesehenen Filme der Benutzer von b und b'. Damit liegt der Wert von g zwischen 0 und 1, und wir erhalten zunächst als Formel:

$$g(b,b') \quad = \quad \frac{2 \cdot |A|}{n+m} \tag{1}$$

Die spirituelle Bewertungsfunktion berechnet den Empfehlungswert eines Films f für einen Benutzer b anhand der Ähnlichkeit zu allen anderen Benutzern. Dabei werden die Ähnlichkeiten mit Benutzern b' addiert. Um nun Werte zwischen 0 und 1 zu erhalten, teilen wir die Summe durch die Anzahl der Summanden, das ist die Anzahl, wie oft der entsprechende Film insgesamt von allen Benutzern gesehen wurde.

$$
\begin{aligned}
a(b',f) &= \begin{cases} 1 & \text{falls } b' \text{ Film } f \text{ gesehen hat} \\ 0 & \text{falls } b' \text{ Film } f \text{ nicht gesehen hat} \end{cases} \\
e_{\text{spirituell}}(b,f) &= \frac{1}{N} \cdot \sum_{b' \in \text{Benutzer}, b \neq b'} a(b',f) \cdot g(b,b') \\
\text{mit } N &= \text{Anzahl der Summanden mit } a(b',f) \neq 0
\end{aligned}
$$

Wahl der Parameter α und β Die globale Wertung eines Films ist nur wichtig, falls für die spirituelle Bewertung nicht genug Daten zur Verfügung stehen, falls also der Benutzer bis jetzt kaum Filme gesehen hat. Umgekehrt kann bei einem ausgeprägten Filmgeschmack auf die globale Wertung verzichtet werden. Es bietet sich also an, α und β in Relation zur Anzahl x der gesehenen Filme eines Nutzers zu setzen:

$$
\begin{aligned}
\alpha &= \frac{1}{2^x} \\
\beta &= 1 - \alpha
\end{aligned}
$$

Die Parameter α und β summieren zu 1 auf, sodass der Empfehlungswert zwischen 0 und 1 liegt. Offensichtlich wird der Wert für α sehr schnell sehr klein und damit wird die globale Bewertung der Filme immer weniger berücksichtigt. Das ist sinnvoll, da es sich bei e_{global} auch eher um eine „Notlösung“ handelt, falls nicht genügend Daten vorhanden sind, um den Filmgeschmack eines Nutzers hinreichend zu bestimmen. So steht uns zum Beispiel am Anfang der Simulation in Teilaufgabe 3 keine spirituelle Bewertung der Filme zur Verfügung. Sobald der Benutzer einige Filme gesehen hat, fällt die globale Gewichtung praktisch weg, da die Vorschläge von $e_{\text{spirituell}}$ viel zutreffender sind.

Für den Umgang mit Nutzern, für die noch wenig Daten vorliegen (insbesondere also neuen Nutzern), kann auch folgende Idee hilfreich sein: Die Filme werden statistisch „geclustert“; es werden also Gruppen von Filmen gebildet, die sich bezüglich der Menge ihrer bisherigen Betrachter ähnlich sind. Einem neuen Nutzer können Repräsentanten dieser Gruppen vorgeschlagen werden, um einen bezüglich der verschiedenen Geschmäcker möglichst repräsentativen Ausschnitt aus der Menge aller Filme anzubieten.

3.2 Erweiterungen

- Negative Wertungen bei abgebrochenen bzw. abgelehnten Filmen
- Freundschaften und damit soziale Filmvorschläge
- Tags
- Bonus für Lieblingsregisseure/-schauspieler

- Bewertung von Filmen durch die Benutzer
- Verwendung externer Quellen, z. B. aus dem Web

Negative Filmwertungen

Insbesondere in Teilaufgabe 3 lohnt es sich, die CPU ein wenig zu erweitern. Es sollen auch abgebrochene Filme im System gespeichert werden und als negative Wertungen eingehen. Dies kann zu folgenden Situationen führen: Einige Nutzer haben die Filme A und B gesehen, E und F jedoch abgebrochen. Über einen weiteren Nutzer ist nur bekannt, dass er den Film E abgebrochen hat. Entsprechend bekommt er höhere Empfehlungswerte für die Filme A und B. Man könnte auch von einem „Antigeschmack“ sprechen. Da diese Erweiterung sehr effektiv und relativ einfach umzusetzen ist, folgt nun eine genauere Beschreibung. Zunächst erweitern wir die Funktion $g(b,b')$ aus (1), welche die Ähnlichkeit zwischen Benutzern b und b' angibt:

