Lösungshinweise und Bewertungskriterien

Bundeswettbewerb Informatik

Allgemeines

Grundsätzliches

Es ist immer wieder bewundernswert, wie viele Ideen, wie viel Wissen, Fleiß und Durchhaltevermögen in den Einsendungen zur 2. Runde eines Bundeswettbewerbs Informatik stecken. Um aber die Allerbesten für die Endrunde zu bestimmen, müssen wir die Arbeiten kritisch begutachten und hohe Anforderungen stellen. Deswegen sind Punktabzüge die Regel und Bewertungen mit Pluspunkten über die Erwartungen hinaus die Ausnahme.

Spannende bzw. schwierige Erweiterungen der Aufgabenstellung sind Extrapunkte wert, wenn sie auch praktisch realisiert wurden. Intensive theoretische Überlegungen wie z. B. ein korrekter Beweis zur Komplexität des Problems werden ebenfalls mit zusätzlichen Punkten belohnt. Weitere Ideen ohne Implementierung und geringe Verbesserungen der bereits implementierten Lösung einer Aufgabe gelten allerdings nicht als geeignete Erweiterungen.

Falls Ihre Einsendung nicht herausragend bewertet wurde, lassen Sie sich nicht entmutigen! Allein durch die Arbeit an den Aufgaben und ihren Lösungen hat jede Teilnehmerin und jeder Teilnehmer viel gelernt; diesen Effekt sollten Sie nicht unterschätzen. Selbst wenn Sie nur die Lösung zu einer Aufgabe einreichen konnten, so kann die Bewertung Ihrer Einsendung bei der Anfertigung künftiger Lösungen hilfreich für Sie sein.

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

Bewertungsbogen

Aus der ersten Runde oder auch aus früheren Wettbewerbsteilnahmen kennen Sie den Bewertungsbogen, der angibt, wie Ihre Einsendung die einzelnen Bewertungskriterien erfüllt hat. Auch in dieser Runde können Sie den Bewertungsbogen im Anmeldesystem AMS einsehen. In der 1. Runde ging die Bewertung noch von 5 Punkten aus, von denen bei Mängeln abgezogen werden konnte. In der 2. Runde geht die Bewertung von zwanzig Punkten aus, und es gibt deutlich mehr Bewertungskriterien als in der 1. Runde, bei denen Punkte abgezogen und manchmal auch hinzuaddiert werden konnten.

Terminlage

Für Abiturientinnen und Abiturienten ist der Terminkonflikt zwischen Abiturvorbereitung und 2. Runde sicher nicht ideal. Doch leider bleibt dem Bundeswettbewerb Informatik nur die erste

Jahreshälfte für diese Runde: In der zweiten Jahreshälfte läuft nämlich die zweite Runde des Mathematikwettbewerbs, dem wir keine Konkurrenz machen wollen. Aber: die Bearbeitungszeit für die Runde beträgt etwa vier Monate. Frühzeitig mit der Bearbeitung der Aufgaben zu beginnen ist der beste Weg, zeitliche Engpässe am Ende der Bearbeitungszeit gerade mit der wichtigen, ausführlichen Dokumentation der Aufgabenlösungen zu vermeiden.

Aufgaben der 2. Runde sind oft deutlich schwerer zu lösen, als sie auf den ersten Blick erscheinen. Erst bei der konkreten Umsetzung einer Lösungsidee stößt man manchmal auf Besonderheiten bzw. noch zu lösende Schwierigkeiten, was dann zusätzlicher Zeit bedarf. Daher ist es sinnvoll, die einzureichenden Aufgaben nicht nacheinander, sondern relativ gleichzeitig zu bearbeiten, um nicht vom zeitlichen Aufwand der jeweiligen Aufgabe unangenehm kurz vor Ablauf der Bearbeitungszeit überrascht zu werden und keine vollständige Lösung mehr zu schaffen.

Dokumentation

Es ist sehr gut nachvollziehbar, dass Sie Ihre Energie bevorzugt in die Lösung der Aufgaben, die Entwicklung Ihrer Ideen und deren Umsetzung in Software fließen lassen. Doch ohne eine verständliche Beschreibung der Lösungsideen und ihrer jeweiligen Umsetzung, eine übersichtliche Dokumentation der wichtigsten Komponenten Ihrer Programme, eine geeignete Kommentierung der Quellcodes und eine ausreichende Zahl sinnvoller Beispiele (welche die verschiedenen, bei der Lösung des Problems zu berücksichtigenden Fälle abdecken) ist eine Einsendung nur wenig wert.

Bewerterinnen und Bewerter können die Qualität Ihrer Aufgabenlösungen nur anhand dieser Informationen vernünftig einschätzen. Mängel in der Dokumentation der Einsendung können nur selten durch Ausprobieren und Testen der Programme ausgeglichen werden – wenn die Programme denn überhaupt ausgeführt werden können: Hier gibt es gelegentlich Probleme, die meist vermieden werden könnten, wenn Lösungsprogramme vor der Einsendung nicht nur auf dem eigenen, sondern auch einmal auf einem fremden Rechner auf Lauffähigkeit getestet würden. Insgesamt sollte die Erstellung der Dokumentation die Programmierarbeit begleiten oder ihr teilweise sogar vorangehen: Wer nicht in der Lage ist, Idee und Modell präzise zu formulieren, bekommt auch keine saubere Umsetzung hin, in welcher Programmiersprache auch immer. Einige unterhaltsame Formulierungsperlen sind im Anhang wiedergegeben.

Bewertungskriterien

Bei den im Folgenden beschriebenen Lösungsideen handelt es sich nicht um perfekte Musterlösungen, sondern um sinnvolle Lösungsvorschläge. Dies sind also nicht die einzigen Lösungswege, die wir gelten ließen. Wir akzeptieren vielmehr in der Regel alle Ansätze, auch ungewöhnliche, kreative Bearbeitungen, die die gestellte Aufgabe vernünftig lösen und entsprechend dokumentiert sind. Einige der Fußnoten in den folgenden Lösungsvorschlägen verweisen auf weiterführende Fachliteratur für besonders Interessierte; Lektüre und Verständnis solcher Literatur wurden von den Teilnehmenden natürlich nicht erwartet.

Unabhängig vom gewählten Lösungsweg gibt es aber Dinge, die auf jeden Fall von einer guten Lösung erwartet wurden. Zu jeder Aufgabe wird deshalb in einem eigenen Abschnitt, jeweils am Ende des Lösungsvorschlags erläutert, auf welche Kriterien bei der Bewertung dieser Aufgabe besonders geachtet wurde. Dabei können durchaus Anforderungen formuliert werden, die

aus der Aufgabenstellung nicht hervorgingen. Letztlich dienen die Bewertungskriterien dazu, die allerbesten unter den sehr vielen guten Einsendungen herauszufinden.

Außerdem gibt es aufgabenunabhängig einige Anforderungen an die Dokumentation (klare Beschreibung der Lösungsidee, genügend aussagekräftige Beispiele und wesentliche Auszüge aus dem Quellcode) einschließlich einer theoretischen Analyse (geeignete Laufzeitüberlegungen bzw. eine Diskussion der Komplexität des Problems) sowie an den Quellcode der implementierten Software (mit übersichtlicher Programmstruktur und verständlicher Kommentierung) und an das lauffähige Programm (ohne Implementierungsfehler). Wünschenswert sind auch Hinweise auf die Grenzen des angewandten Verfahrens sowie sinnvolle Begründungen z. B. für Heuristiken, vorgenommene Vereinfachungen und Näherungen. Geeignete Abbildungen und eigene zusätzliche Eingaben können die Erläuterungen in der Dokumentation gut unterstützen. Die erhaltenen Ergebnisse für die Beispieleingaben (ggf. mit Angaben zur Rechenzeit) sollten leicht nachvollziehbar dargestellt sein, z. B. durch die Ausgabe von Zwischenschritten oder geeignete Visualisierungen. Eine Untersuchung der Skalierbarkeit des eingesetzten Algorithmus hinsichtlich des Umfangs der Eingabedaten ist oft ebenfalls nützlich.

Danksagung

Die Aufgaben wurden vom Aufgabenausschuss des Bundeswettbewerbs Informatik entwickelt: Peter Rossmanith, *RWTH Aachen University* (Vorsitzender); Hanno Baehr, *RWE Supply & Trading GmbH, Essen*; Jens Gallenbacher, *JGU Mainz, ETH Zürich*; Rainer Gemulla, *Universität Mannheim*; Torben Hagerup, *Universität Augsburg*; Christof Hanke, *Berufliches Gymnasium für Informatik, FLB Herford*; Thomas Kesselheim, *Universität Bonn*; Arno Pasternak, *TU Dortmund*; Holger Schlingloff, *Fraunhofer FOKUS, HU Berlin*; Melanie Schmidt, *Universität Köln*; als Gast im Ausschuss: Wolfgang Pohl, *BWINF, Bonn*.

An der Erstellung der im Folgenden beschriebenen Lösungsideen wirkten vor allem folgende Personen mit: Alexandru Duca (Aufgabe 1), Hans-Martin Bartram (Aufgabe 2) und Niccolò Rigi-Luperti (Aufgabe 3). Allen Beteiligten sei für Ihre Mitarbeit hiermit ganz herzlich gedankt.

Aufgabe 1: Flohmarkt in Langdorf

Bei dieser Aufgabe ist es nicht schwer, eine Verwandtschaft mit dem Problem Subset-Sum[1] zu sehen: Gegeben sind eine Menge ganzer Zahlen und eine „Zielsumme“; zu entscheiden ist, ob es eine Teilmenge der Zahlen gibt, die sich genau zur Zielsumme addieren.

In der Tat lässt sich Subset-Sum auf das in dieser Aufgabe gestellte Problem so reduzieren: Man nimmt die Zielsumme von Subset-Sum als Flohmarktlänge und die Werte der Zahlen von Subset-Sum als die Länge der angemeldeten Stände. Alle Stände werden für die volle Zeit angemeldet, also von 8 bis 18 Uhr; damit spielt nur die Länge eine Rolle. Wenn wir für dieses spezielle "Flohmarkt-Problemëntscheiden können, ob wir mit einem Teil der Voranmeldungen den Flohmarkt komplett belegen können, können wir das auch das ursprüngliche Subset-Sum-Problem lösen.

Das Problem Subset-Sum ist als *NP-schwer* bekannt. Damit wissen wir also, dass auch das Flohmarkt-Problem NP-schwer ist und sich daher im Allgemeinen nicht effizient optimal lösen lässt.

Daher werden wir bei den vorliegenden Problemgrößen mit Heuristiken arbeiten müssen: entweder mit solchen, die wir gezielt für die Lösung des Problems entwickeln, oder auch mit einer Instanziierung einer geeigneten *Metaheuristik*[2]. Auf beide Wege gehen wir im Folgenden ein.

1.1 Greedy-Algorithmus

Ein einfacher Lösungsansatz folgt dem Greedy-Prinzip (engl. greedy = gierig). Dabei weisen wir einfach jeder Voranmeldung der Reihe nach den ersten passenden Standplatz zu, wenn es einen gibt. Da ein Greedy-Algorithmus keine weiteren Voranmeldungen berücksichtigt außer der aktuell betrachteten, wird er in vielen Fällen nicht die optimale Entscheidung treffen. Möglicherweise kann es nämlich besser sein, die aktuelle Voranmeldung nicht zu berücksichtigen, um später ein oder mehrere besser passende Voranmeldungen auszuwählen.

Dennoch kann auch ein Greedy-Algorithmus in manchen Fällen (auch bei den vorgegebenen Beispielen) recht gute Ergebnisse liefern und bildet eine gute untere Abschätzung. Das gilt umso mehr, wenn man mit weiteren Verfeinerungen arbeitet.

