40. Bundeswettbewerb Informatik 2021/2022, 1. Runde

Lösungshinweise

Bundeswettbewerb Informatik

Allgemeines

Es war sehr erfreulich, dass wieder sehr viele die Aufgaben bearbeitet und am Bundeswettbewerb Informatik teilgenommen haben. Den Veranstaltern ist bewusst, dass in der Regel viel Arbeit hinter der Erstellung einer Einsendung steckt.

Bei der Bewertung wurden die in den Einsendungen gezeigten Leistungen so gut wie möglich gewürdigt. Das ist nicht immer leicht, insbesondere wenn die Dokumentation nicht die im Aufgabenblatt genannten Anforderungen erfüllt. Bevor mögliche Lösungsideen zu den Aufgaben und Einzelheiten zur Bewertung beschrieben werden, soll deshalb im folgenden Abschnitt auf die Dokumentation näher eingegangen werden; vielleicht helfen diese Anmerkungen bei der nächsten Teilnahme.

Wie auch immer die Einsendung bewertet wurde, es sollte nicht entmutigen! Allein durch die Arbeit an den Aufgaben und ihren Lösungen hat jede Teilnehmerin und jeder Teilnehmer einiges gelernt; diesen Effekt sollte man nicht unterschätzen.

Die Bearbeitungszeit für die 1. Runde beträgt etwa drei Monate. Frühzeitig mit der Bearbeitung der Aufgaben zu beginnen ist der beste Weg, zeitliche Engpässe am Ende der Bearbeitungszeit gerade mit der Dokumentation zu vermeiden. Aufgaben sind gelegentlich schwerer zu bearbeiten, als es auf den ersten Blick erscheinen mag. Erst bei der konkreten Umsetzung einer Lösungsidee oder beim Testen von Beispielen kann man auf Besonderheiten oder Schwierigkeiten stoßen, die zusätzlicher Zeit bedürfen.

Noch etwas Organisatorisches: Sollte der Name auf Urkunde oder Teilnahmebescheinigung falsch geschrieben sein, ist er auch im AMS (login.bwinf.de) falsch eingetragen. Die Teilnahmeunterlagen können gerne neu angefordert werden; dann aber bitte vorher den Eintrag im AMS korrigieren.

Dokumentation

Die Zeit für die Bewertung ist leider begrenzt, weshalb es unmöglich ist, alle eingesandten Programme gründlich zu testen. Die Grundlage der Bewertung ist deshalb die Dokumentation, die, wie im Aufgabenblatt beschrieben, für jede bearbeitete Aufgabe aus den Teilen *Lösungsidee*, *Umsetzung*, *Beispielen* und *Quellcode* bestehen soll. Leider sind die Dokumentationen bei vielen Einsendungen sehr knapp ausgefallen, was oft zu Punktabzügen führte, die das Erreichen der 2. Runde verhinderten. Grundsätzlich kann die Dokumentation einer Aufgabe als „sehr unverständlich oder nicht vollständig“ bewertet werden, wenn die schriftliche Darstellung kaum verständlich ist oder Teile wie Lösungsidee, Umsetzung oder Quellcode komplett fehlen.

Die Beschreibung der *Lösungsidee* sollte keine Bedienungsanleitung des Programms oder eine Wiederholung der Aufgabenstellung sein. Stattdessen soll erläutert werden, welches fachliche Problem hinter der Aufgabe steckt und wie dieses Problem grundsätzlich mit einem Algorithmus sowie Datenstrukturen angegangen wurde. Ein einfacher Grundsatz ist, dass Bezeichner von Programmelementen wie Variablen, Methoden etc. nicht in der Beschreibung einer Lösungsidee verwendet werden, da sie unabhängig von solchen technischen Realisierungsdetails sind. Wenn die Beschreibungen in der Dokumentation nicht auf die Lösungsidee eingehen oder bzgl. der Lösungsidee kaum nachvollziehbar sind, kann es einen Punktabzug geben, weil das „Verfahren unzureichend begründet bzw. schlecht nachvollziehbar“ ist.

Ganz besonders wichtig sind die vorgegebenen (und ggf. weitere) *Beispiele*. Wenn Beispiele und die zugehörigen Ergebnisse in der Dokumentation ganz oder teilweise fehlen, führt das zu Punktabzug. Zur Bewertung ist für jede Aufgabe vorgegeben, zu wie vielen (und teils auch zu welchen) Beispielen Programmausgaben oder Ergebnisse in der Dokumentation erwartet werden. Diese Ergebnisse sollten idealerweise korrekt sein. Punktabzug für falsche Ergebnisse gibt es aber nur, wenn nicht schon für den ursächlichen Mangel ein Punkt abgezogen wurde.

Es ist nicht ausreichend, Beispiele nur in gesonderten Dateien abzugeben, ins Programm einzubauen oder den Bewerterinnen und Bewertern sogar das Erfinden und Testen von geeigneten Beispielen zu überlassen. Beispiele sollen die Korrektheit der Lösung belegen und auch zeigen, wie das Programm mit Sonderfällen umgeht.

Auch *Quellcode*, zumindest dessen für die Lösung wichtigen Teile, gehört in die Dokumentation; Quellcode soll also nicht nur elektronisch als Code-Dateien (als Teil der Implementierung) der Einsendung beigefügt werden. Schließlich gehören zu einer Einsendung als Teil der Implementierung *Programme*, die durch die genaue Angabe der Programmiersprache und des verwendeten Interpreters bzw. Compilers möglichst problemlos lauffähig sind und hierfür bereits fertig (interpretierbar/kompiliert) vorliegen. Die Programme sollten vor der Einsendung nicht nur auf dem eigenen, sondern auch einmal auf einem fremden Rechner hinsichtlich ihrer Lauffähigkeit getestet werden.

Hilfreich ist oft, wenn die Erstellung der Dokumentation die Programmierarbeit begleitet oder ihr teilweise sogar vorangeht. Wer es nicht ausreichend schafft, seine Lösungsidee für Dritte verständlich zu formulieren, dem gelingt meist auch keine fehlerlose Implementierung, egal in welcher Programmiersprache. Daher kann es nützlich sein, von Bekannten die Dokumentation auf Verständlichkeit hin lesen zu lassen, selbst wenn sie nicht vom Fach sind.

Wer sich für die 2. Runde qualifiziert hat, beachte bitte, dass dort deutlich kritischer als in der 1. Runde bewertet wird und höhere Anforderungen an die Qualität der Dokumentationen und Programme gestellt werden. So werden in der 2. Runde unter anderem eine klar beschriebene Lösungsidee (mit geeigneten Laufzeitüberlegungen und nachvollziehbaren Begründungen), übersichtlicher Programmcode (mit verständlicher Kommentierung), genügend aussagekräftige Beispiele (zum Testen des Programms) und ggf. spannende eigene Erweiterungen der Aufgabenstellung (für zusätzliche Punkte) erwartet.

Bewertung

Bei den im Folgenden beschriebenen Lösungsideen handelt es sich um Vorschläge, aber sicher nicht um die einzigen Lösungswege, die korrekt sind. Alle Ansätze, die die gestellte Aufgabe vernünftig lösen und entsprechend dokumentiert sind, werden in der Regel akzeptiert. Einige Dinge gibt es allerdings, die – unabhängig vom gewählten Lösungsweg – auf jeden Fall erwartet

werden. Zu jeder Aufgabe werden deshalb im jeweils letzten Abschnitt die Kriterien näher erläutert, auf die bei der Bewertung dieser Aufgabe besonders geachtet wurde. Die Kriterien sind in der Bewertung, die man im AMS einsehen kann, aufgelistet und geben an, inwieweit die Bearbeitung einer Aufgabe die einzelnen Bewertungskriterien erfüllt. Außerdem gibt es aufgabenunabhängig einige Anforderungen an die Dokumentation, die oben erklärt sind.

In der 1. Runde geht die Bewertung von fünf Punkten pro Aufgabe aus, von denen bei Mängeln Punkte abgezogen werden. Da es nur Punktabzüge gibt, sind die Bewertungskriterien meist negativ formuliert. Wenn das (Negativ-)Kriterium erfüllt ist, gibt es einen Punkt oder gelegentlich auch zwei Punkte Abzug; ansonsten ist die Bearbeitung in Bezug auf dieses Kriterium in Ordnung. Wurde die Aufgabe nur unzureichend bearbeitet, wird ein gesondertes Kriterium angewandt, bei dem es bis zu fünf Punkten Abzug geben kann. Im schlechtesten Fall wird eine Aufgabenbearbeitung mit 0 Punkten bewertet.

Für die Gesamtpunktzahl sind die drei am besten bewerteten Aufgabenbearbeitungen maßgeblich. Es lassen sich also maximal 15 Punkte erreichen. Einen 1. Preis erreicht man mit 14 oder 15 Punkten, einen 2. Preis mit 12 oder 13 Punkten und einen 3. Preis mit 9 bis 11 Punkten. Mit einem 1. oder 2. Preis ist man für die 2. Runde qualifiziert. Kritische Fälle mit nur 11 Punkten sind bereits sehr gründlich und mit viel Wohlwollen geprüft. Leider ließ sich aber nicht verhindern, dass Teilnehmende nicht weitergekommen sind, die nur drei Aufgaben abgegeben haben in der Hoffnung, dass schon alle richtig sein würden; dies ist ziemlich riskant, da sich leicht Fehler einschleichen.

Auch wurde in einigen Fällen die Regelung zur Bearbeitung von Junioraufgaben als Teil einer Einsendung zum Bundeswettbewerb Informatik nicht beachtet. Hierzu ein Zitat aus dem Mantelbogen des Aufgabenblatts: „Die etwas leichteren Junioraufgaben dürfen nur von SchülerInnen vor der Qualifikationsphase des Abiturs bearbeitet werden.“ Nur unter Einhaltung dieser Bedingung können Bearbeitungen von Junioraufgaben im Bundeswettbewerb gewertet werden.

Danksagung

Alle Aufgaben wurden vom Aufgabenausschuss des Bundeswettbewerbs Informatik entwickelt: Peter Rossmanith (Vorsitz), Hanno Baehr, Jens Gallenbacher, Rainer Gemulla, Torben Hagerup, Christof Hanke, Thomas Kesselheim, Arno Pasternak, Holger Schlingloff, Melanie Schmidt sowie (als Gast) Wolfgang Pohl.

An der Erstellung der im folgenden skizzierten Lösungsideen wirkten neben dem Aufgabenausschuss vor allem folgende Personen mit: Jannik Kudla (Junioraufgabe 1), Julian Baldus (Junioraufgabe 2), Alexandru Duca (Aufgabe 1), Marian Dietz und Simon Schwarz (Aufgabe 2), Hannah Rauterberg und Manuel Gundlach (Aufgabe 3), Tim Pokart (Aufgabe 4) und Hans-Martin Bartram (Aufgabe 5). Allen Beteiligten sei für ihre Mitarbeit ganz herzlich gedankt.

Junioraufgabe 1: Zum Winde verweht

J1.1 Lösungsidee

Hinter dieser Aufgabe verbirgt sich ein geometrisches Problem. Es ist hilfreich, die Situation aus Häusern und Windrädern in einer flachen Landschaft zunächst mathematisch zu erfassen. Hierzu betrachten wir die Häuser und Windräder als Punkte in einem zweidimensionalen Koordinatensystem. Mit $H_1, \ldots, H_n$ bezeichnen wir die Punkte der Häuser, mit $W_1, \ldots, W_m$ jene der Windräder. Dabei sind die Häuser von 1 bis n und die Windräder von 1 bis m durchnummeriert.

Wir betrachten nun jedes Windrad separat und bestimmen seine maximale Höhe. Es bezeichne $\mathscr{H}(k)$ die maximale Höhe von Windrad k. Um $\mathscr{H}(k)$ zu bestimmen, reizen wir die 10H-Regel so weit es geht aus: Wir bauen Windrad k so hoch, dass der Mindestabstand $10 \cdot \mathscr{H}(k)$ zu allen Häusern gerade so eingehalten wird.

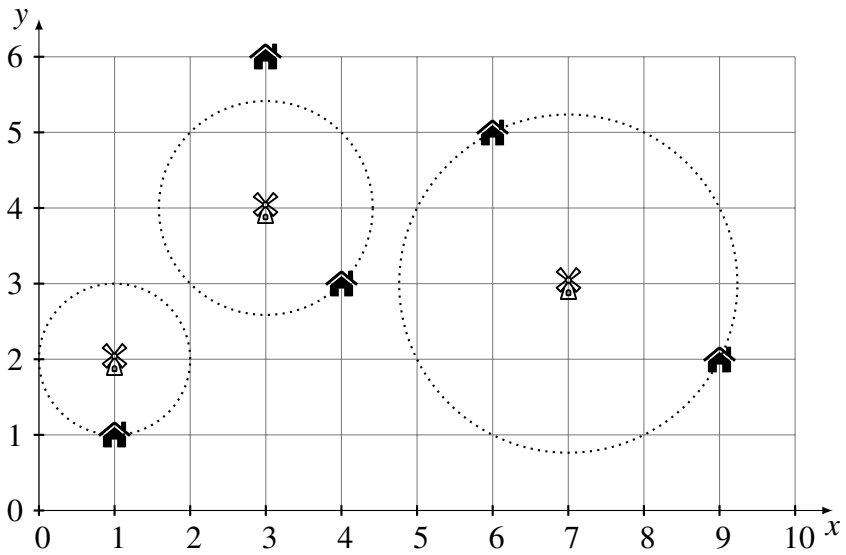


Abbildung J1.1: Beispiel mit $n = 5$ Häusern und $m = 3$ Windrädern

Geometrisch ist die Situation in Abbildung J1.1 dargestellt: Wir ziehen um jedes Windrad einen Kreis mit größtmöglichem Radius, sodass keine Häuser strikt innerhalb des Kreises liegen. Erhält ein Windrad k dann den Radius r, so muss laut 10H-Regel $r \geq 10 \cdot \mathscr{H}(k)$ sein, damit jedes Haus Abstand mindestens $r \geq 10 \cdot \mathscr{H}(k)$ zum Windrad hat. Wegen

$$r \geq 10 \cdot \mathscr{H}(k) \iff \mathscr{H}(k) \leq \frac{r}{10}$$

darf das Windrad maximal Höhe $\frac{r}{10}$ haben.

Damit können wir bereits die Aufgabe lösen: Für jedes Windrad berechnen wir den größtmöglichen Radius r, sodass kein Haus strikt innerhalb des Radius r um das Windrad liegt. Die maximale Höhe des Windrads ist dann $\frac{r}{10}$. Um r zu berechnen, bemerken wir, dass r der Abstand zum nächstgelegenen Haus ist. Wir können also alle Häuser durchgehen, jeweils den Abstand bestimmen und uns den (bisher) kleinsten Abstand merken.

Abstandsberechnung

Es bleibt eine Frage zu klären: Wie berechnen wir den Abstand zwischen einem Windrad k und einem Haus ℓ, also zwischen den Punkten W_k und H_ℓ?

Bemerkung: Diese Frage ist im Rahmen der Lösungsidee zu beantworten. Eine Formel für den Abstand darf ohne Herleitung verwendet bzw. als bekannt vorausgesetzt werden, da oft aus dem Schulunterricht bekannt. Für Interessierte beschreiben wir folgend trotzdem, wie man auf die häufig verwendete und nützliche Formel kommt.

Sind $W_k = \left(x_k^{(W)}, y_k^{(W)}\right)$ und $H_\ell = \left(x_\ell^{(H)}, y_\ell^{(H)}\right)$, so können wir den Abstand $d(W_k, H_\ell)$ von W_k zu H_ℓ nach dem Satz des Pythagoras durch

$$d(W_k, H_\ell) = \sqrt{\left(x_k^{(W)} - x_\ell^{(H)}\right)^2 + \left(y_k^{(W)} - y_\ell^{(H)}\right)^2}$$

berechnen. Die Notation mag etwas sperrig erscheinen, doch der Abstandsformel liegt eine einfache Idee zugrunde: Wir erhalten das in Abbildung J1.2 dargestellte rechtwinklige Dreieck, indem wir den direkten Weg von W_k und H_ℓ (blau) aufteilen in eine x-Komponente und eine y-Komponente (jeweils orange) – die Katheten des Dreiecks.