A	–	Menge der Filme, die b und b' beide gesehen oder beide abgebrochen haben
B_1	–	Menge der Filme, die b abgebrochen und b' nicht gesehen hat
B_2	–	Menge der Filme, die b' abgebrochen und b nicht gesehen hat
n	–	Anzahl der Filme, die b gesehen oder abgebrochen hat
m	–	Anzahl der Filme, die b' gesehen oder abgebrochen hat

Falls b' einen Film nicht gesehen hat, kann dies auch daran liegen, dass unser Benutzer den Film von vornherein nicht mag. In diesem Fall würde die Ähnlichkeit zwischen b und b' steigen, falls wir wissen, dass b genau diesen Film abgebrochen hat. Diese Filme sind in der Menge B_1 enthalten, B_2 enthält den umgekehrten Fall (b' hat abgebrochen und b hat den Film nicht gesehen). Es kann natürlich auch sein, dass ein Film nur aus Zufall (noch) nicht gesehen wurde. Da dies wahrscheinlicher ist, je weniger Filme ein Benutzer insgesamt gesehen hat, und somit weniger Aussagekraft enthält, möchten wir diesen Einfluss stark gewichten, wenn der Benutzer viele Filme gesehen hat und schwach gewichten, wenn er noch nicht sehr viele gesehen hat.

Daher skalieren wir $|B_1|$ und $|B_2|$ mit n und m. Diesmal normieren wir jedoch mit $(n+m)^2$. Man kann sich klar machen, dass der Zähler im folgenden Bruch in jedem Fall kleiner oder gleich $(n+m)^2$ ist und daher $g(b,b')$ immer Werte zwischen 0 und 1 liefert:

$$
g(b,b') = \frac{(n+m) \cdot 2 \cdot |A| + n \cdot |B_1| + m \cdot |B_2|}{(n+m)^2}
$$

Außerdem erweitern wir die Funktion $a(b',f)$ um den Wert -1, falls Benutzer b' Film f abgebrochen hat. Somit gehen abgebrochene Filme negativ in die Bewertung ein. $e_{\text{spirituell}}(b,f)$ braucht dazu nicht verändert werden. Es werden jetzt allerdings Empfehlungswerte zwischen -1 und 1 berechnet, wobei -1 von einem Film im höchsten Maße abrät. Der Filmgeschmack

der Testperson in Teilaufgabe 3 wird so wesentlich schneller erkannt, da durch das Abbrechen[6] der ersten vorgeschlagenen Filme ein „Antigeschmack“ entsteht. Ein Testlauf hat ergeben, dass bereits am dritten Tag vom System gute Vorschläge gemacht werden. Noch effizienter wäre diese Erweiterung natürlich, wenn in den Eingabedaten bereits abgebrochene Filme enthalten wären.

```
Tag 1: was anderes gemacht
Tag 2: was anderes gemacht
Tag 3: The Three Burials of Melquiades Estrada (Vorschlag Nr. 4)
Tag 4: Giant (Vorschlag Nr. 2)
Tag 5: El Dorado (Vorschlag Nr. 1)
Tag 6: Little Big Man (Vorschlag Nr. 1)
Tag 7: Way Out West (Vorschlag Nr. 1)
[...]
```

Abbildung 3.1: Ablaufprotokoll für Teilaufgabe 3 mit Erweiterung des Systems

Freundschaften

Die CPU integriert in ein soziales Netzwerk: Die Benutzer können Profile erstellen und Freundschaften schließen – was an sich ja schon Spaß macht, vgl. MySpace und Nachahmer. Die Freundschaften könnten in den Empfehlungswert eingehen: Ein Freund hat einen bestimmten Film gesehen, damit steigt der Wert für diesen Film.

Tags

Ein Programm kann selber nicht gut entscheiden, ob ein Film „Action“, „Komödie“ oder sogar „Kultfilm“ ist. Menschen können das sehr wohl. Die Benutzer sollten daher die Möglichkeit haben, einzelne Filme mit sogenannten Tags, also kurzen beschreibenden Wörtern, zu versehen. Diese Tags sind einfach in das Empfehlungssystem zu integrieren; so können z. B. einem Benutzer, der gerne Komödien schaut, auch weiterhin Komödien empfohlen werden.

Lieblingsregisseure/-schauspieler

Filmfanatiker haben meistens bestimmte Vorlieben, was Regie und Schauspieler angeht. Das System könnte dies nach einiger Zeit erkennen und entsprechend seine Vorschläge anpassen.

[6] Die Ablehnung aller Vorschläge wird mit dem Abbrechen dieser Filme gleich gesetzt.