1.2 Greedy-Algorithmus mit Heuristik

Insbesondere kann es helfen, die Reihenfolge zu verändern, in der man die Voranmeldungen nach dem Greedy-Prinzip abarbeitet. Recht erfolgreich lassen sich diese beiden Ideen einsetzen:

Zufällige Reihenfolge Die Voranmeldungen werden in eine zufällige Reihenfolge gebracht (randomisiert). Insbesondere wenn viele verschiedene zufällige Reihenfolgen ausprobiert werden (der Greedy-Algorithmus als solcher läuft ja sehr schnell), können hier sehr gute Ergebnisse auftreten.

Gezielte Priorisierung Auch gezielte Veränderungen der Reihenfolge für das Greedy-Verfahren sind denkbar. Weil insgesamt die Mieteinnahmen maximiert werden sollen, liegt nahe,

[1] `https://de.wikipedia.org/wiki/Teilsummenproblem`
[2] `https://de.wikipedia.org/wiki/Metaheuristik`

die Voranmeldungen nach ihren Mietkosten (Länge mal Zeit) absteigend zu sortieren, also die „teuersten“ Stände zu bevorzugen.

1.3 Ganzzahlige Optimierung

Schwierige Probleme lassen sich, wie erwähnt, auch mit Metaheuristiken lösen. Dazu lassen sich, im weiteren Sinne, auch verallgemeinerte Optimierungsmethoden zählen wie die *lineare Optimierung*. Diese Methode findet im allgemeinen optimale Werte einer Zielfunktion, bei der die Werte der Definitionsmenge durch Ungleichungen eingeschränkt werden. Beim Flohmarkt-Problem muss etwa eine Auswahl von Voranmeldungen, die zu Doppelbelegungen führt, ausgeschlossen werden.

Im Folgenden wollen wir unseren Ansatz zur Lösung der Aufgabe mit Hilfe der ganzzahligen Optimierung vorstellen. Dazu führen wir zunächst die grundlegenden Begriffe linearer Programme ein und zeigen dann, wie das Flohmarkt-Problem damit bearbeitet werden kann.

Begriffserklärung

Bei einem **linearen Programm** (engl. **linear program**, LP) ist gegeben:

- endlich viele Entscheidungsvariablen $x_1, \ldots, x_n$
- genau eine lineare Zielfunktion $f(x_1, \ldots, x_n) = c_1 x_1 + \cdots + c_n x_n$ mit $c_1, \ldots, c_n \in \mathbb{R}$
- endlich viele lineare Ungleichungen $a_1 x_1 + \cdots + a_n x_n \leq b$ mit $a_1, \ldots, a_n, b \in \mathbb{R}$

Eine Lösung legt für jede Entscheidungsvariable einen Wert aus $\mathbb{R}$ fest. Eine Lösung heißt gültig, wenn alle Ungleichungen erfüllt sind, und optimal, wenn die Zielfunktion einen maximalen Wert annimmt.

Ein **gemischt-ganzzahliges lineares Programm** (engl. **mixed-integer linear program**, MILP) ist ein LP, bei dem für einige Entscheidungsvariablen die zusätzliche Bedingung gegeben ist, dass diese ganzzahlig sind.

Während LPs in Polynomialzeit lösbar sind, handelt es sich bei MILPs um NP-schwere Probleme, für die mittlerweile fortgeschrittene MILP-Solver existieren. Ein Ausgangspunkt für die Lösungssuche liefert für üblich eine sogenannte LP-Relaxation. Hierbei wird die Ganzzahligkeitsbedingung für alle Entscheidungsvariablen des MILPs vorerst ignoriert und das daraus resultierende LP in kurzer Zeit gelöst. Ab dann können unterschiedliche Methoden (z.B. Branch-and-Bound) angewandt werden, um aus dieser Zwischenlösung, die nicht unbedingt alle Ganzzahligkeitsbedingungen erfüllt, eine gültige und (hoffentlich) optimale Lösung für das MILP abzuleiten.

Bei einem sogenannten Warmstart wird ein MILP-Solver vor der Ausführung mit einer gültigen Lösung für das MILP vertraut gemacht. Dieser zusätzliche Ausgangspunkt kann zu einer Steigerung der Performanz führen.

Formalisierung

Die Aufgabenstellung wird wie folgt auf ein MILP zurückgeführt:

Sei N die Anzahl der Voranmeldungen. Für eine Voranmeldung $i \in \{1, \ldots, N\}$ ist gegeben:

- der Mietbeginn $a_i \in \{8,\ldots,17\}$
- das Mietende $b_i \in \{a_i+1,\ldots,18\}$
- die Länge des Standes $l_i \in \{1,\ldots,1000\}$

Für jede Voranmeldung i werden zwei ganzzahlige Entscheidungsvariablen eingeführt:

- $0 \leq k_i \leq 1$ beschreibt, ob Voranmeldung i angenommen $(k_i = 1)$ oder abgelehnt $(k_i = 0)$ wird.
- $0 \leq x_i \leq 1000 - l_i$ beschreibt den für Voranmeldung i ausgewählten Ort für den Anfang des Standes.

Jede Voranmeldung $i \in \{1,\ldots,N\}$ bringt genau $k_i l_i (b_i - a_i)$ Euro ein, denn:

$$k_i l_i (b_i - a_i) = \begin{cases} l_i(b_i - a_i) & \text{für } k_i = 1 \text{ (Voranmeldung } i \text{ angenommen)} \\ 0 & \text{für } k_i = 0 \text{ (Voranmeldung } i \text{ abgelehnt)} \end{cases}$$

Zu maximieren ist der Profit. Damit lautet die Zielfunktion:

$$f(k_1,\ldots,k_N) = k_1 l_1 (b_1 - a_1) + \cdots + k_N l_N (b_N - a_N)$$

Für den Fall, dass sich zwei Voranmeldungen $i, j \in \{1,\ldots,N\}$ $(i \neq j)$ zeitlich überschneiden, d.h. $b_i > a_j$ und $b_j > a_i$, wird eine räumliche Überschneidung verhindert, wenn genau eine der folgenden zwei Ungleichungen erfüllt ist:

- $x_i + l_i \leq x_j$ (Stand i ist links von Stand j)
- $x_j + l_j \leq x_i$ (Stand j ist links von Stand i)

Um eine räumliche Überschneidung zu verhindern, ist also die zusätzliche Entscheidung zu treffen, wie Stand i und j relativ zueinander stehen bzw. welche der zwei oberen Ungleichungen zu erfüllen ist. Für jedes Paar sich zeitlich überschneidender Voranmeldungen i, j $(i < j)$ wird eine ganzzahlige Entscheidungsvariable eingeführt:

- $0 \leq z_{i,j} \leq 1$ beschreibt, ob Stand j links von Stand i ist $(z_{i,j} = 1)$ oder ob Stand i links von Stand j ist $(z_{i,j} = 0)$.

Ein Ungleichungssystem, dessen Erfüllung die räumliche Überschneidung zweier sich zeitlich überschneidender Voranmeldungen i, j verhindert, sieht wie folgt aus:

$$\begin{aligned} x_i + l_i &\leq x_j + 1000 z_{i,j} + 1000(1 - k_i) + 1000(1 - k_j) \\ x_j + l_j &\leq x_i + 1000(1 - z_{i,j}) + 1000(1 - k_i) + 1000(1 - k_j) \end{aligned}$$

Das Ungleichungssystem baut auf der Idee auf, dass wegen

$$\begin{aligned} x_i \leq 1000 - l_i &\Leftrightarrow x_i + l_i \leq 1000 \\ x_j \leq 1000 - l_j &\Leftrightarrow x_j + l_j \leq 1000 \end{aligned}$$

eine Ungleichung immer erfüllt ist, wenn die rechte Seite um 1000 erhöht wird. Für den Summanden $1000(1 - k_i)$ und analog für $1000(1 - k_j)$ gilt:

$$1000(1 - k_i) = \begin{cases} 0 & \text{für } k_i = 1 \text{ (Voranmeldung } i \text{ angenommen)} \\ 1000 & \text{für } k_i = 0 \text{ (Voranmeldung } i \text{ abgelehnt)} \end{cases}$$

Damit sind beide Ungleichungen immer erfüllt, wenn mindestens eine Voranmeldung i, j abgelehnt wird. Das ist sinnig, weil es gar nicht zu einer Überschneidung kommt, wenn mindestens ein Stand nicht existiert. Wenn beide Voranmeldungen i, j angenommen werden, nehmen die Summanden $1000(1-k_i)$ und $1000(1-k_j)$ den Wert 0 an und das Ungleichungssystem vereinfacht sich zu:

$$x_i + l_i \leq x_j + 1000 z_{i,j}$$
$$x_j + l_j \leq x_i + 1000(1 - z_{i,j})$$

Wenn Stand j links von Stand i ist $(z_{i,j} = 1)$, dann ist die erste Ungleichung erfüllt, während die zweite Ungleichung sich zu $x_j + l_j \leq x_i$ zu vereinfacht. Umgekehrtes ist für $z_{i,j} = 0$ der Fall. Die Wahl des Werts für $z_{i,j}$ legt also fest, welche der beiden Ungleichungen, deren Erfüllung eine räumliche Überschneidungsfreiheit garantiert, zum Erfüllen übrig bleibt.

Implementierung

Zuerst wird mit einem Greedy-Algorithmus eine gültige Lösung für das MILP bestimmt: Alle Voranmeldungen werden absteigend nach dem Profit angeordnet, und der Reihe nach wird der nächstmögliche Standort vergeben. Wenn zu einem späteren Zeitpunkt für eine Voranmeldung kein gültiger Standort existiert, dann wird sie abgelehnt.

Wenn alle Voranmeldungen angenommen werden oder der Profit gleich 10000 ist, dann ist die gefundene Lösung optimal. In jedem anderen Fall wird der MILP-Solver mit der vom Greedy-Algorithmus bestimmten Lösung warm gestartet.

Als MILP-Solver wurde von uns die Python Bibliothek PuLP 2.4 verwendet.

1.4 Weitere mögliche Verfahren

Ausgefeiltere Heuristiken

Neben den einfachen Heuristiken auf Basis des Greedy-Ansatzes, die weiter oben beschrieben wurden, lassen sich auch ausgefeiltere Heuristiken überlegen, die nicht auf eine statisch vorgegebene Sortierung der Voranmeldungen zurückgreifen müssen, sondern stattdessen dynamisch auch die schon vergebenen und noch freien Plätze berücksichtigen können. Dabei können zum Beispiel Voranmeldungen bevorzugt angenommen werden, die an eine Stelle des Zeitplans noch besonders gut reinpassen.

Evolutionäre Algorithmen

Sehr gute Ergebnisse lassen sich bei diesem Problem auch mit evolutionären Algorithmen[3] erzielen.

Dabei wird eine Population von Lösungen als Ausgangspunkt genommen, von denen wiederholt Nachkommen erzeugt werden um neue Populationen von Lösungen zu erzeugen, die aus den besten Lösungen in der Menge der Nachkommen gebildet wird.

[3] `https://de.wikipedia.org/wiki/Evolution%C3%A4rer_Algorithmus`

Die Nachkommen können erzeugt werden indem die Lösungen leicht mutiert werden (etwa durch Einfügen, Verschieben oder Austausch von einzelnen Voranmeldungen).

Auch eine Rekombination von verschiedenen Lösungen ist hier möglich – man spricht dann von genetischen Algorithmen.

1.5 Beispiele

Tabelle 1 zeigt für jede Beispielaufgabe die benötigte Rechenzeit und den Profit von jeder gültigen Zwischenlösung, die der uns verwendete Algorithmus auf Grundlage von ganzzahlige Optimierung bestimmt hat.