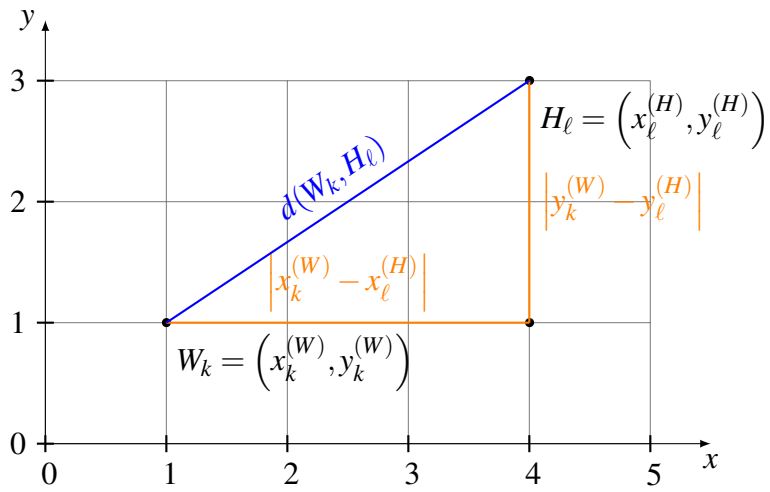


Abbildung J1.2: Abstandsberechnung mit dem Satz des Pythagoras

Die Längen der Katheten sind $\left|x_k^{(W)} - x_\ell^{(H)}\right|$ und $\left|y_k^{(W)} - y_\ell^{(H)}\right|$, wobei wir die Beträge bilden müssen, um in allen vier Situationen „W_k links/rechts von H_ℓ“ und „W_k über/unter H_ℓ“ den Abstand ohne negatives Vorzeichen zu erhalten. Da das Dreieck rechtwinklig ist, gilt nach Pythagoras

$$d(W_k, H_\ell)^2 = \left|x_k^{(W)} - x_\ell^{(H)}\right|^2 + \left|y_k^{(W)} - y_\ell^{(H)}\right|^2 = \left(x_k^{(W)} - x_\ell^{(H)}\right)^2 + \left(y_k^{(W)} - y_\ell^{(H)}\right)^2$$

und Wurzelziehen liefert die oben genannte Formel.

Algorithmus in Pseudocode

Es ist bei informatischen Problemen oft hilfreich, zwischen Lösungsidee und Programmcode eine Abstraktionsebene einzuschieben, den *Pseudocode*. Mittels Pseudocode können wir den

Algorithmus sehr präzise beschrieben, ohne uns gleich mit den Details einer Programmiersprache herumzuschlagen. Siehe Algorithmus 1 für eine Formulierung der oben beschriebenen Lösungsidee in Pseudocode.

Algorithmus 1 Bestimmung der maximalen Höhe aller Windräder

```
1: for k = 1 to m do
2:     r ← ∞                                  ▷ größtmöglicher Radius um Windrad k
3:     for ℓ = 1 to n do
4:         r ← min(r, d(W_k, H_ℓ))            ▷ d(W_k, H_ℓ) via Abstandsformel
5:     end for
6:     H(k) ← r/10
7: end for
8: return max. Höhe H(k) für alle Windräder k = 1, ..., m
```

Ausgehend von diesem Pseudocode kann der Algorithmus sehr systematisch in einer (imperativen) Programmiersprache implementiert werden. Für die Instruktion $r \leftarrow \infty$ setzt man r in der Praxis oft auf einen ausreichend hohen Wert.

Laufzeitbetrachtung

Für fehlende Laufzeitbetrachtungen gibt es bei dieser Aufgabe keinen Punktabzug. Es gehört jedoch zum Algorithmendesign dazu, die Performance auf großen Eingaben mathematisch zu untersuchen. In der 1. Runde werden solche Analysen in der Regel nicht erwartet, aber in der 2. Runde.

Der Pseudocode bietet einen idealen Ausgangspunkt für unsere Laufzeitanalyse. In Abhängigkeit von der Anzahl Häuser n und der Anzahl Windräder m gilt es zu ermitteln, wie viel Zeit die Ausführung des Programms brauchen wird.

Für die Zeilen 2, 4, 6 in Algorithmus 1 werden jeweils konstant viele Rechenoperationen benötigt. Die Laufzeit liegt mithin in $\mathcal{O}(1)$ (O-Notation[1]). Insgesamt werden die Instruktionen in Zeile 2 und Zeile 6 jeweils m-mal ausgeführt, während die Instruktion in Zeile 4 wegen der verschachelten **for**-Schleifen $m \cdot n$-mal ausgeführt wird. Die Gesamtlaufzeit liegt somit in

$$m \cdot \mathcal{O}(1) + m \cdot \mathcal{O}(1) + m \cdot n \cdot \mathcal{O}(1) = \mathcal{O}(m \cdot n),$$

ist also proportional zum Produkt $m \cdot n$.

Ein konkretes Beispiel: Es ist zu erwarten, dass unser Programm bei 300 Häusern und 20 Windkraftanlagen etwa 12-mal so viel Rechenzeit benötigt wie bei 100 Häusern und 5 Windkraftanlagen, denn $300 \cdot 20 = 6\,000$ und $100 \cdot 5 = 500$.

J1.2 Effizientere Lösung

Wir haben eine einfache $\mathcal{O}(mn)$-Lösung gesehen, die jedoch dann an ihre Grenzen stößt, wenn sowohl n als auch m beide groß sind, z. B. $n = m = 100\,000$ bei heutigen Computern. Somit

[1] Siehe `https://en.wikipedia.org/wiki/Big_O_notation`. Die O-Notation ist für Laufzeitangaben zu empfehlen. Jedoch genügt eine präzise Beschreibung des Laufzeitverhaltens in Textform in der Regel auch den BWINF-Anforderungen.

stellt sich die Frage nach einem schnelleren Algorithmus, den wir hier skizzieren wollen. Die Laufzeit liegt in $\mathcal{O}((m+n)\cdot\log(m+n))$, d. h. sie ist nur geringfügig schlechter als linear.

Die hier beschriebene Lösung enthält einige interessante, aber auch sehr fortgeschrittene Konzepte und Ansätze. *Sie geht dabei weit über die Anforderungen einer Junioraufgabe und auch einer Erstrundenaufgabe hinaus.*

Voronoi-Diagramm

Wir berechnen zunächst das Voronoi-Diagramm[2] der Häuser. Dafür existieren verschiedene Algorithmen mit Laufzeit in $\mathcal{O}(n\log n)$, u. a. Fortunes Algorithmus. Das Voronoi-Diagramm unterteilt die Ebene in n polygonale Regionen bzw. „Einzugsgebiete“, die jeweils genau die Punkte enthalten, deren nächstes Haus das Haus in der entsprechenden Region (das sogenannte *Zentrum*) ist, siehe Abbildung J1.3.

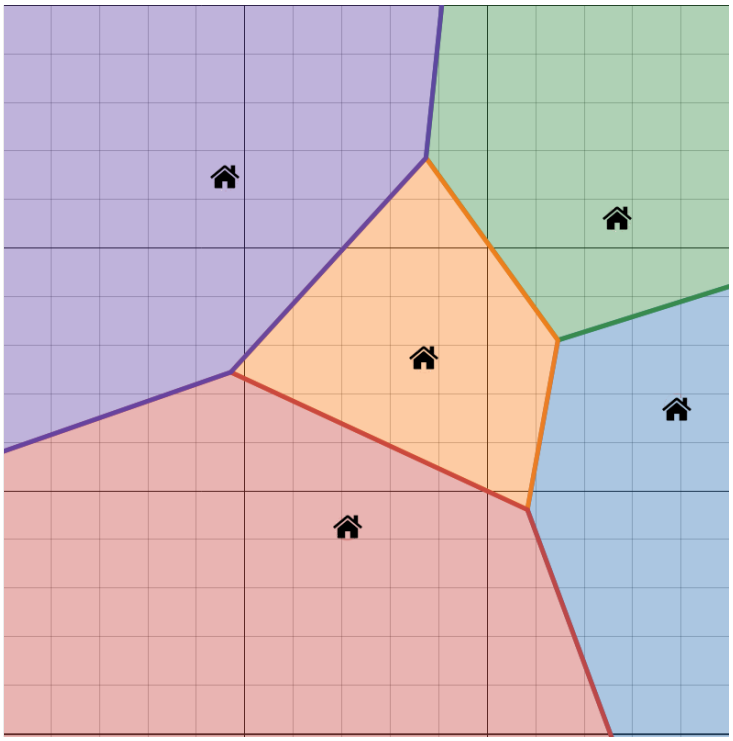


Abbildung J1.3: Beispiel für ein Voronoi-Diagramm der Häuser

Nearest-Neighbor-Abfragen

Die Frage nach dem jeweils nächstgelegenen Haus für alle Windräder ist als *Nearest-Neighbor-Problem* in der Literatur bekannt. Mithilfe des Voronoi-Diagramms können wir für jedes Windrad das nächstgelegene Haus ermitteln, indem wir die Voronoi-Region des Windrads ermitteln. Per Definition ist das Haus im Zentrum der Region das gesuchte Haus.

Wie finden wir zu einem gegebenen Punkt die Voronoi-Region? Die Idee ist ein Sweep-Line-Algorithmus, der die Ecken des Voronoi-Diagramms und die Windräder in einer bestimmten

[2] Jacques Heunis hat in diesem Blog-Post eine sehr schöne Erklärung zu Voronoi-Diagrammen und Fortunes Algorithmus veröffentlicht: `https://jacquesheunis.com/post/fortunes-algorithm/`.

Richtung durchläuft und dabei die folgenden Berechnungen ausführt. Wir stellen uns dabei eine vertikale Gerade (die *Sweep Line*) vor, die sich in horizontaler Richtung bewegt.[3]

- Ecken des Voronoi-Diagramms: Für jede Ecke gehen wir alle angrenzenden Kanten durch. Wir speichern uns die Kanten rechts von der Sweep Line in einer Datenstruktur $\mathscr{D}$ sortiert nach y-Koordinaten. Mit y-Koordinaten sind hier die y-Koordinaten der Schnittpunkte der sich gerade bewegenden Sweepline mit den Kanten in $\mathscr{D}$ gemeint. Kanten links von der Sweep Line werden entsprechend aus $\mathscr{D}$ gelöscht.
- Windräder: Da $\mathscr{D}$ sortiert ist, können wir mit einer binären Suche die zwei Kanten in $\mathscr{D}$ finden, die am nächsten über bzw. unter dem Punkt des Windrads liegen. Diese zwei Kanten grenzen an genau eine gemeinsame Voronoi-Region: Die Region, in der sich das Windrad befindet.

Umsetzung und Laufzeit: Insgesamt wird die Sweep Line $\mathscr{O}(n+m)$-mal bewegt, man nennt die „Zwischenstopps“ auch *Events*. Hier geht ein, dass das Voronoi-Diagramm maximal $2n-5$ Ecken haben kann. Initial müssen die Events nach unserer Bewegungsrichtung sortiert werden, was Zeit $\mathscr{O}((n+m)\log(n+m))$ dauert. Die Binärsuche im Falle eines Windrads dauert Zeit $\mathscr{O}(\log n)$, da auch die Anzahl der Kanten des Voronoi-Diagramms linear bzw. genauer durch $3n-6$ beschränkt ist. Schließlich kann auch das Einfügen und Löschen in bzw. aus $\mathscr{D}$ in Zeit $\mathscr{O}(\log n)$ bewerkstelligt werden, wenn man für $\mathscr{D}$ die richtige Datenstruktur wählt, z. B. einen balancierten binären Suchbaum[4]. Zusammengefasst wird man dann $\mathscr{O}(n+m)$ Events haben, die jeweils Zeit $\mathscr{O}(\log n)$ brauchen, sodass die Gesamtlaufzeit vom Sortieren der Events in Zeit $\mathscr{O}((n+m)\log(n+m))$ dominiert wird.

Bei der Implementierung wird man wahrscheinlich auf weitere Schwierigkeiten und Sonderfälle stoßen, etwa die unbegrenzten Regionen. Wir haben mit der oben erwähnten „Bewegungsrichtung“ versucht, ein wenig zu veranschaulichen, wie tückisch solche Sonderfälle sind und wie sie behandelt werden können. Diese Sonderfälle treten häufiger in der algorithmischen Geometrie auf und es gibt für deren Lösung daher immerhin etablierte Ansätze.

Bemerkung: Es ist auch möglich, die Nearest-Neighbor-Abfragen schon während der Konstruktion des Voronoi-Diagramms mit Fortunes Algorithmus zu beantworten. Hierzu sei auf die Publikation von Dinis und Mamede[5] verwiesen.

J1.3 Beispiele

In Tabelle 2 sind die Ergebnisse für alle vier BWINF-Beispiele dargestellt. Am wichtigsten ist jeweils die Angabe der maximalen Höhen $\mathscr{H}(1),\ldots,\mathscr{H}(m)$ *aller* Windräder. Die Werte sind auf zwei Nachkommastellen gerundet.

In der letzten Spalte kann jeweils die benötigte Rechenzeit des Python-Programms auf einem halbwegs modernen Linux-Laptop abgelesen werden. Der Leser sei dazu angehalten, den etwa proportionalen Zusammenhang zwischen $n \cdot m$ und der Rechenzeit zu überprüfen.

[3] Um auch vertikale Kanten des Voronoi-Diagramms korrekt bearbeiten zu können, wählen wir die Richtung ggf. (leicht) anders.

[4] Ein Kandidat wäre der AVL-Baum: `https://de.wikipedia.org/wiki/AVL-Baum` In vielen Programmiersprachen ist eine passende Datenstruktur schon integriert, beispielsweise in Python, C++ oder Java unter der Bezeichnung `set` bzw. `TreeSet`.

[5] `https://www.researchgate.net/publication/2895260_A_Sweep_Line_Algorithm_for_Nearest_Neighbour_Queries`

#	Häuser n	Windräder m	Max. Höhen $\mathscr{H}(1), \ldots, \mathscr{H}(m)$	Rechenzeit
1	12	3	48.52 m, 158.98 m, 72.41 m	< 1 ms
2	94	15	115.16 m, 201.25 m, 138.85 m, 209.12 m, 132.01 m, 186.16 m, 161.68 m, 133.30 m, 133.54 m, 128.77 m, 91.78 m, 118.28 m, 161.95 m, 142.39 m, 177.04 m	2 ms
3	2382	16	451.57 m, 393.79 m, 336.70 m, 280.74 m, 444.62 m, 385.71 m, 327.11 m, 269.02 m, 440.84 m, 381.25 m, 321.73 m, 262.32 m, 440.31 m, 380.54 m, 320.78 m, 261.02 m	34 ms
4	9993	30	0.00 m, 383.81 m, 262.45 m, 233.99 m, 296.19 m, 71.76 m, 181.41 m, 235.40 m, 343.11 m, 177.90 m, 449.16 m, 408.03 m, 317.95 m, 221.29 m, 520.12 m, 394.71 m, 433.25 m, 703.83 m, 168.42 m, 201.27 m, 139.16 m, 348.99 m, 297.91 m, 110.23 m, 813.60 m, 236.19 m, 391.94 m, 125.78 m, 241.01 m, 625.40 m	246 ms

Tabelle 1: Ausgaben unseres Programms für die BWINF-Beispieleingaben

#	Häuser n	Windräder m	Max. Höhen $\mathscr{H}(1), \ldots, \mathscr{H}(m)$
4 (alt)	114993	30	0.0 m, 179.29 m, 107.89 m, 151.18 m, 115.28 m, 71.76 m, 42.22 m, 58.58 m, 67.58 m, 112.13 m, 63.16 m, 251.61 m, 155.14 m, 118.78 m, 336.01 m, 109.77 m, 260.79 m, 350.22 m, 135.99 m, 68.08 m, 139.16 m, 50.91 m, 72.25 m, 110.23 m, 532.78 m, 94.06 m, 209.56 m, 28.15 m, 76.85 m, 425.83 m

Tabelle 2: Ausgaben unseres Programms für die alte Version der 4. BWINF-Beispieleingabe

Bei dieser Aufgabe bietet sich eine Visualisierung im Stile von Abbildung J1.1 an. Daran lässt sich die Korrektheit der berechneten Radien geometrisch prüfen. Siehe Abbildung J1.4 bis J1.7 für die Visualisierungen zu `landkreis1.txt` bis `landkreis4.txt`. Man beachte, dass in Abbildung J1.6 und J1.7 die Häuser durch orange Punkte dargestellt sind – mehrere Tausend kleine Icons zu rendern ist eben auch nicht immer geschenkt...