Bewertung von Filmen

An sich werden die Filme nur in die Kategorien „gesehen“ und „nicht gesehen“ eingeteilt. Es könnte sinnvoll sein, wenn sich der Benutzer nach dem Film eine Minute Zeit nimmt und eine genauere Bewertung, z. B. in Form einer Schulnote, abgibt. Aber können wir das von unseren faulen couch potatoes erwarten?

Externe Quellen

Es könnte Sinn machen, zusätzlich zu der globalen, spirituellen (und ggf. sozialen) Bewertung noch einen weiteren Faktor einzubeziehen. Zum Beispiel sind die Nutzerwertung eines Films in der IMDB[7] und der Pressespiegel auf rottentomatoes.com[8] ziemlich aussagekräftig.

3.3 Grenzen des Modells

- Zeitliche Änderungen im Filmgeschmack werden nicht berücksichtigt. Es ist im System irrelevant, ob ein Film vor drei Jahren oder drei Wochen gesehen wurde.
- Der aktuelle Filmgeschmack eines Benutzers kann von seiner Stimmung abhängig sein.
- FSK-Altersbeschränkungen werden nicht berücksichtigt.

3.4 Laufzeit und Optimierung

Die globale Bewertungsfunktion durchläuft in einer Schleife alle Filme, liegt also in $O(f)$ (f ist die Anzahl der Filme). Kritisch wird es bei der spirituellen Bewertungsfunktion, da hier in zwei geschachtelten Schleifen alle Filme und alle Benutzer durchlaufen werden. Damit liegt der Algorithmus in $O(nf)$ (n ist die Anzahl der Nutzer). Verdoppelt sich beispielsweise die Nutzerzahl, so würde sich die Laufzeit des Programms für eine Vorschlagsliste der selben Größe ebenfalls verdoppeln.

Zur Lösung dieses Problems könnte man die spirituelle Bewertungsfunktion randomisieren und jeweils nur mit einer begrenzten Zahl von zufällig ausgewählten Nutzern vergleichen. Es ist nämlich davon auszugehen, dass die Anzahl der signifikant unterschiedlichen Filmgeschmäcker ab einer genügend großen Nutzerzahl wesentlich kleiner ist als die Anzahl der Benutzer im System. Anders gesagt, viele Menschen haben einen ähnlichen Filmgeschmack. Es reicht also, eine Teilmenge von zufällig ausgewählten Nutzern zu betrachten. Ist die Menge hinreichend groß (also repräsentativ), so werden alle unterschiedlichen Geschmäcker im annähernd richtigen Verhältnis berücksichtigt. Das Prinzip ist das gleiche wie bei Wahlumfragen: es reicht, einen kleinen Prozentsatz der Bevölkerung zu befragen. Mit dieser Methode

[7] http://www.imdb.com
[8] http://www.rottentomatoes.com

bleibt n in $O(nf)$ konstant und die Laufzeit der spirituellen Bewertungsfunktion hat die Komplexität $O(f)$.

3.5 Programmablaufprotokoll

Mit den BWINF-Testdaten erhält man für die Teilaufgaben 2 und 3 folgende Ergebnisse:

```
Content Provider Unit.
==============================
Benutzeranzahl: 1057
Filmanzahl: 545

Teilaufgabe 2:
Vorschläge für Ada L.
------------------------------
The Terminator
Close Encounters of the Third Kind
Seksmisja
Young Frankenstein
Star Wars: Episode III - Revenge of the Sith

Vorschläge für Edsgar D.
------------------------------
Kagemusha
Rang De Basanti
Judgment at Nuremberg
Mutiny on the Bounty
Ying xiong

Vorschläge für Don K.
------------------------------
No End in Sight
Promises
Some Like It Hot
Fiddler on the Roof
Star Wars: Episode III - Revenge of the Sith

Vorschläge für Alan T.
------------------------------
The Blues Brothers
Mulholland Dr.
Sleuth
The Maltese Falcon
The Sixth Sense
```



```
Teilaufgabe 3:

Tag 1: was anderes gemacht
Tag 2: was anderes gemacht
Tag 3: was anderes gemacht
Tag 4: was anderes gemacht
Tag 5: was anderes gemacht
Tag 6: The Treasure of the Sierra Madre (Vorschlag Nr. 1)
Tag 7: was anderes gemacht
Tag 8: was anderes gemacht
Tag 9: was anderes gemacht
Tag 10: was anderes gemacht
Tag 11: Red River (Vorschlag Nr. 1)
Tag 12: Giù la testa (Vorschlag Nr. 1)
Tag 13: Hud (Vorschlag Nr. 2)
Tag 14: The Gunfighter (Vorschlag Nr. 2)
Tag 15: The Magnificent Seven (Vorschlag Nr. 1)
Tag 16: Butch Cassidy and the Sundance Kid (Vorschlag Nr. 2)
Tag 17: McCabe & Mrs. Miller (Vorschlag Nr. 1)
Tag 18: Destry Rides Again (Vorschlag Nr. 3)
Tag 19: Shane (Vorschlag Nr. 2)
Tag 20: Winchester '73 (Vorschlag Nr. 2)
Tag 21: The Man Who Shot Liberty Valance (Vorschlag Nr. 2)
Tag 22: The Big Country (Vorschlag Nr. 1)
Tag 23: Way Out West (Vorschlag Nr. 2)
Tag 24: Ride the High Country (Vorschlag Nr. 1)
Tag 25: Dances with Wolves (Vorschlag Nr. 1)
Tag 26: Lonely Are the Brave (Vorschlag Nr. 1)
Tag 27: Unforgiven (Vorschlag Nr. 1)
Tag 28: El Dorado (Vorschlag Nr. 1)
Tag 29: Little Big Man (Vorschlag Nr. 1)
Tag 30: Dead Man (Vorschlag Nr. 1)
```