Tabelle 1: Detaillierte Ergebnisse für die vorgegebenen Beispiele

Beispiel	**Ergebnis Greedy**	**1. Ergebnis Solver**	**2. Ergebnis Solver**	**3. Ergebnis Solver**	**4. Ergebnis Solver**	**Optimalitätsbeweis für das beste Ergebnis**
1	12,40s 8028	Der Solver wurde nicht ausgeführt, weil die Greedy Lösung optimal ist.				
2	21,39s 8796	–	–	–	–	–
3	23,68s 8630	–	–	–	–	–
4	0,03s 6867	0,00s 7370	–	–	–	0,01s
5	0,04s 6580	0,56s 6609	0,99s 7257	12,29s 8447	43,96s 8705	202,33s
6	0,03s 9497	0,01s 10000	–	–	–	0,01s
7	16,83s 9753	–	–	–	–	–

Tabelle 2 fasst die besten Ergebnisse zusammen, die von dieser Beispiellösung gefunden wurden. Außerdem werden die besten Ergebnisse aus den eingesendeten Lösungen angegeben.

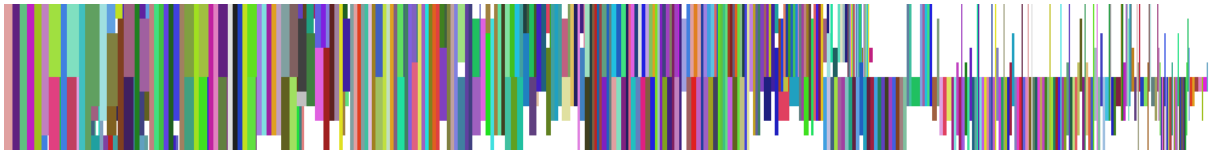
Tabelle 2: Beste Ergebnisse für die vorgegebenen Beispiele

Aufgabe	1	2	3	4	5	6	7
Beispiellösung	8028	8796	8630	7370	8705	10000	9753
Andere Lösungen	8028	9077	8778	7370	8705	10000	10000
Optimal	Ja	?	?	Ja	Ja	Ja	Ja

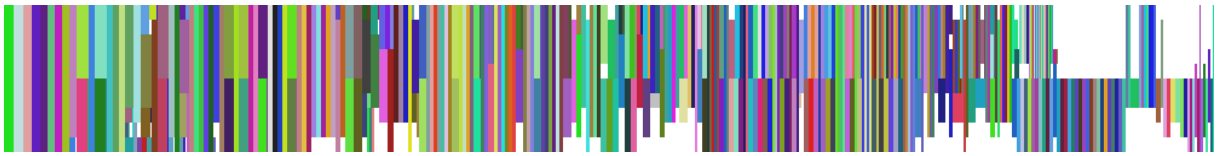
Es folgt für jedes Beispiel eine Veranschaulichung der besten gefundenen Lösung. In den Grafiken entspricht eine Pixelhöhe fünf Minuten und eine Pixelbreite einem Meter. Für jede angenommene Voranmeldung nehmen alle Pixel, welche die zeitliche und räumliche Lage des Standes beschreiben, die gleiche Farbe an.

Für die kleinen Beispiele wird jeweils zusätzlich eine Tabelle angegeben, die für alle angenommenen Voranmeldungen $i \in \{1, \ldots, N\}$ dem vergebenen Standort x_i angibt.

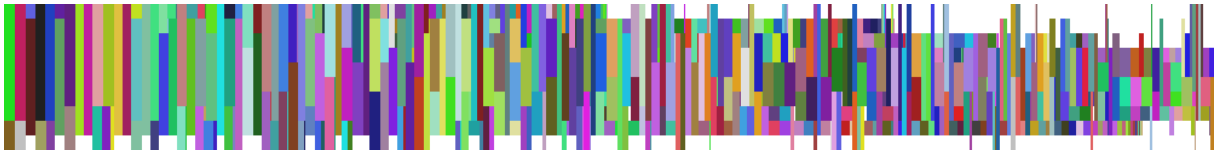
flohmarkt1.txt



flohmarkt2.txt



flohmarkt3.txt



flohmarkt4.txt

i	1	2	4	5	6
x_i	77	0	326	352	829

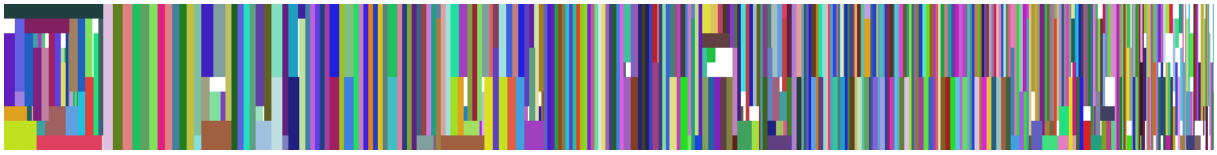
flohmarkt5.txt

i	2	3	6	7	9	14	19	20
x_i	42	0	714	568	969	889	0	885

flohmarkt6.txt

i	1	2	3	4	5	6	7	8	9
x_i	0	420	0	526	249	923	420	429	897

flohmarkt7.txt



1.6 Bewertungskriterien

Die aufgabenspezifischen Bewertungskriterien werden hier erläutert.

1. Lösungsweg

(1) *Problem adäquat modelliert*:
- Der zu einer Voranmeldung zugehörige Stand wird als ein Rechteck mit einer räumlichen und einer zeitlichen Kante verstanden (oder analog).
- Es sollte angegeben sein, wie die Position des Standes einer angenommenen Voranmeldung beschrieben wird.
- Es sollte (am besten in Formeln) angegeben werden, wann genau zwei Stände überschneidungsfrei sind und wann genau nicht.

(2) *Mehrere Lösungsansätze*: Da Heuristiken offensichtlich nicht zu garantiert optimalen Ergebnissen führen, ist es sinnvoll, sich über unterschiedliche Lösungsansätze zumindest Gedanken zu machen. Wer wirklich mehrere und deutlich unterschiedliche Ansätze realisiert hat, hat Pluspunkte verdient.

(3) *Laufzeit des Verfahrens in Ordnung*: Da die Größe des „Platzierungsrechtecks“ (Länge des Flohmarkts mal Stunden) konstant ist, ist die Laufzeit vieler Heuristiken grundsätzlich durch das Sortieren der n Anmeldungen nach unten begrenzt und liegt dann bestenfalls in $O(n \log n)$, häufig wohl in $O(n^2)$. Aber auch höhere polynomielle Laufzeiten sind in Ordnung. Abzüge gibt es natürlich für Verfahren mit exponentiellen Laufzeiten, die ohnehin nicht in der Lage sein sollten, für alle Beispiele in angemessener Zeit Ergebnisse zu berechnen. Auch Verschlechterungen der Laufzeit durch unnötige Verkomplizierungen des Verfahrens können zu Punktabzug führen.

(4) *Speicherbedarf in Ordnung*: Die Belegung des Flohmarktes lässt sich analog repräsentieren: Für jeden „Stundenmeter“ kann ein im Prinzip Boolescher Wert gespeichert werden: frei oder belegt. Die dazu nötigen 10.000 Werte stellen kein Platzproblem dar. Gute Ansätze zur Reduzierung des „Belegungsspeichers“ können mit einem Pluspunkt belohnt werden.

(5) *Verfahren mit korrekten Ergebnissen*: Stände werden einem gültigem Standort zugewiesen und sind paarweise überschneidungsfrei. Auch die Berechnung des Profits sollte fehlerfrei sein.

(6) *Verfahren mit guten Ergebnissen*: Für die kleinen Beispiele 4 und 6 sollten die optimalen Ergebnisse gefunden werden; das schaffen die meisten Ansätze. Bei Beispiel 1 finden praktisch alle einigermaßen brauchbaren Ansätze das optimale Resultat, deshalb wird auch das erwartet. Für die Beispiele 2, 3, 5 (hier ist es schwieriger, das optimale Ergebnis zu ermitteln) und 7 wurden anhand der Einsendungen Grenzwerte bestimmt, nach denen die Punkte vergeben werden. Die nach dieser Tabelle berechneten Abzüge bzw. Zugaben für die einzelnen Beispiele werden zur Gesamtbewertung addiert (wobei -4 die Minimalwertung ist):

Bewertung	1	2	3	4	5	6	7
-1 wenn $<$	8028	8750	8600	7370	8500	10000	9750
$+1$ wenn $\geq$	–	9050	8750	–	8705	–	9900

2. Theoretische Analyse

(1) *Verfahren / Qualität ingesamt gut begründet*:
- Falls ein heuristisches Verfahren gewählt wurde, muss erkannt worden sein, dass dieses nicht gesichert optimale Ergebnisse liefern wird. Für einzelne, insbesondere die kleineren Beispiele kann man außerdem gut erkennen, wenn das eigene Verfahren schwache Ergebnisse liefert. Wurde eine klassische Suche realisiert, muss der Laufzeitnachteil erkannt worden sein. Insgesamt erwarten wir – wie immer – eine einigermaßen selbstkritische Haltung.
- Es sollte begründet sein, warum ein Verfahren gewählt wurde, das nicht gesichert optimale Ergebnisse liefert.
- Für ein heuristisches Verfahren sollte dargelegt werden, welche Ergebnisqualität zu erwarten ist oder warum es möglicherweise bessere Ergebnisse liefert als andere Ansätze.

(2) *Gute Überlegungen zur Laufzeit*: Die Laufzeit des Verfahrens muss nicht zwingend formal, aber nachvollziehbar und korrekt charakterisiert werden.

(3) *Komplexität plausibel charakterisiert*: Es ist nicht allzu schwierig, das Flohmarkt-Problem als NP-schwer zu erkennen. Ist eine entsprechende Aussage vorhanden (verlangt ist sie nicht), sollte sie aber nicht nur spekulativ, sondern zumindest mit sinnvollen Argumenten begründet werden. Wenn diese Argumente besondere Substanz haben oder sogar ansatzweise formal aufgeschrieben sind, kann es Pluspunkte geben.

(4) *Mehrere Ansätze gründlich verglichen*: Wer mehrere Lösungsansätze nicht nur realisiert (s.o.), sondern bezüglich ihrer Leistungsfähigkeit gründlich analysiert und verglichen hat, kann hier noch einmal mit Pluspunkten belohnt werden.

3. Dokumentation

(3) *Vorgegebene Beispiele dokumentiert*: Es sollten alle vorgegebenen Beispiele bearbeitet und dokumentiert worden sein. Wenn nicht alle, aber mindestens 4 der 7 Beispiele dokumentiert sind, gibt es 2 Punkte, bei weniger bis zu 4 Punkte Abzug.

(5) *Ergebnisse nachvollziehbar dargestellt*:
- Zu jedem Beispiel muss der berechnete Profit angegeben werden; ansonsten kann stark abgezogen werden. Weil der Bezugswert 10000 lautet, lässt sich eine Angabe in Prozent glücklicherweise genau so gut lesen wie eine absolute Angabe.
- Außerdem sollte zumindest ausschnittsweise dargestellt werden, wie die gefundene Belegung aussieht. Dabei sollten die unterschiedlichen Stände gut unterscheidbar sein. Für eine grafische Darstellung wie hier in der Beispiellösung kann es Pluspunkte geben.
- Das gilt auch, wenn weitere Werte ausgegeben werden, die die Belegung oder auch das Verhalten des Verfahrens interessant charakterisieren.

Aufgabe 2: Spießgesellen

Bei dieser Aufgabe sollten aus den Beobachtungen von Donald, die immer eine Teilmenge von Obstsorten mit einer Teilmenge von Schüsseln in Beziehung setzen, ermittelt werden, in welcher Menge von Schüsseln Donald seine Lieblingsobstsorten finden kann.