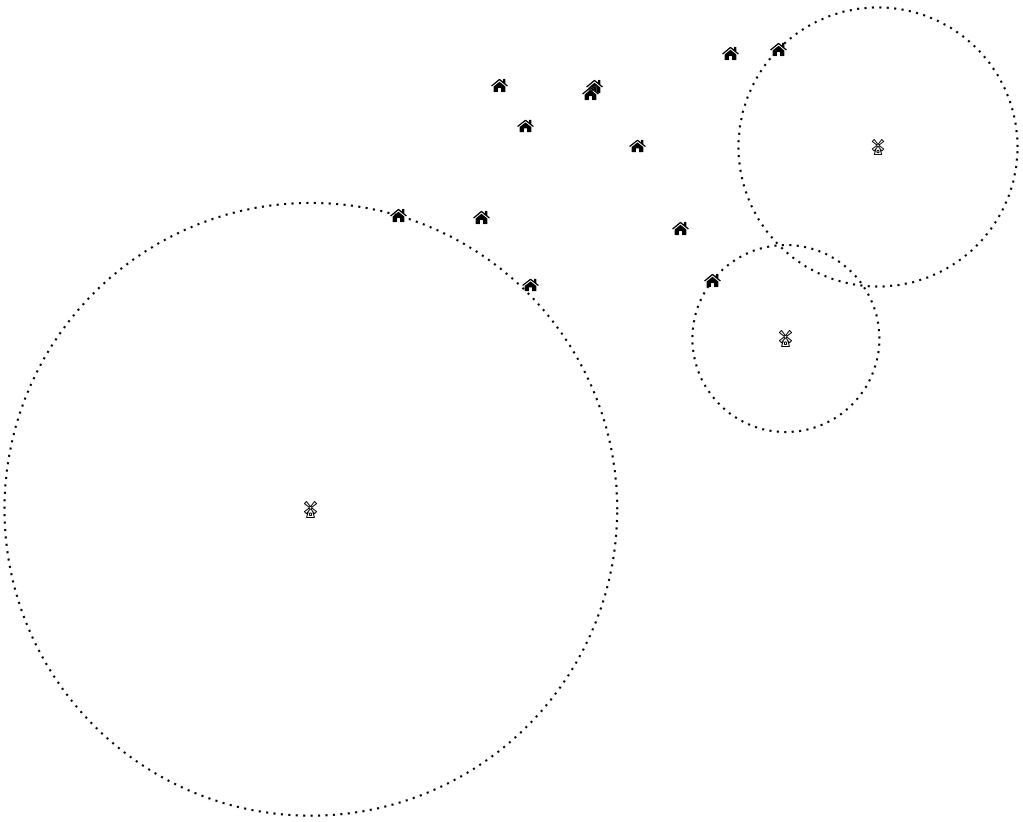


Abbildung J1.4: Visualisierung der Ergebnisse für Landkreis 1

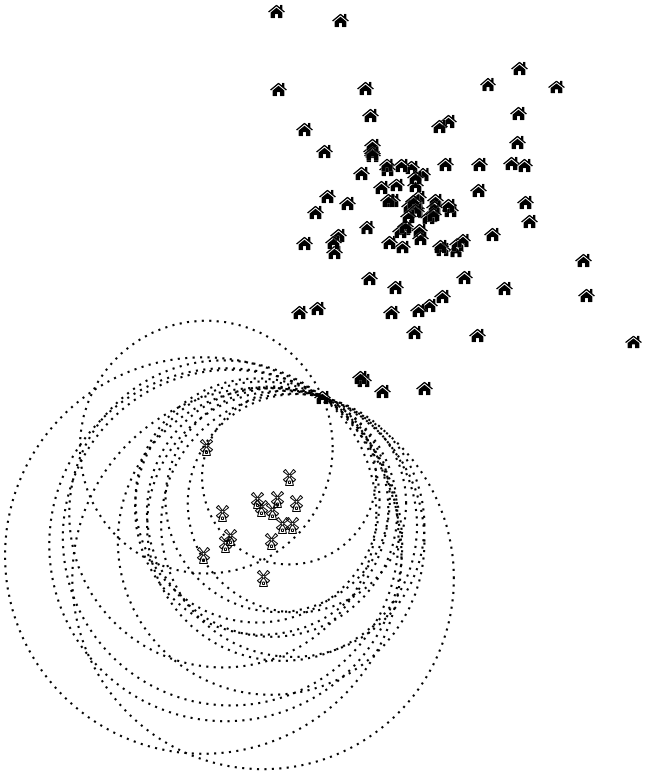


Abbildung J1.5: Visualisierung der Ergebnisse für Landkreis 2

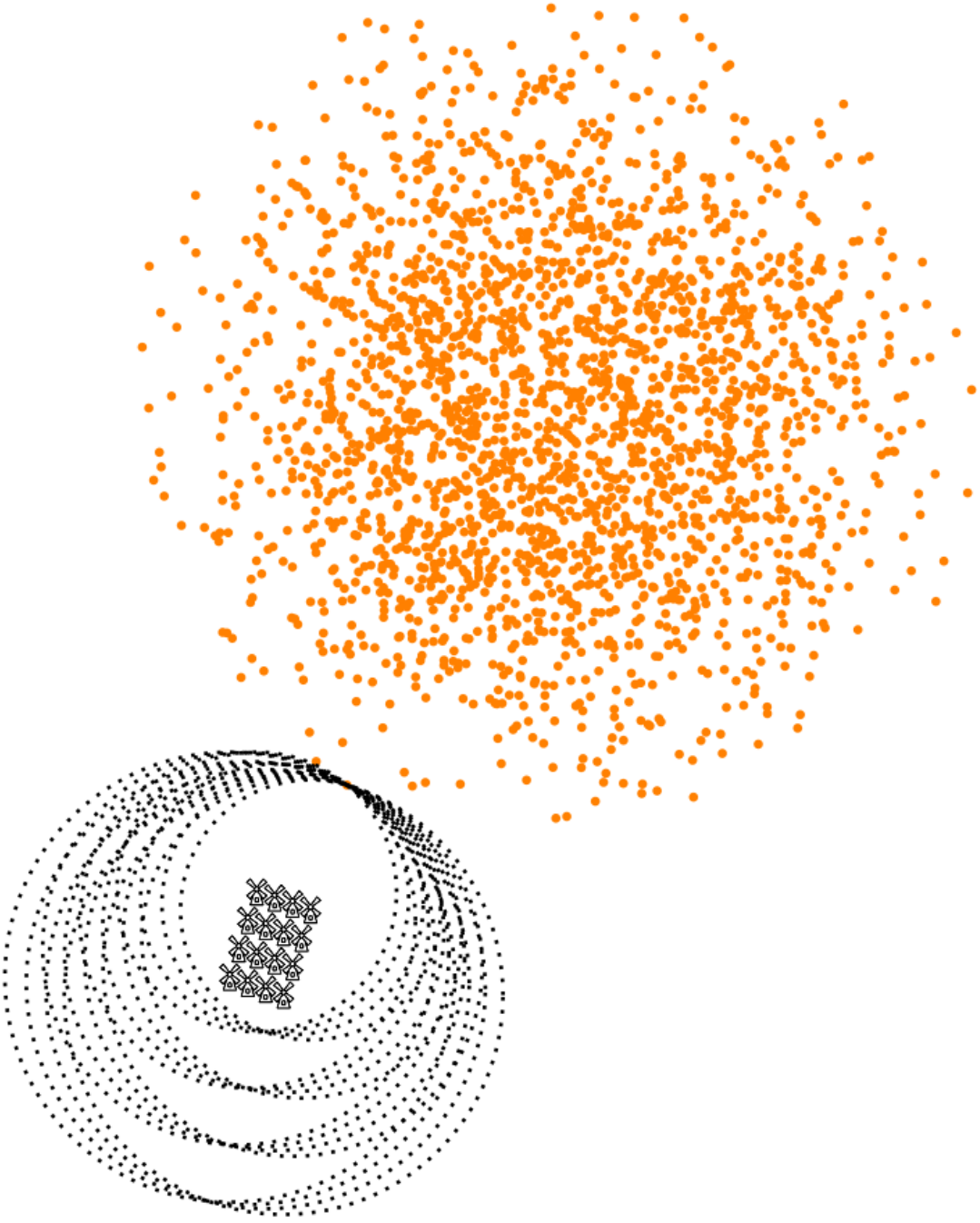


Abbildung J1.6: Visualisierung der Ergebnisse für Landkreis 3

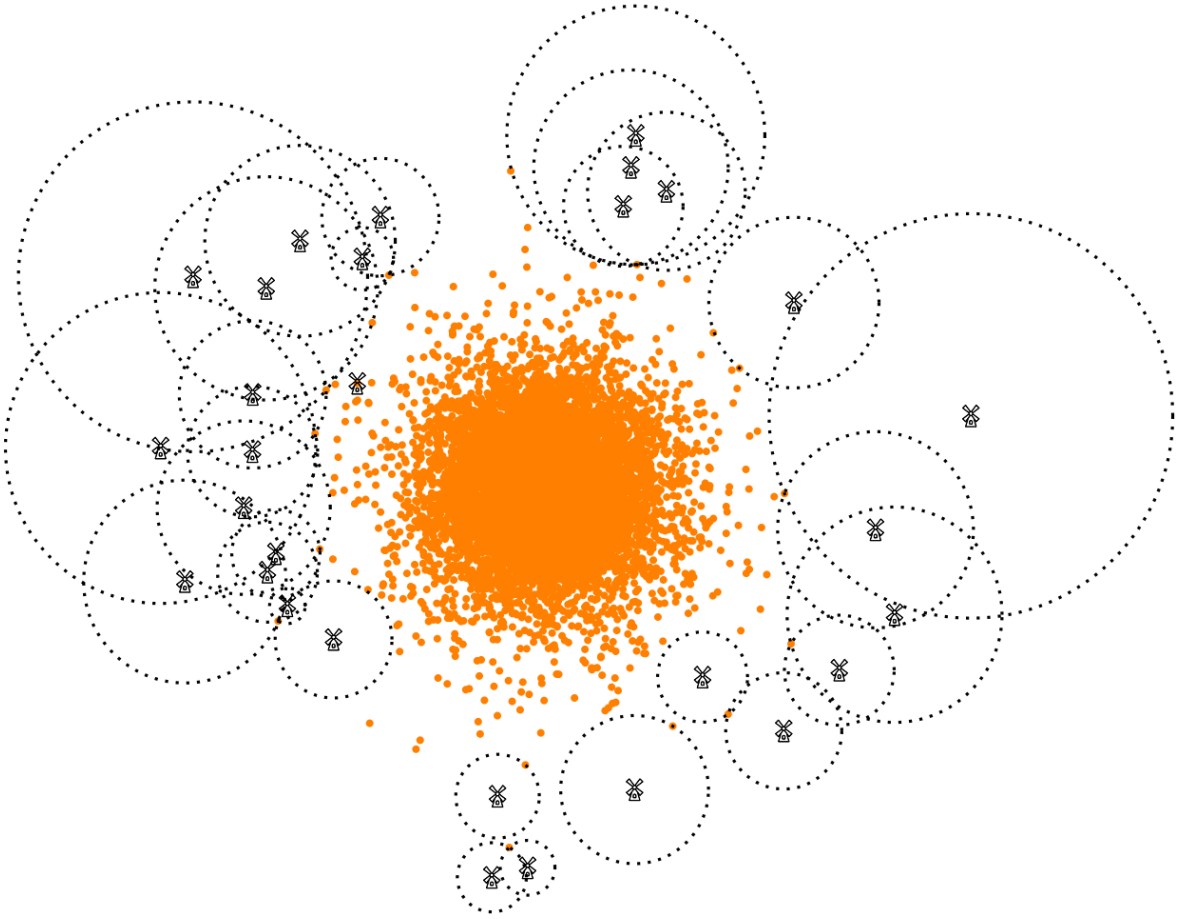


Abbildung J1.7: Visualisierung der Ergebnisse für Landkreis 4

J1.4 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- $[-1]$ **Lösungsverfahren fehlerhaft**
  Das Lösungsverfahren muss die maximalen Höhen für jedes Windrad korrekt berechnen. Den Radius mit 10 zu multiplizieren statt zu dividieren, Abstände falsch zu berechnen usw. führen sämtlich zu Punktabzug.
- $[-1]$ **Abstands- und Distanzberechnung nicht erklärt**
  Es ist zu erklären, wie die maximalen Höhen berechnet werden. Eine Lösung für die Abstandsberechnung ist zumindest zu erwähnen, gleichwohl eine Formel ohne weitere Erklärungen verwendet werden darf.
- $[-1]$ **Ergebnisse schlecht nachvollziehbar**
  Für jedes Beispiel sind die maximalen Höhen aller Windräder abzudrucken, sodass eine Zuordnung anhand der Nummerierung schnell möglich ist. Eine Angabe der Koordinaten ist nicht erforderlich. Werden die maximalen Höhen zu Koordinaten statt Nummern angegeben, ist das akzeptabel.
- $[-1]$ **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**
  Die Dokumentation soll Ergebnisse zu mindestens 3 der vorgegebenen Beispiele (`landkreis1.txt` bis `landkreis4.txt`) enthalten. Dabei wird akzeptiert, wenn die Höhen auf ganze Zahlen gerundet werden.

Junioraufgabe 2: Treffsicherheit

J2.1 Lösungsidee

Betrachten wir zunächst einen konkreten Termin. Um herauszufinden, ob dieser Termin allseits beliebt ist, iterieren wir über alle Mitglieder der Gruppe. Für jedes Mitglied der Gruppe überprüfen wir, ob es einen anderen Termin gibt, den diese Person als besser bewertet hat. Wenn das der Fall ist, bezeichnen wir das als Konflikt. Der Termin ist dann nicht allseits beliebt.

Um einen Konflikt zu lösen, könnten wir entweder die Bewertung des gewählten Termins verbessern oder die Bewertung der anderen Termine verschlechtern. Wir müssen also für jede Person, die einen Konflikt auslöst, mindestens eine Änderung vornehmen. Eine Änderung pro Konflikt ist auch ausreichend, wenn wir immer die Bewertung des gewählten Termins auf grün setzen. Die Anzahl notwendiger Änderungen, um einen Termin allseits beliebt zu machen, ist also die Anzahl der Konflikte.

Um den besten Termin und die minimale Anzahl Änderungen zu finden, können wir also für alle Termine die Anzahl Konflikte berechnen und den Termin mit den wenigsten wählen.

J2.2 Optimierung: Vorberechnung

Die Suche nach Konflikten für einen gewählten Termin kann so umgesetzt werden, dass für alle Personen alle Bewertungen betrachtet werden, um zu prüfen, ob eine davon besser als die Bewertung des gewählten Termins ist. Dies lässt sich jedoch optimieren. Es ist nämlich völlig ausreichend, zu wissen, was die beste Bewertung ist, die eine Person vergeben hat. Wenn die Bewertung für den gewählten Termin nicht dieser besten Bewertung entspricht, gibt es einen Konflikt.

Die beste Bewertung für jede Person können wir zu Beginn vorberechnen und in einem separaten Array speichern. Dieses nutzen wir dann für die folgende Suche nach Konflikten.

J2.3 Laufzeit

Zur Umsetzung der Lösungsidee müssen wir für alle m Termine über alle n Personen iterieren, und für diese Person jeweils vergleichen, ob die Person einen der $m-1$ übrigen Terminvorschläge in der Präferenztabelle besser bewertet hat als den aktuell betrachteten. Bei der unoptimierten Variante iterieren wir dafür für jeden Termin und jede Person einmal über alle anderen Terminvorschläge, machen also ca. $m \cdot n \cdot (m-1)$ Schritte, was einer Laufzeit von $O(m^2 n)$ entspricht.