Das System benötigt eine gewisse Zeit, um sich auf den neuen Benutzer und dessen Geschmack einzustellen. Doch nach knapp 2 Wochen hat es seine Präferenz[9] so gut erkannt, dass es in der Lage ist, durchgängig gute Vorschläge zu erstellen, bis am Ende jeweils schon der erste oder zweite Vorschlag angenommen wird. Wie schon gesagt: Wenn die Ablehnung aller Vorschläge in negative Bewertungen umgesetzt wird, lässt sich die klare Präferenz dieses Nutzers früher erkennen.

Es lässt sich leider schlecht überprüfen, ob die Vorschläge in Teilaufgabe 2 sinnvoll sind und den Benutzern gefallen würden. Wir können allerdings davon ausgehen, dass die Geschmäcker

[9] Unser Benutzer ist ein eingefleischter Western-Fan.

von Ada L.[10], Edsgar D.[11], Don K.[12], Alan T.[13] ähnlich effizient erfasst werden wie in Teilaufgabe 3.

3.6 Bewertungskriterien

- Es muss ein Lösungsweg beschrieben und umgesetzt sein, der für die vom BWINF vorgegebenen Daten brauchbare Ergebnisse liefert. Das bedeutet insbesondere, dass für Teil 1 zumindest eine Lösungsvariante vorgelegt wird, die ausschließlich mit positiven Informationen („Film zu Ende gesehen“) klarkommt. In Teilaufgabe 3 konnte eine Ablehnung aller fünf Vorschläge aber durchaus als so genannte negative Evidenz berücksichtigt werden.
- Das (mathematische) Modell, um Filme vorzuschlagen, sollte gut begründet sein. Folgende Fragen sollten in der Dokumentation beantwortet werden:
  - Was sind die Kriterien, auf die die Empfehlungswürdigkeit eines Films begründet ist?
  - Warum eignet sich dieses Modell, um die Benutzer zufrieden zu stellen?
  - Gibt es Grenzen des Modells?
- Die Qualität des Programms zeigt sich in Teilaufgabe 3. Dem Programm ist eine „Eingewöhnungszeit“ einzuräumen. Mittel- bis langfristig müssen jedoch kontinuierlich gute Vorschläge gemacht werden. Ein abgelehnter Film darf kein zweites Mal vorgeschlagen werden. Gelegentliche Wiederholungen in den Vorschlägen sind akzeptabel.
- Die in Teil 3 geforderte Simulation muss nach den Vorgaben der Aufgabenstellung realisiert sein. Die Liste der Filme, die das neue Mitglied mag, darf dabei nur zur Bestimmung der Nutzerauswahl herangezogen werden.
- Programmablaufprotokolle dürfen nicht fehlen! Zusätzlich zu den in Teilaufgaben 2 und 3 vorgegebenen Beispielen sollten für weitere Beispiele Ablaufprotokolle angegeben werden. Besonders lobenswert ist es, eigene (selbst generierte) Eingabedaten zu verwenden, um etwa für Teil 3 zu prüfen, ob das System auch für einen weniger geschmacksfokussierten Nutzer akzeptabel funktioniert.
- Angesichts des realistischen Anwendungshintergrundes sollten Aspekte der Einsatzfähigkeit des Systems diskutiert werden, insbesondere einige der folgenden Fragen:

[10] Ada Lovelace: mathematikbegeisterte Gräfin und womöglich erste Programmiererin, nach ihr wurde die Programmiersprache „Ada“ benannt.
[11] Edsger Dijkstra: erfand u.a. den berühmten Dijkstra-Algorithmus (kürzeste Pfade in einem Graphen).
[12] Donald Knuth: Autor der Buchreihe „The Art of Computer Programming“, Erfinder und Entwickler von TEX, etc.
[13] Alan Turing: Erfinder des nach ihm benannten Berechnungsmodells, der Turing-Maschine, und damit einer der Urväter der theoretischen Informatik.