Diese Aufgabe lässt sich sehr einfach als ein Bipartite-Matching[4]-Problem formulieren, indem ein vollständiger bipartiter Graph als Ausgangspunkt genommen wird und alle Kanten entfernt werden, die den Beobachtungen widersprechen. Aufgrund der Symmetrie in diesem Problem – die beobachteten Obstsorten können nicht nur nicht in den nicht-beobachteten Schüsseln vorkommen, sondern auch die nicht-beobachteten Obstsorten können nicht in den beobachteten Schüsseln vorkommen – ist diese Aufgabe jedoch wesentlich einfacher lösbar als Bipartite-Matching.

Wir wollen im Folgenden eine solche einfachere Lösung vorstellen.

2.1 Lösungsidee

Jeder Obstsorte der Wunschmenge wird eineindeutig eine Menge von Schüsseln zugeordnet; bei k Obstsorten sind das die Schüsselmengen M_1 bis M_k. Diese Mengen beschreiben, in welchen Schüsseln sich die zugeordnete Obstsorte befinden kann. Zunächst sind diese die Mengen alle gleich und enthalten alle Schüsseln. Nun werden nach und nach die Beobachtungen verarbeitet. Eine Beobachtung bzw. ein beobachteter Spieß ist ein Paar aus zwei Mengen: der Menge von Obstsorten auf dem Spieß und der Menge der für das Zusammenstellen des Spießes benutzten Schüsseln. Diese Information wird genutzt, um für jede Obstsorte aus ihrer Schüsselmenge diejenigen Schüsseln zu entfernen, welche für als Behälter dieser Sorte nicht mehr in Frage kommen. Hierbei wird nach Algorithmus 1 vorgegangen.

Algorithmus 1 Rausstreichen

```
1: for all Spieß in Beobachtungen do
2:     for all Obstsorte in Wunschmenge do
3:         if Obstsorte in Spieß then
4:             entferne alle Schüsseln, die für den Spieß nicht benutzt wurden
5:         else
6:             entferne alle Schüsseln, die für den Spieß benutzt wurden
7:         end if
8:     end for
9: end for
```

Abschließend werden alle Schüsselmengen M_1 bis M_k vereinigt und die Anzahl der Elemente der Vereinigungsmenge bestimmt. Ist diese Anzahl gleich der Anzahl der Wunschsorten, so wurde eine Lösung gefunden. Ist die Anzahl der Elemente in der Vereinigungsmenge größer als die Anzahl der Wunschsorten, dann lässt sich für die Wunschsorten keine eindeutige Schüsselmenge angeben, und die Schüsseln aus der Vereinigungsmenge enthalten auch unerwünschte Obstsorten.

[4] `https://de.wikipedia.org/wiki/Matching_(Graphentheorie)`

2.2 Beispielvorgehen

Im Folgenden wird der Algorithmus anhand des Beispiels aus der Aufgabenstellung erklärt. Es wird mit den Mengen begonnen, die alle Elemente enthalten:

Wunschsorten	Apfel	Weintraube	Brombeere
Schüsseln	$\{1,2,3,4,5,6\}$	$\{1,2,3,4,5,6\}$	$\{1,2,3,4,5,6\}$

Nun wird die Beobachtung von Micky verwendet, um einige Schüsseln zu streichen. Micky hat auf seinem Spieß Apfel, Banane und Brombeere und hat die Schüsseln 1, 4 und 5 benutzt. So werden bei Apfel und Brombeere die Schüsseln gestrichen, welche nicht von Micky benutzt wurden ($\{2,3,6\}$). Da Weintraube nicht auf Mickys Spieß vorkam, werden für diese Obstsorte alle von Micky benutzten Schüsseln ($\{1,4,5\}$) gestrichen.

Wunschsorten	Apfel	Weintraube	Brombeere
Schüsseln	$\{1,4,5\}$	$\{2,3,6\}$	$\{1,4,5\}$

Das gleiche geschieht nun auch mit Minnies Spieß ({ Weintraube, Banane, Pflaume }; $\{3,5,6\}$). Weintraube gehört zu den Wunschsorten, Banane und Pflaume tun dies aber nicht.

Wunschsorten	Apfel	Weintraube	Brombeere
Schüsseln	$\{1,4\}$	$\{3,6\}$	$\{1,4\}$

Bei Gustavs Spieß ({ Apfel, Brombeere, Erdbeere }; $\{1,2,4\}$) gehören Apfel und Brombeere zu den Wunschsorten, Erdbeere jedoch nicht.

Wunschsorten	Apfel	Weintraube	Brombeere
Schüsseln	$\{1,4\}$	$\{3,6\}$	$\{1,4\}$

Da Gustav jedoch genau die Komplementärmenge von Minnie hat, können anhand seines Spießes keine weiteren Schüsseln gestrichen werden. Mit Daisys Spieß ({ Erdbeere, Pflaume }; $\{2,6\}$) wird jedoch die Schüssel 6 für die Weintrauben ausgeschlossen.

Wunschsorten	Apfel	Weintraube	Brombeere
Schüsseln	$\{1,4\}$	$\{3\}$	$\{1,4\}$

Damit sind alle Beobachtungen verarbeitet. Die Vereinigung der drei resultierenden Schüsselmengen ist $\{1,3,4\}$ und hat genau so viele Elemente wie Wunschsorten, nämlich drei. Das bedeutet, dass $\{1,3,4\}$ genau die Schüsseln sind, aus denen sich Donald bedienen muss. Es ist dabei kein Problem, dass für Apfel und Brombeere nicht eindeutig bestimmt werden konnte, in welcher Schüssel diese Sorten sich befinden. Entscheidend ist nur, dass Donald diese beiden Sorten auf seinen Spieß bekommt, indem er sich aus den Schüsseln 1 und 4 bedient.

2.3 Korrektheit des Algorithmus

Bei der Betrachtung der Lösbarkeit einer Eingabe fällt auf: Wenn zwei Obstsorten immer gemeinsam auftreten oder gemeinsam nicht auftreten, können ihre Schüsseln nicht von einander unterschieden werden. Wenn nun eine dieser Obstsorten zur Wunschmenge gehört und die andere nicht, existiert keine Lösung. Existiert kein Paar von Obstsorten, das diese Bedingungen erfüllt, muss eine eindeutige Lösung existieren.

Wenn eine Lösung existiert, so wird sie auch von dem Algorithmus gefunden. Für jedes Paar von Obstsorten, bei dem genau eine Obstsorte zur Wunschmenge gehört, muss gelten, dass ein Spieß existiert, bei dem genau eine der beiden Obstsorten vorkommt. Bei Betrachtung dieses

Spießes werden nun alle Schüsseln gestrichen, die für die beiden Obstsorten noch in Frage kommen. Sei die Menge der benutzten Schüsseln für den Spieß M und die Menge der nicht benutzten Schüsseln $\overline{M}$. Nun wird die Menge der Schüsseln einer Obstsorte mit M geschnitten und die der anderen Obstsorte mit $\overline{M}$ geschnitten.

Da keine Schüssel sowohl in M als auch $\overline{M}$ existieren kann, können sich auch die Schüsselmengen der beiden Obstsorten keine gemeinsamen Schüsseln mehr teilen. Wird nun eine Schüssel aus der Schüsselmenge der Wunschsorte ausgewählt, so ist sicher, dass die zweite Obstsorte sich nicht darin befindet. Geschieht dies für alle eben genannten Paare, so gibt es eine eindeutige Lösung. Für alle Schüsseln in den Schüsselmengen kann ausgeschlossen werden, dass sie keine Wunschsorte enthält.

2.4 Implementierung

Ein großer Faktor für die Performanz des Algorithmus ist die Implementierung der Mengen. Hierfür gibt es einige Möglichkeiten, wobei die einfachsten eine Liste oder ein Hashset von natürlichen Zahlen sind. Eine sehr performante Lösung sind jedoch Bitvektoren (manchmal auch Bitarrays genannt). Hierbei wird die Information ob eine Schüssel i in der Menge vorkommt oder nicht in dem Bit an der Stelle i gespeichert. Um schnell zu überprüfen ob eine Obstsorte auf einem Spieß zur Wunschmenge gehört, kann für die Wunschmenge Hashset oder ein Boolarray verwendet werden. In der untenstehenden Implementation wird eine Hashmap (in C# Dictionary genannt) verwendet um eine Abbildung von Obstsorten nach Indizes umzusetzen. Diese Indizes werden dann für ein Boolarray verwendet.

2.5 Laufzeitanalyse

Wenn die Anzahl der Obstsorten (und Schüsseln) N, die Anzahl der Wunschsorten K und die Anzahl Beobachtungen L ist, so hat der Algorithmus folgende Laufzeit:

$$T(N,K,L) = \mathcal{O}(L \cdot (N+K))$$

Die Laufzeit für eine Beobachtung setzt sich zusammen aus der Zeit die benötigt wird um die maximal N Obstsorten auf einem Spieß einzulesen und folgenden K Tests ob Wunschsorte auf dem Obstspieß vor kam oder nicht. Die Mengenoperationen Negation, Schnitt und Vereinigung, die zum entfernen der Schüsseln verwendet werden, haben bei kleinen N konstante Laufzeit. Das Überprüfen ob eine Obstsorte zur Wunschmenge gehört, ist ebenfalls in konstanter Zeit möglich.

Wird die Menge der Obstsorten jedoch viel größer als in den Beispielen, können die Mengenoperationen nicht mehr in konstanter Zeit bearbeitet werden. So würde folgende Laufzeit resultieren:

$$T(N,K,L) = \mathcal{O}(L \cdot (N+N \cdot K))$$

2.6 Beispiele

Es folgen die Lösungen für Beispieldatein:

spiesse1.txt

Clementine	Erdbeere	Grapefruit	Himbeere	Johannisbeere
$\{1\}$	$\{2,4\}$	$\{7\}$	$\{2,4\}$	$\{5\}$

Lösung gefunden: $\{1,2,4,5,7\}$

spiesse2.txt

Apfel	Banane	Clementine	Himbeere	Kiwi	Litschi
$\{1\}$	$\{5,10,11\}$	$\{5,10,11\}$	$\{5,10,11\}$	$\{6\}$	$\{7\}$

Lösung gefunden: $\{1,5,6,7,10,11\}$

spiesse3.txt

Clementine	Erdbeere	Feige	Himbeere	Ingwer	Kiwi	Litschi
$\{5\}$	$\{8\}$	$\{7,10\}$	$\{1\}$	$\{7,10\}$	$\{12\}$	$\{2,11\}$

Keine eindeutige Lösung gefunden: $\{1,2,5,7,8,10,11,12\}$
Es gibt keine eindeutige Schüsselmenge für Litschi und keine weitere(n) Wunschsorte(n) mit der gleichen Schüsselmenge.

spiesse4.txt

Apfel	Feige	Grapefruit	Ingwer	Kiwi	Nektarine	Orange	Pflaume
$\{9\}$	$\{13\}$	$\{8\}$	$\{6\}$	$\{2\}$	$\{7\}$	$\{14\}$	$\{12\}$

Lösung gefunden: $\{2,6,7,8,9,12,13,14\}$

spiesse5.txt

Apfel	Banane	Clementine	Dattel	Grapefruit	Himbeere	Mango
$\{1,4,19\}$	$\{3,9\}$	$\{20\}$	$\{6\}$	$\{1,4,19\}$	$\{5\}$	$\{1,4,19\}$
Nektarine	Orange	Pflaume	Quitte	Sauerkirsche	Tamarinde	
$\{14\}$	$\{2,16\}$	$\{10\}$	$\{3,9\}$	$\{2,16\}$	$\{12\}$	

Lösung gefunden: $\{1,2,3,4,5,6,9,10,12,14,16,19,20,\}$

spiesse6.txt

Clementine	Erdbeere	Himbeere	Orange	Quitte	Rosine	Ugli	Vogelbeere
$\{7\}$	$\{10\}$	$\{18\}$	$\{20\}$	$\{4\}$	$\{11,15\}$	$\{11,15\}$	$\{6\}$

Lösung gefunden: $\{4,6,7,10,11,15,18,20\}$

spiesse7.txt

Apfel	Clementine	Dattel	Grapefruit	Mango	Sauerkirsche
$\{3,10,20,26\}$	$\{24\}$	$\{6,16,17\}$	$\{3,10,20,26\}$	$\{6,16,17\}$	$\{8,14\}$
Tamarinde	Ugli	Vogelbeere	Xenia	Yuzu	Zitrone
$\{5,23\}$	$\{18,25\}$	$\{6,16,17\}$	$\{3,10,20,26\}$	$\{8,14\}$	$\{5,23\}$

Keine eindeutige Lösung gefunden: $\{3,5,6,8,10,14,16,17,18,20,23,24,25,26\}$
Es gibt keine eindeutige Schüsselmenge für Ugli und keine weitere(n) Wunschsorte(n) mit der gleichen Schüsselmenge.
Es gibt keine eindeutige Schüsselmenge für Apfel, Grapefruit, Xenia und keine weitere(n) Wunschsorte(n) mit der gleichen Schüsselmenge.