Bei der optimierten Version muss für jeden Termin und jede Person nur überprüft werden, ob die Bewertung der gespeicherten besten Bewertung dieser Person entspricht. Es werden also nur $m \cdot n$ Schritte benötigt. Die Vorberechnung können wir durch einmaliges Iterieren über die gesamte Präferenztabelle umsetzen. Dies braucht auch nur $m \cdot n$ Schritte. Damit liegt die Laufzeit insgesamt nur noch bei $O(mn)$.

In den gegebenen Beispieldateien ist $n \leq 50$ und $m \leq 80$. Damit lässt sich auch die unoptimierte Variante locker in unter einer Sekunde auf allen Testfällen ausführen. Für die optimierte Variante wäre das auch z. B. noch für $n = 1000$ und $m = 1000$ möglich.

J2.4 Lösung der Beispieltestfälle

Die gesuchten Werte sind in der folgenden Tabelle angegeben. Bei den Testfällen `praeferenzen1.txt` und `praeferenzen5.txt` gibt es je zwei verschiedene Termine, die gleich gut sind.

Testfall	minimale Anzahl Änderungen	beste Termine
`praeferenzen0.txt`	2	6
`praeferenzen1.txt`	1	2 oder 3
`praeferenzen2.txt`	0	4
`praeferenzen3.txt`	7	18
`praeferenzen4.txt`	14	22
`praeferenzen5.txt`	34	31 oder 56

Im Folgenden sind die kleinsten 4 Testfälle nochmal im Detail dargestellt. Dabei ist ein bestmöglicher Termin grau unterlegt. Die grün unterlegten Tabelleneinträge müssen geändert werden, damit der Termin allseits beliebt wird.

`praeferenzen0.txt`

?			?	?		
X	X	X	?	X	X	X
X	?	?	?	X	?	X
	?	X	X	?		
?	X	?	X		?	?

`praeferenzen1.txt`

?	?	X	X	?
X	?	?	?	?
X	X		?	?
X			?	?

`praeferenzen2.txt`

	?								
X	X	X		?	?	X			?
?	?			?	?	?			?
X	X	X	?	X	?	?	X	?	?
?	?	?	?	X	X	?	?	X	?
?	?	X		?	?	?	?	?	X
X	?	X		X	X	X	X		X
?	X	?		X	?	X	?	X	X

`praeferenzen3.txt`

		?	?		X	?	?	X	?	X	X	?	X		X	?	X	X	?
X	X	?		?	?	?		X	?		?	?	?	X			X	X	?
?	?		?	X	X	X	X	?	?	?	X	?	?	?	?	?		?	X
X	X	?	?	X			X			?	X	X	?	X	?		?	?	X
?	X	?	?	X	?	X	X	?	X	?	X	X	?	?	?	?		X	
	X	X	?	?	?	?	?		?	?	?	?	X	X	?	?	X	?	X
?	?	X	X	?	?	?	X		?		?	?	?	X	?	?			?
		X	?	X		?	X	X	?	?	X	?	?	X		?	X		?
?	?	?	?		?	?	?	?	?	?	?	?	X	?	?	?	X	X	X
X	?	?	?	X	?	?	?		X	?	X		?		?			X	
X	?	?	X	?	?		?	X	?	?	?	X	?	?		?		X	
	?		?	X	?	?	X	X	?			?	?	?	?	?		?	?
	?	X	?	?	?	?	X	?	X	?	X	X	?	?	?	X	?		X

J2.5 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- $[-1]$ **Lösungsverfahren fehlerhaft**
  Vorgesehen ist, dass bei jeder Änderung ein beliebiger Tabelleneintrag auf eine beliebige Bewertung gesetzt werden kann, also eine Änderung von „rot“ auf „grün“ nur einer Änderung entspricht. Wenn abweichende Annahmen getroffen werden, die die möglichen Änderungen einschränken, müssen diese zumindest in der Dokumentation explizit erwähnt werden.
- $[-1]$ **Allseits beliebter Termin fehlerhaft umgesetzt**
  Ein allseits beliebter Termin sollte ein Termin sein, sodass für keine Person ein Termin existiert, welcher eine höhere Präferenz hätte. Dieser ist nicht notwendigerweise bei allen Personen „grün“ oder hat die wenigsten „roten“ Einträge. Davon auszugehen, dass der Termin mit der geringsten Summe zwangsläufig auch der beste Termin wird, ist nicht korrekt. Ist das Lösungsverfahren nur in diesem Punkt fehlerhaft, wird nur hier abgezogen und nicht zusätzlich bei Kriterium 4.
- $[-1]$ **Ergebnisse schlecht nachvollziehbar**
  Zu einer vollständigen Ausgabe gehört mindestens die minimal nötige Anzahl Änderungen sowie die Angabe eines finalen Termins, der mit dieser Anzahl Änderungen allseits beliebt wird. Die Angabe der konkreten Änderungen ist nicht nötig. Wenn es gleichwertige „allseits beliebte Termine“ gibt, dann reicht es, wenn einer angegeben wird.
- $[-1]$
  Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele Die Dokumentation soll Ergebnisse zu mindestens 3 der vorgegebenen Beispiele (`praeferenzen1.txt` bis `praeferenzen5.txt`) enthalten.

Aufgabe 1: Schiebeparkplatz

1.1 Lösungsidee

Sei Q ein querstehendes Auto, das ein parkendes Auto P blockiert. Die für Q erstbeste Position weiter links (bzw. rechts), welche P entsperrt, wird linke (bzw. rechte) Schlüsselposition von Q genannt. Für diese gilt genau einer der folgenden drei Fälle:

1. Die Schlüsselposition ist frei.
2. Die Schlüsselposition liegt außerhalb des Parkplatzes und ist deswegen nicht erreichbar.
3. Die Schlüsselposition ist durch ein weiteres querstehendes Auto blockiert. Die für das weitere Auto erstbeste Position weiter links (bzw. rechts), welche die Schlüsselposition entsperrt, ist die Schlüsselposition des weiteren Autos, für die wieder genau einer dieser drei Fälle gilt.

Im Allgemeinen wiederholt sich der dritte Fall in einer Kaskade, die in den ersten oder zweiten Fall mündet. Es ergeben sich die folgenden zwei Möglichkeiten:

a. Die Kaskade mündet in Fall 1: Beginnend mit dem letzten Auto, dessen Schlüsselposition frei ist, nehmen alle in der Kaskade involvierten Autos der Reihe nach ihre Schlüsselposition ein. Aus einer Protokollierung dieser Züge geht die kleinste Anzahl an Zügen hervor, die benötigt wird, um Q auf die linke (bzw. rechte) Schlüsselposition zu bringen.

b. Die Kaskade mündet in Fall 2: Für Q ist das Erreichen seiner linken (bzw. rechten) Schlüsselposition nicht möglich.

Nach diesem Verfahren lässt sich bestimmen, ob und in wie vielen Zügen die linke (bzw. rechte) Schlüsselposition für Q erreichbar ist. Um P zu entsperren, wird eine mit den wenigsten Zügen erreichbare Schlüsselposition von Q gewählt und die Zugfolge zum Erreichen dieser ausgegeben.

1.2 Beispiele

Da primär die Anzahl der Züge das Optimalitätskriterium ist (also, dass möglichst wenig Autos bewegt werden müssen), existieren im Allgemeinen mehrere optimale Lösungen. Bei den folgenden Programmausgaben handelt es sich jeweils um eine optimale Lösung, bei der die querstehenden Autos eine minimale Distanz zurücklegen.

parkplatz0.txt

```
A B C D E F G
_ _ H H _ I I

A:
B:
C: H 1 rechts
D: H 1 links
E:
F: H 1 links, I 2 links
G: I 1 links
```

parkplatz1.txt

```
A B C D E F G H I J K L M N
_ O O P P _ Q Q _ _ R R _ _

A:
B: P 1 rechts, O 1 rechts
C: O 1 links
D: P 1 rechts
E: O 1 links, P 1 links
F:
G: Q 1 rechts
H: Q 1 links
I:
J:
K: R 1 rechts
L: R 1 links
M:
N:
```

parkplatz2.txt

```
A B C D E F G H I J K L M N
_ _ O O _ P P Q Q R R _ S S

A:
B:
C: O 1 rechts
D: O 1 links
E:
F: O 1 links, P 2 links
G: P 1 links
H: R 1 rechts, Q 1 rechts
I: P 1 links, Q 1 links
J: R 1 rechts
K: P 1 links, Q 1 links, R 1 links
L:
M: P 1 links, Q 1 links, R 1 links,
   S 2 links
N: S 1 links
```

parkplatz3.txt

```
A B C D E F G H I J K L M N
_ O O _ P P _ _ Q Q R R S S

A:
B: O 1 rechts
C: O 1 links
D:
E: P 1 rechts
F: P 1 links
G:
H:
I: Q 2 links
J: Q 1 links
K: Q 2 links, R 2 links
L: Q 1 links, R 1 links
M: Q 2 links, R 2 links, S 2 links
N: Q 1 links, R 1 links, S 1 links
```

parkplatz4.txt

```
A B C D E F G H I J K L M N O P
Q Q R R _ _ S S _ _ T T _ U U _

A: R 1 rechts, Q 1 rechts
B: R 2 rechts, Q 2 rechts
C: R 1 rechts
D: R 2 rechts
E:
F:
G: S 1 rechts
H: S 1 links
I:
J:
K: T 1 rechts
L: T 1 links
M:
N: U 1 rechts
O: U 1 links
P:
```

parkplatz5.txt

```
A B C D E F G H I J K L M N O
_ _ P P Q Q _ _ R R _ _ S S _

A:
B:
C: P 2 links
D: P 1 links
E: Q 1 rechts
F: Q 2 rechts
G:
H:
I: R 1 rechts
J: R 1 links
K:
L:
M: S 1 rechts
N: S 1 links
O:
```

1.3 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- [−1] **Modellierung fehlerhaft**
  Die Ausgangssituation des Parkplatzes muss korrekt eingelesen und umgesetzt werden. Die Grenzen des Parkplatzes müssen beachtet werden. So darf z. B. ein Auto, welches bereits ganz links geparkt ist, nicht nach links verschoben werden. Autos können nur in vorgesehene Richtungen bewegt werden. So sollten querstehende Autos nur nach links oder rechts verschoben werden.
- [−1] **Lösungsverfahren fehlerhaft**
  Das Verfahren muss eine in Bezug auf die Anzahl der zu verschiebenden Autos optimale Lösung bestimmen. Können zum Beispiel (siehe `parkplatz5.txt`, Auto `F`) 2 Autos jeweils um einen Schritt nach links verschoben werden oder ein Auto um 2 Schritte nach rechts, so ist Letzteres zu wählen.
- [−1] **Verfahren bzw. Implementierung unnötig aufwendig / ineffizient**
  Bei der Berechnung der nötigen Verschiebungen sollten nicht mehr Autos betrachtet werden als nötig.
- [−1] **Ergebnisse schlecht nachvollziehbar**
  Die berechneten Verschiebungen müssen verständlich aufgeschrieben sein. Dabei dürfen die Verschiebungen je Auto in beliebiger Reihenfolge angegeben werden. Es ist schön, wenn die Ausgangssituation des Parkplatzes jeweils veranschaulicht ist; das wird aber nicht erwartet.
- [−1] **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**
  Die Dokumentation soll Ergebnisse zu mindestens 3 der vorgegebenen Beispiele (`parkplatz1.txt` bis `parkplatz5.txt`) enthalten.

Aufgabe 2: Vollgeladen

2.1 Lösungsidee

Ein einfacheres Problem

Um Lara und Paul bei der Auswahl der Hotels zu helfen, versuchen wir zuerst, das Problem in ein leichteres zu überführen. Dazu betrachten wir das folgende Problem:

„Ist es möglich, dass Familie Meier ankommt, wenn alle Hotels – unabhängig von der Bewertung – an der Route benutzt werden dürfen?“

Damit Familie Meier ankommen kann, muss sie eine Auswahl an Hotels treffen. Wir bezeichnen die Hotels, in denen Familie Meier übernachten möchte, als $H_1, \ldots, H_i$. Dabei steht H_1 für das erste Hotel an der Route und H_i für das letzte. Laut Aufgabenstellung darf die Reise maximal 5 Tage dauern. Daher dürfen maximal 4 Hotels zur Übernachtung ausgewählt werden. Natürlich muss Familie Meier aber nicht 4 Nächte in Hotels verbringen, sofern die Reise bereits früher beendet ist. Formal bedeutet dies, dass $i \leq 4$ gelten muss.

Ein Sonderfall tritt auf, wenn unsere Reise weniger als 6h dauert und daher ohne Übernachtung gefahren werden kann. Diesen Fall sollte unser Programm gesondert überprüfen und eine informative Meldung ausgeben. In der folgenden Lösung werden wir diesen Fall daher nicht weiter betrachten.

Damit die Auswahl $H_1, \ldots, H_i$ der Hotels gültig ist, muss sie die folgenden Bedingungen erfüllen:

- Es wurden maximal 4 Hotels ausgewählt, also $i \leq 4$.
- Das Hotel H_1 ist vom Start „erreichbar“, also maximal 6h vom Start entfernt.
- Jedes weitere Hotel H_j mit $j \geq 2$ ist vom vorherigen Hotel H_{j-1} in max. 6h erreichbar.
- Vom letzten Hotel H_i ist das Ziel der Reise in höchstens 6h erreichbar.

Um also festzustellen, ob Familie Meier ihr Ziel erreichen kann, müssen wir überprüfen, ob eine solche gültige Auswahl an Hotels aus unserer Liste existiert. Dazu benutzen wir die folgende Beobachtung: „Es ist immer optimal, das am weitesten entfernte noch erreichbare Hotel als nächstes auszuwählen“. Diese Beobachtung ergibt sich intuitiv daraus, dass es nie falsch sein kann, noch mögliche Strecke am aktuellen Tag zurückzulegen, anstatt sie auf den nächsten Tag aufzuschieben – schließlich kann dann am nächsten Tag eventuell noch weiter gefahren werden[6].

Mit dieser Beobachtung können wir also einen Algorithmus entwickeln, der prüft, ob in einer Liste L an Hotels eine gültige Auswahl existiert. Dazu nehmen wir vorerst an, dass die Liste L aufsteigend sortiert nach Position der Hotels ist – d.h. das erste Hotel an der Route steht auch an erster Stelle der Liste. Aus dieser Liste wählen wir nun das am weitesten noch vom Start erreichbare Hotel aus (dieses wird H_1). Im nächsten Schritt wählen wird dann das am weitesten entfernte, noch von H_1 erreichbare Hotel (dieses wird H_2). Diesen Prozess wiederholen wir, bis wir das Ziel erreicht haben. In diesem Fall gibt unser Algorithmus *true* zurück.

Natürlich kann es aber auch passieren, dass wir das Ziel nicht erreichen können. Zum Beispiel ist es möglich, dass kein Hotel von unserem aktuellen Punkt mehr erreicht werden kann, oder

[6] Angelehnt an das bekannte Sprichwort „Was du heute kannst besorgen, das verschiebe nicht auf morgen.“

dass Familie Meier das Ziel nicht mit 4 Übernachtungen erreichen kann. In einem solchen Fall gibt unser Algorithmus den Wert $false$ zurück.

Im folgenden Pseudocode nehmen wir an, dass die Variable n die Anzahl der Hotels enthält, und die Position unseres Ziels in der Variable *Ziel* gespeichert ist. Unser Algorithmus sieht dann wie folgt aus, wobei j den Index des am weitesten entfernten, aber immer noch erreichbaren Hotels darstellt:

Algorithmus 2 Lösung des einfacheren Problems

```
1:  procedure ZielErreichbar(L)
2:      letzte_position ← 0
3:      j ← 1
4:      if hotel_position(L[1]) > 6h then
5:          return false                    ▷ Das erste Hotel ist nicht vom Start erreichbar
6:      end if
7:      for i ← 1,...,4 do
8:          while j < n and hotel_position(L[j+1]) − letzte_position ≤ 6h do
9:              j ← j+1
10:         end while
11:         H_i ← L[j]                      ▷ H_i ist das am weitesten entfernte erreichbare Hotel.
12:         letzte_position ← hotel_position(H_i)
13:         if Ziel − letzte_position ≤ 6h then
14:             return true                 ▷ Das Ziel ist vom aktuellen Hotel erreichbar.
15:         end if
16:     end for
17:     return false                        ▷ Das Ziel ist nicht mit 4 Übernachtungen erreichbar.
18: end procedure
```

Für eine Laufzeitanalyse beobachten wir, dass in der *innersten* while-Schleife j immer um eins erhöht wird. Da die Schleife aber sofort beendet wird, sobald $j > n$, kann diese Schleife höchstens n-mal ausgeführt werden. Da die äußere Schleife nur konstant oft durchlaufen wird, ergibt sich damit insgesamt eine Laufzeit von $\mathcal{O}(n)$.