  - Sind Laufzeit und Speicherbedarf akzeptabel? (Dies sollte für das realisierte System der Fall sein, es sollte also auch mit hohen Nutzerzahlen zurecht kommen – und mit einer großen Anzahl von Filmen.)
  - Was wäre für ein System mit besonders vielen Nutzern problematisch, gibt es Optimierungsvorschläge?
  - Steigt die Qualität der Vorschläge mit einer steigenden Anzahl Nutzern?
  - Für welche Art der Nutzung ist das System besonders geeignet, d.h. wie kann ein Nutzer vorgehen, um das System optimal nutzen zu können?

Quellen

Im Zuge der Ringvorlesung „Perspektiven der Informatik“ an der Universität des Saarlandes (an der die Autoren dieses Abschnitts studieren) war der Vortrag von Dr. Ralf Schenkel zum Thema „Informationssuche in sozialen Netzen“[14] sehr hilfreich für die Lösung der Aufgabe. „Algorithmen: Eine Einführung“ von Cormen et al. war vor allem für die Implementierung der Hash-Datenstruktur hilfreich und führt auch in die zugrunde liegende Theorie ein.

[14] Folien: http://rw4.cs.uni-sb.de/teaching/perspektiven08/slides/Perspektiven020209.ppt

Perlen der Informatik – aus den Einsendungen

Allgemeines

Worte des Wettbewerbs: Die Implantierung in C#, Permutationsalgorithmius, rekursieve Funktion, uhrsprünglicher Punkt

Wenn ich mich haargenau an die Anleitung der Aufgabenstellung halten würde, dann würde mein Algorithmus komplett versagen. Aber damit kann ich leben, …

Die letzten zwei Wochen waren wohl die anspruchsvollsten und härtesten Wochen meines noch jungen Lebens.

Aufgabe 1: k-Sortierer

Man kann dieses Problem grafisch auch als Graf darstellen.

Ein trivialer Grund, warum man einen Autoreifen nicht mehr benutzen sollte, ist z.B. dass er viereckig ist.

An diesen Kabeln sitzen die jeweiligen 1-Sortierer.

… die Differenz $xa + xb + xc$ …

[Der Wert] … errechnet sich aus der Anzahl … mal eins.

Aufgabe 2: Kunst mit drei Ecken

All diese geometrischen Berechnungen haben in der Informatik einen natürlichen Feind, welcher nur auf sie lauert: sein Name ist „Division durch Null“, und er steht ganz oben in der Nahrungskette.

Wir wollen dabei entartete Dreiecke mit einem Innenwinkel von π nicht zulassen, da dies D. Trilikos vermutlich nicht gefallen würde, denn sonst hätte er seinen Künstlernamen nicht so gewählt. Würde er gerade Striche auf einem Blatt Papier bevorzugen, müsste er sich wohl eher L. Bilikos nennen.

Sich überschneidende Dreiecke sehen eher wie ein großes Chaos aus oder ein Familientreffen, gemalt von Pablo Picasso.

Um nicht zu verwirren, werden keine Zahlen präsentiert, sondern ein netter Schieberegler, der so abstrakt ist, dass man nicht so genau weiß, was man tatsächlich wie einstellt – aber trotzdem schön und praktisch.

Diese Vorgehensweise nennt sich „genetisches Programmieren“. […] Dieses Verfahren hat sich in den letzten Millionen Jahren als äußerst effektiv erwiesen.

Der Kreis beginnt von Forne.

Aufgabe 3: Content Provider Unit

Der Besitzer der CPU heißt Karl West, nur für die Gleichberechtigung.

Ich würde auch den Film „Aliens“ vorschlagen, wenn der Film „Alien“ gesehen wurde. In beiden Filmen geht es offensichtlich um wenigstens ein Alien.

… für den Fall, dass ein Mitglied alle Filme gesehen hat. Wer allerdings so hirnkrank ist, braucht keine CPU mehr.

Als Standard-Abweichung habe ich 1,54 errechnet – ich übernehme keine Gewähr dafür, dass der Wert stimmt, schließlich wusste ich in der Schule noch nicht, dass ich Stochastik sogar mal wirklich brauchen könnte!

Die Lösung dieses Problems liegt im Erfolg.