2.7 Bewertungskriterien

Die aufgabenspezifischen Bewertungskriterien werden hier erläutert.

1. Lösungsweg

(1) *Problem adäquat modelliert*: Die Aufgabe lässt sich mit bipartitem Matching lösen, es geht aber auch ohne graphentheoretische Überlegungen. Entscheidend ist die Verwendung von Mengen, insbesondere bei den Beobachtungen; diese enthalten keine über Mengen hinausgehende Information (etwa eine Reihenfolge).
(2) *Beobachtungen werden voll ausgenutzt*: Das Verfahren soll alle in den Beobachtungen enthaltene Information ausnutzen (ggf. auch implizit), also anhand der Beobachtung die Schüsselmengen sowohl der beobachteten als auch der nicht beobachteten Sorten eingrenzen.
(3) *Verfahren läuft komplett automatisch ab*: Sollte das Problem mit Hilfe eines externen Moduls angegangen werden, muss der Input für dieses Modul automatisch aus den gemäß Aufgabenstellung eingelesenen Daten generiert werden, und bei Bedarf ist auch die Ausgabe des Moduls noch automatisch zu verarbeiten.
(4) *Laufzeit des Verfahrens in Ordnung*: Die Aktualisierung der Zuordnungen zwischen Schüsseln und Obstsorten anhand einer Beobachtung ist auf jeden Fall in $O(N^2)$ zu erledigen; eine schwächere Laufzeit ist nicht akzeptabel.
(5) *Speicherbedarf in Ordnung*: Die Zuordnungen zwischen Schüsseln und Obstsorten lassen sich in einer Matrix der Größe N^2 speichern. Diese Größenordnung sollte nicht überschritten werden.
(6) *Verfahren mit korrekten Ergebnissen*: Das Verfahren sollte in allen Fällen korrekte Ergebnisse liefern, also korrekt bestimmen, (a) ob die Schüsselmenge der Wunschsorten eindeutig angegeben werden kann und (wenn ja) (b) welche Menge dies ist. Auch der in Beispiel 1 gegebene Fall, dass es zu einer Wunschsorte keine Beobachtung gibt, muss korrekt behandelt werden.

2. Theoretische Analyse

(1) *Verfahren / Qualität ingesamt gut begründet*: Es ist nicht schwierig, ein korrekt arbeitendes Verfahren zur Lösung dieser Aufgabe zu finden. Umso wichtiger ist es, dass die Korrektheit des Verfahrens begründet wird. Falls explizit auf ein bekanntes Verfahren wie bipartites Matching (mit brauchbarer Referenz) oder auf eine externe Lösungskomponente (wie etwa einen constraint solver) zurückgegriffen wird, muss dazu sauber begründet werden, warum diese Aufgabe mit dem Verfahren bzw. der Komponente gelöst werden kann. Wurde das Lösungsverfahren selbst entwickelt, ist eine gut nachvollziehbare Argumentation zur Korrektheit Pflicht. Dabei genügt es nicht zu begründen oder festzustellen, dass die üblicherweise mit jeder Beobachtung vorgenommenen Einschränkungen der Schüsselmengen logisch richtig sind. Außerdem muss erklärt werden, dass das Verfahren „vollständig“ ist, also eine eindeutige Schüsselmenge immer findet, wenn es sie gibt. Dieser zweite Teil fehlte häufig; dann wurden 2 Punkte abgezogen.
(2) *Gute Überlegungen zur Laufzeit des Verfahrens*: Die Laufzeit des Verfahrens muss nicht zwingend formal, aber nachvollziehbar und korrekt charakterisiert werden.
(3) *Bedeutung von Mengenoperationen erkannt*: Für die allermeisten Lösungsansätze spielen Operationen auf Mengen eine (auch für die Laufzeit) entscheidende Rolle. Dies sollte

zumindest implizit erkannt und bei der Umsetzung beachtet worden sein. Es wird z. B. anerkannt, wenn Mengen als Bitvektoren oder Zuordnungen über Hashsets oder Zuordnungsmatrizen realisiert sind und dies in der Dokumentation auch angegeben ist. Natürlich kann man die Aufgabe auch mit einfachen Listen etc. lösen; aber in der 2. Runde sollte eine Lösung so gut wie möglich sein, und viele Teilnehmende haben den Vorteil spezifischer Datenstrukturen erkannt und umgesetzt.

3. Dokumentation

(3) *Teil (a) bearbeitet*: Teilaufgabe (a) sollte die Teilnehmenden anregen, sich zunächst anhand des Beispiels aus der Aufgabenstellung Gedanken über das zu lösende Problem zu machen. Es ist in Ordnung, wenn die Dokumentation nicht entsprechend gegliedert ist und unmittelbar eine allgemeine Lösungsidee angibt. Das Beispiel aus der Aufgabenstellung kann dann zur Demonstration der Idee genutzt werden; jedenfalls sollte es nicht komplett ignoriert werden.

(4) *Vorgegebene Beispiele dokumentiert*: Es sollten alle vorgegebenen Beispiele bearbeitet und dokumentiert worden sein. Wenn nicht zu allen, aber zu mindestens 4 der 7 Beispiele Ergebnisse dokumentiert sind und darunter mindestens eines ohne eindeutige Antwort (Beispiele 3 und 7) ist, gibt es 2 Punkte, bei weniger bis zu 4 Punkte Abzug.

(6) *Ergebnisse nachvollziehbar dargestellt*: Zunächst muss klar ausgegeben werden, ob es eine eindeutige Schüsselmenge gibt oder nicht; ansonsten wird 1 Punkt abgezogen. Dann genügt es *nicht*, einfach nur die Schüsseln anzugeben, aus denen Donald Obst nehmen sollte, auch wenn die Aufgabenstellung nicht mehr verlangt. Die Zuordnung zwischen Obstsorten und Schüsseln muss mindestens für die Wunschsorten angegeben werden; nur so kann einigermaßen nachvollzogen werden, wie es zu Donalds Schüsselmenge kommt. Erfreulicherweise wird das von den meisten Einsendungen auch so gehandhabt.

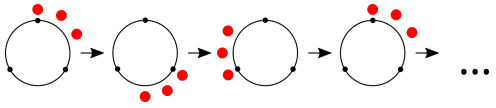
(7) *Informative Meldung bei Nicht-Eindeutigkeit*: Falls es nicht möglich ist, für die Wunschsorten eine eindeutige Schüsselmenge zu bestimmen, verlangt die Aufgabenstellung explizit eine „möglichst informative Meldung“. Hier wird erwartet, dass angegeben wird, bzgl. welcher Obstsorte(n) es noch Klärungsbedarf gibt; so wird es Donald ermöglicht, bei Bedarf nach entsprechenden Spießen Ausschau zu halten.

Aufgabe 3: Eisbudendilemma

Bei dieser Aufgabe sollen die Standorte der Eisbuden so verteilt werden, dass keine andere Verteilung bei einer Abstimmung mehr als die Hälfte der Stimmen erhält.

Eine naheliegende Frage, die man sich stellen kann, ist: Ist es möglich, dass man durch eine Reihe von erfolgreichen Abstimmungen wieder bei einer Verteilung landet, die es schon einmal gab? Wenn das nämlich nicht der Fall wäre, so ließe sich einfach eine stabile Verteilung finden, indem man so lange nach „besseren“ Verteilungen sucht, bis man keine mehr findet.

Leider ist das aber doch der Fall, wie das folgende Beispiel zeigt. Nimmt man einen See mit Umfang 12 und platziert die Häuser an den Positionen 0, 4 und 8, so lassen sich die folgenden Verteilungen der Eisbuden (0,1,2), (4,5,6) und (8,9,10) in dieser Reihenfolge immer wieder ineinander überführen, es werden immer zwei Häuse für die angegebene Verschiebung der Eisbuden stimmen:

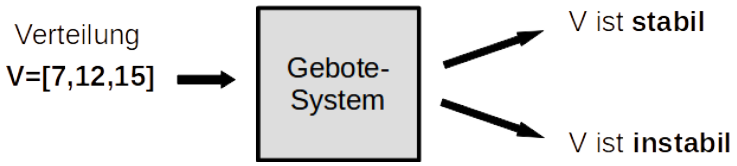


Wir brauchen also ein aufwändigeres Verfahren, um diese Aufgabe zu lösen. Wir wollen hier die grundlegendsten Verfahren vorstellen.

Im Folgenden sei L der Umfang der Siedlung (also die Anzahl der möglichen Adressen der Häuser und Eisbuden) und N die Anzahl der Häuser der Siedlung. Ein Tripel aus drei Eisbuden-Adressen $i, j, k \in \{0, \ldots, L-1\}$ bezeichnen wir als *Verteilung* $V = [i, j, k]$ von Eisbuden.

3.1 Gebote-System

Wir lösen diese Aufgabe mit Hilfe eines Verfahrens, das wir *Gebote-System* nennen wollen. Das Gebote-System ist eine Methode, die eine Verteilung $V = [i, j, k]$ (ohne Beschränkung der Allgemeinheit mit $i < j < k$) von drei Eisbuden als Input bekommt und ausrechnet, ob diese Verteilung stabil ist oder nicht.



Wenn man nun kurz das Gebote-System als eine gegebene „Black-Box“ annimmt, dann haben wir damit diese Aufgabe gelöst: Wir können alle möglichen Eisbudenverteilungen $V = [i, j, k]$ nacheinander in das Gebote-System füttern, also beginnend mit $V = [0,1,2], V = [0,1,3], V = [0,1,4], \ldots$ und bei z.B. einer Siedlung mit Umfang $L = 17$ endend mit $V = [14,15,16]$. Wenn sich dabei herausstellt, dass irgendeine dieser Verteilungen stabil ist, wäre damit gezeigt, dass die Siedlung eben eine stabile Eisbudenverteilung besitzt. Andernfalls wäre gezeigt, dass die Siedlung eben keine einzige stabile Eisbudenverteilung besitzt. Beides löst die Aufgabenstellung.

Für die folgenden Beschreibungen nutzen wir als Beispiel eine kleine Siedlung mit Umfang $L = 17$ und $N = 9$ Häusern. Die Siedlung hat Häuser an den Adressen 0, 3, 6, 7, 8, 11, 13, 14 und 15 und ist in Abbildung 3.1 gezeigt.

Verteilung V

Das Gebote-System braucht eine Verteilung $V = [i, j, k]$ als Input. Wir wählen jetzt zu Demonstrationszwecken eine aus, nämlich die Verteilung $V = [7, 12, 15]$; bei dieser Verteilung sind alle Dynamiken des Gebote-Systems gut zu sehen. Diese Verteilung soll beispielhaft geprüft werden; es wird also ermittelt, ob die Platzierung der drei Eisbuden auf die Adressen 7, 12 und 15 eine stabile Verteilung ist.