Filtern der Hotels Nun, da wir eine Lösung für das obige Problem gefunden haben, wollen wir nun auch die Bewertungen der Hotels in unser Programm einfließen lassen. Dazu stellen wir uns die Frage „Ist es möglich, dass Familie Meier ankommt, wenn alle Hotels mit mindestens Bewertung X an der Route benutzt werden dürfen?“ Eine Bewertung X, für welche Familie Meier ihr Ziel erreichen kann, nennen wir im folgenden auch eine *gültige* Bewertung.

Um dieses Problem zu lösen, können wir unseren obigen Algorithmus benutzen. Jedoch schränken wir nun die Liste L an verfügbaren Hotels soweit ein, dass nur noch Hotels mit einer Bewertung besser oder gleich X auf der Liste auftauchen. Wenn es nun möglich ist, mit den Hotels aus L das Ziel zu erreichen, ist X also eine gültige Bewertung. Damit wissen wir, dass die Bewertung aller ausgewählten Hotels mindestens X sein muss. Insbesondere bedeutet das auch, dass die schlechteste Bewertung eines ausgewählten Hotels mindestens X ist.

Finden der optimalen Bewertung

Lineare Suche Nun sind wir bereit, unser originales Problem zu lösen. Dazu müssen wir die beste gültige Bewertung X finden. Um diese zu finden, können wir einfach absteigend über alle möglichen Bewertungen iterieren und überprüfen, ob diese gültig ist. Die erste gültige Bewertung, welche wir in unserer Schleife finden, ist also die höchste und damit unsere gesuchte Lösung.

Insgesamt iteriert unser Algorithmus also im schlimmsten Fall einmal über alle möglichen Bewertungen. Da es nicht mehr mögliche Bewertungen als Hotels geben kann, kann es also höchstens n (wobei n die Anzahl an Hotels ist) Iterationen geben. In jeder Iteration wird überprüft, ob die Bewertung gültig ist. Dazu wird in jeder Iteration die oben beschriebene Methode benutzt, welche $\mathcal{O}(n)$ Zeit benötigt. Insgesamt ergibt sich also eine Laufzeit von $\mathcal{O}(n^2)$.

In allen BWINF-Beispielen tauchen jedoch nur die Bewertungen $0,0-5,0$ auf. Da unter dieser Annahme nur 51 verschiedene Bewertungen möglich sind, ist unsere Laufzeit (bei korrekter Behandlung von doppelten Bewertungen) in der Praxis auf den BWINF-Beispielen etwas schneller als die theoretische Analyse ergeben würde.

Binäre Suche Die Laufzeit zum Finden des optimalen Hotels lässt sich mittels *binärer Suche* noch weiter verbessern. Dazu verwenden wir die folgenden beiden Beobachtungen[7]:

- Wenn die Bewertung X gültig ist, dann ist eine niedrigere Bewertung als X ebenfalls gültig.
- Wenn die Bewertung X nicht gültig ist, dann kann eine höhere Bewertung als X ebenfalls nicht gültig sein.

Wenn wir nun also eine Liste an sortierten, möglichen Bewertungen haben, können wir für den *Median aller Bewertungen* prüfen, ob dieser gültig ist. Dort können die folgenden beiden Fälle auftreten:

1. Der Median ist eine gültige Bewertung: Dann müssen wir für alle niedrigeren Bewertungen nicht mehr prüfen, ob diese gültig sind – da diese garantiert gültig sind.
2. Der Median ist keine gültige Bewertung: Dann müssen wir für alle höheren Bewertungen die Gültigkeit nicht überprüfen, da diese garantiert ungültig sind.

Da für den Median an Bewertungen jeweils die Hälfte aller Bewertungen niedriger oder höher ist, können wir in diesem Schritt die Hälfte aller möglichen Bewertungen ausschließen. Diesen Prozess können wir iterativ fortsetzen, bis nur noch eine Bewertung übrig bleibt. Da wir nach diesem Prozess die Gültigkeit aller Bewertungen kennen, können wir sehr leicht die optimale Bewertung herausfinden.

Unser Verfahren ist sehr effizient, da in jedem Schritt die Hälfte aller restlichen Bewertungen als Lösung ausgeschlossen wird. Damit kann es maximal $\log n$ viele Schritte geben. Weiterhin wird in jedem Schritt überprüft, ob eine Bewertung gültig ist, was $\mathcal{O}(n)$ dauert. Insgesamt ergibt sich somit eine Laufzeit von $\mathcal{O}(n \cdot \log n)$.

[7] Mathematik-Profis werden bemerken, dass beide Beobachtungen äquivalent sind.

2.2 Alternative Lösungsideen

Naives Probieren

Um die Hotels $H_1, \ldots, H_i$ zu finden, ist es auch möglich, alle Kombinationen an Hotels durchzuprobieren. Die Überprüfung, ob eine Auswahl gültig ist, kann dann anhand der Kriterien in jeweils konstanter Zeit für feste $H_1, \ldots, H_i$ passieren. Da maximal 4 Hotels ausgewählt werden müssen, ergibt sich somit eine (sehr ineffiziente) Laufzeit von $\mathcal{O}(n^4)$. Mit effizientem Code und durch einige Optimierungen (z.B. nur aufsteigende Hotels auswählen) können aber auch mit dieser Methode für alle BWINF-Beispiele Lösungen in adäquater Zeit gefunden werden. Dieser Ansatz ist jedoch trotzdem aufgrund seiner Laufzeit nicht zu empfehlen, und kann auch schon bei etwas mehr Hotels als in den vorgegebenen Beispielen keine Lösung mehr finden.

Dynamische Programmierung

Eine weitere alternative Lösung der Aufgabe ergibt sich mit Hilfe von *dynamischer Programmierung* (DP). Mit zweidimensionaler dynamischer Programmierung kann $\mathrm{dp}[i][j]$ berechnet werden. Wir definieren diesen Wert als die optimale Bewertung, die erreicht werden kann, wenn wir die Fahrt an Position i beginnen und noch j Übernachtungen nutzen können.

Unter dieser Definition kann dieser Wert (in Abhängigkeit von anderen zuvor berechneten Werten) folgendermaßen berechnet werden:

$$\mathrm{dp}[i][j] = \max_{\text{Hotel } \ell \text{ an Position } i<k\leq i+6h} \min(\mathrm{dp}[k][j-1], \mathrm{bewertung}(\ell))$$

Aufgepasst werden muss in Randfällen (z.B. $j = 0$). Das Endergebnis steht dann nach der Berechnung in $\mathrm{dp}[0][4]$. Die entsprechenden Hotelposition können mit dieser Methode ebenfalls berechnet werden, indem zusätzlich zu $\mathrm{dp}[i][j]$ auch das dort verwendete Hotel ℓ gespeichert wird.

Der Wert $\mathrm{dp}[i][j]$ wird nur benötigt, falls $i = 0$ oder falls es ein Hotel an Position i gibt. j kann nur Zahlen zwischen 0 und 4 annehmen. Somit müssen insgesamt lediglich $5n$ verschiedene DP-Werte berechnet werden. Die Berechnung eines DP-Eintrags $\mathrm{dp}[i][j]$ kann mithilfe der Formel oben in Laufzeit $\mathcal{O}(n)$ implementiert werden (indem für jedes Hotel überprüft wird, ob dessen Position zwischen i und $i+6h$ liegt). Dadurch beträgt die asymptotische Laufzeit insgesamt $\mathcal{O}(n^2)$, aber es gibt auch weitere Möglichkeiten für eine Implementierung mithilfe von DP, mit denen andere Laufzeiten erreicht werden.

2.3 Beispiele

Angewandt auf die Eingaben auf der BWINF-Website liefert das Programm folgende Hotels, die ausgewählt werden können, um eine bestmögliche Bewertung des schlechtesten Hotels zu erreichen. Angegeben ist jeweils die Position des Hotels, sowie die zugehörige Bewertung. Eine korrekte Lösung kann auch andere Hotels auswählen, solange deren minimale Bewertung dieselbe bleibt.

	minimale Bewertung	Hotel 1	Hotel 2	Hotel 3	Hotel 4
hotels1.txt	**2.7**	347 (2.7)	687 (4.4)	1007 (2.8)	1360 (2.8)
hotels2.txt	**2.3**	341 (2.3)	700 (3.0)	1053 (4.8)	1380 (5.0)
hotels3.txt	**0.3**	360 (1.3)	717 (0.3)	1076 (3.8)	1433 (1.7)
hotels4.txt	**4.6**	340 (4.6)	676 (4.6)	1032 (4.9)	1316 (4.9)
hotels5.txt	**5.0**	317 (5.0)	636 (5.0)	987 (5.0)	1286 (5.0)

2.4 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- $[-1]$ **Lösungsverfahren fehlerhaft**
  Der Fall, dass mehrere Hotels unterschiedlicher Bewertung an der gleichen Stelle stehen, muss korrekt behandelt werden. Dies bezieht sich insbesondere auf `hotels2.txt`, dort steht an Stelle 341 ein Hotel mit Bewertung 2.2 und eines mit 2.3. Bei Hotelbewertungen darf davon ausgegangen werden, dass diese im Bereich $0.0 - 5.0$ liegen.
- $[-1]$ **Bedingungen nicht korrekt eingehalten**
  Entsprechend der Aufgabenstellung dürfen nur ≤ 4 Übernachtungen genutzt werden. Dies bezieht sich insbesondere auf `hotels3.txt`, da dort mit mehr Übernachtungen eine bessere Minimalbewertung erzielt werden kann. Es wird maximal 6h am Tag gefahren.
- $[-1]$ **Verfahren bzw. Implementierung unnötig aufwendig / ineffizient**
  Auch auf den größeren Musterbeispielen sollte eine Lösung in angemessener Zeit terminieren. Auch bei „naiven“ $\mathcal{O}(n^4)$-Lösungen sollte der Output innerhalb weniger Minuten für alle Beispiele generiert werden. Ein konstanter Faktor ist nur problematisch, wenn die Laufzeit bereits in $\mathcal{O}(n^4)$ liegt. Das Verfahren sollte insbesondere keine unnötigen Dinge tun, z. B. alle möglichen Bewertungen von $0,0$ bis $5,0$ in Schritten von $0,1$ durchsuchen, anstatt nur die in der Eingabe vorkommenden Bewertungen.
- $[-1]$ **Ergebnisse schlecht nachvollziehbar**
  Die niedrigste Bewertung der ausgewählten Hotels muss klar ersichtlich sein. Die Liste aller ausgewählten Hotels wird ausgegeben.
- $[-1]$ **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**
  Die Dokumentation soll Ergebnisse zu mindestens 3 der vorgegebenen Beispiele (`hotels1.txt` bis `hotels5.txt`) enthalten.

Aufgabe 3: Wortsuche

3.1 Lösungsidee

Schwierigkeitsstufen

Bevor wir eine Wortsuche erzeugen können, benötigen wir Kriterien, welche sich eignen, die Rätsel intuitiv voneinander zu unterscheiden. In der Aufgabenstellung finden sich bereits zwei Beispiele, von denen wir einige Kriterien ableiten können:

- **Richtung:** Es ist intuitiv leichter, Wörter zu finden, welche nur in Zeilen oder nur in Spalten stehen, als Wörter, die diagonal geschrieben sind.
- **Leserichtung:** Vorwärts eingefügte Wörter sind einfacher zu finden als rückwärts geschriebene Wörter.
- **Größe:** Kleine Wortsuchen sind einfacher zu lösen als große.
- **Auffüllen:** Eine Wortsuche, welche mit komplett zufälligen Buchstaben aufgefüllt wurde, ist einfacher zu lösen, als eine Wortsuche, welche mit Fragmenten der eingefügten Wörter aufgefüllt wurde.

Auf diesen Kriterien basieren wir die Definition der Schwierigkeitsstufen. Dabei gehen wir davon aus, dass wie in den Beispielen die Größe der Wortsuche festgelegt ist:

(1) einfach

- Alle Wörter werden vorwärts eingefügt.
- Die Richtungen sind von links nach rechts und von oben nach unten.
- Die Leerstellen werden mit zufälligen Buchstaben aufgefüllt.

(2) mittel

- Die Wörter werden vorwärts oder rückwärts eingefügt.
- Alle Richtungen sind erlaubt (links nach rechts, oben nach unten und diagonal).
- Die Leerstellen werden mit zufälligen Buchstaben aufgefüllt.

(3) schwer

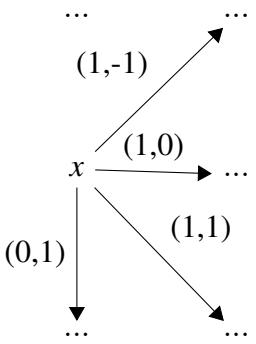
- Die Wörter werden vorwärts und rückwärts eingefügt.
- Alle Richtungen sind erlaubt (links nach rechts, oben nach unten und diagonal).
- Die Leerstellen werden mit Buchstaben aufgefüllt, welche bereits in den eingefügten Wörtern vorkommen.

Die Struktur einer Wortsuche

Eine Wortsuche lässt sich beschreiben als eine $h \times b$-Matrix, wobei h die Anzahl der Zeilen und b die Anzahl der Spalten ist. Ein eingefügtes Wort können wir dann durch einen Startindex (i, j) mit $0 \leq i \leq h-1$ und $0 \leq j \leq b-1$ sowie eine Richtung (d_x, d_y) beschreiben. Dabei beschreibt d_x die horizontale Veränderung, also die Veränderung von j, und d_y die vertikale Veränderung, also die Veränderung von i. Um eine Richtung zu definieren, müssen wir verstehen, wie sich die Indizes i und j verhalten, wenn wir über das einzufügende Wort in der

Matrix iterieren. Fügen wir zum Beispiel ein Wort von links nach rechts ein, dann bleibt der Reihenindex i unverändert und der Spaltenindex j steigt mit jedem weiteren Buchstaben. Von der Startposition aus können wir also die Position jedes weiteren Buchstabens bestimmen, indem wir i unverändert lassen und $j+1$ rechnen. Somit ergeben sich vier Richtungen (d_x, d_y):

1. $(1,0)$ von links nach rechts
2. $(0,1)$ von oben nach unten
3. $(1,-1)$ diagonal nach oben
4. $(1,1)$ diagonal nach unten



Alle anderen Richtungen ergeben sich durch das Rückwärts-Einfügen des Wortes.

Das Einfügen eines Wortes

Um eine zufriedenstellend schwierige Wortsuche zu erstellen, müssen die Wörter an zufälligen Positionen (in einer zufälligen erlaubten Richtung) eingefügt werden. Dabei bietet es sich an, die Wörter zu Beginn nach Länge zu sortieren und mit dem längsten Wort anzufangen. Falls das längste Wort länger als $max(h,b)$ ist, so kann jetzt bereits abgebrochen werden, da es keine Richtung gibt, in der dieses Wort in die Matrix passt. Basierend auf der Wortlänge l und der Richtung ergibt sich ein Bereich, in welchem es möglich ist, das Wort einzufügen:

$$\begin{cases} 0 \leq i \leq h-l & \text{für } d_y \geq 0 \\ l-1 \leq i \leq h-1 & \text{sonst} \end{cases} \quad \text{und} \quad \begin{cases} 0 \leq j \leq b-l & \text{für } d_x = 0 \\ 0 \leq j \leq b-1 & \text{sonst} \end{cases} \tag{3.1}$$

Eine gültige Position für ein Wort ergibt sich aus den Koordinaten (i,j) innerhalb des validen Bereiches sowie einer Richtung (d_x, d_y), sodass für jeden Buchstaben des Wortes in der Zelle (i,j) in der Matrix entweder nichts oder bereits der korrekte Buchstabe steht. Für jedes Wort in der sortierten Wortliste können nun die gültigen Positionen berechnet werden und eine zufällige ausgewählt. Falls sich für ein Wort keine gültige Position mehr ergibt, dann bricht der Algorithmus ab und beginnt von vorne. Ein Backtracking zum letzten Wort, welches mehrere gültige Positionen hat, d. h. ein Neustart von dort aus garantiert nicht notwendigerweise, dass diesmal für ein späteres Wort eine gültige Position gefunden wird, da der Fehler bereits beim zufälligen Einfügen eines früheren Wortes zustande gekommen sein kann.