Die Verteilung ist in Abbildung 3.2 abgebildet.

Distanzen

Nachdem die Verteilung festgelegt ist, wird für jedes Haus seine Distanz d zur nächstgelegenen Eisbude berechnet. Das geht im Prinzip ganz direkt; man muss nur aufpassen, dass der kürzeste Weg unter Umständen auch über die Adresse 0 führen und die Differenz von zwei Adressen dann eine negative Zahl sein kann. Solche Fälle müssen mit einer Modulo-Funktion oder einer entsprechenden Fallunterscheidung abgefangen werden, damit immer positive Distanzen herauskommen. Die kürzeste Distanz zur nächsten Eisbude muss für jedes Haus bekannt sein, damit es (bzw. seine Bewohner) im nächsten Schritt korrekt seine Stimmen abgeben kann.

Stimmen

Jetzt führen wir eine auf einzelne Adressen bezogene Stimmabgabe durch. Diese Stimmen werden der „Rohstoff“ für die nächsten Schritte sein.

> Jedes Haus gibt eine Stimme ab für jede Adresse, die näher liegt als die nächstgelegene Eisbude.

Schauen wir uns in Abbildung 3.3 kurz an, wie diese Stimmabgabe z.B. für ein Haus mit Distanz $d = 3$ aussieht. Wegen $d = 3$ gibt es fünf Adressen, die näher an dem Haus liegen als die nächstgelegene Eisbude. Für jede dieser fünf Adressen gibt das Haus eine Stimme ab.

Man erkennt, dass die abgegebenen Stimmen exakt dem Stimmverhalten gegenüber neuen potentiellen Verteilungen entsprechen. Jedes Haus stimmt genau dann mit „Ja“ für eine neue Verteilung, wenn diese eine (Eisbuden-)Adresse enthält, für die das Haus eine Stimme abgegeben hat. Für jede Adresse lässt sich an der Summe aller abgegebenen Stimmen ablesen, wie viele Ja-Stimmen die Adresse bei einer Dorfabstimmung „bringen würde“. Das Ergebnis der

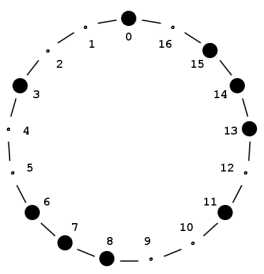


Abbildung 3.1: Eine Beispielsiedlung.

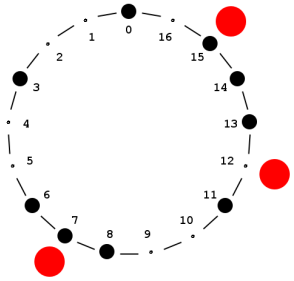


Abbildung 3.2: Die Verteilung $V = [7, 12, 15]$ der drei Eisbuden in der Beispielsiedlung.

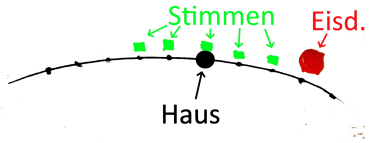


Abbildung 3.3: Ein Haus gibt seine Stimmen ab.

Adress-Stimmabgabe für unsere Beispiel-Siedlung ist in Abbildung 3.4 gezeigt und in Tabelle 3 ausgeschrieben.

In Tabelle 3 gibt es noch die Spalte „Häuser“; sie protokolliert, von welchen Häusern die Stimmen jeweils kommen. Das wird später im Abschnitt „Mergen“ wichtig, damit einzelne Häuser nicht doppelt gezählt werden.

Gebote (Den Beitrag einzelner Adressen bestimmen)

Die Daten aus Tabelle 3 schreiben wir nun leicht um, um sie kompakter und modularer zu haben. Jede Zeile in Tabelle 3, also die Abstimmungsinformation zu jeder Adresse, wird zusammengefasst zu einem neuen Objekt „Gebot“. Ein einzelnes Gebot speichert damit für eine Adresse deren Anzahl an Stimmen und die Liste der Häuser, die für die Adresse gestimmt haben.

Adresse	Stimmen	Häuser	Adresse	Stimmen	Häuser
0	2	0, 3	9	0	–
1	2	0, 3	10	0	–
2	1	3	11	1	11
3	1	3	12	0	–
4	1	3	13	1	13
5	1	3	14	1	14
6	2	3, 6	15	0	–
7	0	–	16	1	16
8	1	8			

Tabelle 3: Das Abstimmungsverhalten der Häuser.

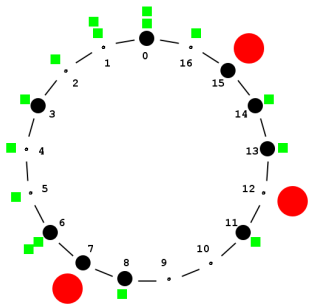


Abbildung 3.4: Das Abstimmungsergebnis, die grünen Quadrate stehen jeweils für eine Stimme.

$G_0 : [a = 0, s = 2, h = [0, 3]]$
$G_1 : [a = 1, s = 2, h = [0, 3]]$
$G_2 : [a = 2, s = 1, h = [3]]$
$G_3 : [a = 3, s = 1, h = [3]]$
$G_4 : [a = 4, s = 1, h = [3]]$
$G_5 : [a = 5, s = 1, h = [3]]$
$G_6 : [a = 6, s = 2, h = [3, 6]]$
$G_8 : [a = 8, s = 1, h = [8]]$
$G_{11} : [a = 11, s = 1, h = [11]]$
$G_{13} : [a = 13, s = 1, h = [13]]$
$G_{14} : [a = 14, s = 1, h = [14]]$
$G_{16} : [a = 16, s = 1, h = [16]]$

Die Gebote packen wir in eine Liste g, sortiert nach der Stimmanzahl s. Die Sortierung sorgt dafür, dass am Anfang der Liste gleich die vielversprechendsten Gebote stehen und dass im späteren Verlauf die Stimmen der restlichen Gebote „hinten raus“ gezielt abgeschätzt werden können, was Optimierungen erlaubt (siehe im späteren Abschnitt „Frühzeitiger Abbruch“).

Gebote	$G = [$	$G_0,$	$G_1,$	$G_6,$	$G_2,$	$G_3,$	$G_4,$	$G_5,$	$G_8,$	$G_{11},$	$G_{13},$	$G_{14},$	$G_{16}].$
Stimmanzahlen	$S = [$	2,	2,	2,	1,	1,	1,	1,	1,	1,	1,	1,	1]

Beim Umwandeln in Gebote lassen wir diejenigen Adressen wegfallen, die keine Stimme erhalten haben. Diese Adressen können bei Abstimmungen über Verteilungen keine Ja-Stimme beitragen und kommen damit als potentielle neue Standorte nicht in Frage.

Gebote-Iteration (Neue Verteilungen prüfen)

Wir haben nun die nötige Vorarbeit geleistet und können ein *Stabilitätskriterium* aufstellen, anhand dessen entschieden werden kann, ob eine zu prüfende Verteilung V stabil ist oder nicht. Als Beispiel verwenden wir weiter die Verteilung $V = [7, 12, 15]$.

> *Stabilitätskriterium für Verteilung V*: Lässt sich aus der Geboteliste G ein Tripel von drei Geboten $[G_i, G_j, G_k]$ wählen, die zusammen die Mehrheit der Stimmen, also mindestens $\lfloor (N/2+1) \rfloor$ Stimmen (in unserem Beispiel 5 Stimmen) auf sich vereinigen können? Falls ja, dann gibt es für die neue Verteilung $V' = [i, j, k]$ eine Mehrheit, und V ist instabil. Falls nein, ist V stabil.

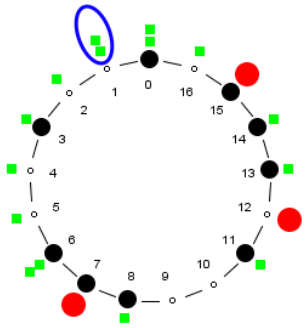


Abbildung 3.5: Das Gebot $G_1 = [a = 1, s = 2, h = [0,3]]$. Hier ist a die Adresse, s ihre Anzahl an Stimmen und h die Liste der für die Adresse stimmenden Häuser.

Wir überprüfen dieses Kriterium, indem wir mit drei geschachtelten Schleifen alle möglichen Tripel $[G_i, G_j, G_k]$ (weiterhin mit $i < j < k$) betrachten und für jedes Tripel seine Stimmanzahl ausrechnen; das nennen wir „Gebote-Iteration“.

Zwei Fälle können während der Gebote-Iteration eintreten:

Fall 1: Wir treffen irgendwann auf ein Tripel $T = [G_i, G_j, G_k]$, das tatsächlich $\lfloor (N/2+1) \rfloor$ oder mehr Stimmen auf sich vereinen kann. Dieses Tripel repräsentiert dann eine neue Verteilung $V' = [i, j, k]$, die bei einer Abstimmung in der Siedlung eine Mehrheit der Stimmen bekommen würde. Damit wäre gezeigt, dass die aktuelle Verteilung $V = [7, 12, 15]$ instabil ist, nämlich (mindestens) gegenüber V'.

Fall 2: Wir treffen niemals auf ein Tripel $T = [G_i, G_j, G_k]$ mit $\lfloor (N/2+1) \rfloor$ oder mehr Stimmen. Daraus können wir schließen, dass die aktuelle Verteilung $V = [7, 12, 15]$ stabil sein muss. Es schadet dabei nicht, dass nicht für alle möglichen neuen Verteilungen $V' = [i, j, k]$ auch eine Gebotetripel $T = [G_i, G_j, G_k]$ existiert. Denn wir hatten nur für diejenigen Adressen keine Gebote gespeichert, die von keinem Haus eine Stimme bekommen hatten. Diese „0-Adressen“ können nicht Bestandteil einer neuen „Mehrheits-Verteilung“ sein.

Merging Die Methode stimmanzahl(G_i, G_j, G_k) berechnet die Anzahl der Stimmen eines Gebote-Tripels als die Kardinalität (Anzahl der Elemente) der Vereinigung der Häusermengen der einzelnen Gebote: $|G_i.h \cap G_j.h \cap G_k.h|$. Dafür werden die Häuserlisten der ersten beiden Gebote G_i und G_j durch ein Reißverschlussverfahren miteinander gemerged. Die Laufzeit skaliert dabei linear mit den Listenlängen. Die so entstandene Menge der gemeinsamen Häuser wird dann nochmal mit der Häuserliste des dritten Gebots G_k gemerged; so entsteht die gesamte Liste der gemeinsamen Häuser. Die Anzahl der Häuser in dieser finalen Häuserliste ist dann die Stimmanzahl für dieses Gebote-Tripel.

Es reicht insbesondere nicht aus, die drei Stimmanzahlen der Gebote einfach zu addieren ($G_i.s + G_j.s + G_k.s$), denn dabei können Dopplungen auftreten. Wenn ein Haus bei mehreren Geboten seine Stimme abgegeben hat, würde das beim Addieren nicht bemerkt werden und das Haus fälschlicherweise doppelt (oder sogar dreifach) gezählt werden.

Zurück zu unserem Beispiel: Das allererste Tripel, das in den Schleifen betrachtet wird, ist $[G_0, G_1, G_6]$. Das Mergen durch stimmanzahl(G_0, G_1, G_6) ergibt dann eine finale Häuserliste $H = [S = 3, h = [0,3,6]]$. Sie zeigt, dass dieses Tripel nur auf 3 Stimmen kommt, also keine

Mehrheit erreicht, die für $N = 9$ Häuser bei 5 Stimmen liegt. Die Gebote-Iteration würde also weiter zum nächsten Tripel gehen.