Generieren der Wortsuche

Bei komplexeren Wortsuchen (mehr Wörter, weniger Dimensionen etc.) ist nicht garantiert, dass der Algorithmus eine Lösung zurück gibt, auch wenn theoretisch eine Lösung möglich wäre. Die Laufzeit des naiven Algorithmus wird optimiert, indem

- zu Beginn geprüft wird, ob eine Lösung theoretisch möglich ist, indem die Dimensionen mit der Länge des längsten Wortes verglichen wird und
- der Bereich, über den iteriert wird, um gültige Positionen zu finden, eingeschränkt wird.

Algorithmus 3 Generierung einer Wortsuche

```
1:  procedure WORTSUCHE(b, h, wortliste)
2:      if max_{wort∈wortliste}(länge(wort)) > max(h, b) then return false
3:      end if
4:      richtungen ← (0,1), (1,0), (1,1), (1,−1)
5:      sortiere(wortliste, wortlänge, absteigend)          ▷ Fange mit dem längsten Wort an
6:      for wort ∈ wortliste do
7:          permutiere_zufällig(richtungen)
8:          for (d_x, d_y) ∈ richtungen do                  ▷ Fange mit einer zufälligen Richtung an
9:              # Bestimmt den gültigen Bereich nach 3.1
10:             i_unten, i_oben, j_unten, j_oben ← gültigerBereich(n, h, länge(wort))
11:             # Iteriere über den Bereich und bestimme alle gültigen Positionen
12:             for i ← i_unten, ..., i_oben do
13:                 for j ← j_unten, ..., j_oben do
14:                     koordinaten ← validePositionen(i, j, d_x, d_y)
15:                 end for
16:             end for
17:             if koordinaten then                         ▷ Falls eine gültige Position gefunden wurde
18:                 # Füge an einer zufälligen gültigen Position ein
19:                 permutiere_zufällig(koordinaten)
20:                 fügeWortEin(wort, koordinaten, d_x, d_y)
21:                 break
22:             end if
23:         end for
24:     end for
25:     fülleMitBuchstabenAuf()                             ▷ Alle Wörter sind eingefügt
26:     return true
27: end procedure
```

Der Algorithmus wird maximal 10 mal ausgeführt, danach wird davon ausgegangen, dass keine Lösung existiert. Für die bereitgestellten Beispiele benötigt der Algorithmus meist nur einen Versuch, um alle Wörter einzufügen.

Für jedes Wort gibt es höchstens $b \cdot h$ mögliche Startpositionen und 4 mögliche Richtungen. Das Wort einzufügen bzw. zu überprüfen, ob das Wort so eingefügt werden kann, benötigt $\mathcal{O}(\text{länge(wort)})$ Anweisungen. Ist w die Anzahl der Wörter und l die Länge des längsten Wortes, so lässt sich die Laufzeit daher durch $\mathcal{O}(w \cdot l \cdot b \cdot h)$ nach oben abschätzen.

Die Änderung der Schwierigkeitsstufen ist der Übersicht halber nicht im Pseudocode enthalten. Im Falle von *einfach* und *mittel* wird die Schwierigkeitsstufe durch eine entsprechende Auswahl aus *richtungen* erzwungen. Für Schwierigkeit *schwer* soll die Wortsuche nicht mit zufälligen Buchstaben aufgefüllt werden, sondern mit Buchstaben oder Fragmenten, welche in den Wörtern aus der Wortliste vorkommen. Dies führt im Beispiel `worte5.txt` (wo nur **DAS** in eine 30×30-Matrix eingefügt werden soll) dazu, dass mit sehr hoher Wahrscheinlichkeit mehrfach das Wort **DAS** entsteht. Dies ist zwar laut Aufgabenstellung nicht explizit verboten, führt allerdings dazu, dass nach den obigen Definitionen der Schwierigkeitsgrad *schwer* sehr viel einfacher wäre als *einfach*. Dies sollte auffallen und zum Beispiel gelöst werden, indem beim Auffüllen der Matrix mit den zufälligen Buchstaben getestet wird, ob ein neues Wort entsteht, welches bereits in der Wortliste enthalten ist.

3.2 Beispiele

Angewandt auf die Eingaben auf der BWINF-Website liefert das Programm folgende Wortsuchen. Da zu erwarten ist, dass alle Wortsuchen voneinander abweichen, sind hier nur ein paar repräsentative Beispiele gezeigt.

worte0.txt:

```
LEVEL 1:
E K E V A
T O R F Z
G V O R Q
Q N C A Q
X Z Y D X

LEVEL 2:
P V R J G
E T R A O
U V O O D
A N A R V
Q Q D K F

LEVEL 3:
V R E V A
R A T T R
O D O V D
V R R F R
F O O D R

SUCHE:
TORF, VOR,
RAD, EVA
```

worte1.txt:

```
LEVEL 1:
O I N F O O
U B D D A G
G Q E I N J
L E C U N D
E R P R N D
R X X S P U

LEVEL 2:
Y N E D S W
F A U U L O
D Q N S F K
R D X N U Q
P E I I C M
H P R E J P

LEVEL 3:
D A N U I N
F U F E N A
E F I A F N
A N E D O O
D E R U N E
F R D D N U

SUCHE:
INFO, EIN,
UND, DA,
ER, DU
```

worte2.txt:

```
LEVEL 1:
J I W O P B F T J F
K N H G C I T A M E
Z B Q W O L G S G S
M A U S M D A T J T
U L Y W P S F A W P
M N Z F U C A T A L
V U S B T H A U D A
W R K T E I M R D T
S B H W R R H N M T
F B Y Z K M C Y N E

LEVEL 2:
U S B R E F G M N Q
M P A U R E J R O C
C S Z T M S Z I Y B
O U O A D T F H X U
M S U T V P J C A M
P S E S K L Z S B H
U W D A U A M D S V
T D W T X T B L P V
E W P V Z T W I W Q
R D H S Z E S B I I

LEVEL 3:
P E B L E F A U M M
A T I S U A M A M C
M T L U M A M M O T
U A D M S E E M A A
M L S A M B P M U S
F P C P E U P P E T
L T H F T U M P E A
P S I E A A A A A T
U E R F A L U U E U
M F M E L P M L A R

SUCHE:
BILDSCHIRM,
FESTPLATTE,
TASTATUR, COMPUTER,
MAUS, USB
```

worte4.txt:

LEVEL 1:

```
F O R D K T R G I Y R U C E Q D A R C H I V C J J J W B M J N S
N R G A Q E F A B R U F D A T U M L Y P A F U Y W Q J B S N F I
Z K G E R X S T H ö H E N M J S V I S D Z O Y B F A V K T M U O
C O R Q H T V B X A C H T U N G K T O Q E W N E G H N L V O S A
O Y E W Y H G I K Q Z F Q M U K D E R M G K F G G N I S U U S U
M R Z V F A R B L E G E N D E D I R T X E F A R N Y A Q N U B V
M A I U F Y N R N F T E Z Q C C U A J F O X U I M D J N K V A W
O F L H A P O E L W E S A U T T R T L E Q H D F A O E A C Q L U
N A S V C X K C Z X M L N K Y M C U C P U M K F U I N V E Y L C
S D Y F E G C O O R D I N A T E A R U S E W F S T G O I Z Z D D
C Y F E X A M R B N E T R C Y K N U D J L P B K O Y C G R U A H
A D N H I U Q D I N T E R N E T Q U E L L E D L A W O A E J T X
T K T H R S I P C O M M O N S W K B E L E G E ä R I M T E X E J
P G G I G Z J U F Y A R C H I V B O T Q U J C R C K M I D E N U
ö S T E R R E I C H B E Z O G E N E K M E O W U H T O O K Y C G
P O S I T I O N S K A R T E W I K I D A T A Z N I I N N P A E F
L V B A B E L C R L O Y L B Z Z I T A T I O N G V O S S M B D M
A N S U P I N G D I T M T J I N F O B T A J D S H N W L E O B U
A F E R P L S B S M I L E Y H E T N E Y B S T H X A B E D O N L
W J T N S S C C S L U M N L B X E L G B T Z I I S R C I A L E T
J I N F O R M A T I O N F L J G R A R F I M L N S Y F S I A E I
P I W E B A R C H I V N V I J Y Y N I L D E U W G Z O T L N S L
S D I S K U S S I O N S S E I T E G F I R W D E S Y L E L D T I
A C W M H I X X R B E N U T Z E R S F Z F A U I A H G K E P Q N
O B T T J C F C B A U S T E L L E H S E W P K S M T E S N M B G
E G R O X O S O Q J C V T A I G E K K N I P M P B N N S S R A U
T R S K U O D G R H W Y W K B T Q J L Z K E U E K W L O P G L A
A W A A A R P O K C H A R T S F R A ä U I N S R A O E H I A L L
U U S T I D A R C H I V I E R U N G R M S R I S L O I O E B M I
V A K E Q I P C S Z J M P G G Y O Q U S O E K O E Z S A G S U E
V K J G V N Q E E R L E D I G T J W N T U C C N N X T Z E A S C
Q Z K O Y A D N R Y N A V F R A M E G E R H H E D A E F L T I I
R J T R Y T W T T J O E L D D B J L X L C T A N E I X F Q Z C M
P I C I V E H E A O R C R O M C N P J L E F R L R R I S A C T D
H I G E K M P R X B K A S T E N R T Q U J H T E S W A P X F P B
Q K Z G A A S I O B O E O A U R F F J N Z V S I T V K K G I H Y
H Q Q R L P I P B U J A Q Q T A R A E G O L L S I Z V P V L T N
L H Y A O T P C O C O L Q R K E J Z S B E L F T L Y P R O M F G
P Z V P V Q A I X E Q A Y G A N G N M R X D T E H N S D Q C X R
V S C H I N N O K L M W B L W V R A W I P Y H G I J X S P O R H
```

SUCHE:

BEGRIFFSKLäRUNGSHINWEIS, NAVIGATIONSLEISTE, öSTERREICHBEZOGEN, DISKUSSIONSSEITE, LIZENZUMSTELLUNG, MEDAILLENSPIEGEL, BEGRIFFSKLäRUNG, INTERNETQUELLE, KATEGORIEGRAPH, PERSONENLEISTE, POSITIONSKARTE, COORDINATEMAP, FUSSBALLDATEN,

ARCHIVIERUNG, FOLGENLEISTE, KALENDERSTIL, MULTILINGUAL,
FARBLEGENDE, INFORMATION,MUSIKCHARTS, WAPPENRECHT, ABRUFDATUM,
AUTOARCHIV, COMMONSCAT, COORDINATE, WIKISOURCE, WIKTIONARY,
ARCHIVBOT, BAUSTELLE, BIBRECORD, GEOQUELLE, LITERATUR,
NOCOMMONS, WEBARCHIV, ALLMUSIC, BENUTZER, ERLEDIGT, NAVFRAME,
WIKIDATA, ZITATION, ACHTUNG, BOOLAND, COMMONS, TAXOBOX, ABSATZ,
ARCHIV, BELEGE, CENTER, CHARTS, KASTEN, SMILEY, BABEL, BBKL,
FILM, GNIS, HöHE, IMDB, INFO, LANG, PING, SORT, TEXT, AUS,
AUT, BEL, BGR, CAN, COL, DDB, DEU, DOI, FNZ, FRA, GER, IPA,
PRO, FN

worte5.txt:
LEVEL 1:

U I O R U J Q K Q L Y S P B B E Y K I D S Q L V M B S U Q A
H K L J A E X M F N Y A F A X B Q S D Y K U C Q I R F N B B
L K T Z O Z F V R C F H H C O M F D I V X D G E P H I K Z B
M D V T O D S E B W V S L J F A E A K A B F M Q M O Y N M V
N P A H A O U O C M O P K Q S L Q K M D W J B I B Y H V K L
H Y L D G F L Y C S X C R G Q J Y I M R V Z M L Z W K P P M
O P V W H C U T U L X S Y T Z K N X J V J C M H S I L K O E
G Q B V Z L Y O J C Z X S X Z P Z A O R S Y U L B M X D D X
K H W B I H D I G S A A N L O F A W L I R C W J R M N N J C
Q M Z I P Y S G N W K P B P G D O V P S H F X W S Y X U K W
R O O M X B C S E K G C A U F C U N M I W A N X N U T E E L
F V S P N F J L K W P D F N R O W N A S R U F L R A S A I J
V T E V D Z H M U N M A P F X H W A M J D U C C O J X T B I
H J G P L P V C Y X X M T Y J Q Z C B F Q T S R U Z T B P A
A K R Y U V I E S N M U D R Q G D Y P Y H C B Z J X M Z E R
S L U U N H Y W M V K L X K Y N D D A S K W Z H X B D T I W
Y E J E X C V J K C S B K G O I H O R B D L B T O X P X W H
U W D K A E H D Y X M B X L C Z A R Y O G M N X S A T X V Q
V S K M I H C D L Y Z C N B N I U F J M R G J S G F L X C A
E W M X T I Y U G R W W L E G X R E K T U T X P K W R J O B
D E I N X F W P Q D D N T C J T V L H U I C W D Y Z G D M N
Y P I H E S N Q F D Y N I P M R A W R Y I Z Z T E Y F E R T
O N V H D Q O K T W Z F A Z T D X Y B D L K I K J T N N B Q
C H U T V E P R F R O V S L G T W E Z Z H V T V C E C R D L
H W S J H N G I M E N P R E B I Y V P D I N T H X S I R E Q
Z R T X K V A Z E C I E R M Y P A F C H E J P W U H L N B R
N J Q N P M O H I Q W I Z G N J Z M Y P J V J U Y W U O R F
D M D F L D B S A A R M E S T G A A C L H O A W O U K D A M
Z C T U U W L V V D P Q V W G M Y I X H L D S A C X C Z X V
A V G X B H M F S S E U N W I Q T P V M W Y E Y Q R I V S U

```
LEVEL 3:
D S S S S S D U S D S S D A A A A S D O D A D A U D A E D S
S O D O S A S D D S U D U U U D D S U U S S S O A O O O U O
O O S S A U S A S U S A S O A S S D D S D S U O A D O A A S
S A U U A D X G S U S U S D A L A O O A O U O U D S O A A S
F U O O S T U A A S S S D A D A M D D S U U A S S U S C U S
O S O O S O S D A M D A X D S A S S A A O S O A A S U D S U
A A D O O O A A D A D U S O U U O U O Y H S U S S O O S U S
D A O S A U U U A U U S A S D O D U A D A O O O A D S O O D
A A U D D D U O O A U A O A S O S S O B S U D D O S D D U O
H A A U O O U O D S S O D U A S U S A S U S U U D D U S A U
D U D D A U D O S O D U S U S D D D A R A D A U D A J O O P
A U U A O A A A D U O O O D S O D S A O D Q D U D O M A D O
I A O O D S O S U A S D A D A A D S D O D U A O A O U U U S
O O A A D O A O O U D A D S C D O U D S O O U A J D O A U A
O A U U D U O U D D A U Q U U O U O A U A D D D U O A A O S
U A D A E S A U S D T D S D D D D S A E A S O A O A D O A O
O D O D O S D O O D A U A A O O D S S U O O U U U S D Q A S
O U A O O A A A D D U D V U S A S U S S O O O D S U O A O S
O U O B U O A O D A O U O S D S S S D S A U U S S D A O A S
S D A T S O A D O U A A A D O U U A S O O O O O O U A A S D
A A D S U U D U S O O Z U A D U A S S A A S D O A O O Q S D
W O O S D U D A A O A D D G O U D U D O S S S U A S O O O O
A D D U S D S U O O O A D O O A O A O A D S S D A W D U A S
A D A A A A A D D U K D A D D D O U O S U D A U D D S D A A
S O F U L U S G A U S S U A D A U U O D D D S U O A S F U S
S O A D D O D U A U A D O D U O O S U O O S D D U H X O A S
U D U A D O D U D A A O O U U U S S D O S A O U D A A Q U S
S U A O A S A U U A O O A A O D A O A S O U A A S A S U O U
D S U Q D O O O A U U S O O D O U A A A O A A O S A G U A O
D O O A D O O S D O U U A S U A A W O U A A S O O O S O D A

SUCHE:
DAS
```