Frühzeitiger Abbruch Tatsächlich muss man die Gebote-Iteration meist nicht komplett durchführen. Denn ab einem bestimmten Punkt können sämtliche noch folgenden Tripel mit einem Schlag ausgeschlossen werden, nämlich sobald rechnerisch klar ist dass sie keine Chance mehr auf eine Mehrheit haben. Dieser spezielle Punkt ist in unserem Beispiel in dieser Situation erreicht:

$$\begin{array}{lccccccccccccc} & & & i & & j & & k & & & & & & \\ G_s = [& 2, & 2, & 2, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1, & 1 \ ] \end{array}$$

Mit Blick auf dieses Beispiel: Da der Index i nur noch weiter nach rechts wandern wird und Index j und k nur noch über Gebote mit 1 Stimme iterieren, ist ab der gezeigten Situation klar, dass alle nachfolgenden Tripel nur noch maximal 4 Stimmen erreichen können, also keine Mehrheit. Man kann ab diesem Punkt also die Suche beenden und sofort alle drei Schleifen verlassen. Hat bis hierhin kein Tripel eine Mehrheit hatte, dann wird es auch später keines mehr geben. Man kann in dem Fall also sofort die Schlussfolgerung ziehen, dass V stabil ist.

Tatsächlich sind noch einige weitere Abkürzungen möglich, nämlich auch in den tieferen Ebenen der drei Schleifen. Diese erlauben dann das Überspringen von jeweils lokalen Abschnitten; das beschleunigt insgesamt aber nur nochmal um ca. zehn Prozent und wird deshalb hier nicht vertieft. Ein weiteres Detail ist, dass man den Merge der ersten beiden Gebote bereits in der zweiten Schleife durchführen und dann mehrmals wieder verwenden kann, was nochmal Zeit spart.

Hiermit ist das Gebote-System fertig erklärt. Die „Black-Box“ steht. Wir können sie jetzt mit einer beliebigen Verteilung V füttern und bekommen zurück, ob V stabil ist oder nicht.

3.2 Siedlungs-Iteration

Damit ergibt sich der einfachste Ansatz, um die Aufgabe Eisbudendilemma mit Hilfe des Gebote-Systems zu lösen: Es wird über alle möglichen Verteilungen $V = [i, j, k]$ iteriert (mit $i, j, k \in \{0, .., L-1\}$ und $i < j < k$), und jede Verteilung wird mit dem Gebote-System auf ihre Stabilität prüft.

Dieser Ansatz löst die Aufgabe. Entweder wir stoßen irgendwann auf eine Verteilung $V = [i, j, k]$, die im Gebote-System als stabil identifiziert wird; dann können wir diese stabile Verteilung als Endergebnis ausgeben (oder weitersuchen, um mögliche weitere stabile Verteilungen zu finden). Oder wir stoßen niemals auf eine stabile Verteilung; dann können wir ausgeben, dass die Siedlung eben keine stabilen Verteilungen besitzt.

3.3 Optimierungen

Es sind aber auch noch weitergehende Optimierungen und effizientere Verfahren denkbar. Beispielhaft wollen wir hier eine Optimierung vorstellen.

Der Ansatz der Optimierung ist, dass instabile Verteilungen V oft bereits viel schneller als instabil identifiziert werden können, durch einen direkten Vergleich mit einer Referenzverteilung V'. Dabei werden die einzelnen Distanzen der Häuser in den beiden Verteilungen V und V' in

linearer Laufzeit $\mathcal{O}(N)$ miteinander verglichen, und wenn in der Referenzverteilung V' mehr als die Hälfte der Häuser eine kürzere Distanz haben, ist gezeigt dass V instabil ist.

In der Siedlungs-Iteration wird dieser direkte Vergleich nun dem Gebote-System vorgeschaltet. Die Durchführung ergibt dann, dass etwa 95-99% aller Verteilungen V der Siedlungs-Iteration bereits durch diesen direkten Vergleich als instabil erkannt werden, was dann den deutlich teureren Aufruf des Gebote-Systems mit $\mathcal{O}(L^3 \cdot N)$ erspart. Die restlichen 1-5 % der Verteilungen, die nicht auf diese Weise aussortiert werden können, müssen dann weiterhin mit dem Gebote-System überprüft werden; aber ihre Zahl ist nun deutlich verringert.

Als Metrik für eine besonders gute Referenzverteilung V' nutzen wir die Gesamtdistanz: Die Summe der Distanzen der einzelnen Häuser zu den Eisbuden. Die intuitive Idee dabei ist, dass eine Verteilung im Allgemeinen „besser“ ist, je kleiner die Distanzen zwischen Eisbuden und Häusern sind. Das ist natürlich kein scharfes Kriterium, aber es reicht aus, um aus den Millionen an verfügbaren Verteilungen einige wenige überdurchschnittlich gute herauszufischen. Ganz zu Beginn des Algorithmus generieren wir deshalb mehrere tausend zufällige Verteilungen und speichern die besten fünf davon ab, d. h. die mit den niedrigsten Gesamtdistanzen. Diese ausgewählten Verteilungen sind dann die Referenzverteilungen V' für den restlichen Ablauf. Wenn sie dann 99% der Verteilungen als instabil aussortieren können beschleunigt sich die Laufzeit um den Faktor 100. Referenzverteilungen V' mit noch niedrigeren Gesamtdistanzen können mithilfe eines Gradient-Descent Verfahren gefunden werden. Dabei startet man mit einer zufälligen Verteilung und „ruckelt“ danach iterativ an den Eisbuden, um die Gesamtdistanz jeweils ein bisschen weiter zu verringern, bis man mit der Verteilung an einem lokalen Minimum ankommt. In der Nähe dieser lokalen Minima liegen interessanterweise oft auch schon die stabilen Verteilungen. Bei den Beispielsiedlungen 5 und 7 können durch das Gradient-Descent Verfahren so die ersten stabilen Verteilungen nach ca. 0,01 s gefunden werden.

3.4 Laufzeitanalyse

Hier analysieren wir, wie die Laufzeit unseres Algorithmus von den Größen L und N (Umfang und Anzahl Häuser) einer Siedlung abhängt bzw. wie sie mit dieser skaliert.

Gebote-System:

- Distanzen berechnen: $\mathcal{O}(N)$
- Stimmen setzten: $\mathcal{O}(N \cdot L)$
- Gebote sortieren: $\mathcal{O}(G \log G)$
- Gebote-Iteration: $\mathcal{O}(G^3)$
- Mergen je Tripel: $\mathcal{O}(N)$

Die teuerste Aktion im Gebote-System ist die Gebote-Iteration mit $\mathcal{O}(G^3)$, G steht hier für die Anzahl an Geboten. Diese Anzahl ist sehr ähnlich wie Anzahl an Adressen ingesamt, wir können hier deshalb gut G durch L ersetzen. Bei den Tripeln muss außerdem jedes noch gemerged werden, das kostet jeweils $\mathcal{O}(N)$, insgesamt kommt das Gebote-System damit auf eine Laufzeit in $\mathcal{O}(L^3 \cdot N)$.

Die Siedlungs-Iteration ruft für jede mögliche Verteilung genau einmal das Gebote-System auf. Die Anzahl an Verteilungen lässt sich glücklicherweise direkt mithilfe des Binomialkoeffizenten angeben, es sind $|V| = \binom{L}{3}$ Verteilungen (gesprochen „L wähle 3“, weil jede Verteilung eine Auswahl von 3 Adressen aus L Adressen ist). Der Binomialkoeffizient lässt sich nun weiter

vereinfachen, allgemein ist er $\binom{n}{k} = n!/(n-k)!/k!$, d.h. in unserem Fall $|V| = (L3) = L!/(L-3!)/3!$. Das obere $L!$ kann umgeschrieben werden in $L! = L \cdot (L-1) \cdot (L-2) \cdot (L-3)!$, dadurch lässt sich nun das $(L-3)!$ über und unter dem Bruch rauskürzen. Wir erhalten $|V| = 1/6L \cdot (L-1) \cdot (L-2) = 1/6L^3 - 1/2L^2 + 1/3L$, d. h. die Anzahl an Verteilungen $|V|$ skaliert mit $\mathcal{O}(L^3)$ und damit skaliert auch die Siedlungs-Iteration mit $\mathcal{O}(L^3)$. Die selbe Rechnung haben wir auch bereits für die Gebote-Iteration gemacht, um auf die Skalierung G^3 zu kommen („G wähle 3“).

$$\begin{aligned} \text{Laufzeit}_{\text{Gesamt}} &= \text{Laufzeit}_{\text{Gebote-System}} \cdot \text{Laufzeit}_{\text{Aufrufe}} && (3.1)\\ &\in \mathcal{O}(L^3 \cdot N \cdot L^3) && (3.2)\\ &= \mathcal{O}(L^6 \cdot N). && (3.3) \end{aligned}$$

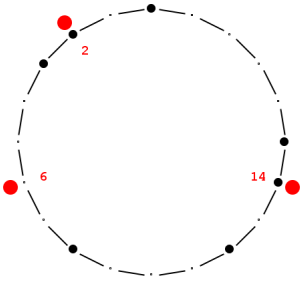
Die Gesamtlaufzeit des Algorithmus ist somit gegeben durch $\mathcal{O}(L^6 \cdot N)$.

3.5 Ergebnisse

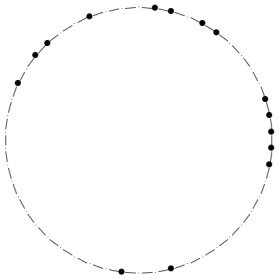
Hier sind Ergebnisse für die sieben Beispieleingaben. Man sieht, dass die Siedlungen 1, 3, 5 und 7 stabil sind und die Siedlungen 2, 4 und 6 instabil.

Die Anzahl der stabilen Verteilungen ist außerdem erstaunlich klein; bei `eisbuden5.txt` gibt es z. B. nur zwei stabile Verteilungen, obwohl die Siedlung insgesamt über 2 Millionen Verteilungen besitzt. Die stabilen Verteilungen von z.B. Beispiel 5 und Beispiel 7 sind zudem alle nur „gerade so“ stabil, denn sie haben jeweils Gegenvorschläge mit 50 % Ja-Stimmen (18 von 36 bzw. 20 von 40), die also nur an einer fehlenden Stimme scheitern. Eine mögliche Erweiterung wäre also zu ermitteln, ob es reicht, bereits ein einzelnes Haus zu entfernen oder nur umzustellen, um die Stabilitätseigenschaften einer Siedlung zu ändern.