3.3 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- $[-1]$ **Lösungsverfahren fehlerhaft**
  Bei einer gültigen Lösung müssen alle Worte aus der Wortliste korrekt eingefügt worden sein. Wenn beim Auffüllen der Leerstellen zufällig Worte aus der Wortliste entstehen, dann führt dies nicht zu Punktabzug. In üblichen Wortsuchen sollte das zwar nicht vorkommen, aber die Aufgabenstellung war in diesem Punkt nicht klar.
- $[-1]$ **Kriterien ungeeignet definiert**
  Die Aufgabe fordert, dass Kriterien definiert werden, welche die Schwierigkeit von Wortsuchen intuitiv voneinander unterscheiden. Diese Kriterien sollten einerseits eine Klassifizierung von gegebenen Rätseln ermöglichen und andererseits als Basis zur Erzeugung neuer Rätsel geeignet sein. Spätestens für die Definition der Schwierigkeitsstufen sollten Kriterien erkennbar sein.
- $[-1]$ **Schwierigkeitsgrade ungeeignet definiert**
  Die Aufgabe fordert, dass ≥ 3 Schwierigkeitsgrade definiert werden, welche auf den vorher definierten Kriterien basieren. Die Anwendung der Schwierigkeitsgrade auf eine Wortliste sollte zu merklich unterschiedlichen Wortsuchen führen. Es muss betrachtet werden, dass mit Regeln, die auf dem zufälligen Einfügen von Fragmenten basieren, weitere Worte entstehen können. Dies kann dazu führen, dass Wortsuchen, welche mit schwierigeren Regeln generiert wurden, einfacher zu lösen sind (z. B. `worte5.txt`).
- $[-1]$ **Verfahren bzw. Implementierung unnötig aufwendig / ineffizient**
  Es sollte sinnvolle Bedingungen geben, die den Algorithmus abbrechen lassen, falls nach mehrfachen Versuchen keine Wortsuche erzeugt werden konnte. Ein Verfahren ist akzeptabel, wenn innerhalb weniger Sekunden eine Wortsuche generiert wird.
- $[-1]$ **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**
  Die Dokumentation soll Ergebnisse zu mindestens 3 der vorgegebenen Beispiele (`worte1.txt` bis `worte5.txt`) enthalten. Zu jedem dokumentierten Beispiel sollten idealerweise Wortsuchen mit jeweils drei Schwierigkeitsgraden angegeben werden. Sollten es weniger sein, ist das nur akzeptabel, wenn die Beispiele insgesamt einen guten Überblick vermitteln.

Aufgabe 4: Würfelglück

4.1 Lösungsidee

Ziel soll es sein, den Freunden eine verlässliche Auskunft über die Gewinnwahrscheinlichkeiten der einzelnen Würfel zu geben. Dafür bietet es sich an, einen *Monte-Carlo-Algorithmus*[8] zu verwenden, der für verschiedene Würfel jeweils eine große Anzahl von Partien zweier MENSCH-ÄRGERE-DICH-NICHT-Spieler simuliert. Je nachdem, wie oft ein Spieler mit einem Würfel gewinnt, kann dann eine Aussage über die Güte des Würfels getroffen werden.

Regeln

Besonders wichtig ist, dass der Algorithmus sehr genau den Regeln des Brettspiels entspricht, damit das Spiel der Freunde authentisch simuliert wird. Die Regeln werden deswegen hier nochmal zusammengefasst aufgeführt.

Zunächst gibt es die Regeln des Herstellers, die der Einfachheit halber im Wettbewerb angepasst wurden:

H1 Zum Spielen stehen vier Farben (schwarz, gelb, grün und rot) zur Verfügung. Gespielt wird auf einem Spielbrett mit insgesamt 40 Lauffeldern sowie 4 Zielfeldern der jeweiligen Farbe; genau wie im Materialverzeichnis beschrieben. Jede Farbe verfügt über ein markiertes Anfangsfeld sowie vier Figuren, genannt Steine.

H2 Jeder Spieler startet mit einem Spielstein auf dem Anfangsfeld (A-Feld), sowie den restlichen drei auf der Reservebank (B-Felder).

H3 Nach jeder Runde wechselt der startende Spieler (hier weichen die Regeln der Einfachheit halber und da es die Genauigkeit der ermittelten Werte erhöht vom Original ab, in welchem der Startspieler durch die höchste gewürfelte Zahl bestimmt wird).

H4 Ist ein Spieler am Zug, würfelt er und bewegt den vordersten Stein, der gezogen werden kann, um die Anzahl an geworfenen Augen in Richtung Ziel. Ausnahmen dieser Regel sind:

- Wenn ein Stein auf dem Anfangsfeld des Spielers steht und der Spieler noch nicht alle seine Steine auf dem Feld hat. Dann ist der Spieler, wenn möglich, gezwungen, das Anfangsfeld zu räumen und den blockierenden Stein zu bewegen.
- Wenn der Spieler eine 6 würfelt, muss er, wenn möglich, einen neuen Stein auf das Anfangsfeld stellen.
- Wenn der Spieler keinen Stein um die gewürfelte Anzahl Augen bewegen kann, verfällt sein Zug.

H5 Nach jedem Zug wird der Spieler gewechselt, außer, wenn eine 6 gewürfelt wurde.

H6 Ein Zug ist möglich, wenn entweder kein oder ein gegnerischer Stein auf dem Feld ist, auf das der Spieler ziehen möchte. Wenn ein gegnerischer Stein auf dem Feld ist, wird dieser geschlagen und vom Feld zurück auf die Reservebank gestellt.

[8] https://de.wikipedia.org/wiki/Monte-Carlo-Simulation

H7 Wenn ein Spieler mit einem Stein die ganze Laufbahn entlang gelaufen ist, so zieht er seine Figur in die Zielfelder weiter. Dort darf die Figur in den nächsten Zügen explizit weitergezogen und andere dürfen übersprungen werden.

H8 Wenn ein Spieler alle seine vier Steine in die vier Zielfelder gebracht hat, hat er das Spiel gewonnen.

Neben den im Wettbewerb festgelegten Regeln gibt es noch eine Besonderheit, über die man sich Gedanken machen muss: Bei einigen Würfelpaaren kann es zu einem sogenannten Deadlock des Spiels kommen. Das bedeutet, dass keiner der beiden Spieler mehr einen seiner Steine mit seinem Würfel ziehen kann. Hierfür sind mehrere Lösungen denkbar, in dieser Ausarbeitung wird das Spiel bei einem Feststecken beider Spieler als unentschieden gewertet. Denkbar ist beispielsweise auch, den zuerst feststeckenden Spieler verlieren zu lassen, oder den Gewinner in diesem Fall zu würfeln. Der Umgang mit diesem Problem sollte aber auf jeden Fall explizit beschrieben werden.

Da die Freunde womöglich mit anderen Regeln als denen des BWInf spielen, bietet es sich an, noch einige Variationen auszuprobieren. An vorderster Stelle steht jedoch die Simulation entsprechend den Regeln der Aufgabe. Regelvarianten dürfen nicht stattdessen, sondern nur in zusätzlichen Simulationen benutzt werden. Ein paar ausgewählte Variationen sind:

Z1 Der beginnende Spieler wird durch das Würfeln der höchsten Augenzahl in einer Vorrunde bestimmt.

Z2 Außerdem spielen die Freunde Anna, Barbara und Clemens zumindest zu dritt. Eine Spielsimulation mit drei oder vier Spielern und den unterschiedlichen Würfelkombinationen ist also ebenfalls denkbar.

Z3 Beim BWInf wird vorrangig der vorderste Stein gezogen, im echten Spiel werden die Spieler aber besonders hintere Steine ziehen, wenn sich dadurch ein gegnerischer Stein schlagen lässt. Eine Priorisierung von Schlagzügen ist also sehr sinnvoll.

Z4 Ebenfalls gängige Anpassungen sind, dass man nicht mit einem Spielstein auf dem Startblock startet,

Z5 dass man nach einer 6 dreimal würfeln darf, sobald man keinen Stein im Feld hat, oder

Z6 dass ein Rückwärtsschlagen, bei dem man Steine hinter sich schlägt, erlaubt ist.

Z7 Außerdem kann man ein Überhol- oder Zugverbot in den Zielfeldern einrichten.

Natürlich ist auch denkbar, eine tiefere Strategie der Spieler zu programmieren, allerdings ist dies hier nicht erfasst und wäre dann sehr spezifisch für die drei Freunde, die der Autor nicht persönlich kennt.

Spielfeldimplementation

Das Spielfeld kann dadurch implementiert werden, dass jeder Farbe ein Array o. Ä. mit vier Ganzzahleinträgen zugeordnet wird, das die Positionen seiner Steine speichert. Ein Zug kann dann durch Ändern der entsprechenden Einträge sehr leicht implementiert werden. Um die einzelnen Würfe der Spieler zu simulieren, wird ein gleichverteilter Pseudozufallszahlengenerator verwendet, der aus den gegebenen Augenzahlen (pseudo)zufällig eine auswählt. Diese Generatoren sind für die meisten Programmiersprachen bereits vorhanden und sollten genutzt werden.

Andere Implementationen, wie etwa das gesamte Spielfeld abzubilden, sind ebenfalls möglich, aber weniger effizient.

Der Spielablauf wurde realisiert, indem die vorher genannten Regeln des Wettbewerbs inklusive möglicher Zusatzregeln penibel eingehalten werden. Dadurch entsteht ein robustes Programm, welches für eine bestimmte Kombination geworfener Würfelaugen immer das selbe und entsprechend den Regeln korrekte Spiel simuliert.

Umsetzung der Simulation

In der Simulation von N Spielen im Zweispielermodus beginnen beide Spieler A und B abwechselnd und nehmen dann den Regeln entsprechend ihre Züge vor. Gezählt wird, wie viele Spiele G ein bestimmter Spieler, hier immer Spieler A, gewinnt. Seine Gewinnwahrscheinlichkeit $\hat{p}_A = G/N$ gibt dann an, mit welcher Wahrscheinlichkeit Spieler A mit seinem Würfel gegen Spieler B mit dessen Würfel gewinnt.

Es ist möglich, die erhaltenen Gewinnwahrscheinlichkeiten zu spiegeln: Wenn Spieler A mit Würfel 1 gegen Spieler B mit Würfel 2 in 75% der Fälle gewinnt, sollte Spieler A mit Würfel 2 gegen Spieler B mit Würfel 1 nur in 25% der Spiele gewinnen – vorausgesetzt, dass es keine Deadlocks (s. o.) gibt.

Um eine Rangliste der besten Würfel aufstellen zu können, berechnen wir die mittlere Gewinnwahrscheinlichkeit $\overline{p}_A$ gegen die anderen Würfel ohne sich selbst. Diese entspricht der Wahrscheinlichkeit, gegen einen uninformierten Gegner zu gewinnen, wenn dieser einen beliebigen anderen Würfel wählt. Den Würfel, der am häufigsten gewinnt, schätzen wir als den „besten“ ein.

Kennt man den Würfel seines Gegners, kann man natürlich direkt den gegen seine Wahl besten Würfel wählen; bei bekannten Vorlieben kann man mit den Wahrscheinlichkeiten seiner Auswahl gewichten. Denkbar wäre auch ein Ligasystem, um den besten Würfel zu bestimmen.

Laufzeit

Die Länge einzelner Spiele ist nicht vorhersehbar; eine konkrete Laufzeitanalyse anzufertigen, ist dadurch nicht direkt möglich. Da einzelne Spiele beliebig viele Züge benötigen können (denkbar wäre etwa, dass beide Spieler vor ihrem Ziel stehen und immer wieder eine unpassende Augenzahl zum Fertigstellen des Spiels werfen), gibt es keine maximale Laufzeit. Ermitteln wir eine mittlere Spiellänge I_{avg}, ist die Average-Case-Laufzeit $\mathcal{O}(I_{\text{avg}}N)$ linear in der Anzahl der Wiederholungen N.

Um ein Gefühl für die Spiellänge I der Simulationen zu erhalten, betrachten wir die Länge vieler simulierter Spiele, wie in Abbildung 4.1 dargestellt. Erwartungsgemäß streuen die Spiellängen relativ regelmäßig um den Mittelwert $I_{\text{avg}} = 108$.

Mathematische Gedanken zur Sicherheit des Ergebnisses

Dass unsere Methode funktioniert, verdanken wir dem *Gesetz der großen Zahlen*[9] aus der Stochastik. Es garantiert (vereinfacht gesagt), dass unsere Schätzung für die Gewinnwahrschein-

[9] https://de.wikipedia.org/wiki/Gesetz_der_großen_Zahlen

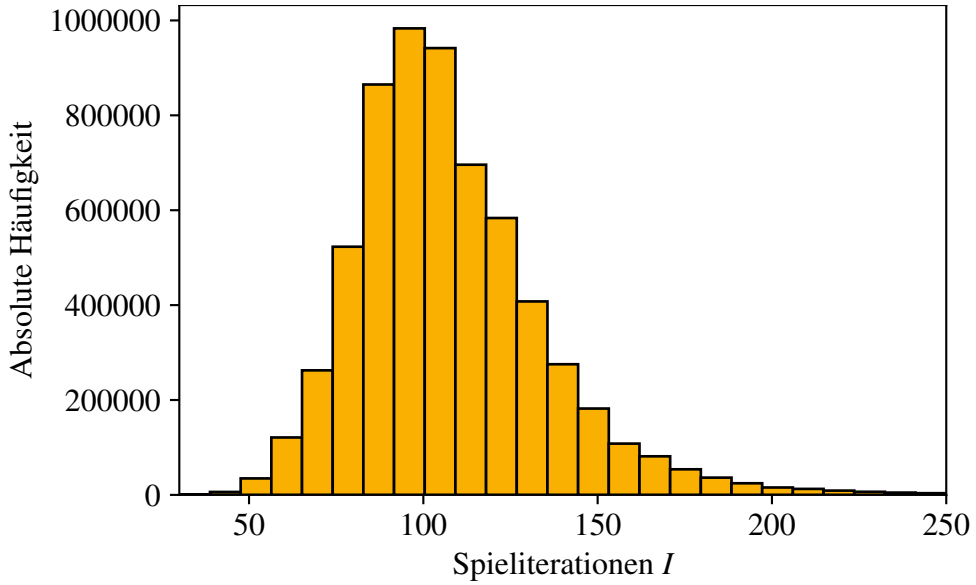


Abbildung 4.1: Verteilungsfunktion der Länge I von $N = 6250000$ Spielen mit den Würfeln aus `wuerfel0.txt` ohne den Würfel W_3. Das mittlere Spiel ist hier $I_{\text{avg}} = 108$ Züge lang, wobei das längste Spiel über $I_{\text{max}} = 503$ Züge ging.

lichkeiten immer genauer wird, je öfter wir unser Zufallsexperiment wiederholen. Als gewissenhafte Programmierer wollen wir den Freunden aber auch ungefähr sagen können, wie genau denn jetzt die erhaltenen Schätzwerte sind.