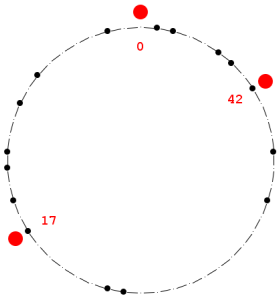
Datei	Anzahl stabile Verteilungen	Liste der stabilen Verteilungen	Dauer mit „Siedlungs-Iteration“ einfach	Dauer mit „Siedlungs-Iteration“ optimiert
eisbuden1.txt L=20, N=7	11	[2,5,14], [2,6,14] [2,7,14], [2,8,14] [2,9,14], [2,10,14] [2,10,15], [2,11,14] [2,11,15], [2,12,14] [2, 12, 15]	0.009 s	0.010 s
eisbuden2.txt L=50, N=15	keine	–	0.150 s	0.033 s
eisbuden3.txt L=50, N=16	2	[0, 17, 42] [1, 17, 42]	0.166 s	0.054 s
eisbuden4.txt L=100, N=19	keine	–	2.5 s	0.223 s
eisbuden5.txt L=247, N=24	6	[83, 128, 231] [83, 129, 231] [83, 129, 232] [83, 130, 231] [83, 130, 232] [83, 130, 233]	111.6 s	2.93 s
eisbuden6.txt L=437, N=36	keine	–	1616 s (27 min)	23.8 s
eisbuden7.txt L=625, N=40	16	[114,285,416] [114,285,417] [114,285,418] [114,285,419] [114,285,420] [114,289,416] [114,289,417] [114,289,418] [114,289,419] [114,289,420] [115,285,420] [115,289,420] [116,285,420] [116,289,420] [117,285,420] [117,289,420]	5083 s (85 min)	47.6 s



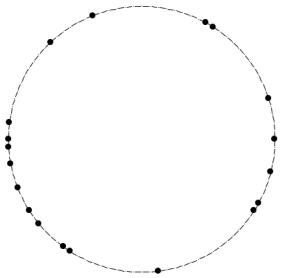
eisbuden1.txt



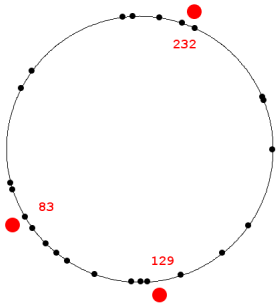
eisbuden2.txt



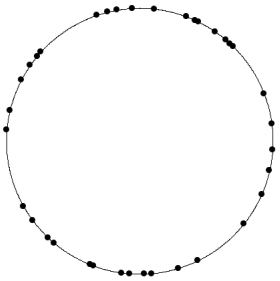
eisbuden3.txt



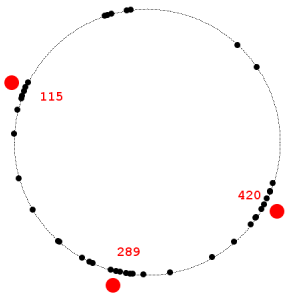
eisbuden4.txt



eisbuden5.txt



eisbuden6.txt



eisbuden7.txt

3.6 Bewertungskriterien

Die aufgabenspezifischen Bewertungskriterien werden hier erläutert.

1. Lösungsweg

(1) *Problem adäquat modelliert*
- Die kürzeste Distanz zwischen zwei Adressen (also zwischen zwei Häusern und insbesondere zwischen einem Haus und einer Eisbude) muss korrekt definiert und berechnet werden, also auch über die Adresse 0 hinweg.
- Entscheidend für eine korrekte Lösung ist die Relation zwischen zwei Eisbuden-Verteilungen V und V', die besagt, dass bei aktueller Verteilung V der neue Vorschlag V' eine Mehrheit erhält. Dies sollte erkannt und sauber beschrieben sein.
- Darüber hinaus, in der Regel auf Grundlage der obigen Relation, sollte ein Stabilitätskriterium für eine Verteilung V beschrieben sein: V ist stabil, wenn es keine „bessere Verteilung" V' im obigen Sinne gibt.

(2) *Verfahren nicht unnötig ineffizient*: Unabhängig von der erreichten Größenordnung der Laufzeit (vgl. nächstes Kriterium) sollten unnötige Berechnungen auf jeden Fall vermieden werden. Ein Beispiel: Die Menge der zu prüfenden Verteilungen $V = [i, j, k]$ sollte nicht alle beliebigen Kombinationen von i, j, k enthalten, sondern kann insbesondere durch $i < j < k$ eingeschränkt werden.

(3) *Laufzeit des Verfahrens in Ordnung*: Der direkte Weg, für alle Verteilungen jeweils alle anderen Verteilungen zu betrachten, kommt auf eine Laufzeit von $\mathrm{O}(L^6)$. Bleibt es dabei, werden Punkte abgezogen. Es wird erwartet, dass insbesondere bei der Betrachtung der anderen Verteilungen deutlich optimiert wird, ob mit guter Heuristik oder auf analytischer Grundlage. Ein linearer Aufwand an dieser Stelle und damit insgesamt eine Laufzeit von $\mathrm{O}(L^4)$ ist dabei der Maßstab (ein möglicher zusätzlicher Faktor N der Häuser sei ignoriert); auch die größeren Beispiele sollten in weniger als einer Minute (wie in dieser Beispiellösung) oder wenigen Sekunden bearbeitet werden können. Bei $\mathrm{O}(L^3)$ (also z. B. konstanter Laufzeit für die Betrachtung anderer Verteilungen) gibt es einen Pluspunkt. Mit hohem Aufwand lassen sich sogar noch bessere Laufzeiten erreichen (evtl. bei Beschränkung auf Bestimmung nur einer stabilen Konstellation); entsprechend kann es dann auch mehr Pluspunkte geben.

(4) *Speicherbedarf in Ordnung*: Der Speicherbedarf ist bei dieser Aufgabe eher unkritisch. Es ist kein Problem, z. B. mit quadratisch bzgl. der Anzahl N der Häuser wachsendem Aufwand Ergebnisse von Vorberechnungen abzulegen. Punktabzüge kann es für einen unnötig verschwenderischen Gebrauch von Speicherplatz geben, etwa wenn die zu prüfenden Verteilungen explizit abgespeichert werden und dann dieser Speicher durchlaufen wird (anstelle über Adresskombinationen ohne Speicherung zu iterieren). Werden Datenstrukturen verwendet, mit denen besonders (zugriffs-)effizient wichtige Informationen gespeichert werden, sind hier Pluspunkte möglich (falls nicht schon bei der Laufzeit Pluspunkte dafür vergeben wurden).

(5) *Verfahren mit korrekten Ergebnissen*: Das Verfahren sollte sicher stabile Standorte finden bzw. ermitteln können, dass es keine stabilen Standorte gibt. Die Aufgabenstellung sagt nicht ganz klar, ob nur eine stabile Konstellation oder alle gefunden werden sollen; deshalb genügt es, wenn das Verfahren bei Vorhandensein stabiler Konstellationen nur eine davon ausgibt. Ansätze, die komplett falsche Ergebnisse berechnen (etwa auf Basis der Überlegung, dass stabile Verteilungen nur Adressen von Häusern enthalten sollten),

führen zu stärkerem Punktabzug. Bei Fehlern nur in besonderen Fällen (gelegentlich gibt es Probleme mit Beispiel 5) genügt ein kleinerer Abzug.

2. Theoretische Analyse

(1) *Verfahren / Qualität ingesamt gut begründet*: Neben den allgemeinen Standards ist hier besonders darauf zu achten, dass die Korrektheit des Verfahrens gut begründet wird. Besonders schade ist, wenn offensichtlich falsche Ergebnisse nicht erkannt wurden. Zumindest für Beispiel 1 ist es nämlich gut möglich, die Ergebnisse von Hand zu überprüfen.

(2) *Gute Überlegungen zur Laufzeit*: Die Laufzeit des Standard-Ansatzes ist recht gut zu charakterisieren, so dass hier auf jeden Fall eine Angabe erwartet wird.

(3) *Gute analytische Überlegungen*: Eine Verbesserung des direkten Ansatzes mit einer Laufzeit von $O(L^6)$ lässt sich insbesondere dann erreichen, wenn das Problem (in der Regel die Suche nach besseren Verteilungen) gründlich und korrekt analysiert wurde. Einige Einsendungen leisten hier sehr viel. Das kann je nach Stringenz und Nachvollziehbarkeit der Überlegungen besonders belohnt werden, selbst wenn diese bereits zu Pluspunkten bei der Laufzeit geführt haben sollten.

3. Dokumentation

(3) *Vorgegebene Beispiele dokumentiert*: Es sollten alle vorgegebenen Beispiele bearbeitet und dokumentiert worden sein. Wenn nicht alle, aber mindestens 4 der 7 Beispiele dokumentiert wurden (und darunter eines der Beispiele ohne stabile Verteilung ist), gibt es 2 Punkte, bei weniger bis zu 4 Punkte Abzug.

(5) *Ergebnisse nachvollziehbar dargestellt*: Die Angabe einer stabilen Verteilung als Zahlentripel genügt, auch wenn das Verfahren alle stabilen Verteilungen finden kann. Die Angabe konkreter Laufzeiten ist interessant, aber nicht gefordert. Pluspunkte kann es für Ausgaben geben, die das Nachvollziehen der Ergebnisse erleichtern.

Aus den Einsendungen: Perlen der Informatik

Allgemeines

Wie kann man diese Aufgabe lösen? Das war die Frage.

[Codeansatz erläutern] *– in roter Schrift. Codeansatz wurde nicht erläutert.*

Der Graph dazu verläuft ähnlich, wie bei einer asymptotischen Funktion.

Der Algorithmus lief in Lichtgeschwindigkeit

Wie ihnen vermutlich auffällt, hat Aufgabe 2 „etwas“ höhere Qualität als Aufgabe 3. Ein „geringfügiger“ Fehler in meiner Zeitplanung meinerseits ist dafür verantwortlich.

Insgesamt können wir so eine deutliche Verbesserung der Laufzeit erzielen, besonders in Kombination mit Diensten wie Amazon Web Services (AWS).

Wenn wir im Programm in die Datei main.c schauen, können wir nachvollziehen, wie es den oben beschriebenen Algorithmus durchführt.

Ich wusste eigentlich selbst nicht mehr, was genau das Programm macht und konnte darum auch Fehlermeldungen nicht mehr lösen.

Ich habe dieses Programm in C++ geschrieben, dies hatte keinen speziellen Grund, bin nur Fan von der Sprache. *– Das ist eigentlich immer der Grund – nur gibt das niemand gerne zu!*

Ich möchte mich bei der Person, die sich das hier durchlesen musste, entschuldigen. Bitte verklagen Sie mich nicht.

Aufgabe 1: Flohmarkt in Langdorf

Deshalb habe ich das Problem vereinfacht, indem ich mit Mietern Tetris gespielt habe ;).

Beispiele
Flohmarkt4.txt:
Bei der erforderlichen Eingabe wird „flohmarkt5.txt“ eingegeben [. . .]

Nebenbei bemerkt liegt dieser Wert unter dem Grundfreibetrag, was bedeutet, dass die Veranstalter eigentlich keine Steuer abführen müssen, wenn sie dieses Jahr keine weiteren Einnahmen planen. Wenn der Wert über 9400 Euro liegen würde, müsste man noch die Steuer bezahlen und die Einkommen wären geringer :-). *– Zum Glück wird die Steuer nur auf den Anteil gezahlt, der ÜBER dem Steuerfreibetrag liegt!*

[. . .] weil der EM-Algorithmus nicht auf eine perfekte, sondern auf eine optimale Verteilung setzt.

Aufgabe 2: Spießgesellen

Zuerst einmal habe ich es dieses Mal nicht objektorientiert umgesetzt, da ich davon bei Aufgabe 2 Kopfschmerzen bekommen habe.

```
return schusseln
```

```
int AnzahlFruechteZumLoeschenAusNichtEindeutigZugeordneteWunschsorten
```

Fußnote: Kleine Bemerkung: Ich habe das Wort Obst manchmal mit Früchten vertauscht.

Fußnote: Ursprünglich Obst, was aber im Laufe der Bearbeitung immer mehr zu Früchten wurde.

Der Einfachheit halber ist die Brombeere hier als C (Calciumhaltige Brombeere) dargestellt.

Meine Ausgabe gibt Donald eine sehr ausschlaggebende Aussage.

Das Besorgniserregende ist in diesem Beispiel nicht die Menge an ungenau bestimmten Schüsseln, sondern eher, dass Donald den Wunsch verspürt, sich Vogelbeeren auf den Spieß zu tun.

Aufgabe 3: Eisbudendilemma

Zuerst fallen bei einer Analyse der Aufgabenstellung verschiedene Parallelen zu neuronalen Netzen auf. *– Na ja: NN tun was, aber keiner weiß warum. So ist der Teilnehmer die Aufgabe dann auch angegangen. ;)*

Ist die Eisbude einmal um den See herum gelaufen, wird sie wieder auf null gesetzt.