Dafür ist es wichtig, zwischen den Ergebnissen unserer Simulation und der wahren Gewinnwahrscheinlichkeit des Würfels zu unterscheiden. Je nachdem, wie unser Programm die einzelnen Spieler würfeln lässt, ergeben sich ganz unterschiedliche Spiele. Am Ende unseres Programms können also immer wieder andere Gewinnwahrscheinlichkeiten herauskommen, obwohl wir prinzipiell nichts geändert haben. Natürlich gibt es in der Realität aber eine konkrete, wahre Gewinnwahrscheinlichkeit der Würfel. Sie unterscheidet sich von unseren Simulationen dadurch, dass sie eben keine Schätzung ist und nicht vom (Pseudo)zufall der durchgeführten Simulationen abhängt.

Nun stellt sich die Frage: Wie nah ist diese Schätzung am wahren Wert?

In statistischer Sprechweise stellen wir die Frage nach der Sicherheit unserer Ergebnisse. Wir wollen also relativ sicher sagen können, dass die den Freunden gegebenen Werte beispielsweise um höchstens einen Prozentpunkt von den korrekten Werten abweichen. Eine genauere mathematische Auseinandersetzung mit dieser Frage folgt im nächsten Abschnitt; diese ist aber nicht für das Verständnis des grundlegenden Sachverhaltes notwendig und richtet sich nur an besonders Interessierte.

Um trotzdem die Genauigkeit einordnen zu können, wiederholen wir einfach oft unsere Simulation mit verschiedenen Startwerten und vergleichen die erhaltenen Gewinnwahrscheinlichkeitsschätzungen, wie in Abbildung 4.2. Dort ist der konkrete Verlauf der Schätzung einer Simulation sowie der resultierende Mittelwert von insgesamt 100 Simulationen dargestellt. Der für uns interessante Streubereich der Schätzungen ist in Gelb markiert und tatsächlich beobachten wir, dass dieser immer enger wird und die Schätzwerte der einzelnen Simulationen immer näher an einen gemeinsamen Mittelwert kommen. Die Garantie des *Gesetzes der großen Zahlen* scheint also bestätigt: die Schätzungen der einzelnen Simulationen streben mit mehr Wiederholungen immer genauer gegen einen gemeinsamen Schätzwert, der vermeintlich wahren Gewinnwahr-

scheinlichkeit.

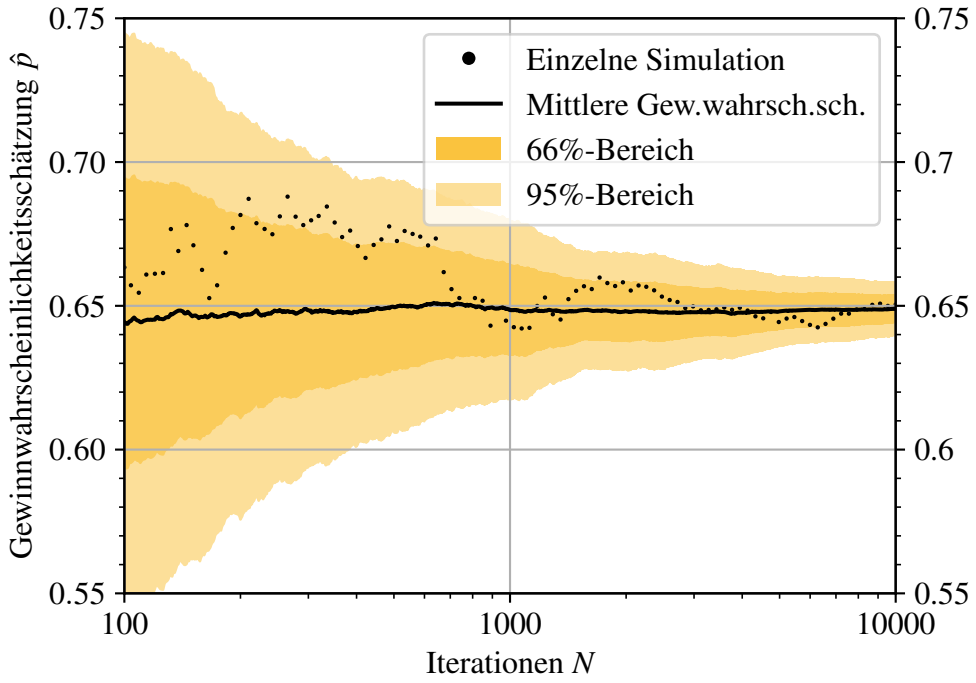


Abbildung 4.2: Verlauf von Erwartungswert und Streuung der mittleren Gewinnwahrscheinlichkeiten bei 100 simulierten Partien MENSCH-ÄRGERE-DICH-NICHT, wobei der Standardwürfel mit sechs Seiten gegen den mit zwölf Seiten spielt. Die geschätzte Gewinnwahrscheinlichkeit einer Simulation ist mit schwarzen Punkten dargestellt. Die Bereiche, in denen jeweils 66% bzw. 95% der Simulationen liegen, sind dunkel- bzw. hellgelb ausgefüllt.

Genauere mathematische Auseinandersetzung

Wir erinnern uns an die Fragestellung: Wie nah ist diese Schätzung am wahren Wert?

Um zu einer Antwort zu kommen, betrachten wir unsere Simulation als viele einzelne, stochastisch unabhängige Spiele, sogenannte *Bernoulli-Experimente*[10]. In jedem Spiel kann hierbei Spieler A entweder gewinnen oder verlieren (entsprechend würde Spieler B entweder verlieren oder gewinnen). Die wahre Wahrscheinlichkeit, mit der Spieler A gewinnt, ist p. In unserer Simulation zählen wir, wie viele der N Spiele der Spieler A gegen seinen Gegner gewinnt; diese Zahl nennen wir G, die Anzahl gewonnener Spiele. Daraus können wir als Näherung für die wahre Wahrscheinlichkeit p eine Schätzung $\hat{p} = G/N$ aufstellen.

Mathematisch wollen wir nun die Standardabweichung der geschätzten Gewinnwahrscheinlichkeiten $\hat{p}$ von der wahren Gewinnwahrscheinlichkeit p abschätzen. Diese gibt an, wie weit unsere Schätzung in $2/3$ der Fälle beziehungsweise bei der doppelten Standardabweichung in 95% der Fälle vom wahren Wert abweicht. Oft wählt man tatsächlich die doppelte Standardabweichung, da mit 95% „fast jeder“ Fall abgedeckt ist, wir betrachten also das 2σ-*Signifikanzniveau*, unsere Ergebnisse haben damit eine Sicherheit von 95%. Außerdem müssen wir uns auf eine Genauigkeit Δ der Gewinnwahrscheinlichkeit einigen. Diese gibt an, wie weit die einzelnen

[10] https://en.wikipedia.org/wiki/Bernoulli_trial

Schätzungen im Rahmen unserer Sicherheit vom wahren Wert abweichen können; eine Genauigkeit von $\Delta = 1\%$ erscheint hierbei sinnvoll.

Der freie Parameter, den wir anpassen müssen, um unsere Genauigkeitsansprüche zu erfüllen, ist nun die Anzahl N von Wiederholungen unserer Simulation. Umso genauer das Ergebnis werden soll, umso mehr Wiederholungen müssen wir machen. Allerdings wollen die Freunde möglichst schnell anfangen, zu spielen, und nicht ewig auf ein Ergebnis warten. Die uns interessierende Größe ist die Standardabweichung unserer Schätzung[11] $\sigma_{\hat{p}} = \frac{\sigma}{\sqrt{N}}$ mit der Varianz eines einzelnen Spiels $\sigma = \sqrt{p(1-p)}$.

Unsere Bedingungen zusammengefasst, wollen wir also, dass die doppelte Abweichung des Mittelwertes $2\sigma_{\hat{p}} \leq \Delta$ kleiner oder gleich unserer gewünschten Genauigkeit ist. Entsprechend gilt mit $p(1-p) < \frac{1}{4}$

$$2\sigma_{\hat{p}} = 2\sqrt{\frac{p(1-p)}{N}} < \sqrt{\frac{1}{N}} \overset{!}{\leq} \Delta \implies N \geq \Delta^{-2} = 10000. \tag{4.1}$$

Wenn wir also ungefähr zehntausend Simulationen pro Würfelkombination spielen, können wir mit 95%-iger Sicherheit sagen, dass die erhaltenen Ergebnisse höchstens um einen Prozentpunkt abweichen.

Betrachten wir wieder Abbildung 4.2, so sehen wir, dass die Gewinnwahrscheinlichkeiten nach 10000 Iterationen tatsächlich um circa $\pm 1\%$ um den gemeinsamen Mittelwert streuen.

4.2 Beispiele

Für die vom BWInf vorgegebenen Regeln ergeben sich die hier aufgeführten Gewinnwahrscheinlichkeiten. Diese wurden entgegen der obigen Rechnung mit $N = 120000$ Iterationen ermittelt, um eine Genauigkeit von $\Delta = 0.3\%$ zu garantieren. Als Beispiele wurden die vom BWInf zur Verfügung gestellten Würfelschachteln gewählt.

Es sei nochmal darauf hingewiesen, dass die Ergebnisse sehr empfindlich auf eine korrekte Implementation der Regeln reagieren. Schon kleine Ungenauigkeiten führen zu teilweise gravierenden Unterschieden, weswegen ein genauer Vergleich immer sehr schwierig ist. Ebenso maßgebend ist auch der Umgang mit den Deadlocks; um hier eine Vergleichbarkeit zu ermöglichen, ist in Tabelle 7 deren Quote für die Datei `wuerfel1.txt` aufgeführt.

Die Implementation der Beispiellösung braucht deutlich weniger als 1 s pro 10000 Iterationen; mehr als 10 s sollten für 10000 Iterationen nicht benötigt werden.

[11] https://de.wikipedia.org/wiki/Varianz_(Stochastik)#Summen_und_Produkte

		Würfel Sp. *B*							
		W_1	W_2	W_3	W_4	W_5	W_6	$\overline{p}_A$	Pl.
Würfel Sp. *A*	W_1	50,1	1,5	100,0	65,1	64,7	88,4	63,9	2
	W_2	98,5	49,8	100,0	98,6	96,7	99,0	98,6	1
	W_3	0,0	0,0	-	0,0	0,0	0,0	0,0	6
	W_4	35,1	1,5	100,0	50,3	51,5	79,3	53,5	3
	W_5	35,2	3,3	100,0	48,6	50,1	75,9	52,6	4
	W_6	11,7	1,1	100,0	20,8	23,9	49,8	31,5	5

Tabelle 3: Gewinnwahrscheinlichkeiten von Spieler *A* in Prozent für `wuerfel0.txt`. Die Würfel der Schachtel sind $W_1 = (1,2,3,4,5,6)$, $W_2 = (1,1,1,6,6,6)$, $W_3 = (1,2,3,4)$, $W_4 = (0,1,2,\ldots,8,9)$, $W_5 = (1,2,\ldots,11,12)$ und $W_6 = (1,2,\ldots,19,20)$.

		Würfel Sp. *B*							
		W_1	W_2	W_3	W_4	W_5	W_6	$\overline{p}_A$	Pl.
Würfel Sp. *A*	W_1	50,1	43,8	50,7	54,5	69,5	81,1	59,9	2
	W_2	56,4	50,0	51,8	54,5	66,0	74,7	60,7	1
	W_3	49,6	48,0	49,9	51,6	61,5	68,5	55,8	3
	W_4	45,4	45,5	48,7	50,0	59,5	65,7	53,0	4
	W_5	30,5	33,8	38,6	40,7	49,9	56,1	40,0	5
	W_6	18,8	25,2	31,4	34,3	44,0	50,0	30,7	6

Tabelle 4: Gewinnwahrscheinlichkeiten von Spieler *A* in Prozent für `wuerfel1.txt`. Die Würfel der Schachtel sind $W_1 = (1,2,3,4,5,6)$, $W_2 = (2,3,4,5,6,7)$, $W_3 = (3,4,5,6,7,8)$, $W_4 = (4,5,6,7,8,9)$, $W_5 = (5,6,7,8,9,10)$ und $W_6 = (6,7,8,9,10,11)$.

		Würfel Sp. *B*						
		W_1	W_2	W_3	W_4	W_5	$\overline{p}_A$	Pl.
Würfel Sp. *A*	W_1	50,0	1,0	0,0	0,0	0,0	0,3	5
	W_2	99,1	50,2	3,8	0,1	0,0	25,8	4
	W_3	100,0	96,2	49,9	6,6	0,2	50,8	3
	W_4	100,0	99,9	93,3	50,0	8,3	75,4	2
	W_5	100,0	100,0	99,8	91,8	50,1	97,9	1

Tabelle 5: Gewinnwahrscheinlichkeiten von Spieler *A* in Prozent für `wuerfel2.txt`. Die Würfel der Schachtel sind $W_1 = (1,1,1,1,1,6)$, $W_2 = (1,1,1,1,6,6)$, $W_3 = (1,1,1,6,6,6)$, $W_4 = (1,1,6,6,6,6)$ und $W_5 = (1,6,6,6,6,6)$.

		Würfel Sp. B							
		W_1	W_2	W_3	W_4	W_5	W_6	$\overline{p}_A$	Pl.
Würfel Sp. A	W_1	49,9	70,8	65,9	79,8	77,2	93,2	77,4	1
	W_2	28,8	49,8	47,5	64,7	64,7	88,4	58,8	3
	W_3	33,9	52,7	50,3	65,1	64,5	87,0	60,6	2
	W_4	20,2	35,0	34,9	50,2	51,6	79,6	44,3	4
	W_5	22,8	35,2	35,5	48,3	49,9	75,8	43,5	5
	W_6	6,7	11,5	13,2	20,5	24,0	50,1	15,2	6

Tabelle 6: Gewinnwahrscheinlichkeiten von Spieler A in Prozent für `wuerfel3.txt`. Die Würfel der Schachtel sind $W_1 = (1,2,5,6)$, $W_2 = (1,2,3,4,5,6)$, $W_3 = (1,2,\ldots,7,8)$, $W_4 = (0,1,\ldots,8,9)$, $W_5 = (1,2,\ldots,11,12)$ und $W_6 = (1,2,\ldots,19,20)$.

Besonderheiten der Partien im Deadlock

Wie bereits erwähnt, ist es besonders wichtig, wie die Deadlocks im Spiel behandelt werden. In der folgenden Übersicht sind die Wahrscheinlichkeiten eines Deadlocks beim MENSCH-ÄRGERE-DICH-NICHT mit der Würfelschachtel aus `wuerfel1.txt` gegeben. Dass eine umfassende Diskussion der Auswirkungen tatsächlich wichtig ist, sieht man vor allem daran, dass teilweise die Hälfte der Spiele in einem Patt der beiden Spieler endet.

		Würfel Sp. B					
		W_1	W_2	W_3	W_4	W_5	W_6
Würfel Sp. A	W_1	0,0	0,0	0,0	0,0	0,0	0,0
	W_2	0,0	6,2	10,7	12,7	17,9	21,0
	W_3	0,0	10,7	19,0	23,0	31,5	36,9
	W_4	0,0	12,8	23,1	28,5	39,1	44,7
	W_5	0,0	17,7	31,6	39,3	54,3	62,5
	W_6	0,0	21,1	37,0	45,3	62,4	71,9

Tabelle 7: Häufigkeit eines Deadlocks in Prozent bei den verschiedenen Würfelpartien in `wuerfel1.txt`.

Werden einfach alle Partien ohne eindeutigen Sieger wiederholt, ergeben sich für die Beispielwürfel die folgenden Gewinnwahrscheinlichkeiten. Ihre Rangfolge ändert sich nicht.

		Würfel Sp. B							
		W_1	W_2	W_3	W_4	W_5	W_6	$\overline{p}_A$	Pl.
Würfel Sp. A	W_1	50,1	43,8	50,7	54,5	69,5	81,1	59,9	2
	W_2	56,4	50,1	52,0	55,1	69,5	81,2	62,8	1
	W_3	49,6	47,8	49,9	52,1	66,7	79,3	59,1	3
	W_4	45,4	44,8	48,3	50,0	65,7	78,4	56,5	4
	W_5	30,5	30,3	33,4	34,6	49,9	66,3	39,0	5
	W_6	18,8	18,6	20,5	21,4	34,0	50,2	22,7	6

Tabelle 8: Gewinnwahrscheinlichkeiten von Spieler A in Prozent für `wuerfel1.txt`, wenn alle Partien die im Deadock enden ignoriert werden.

Weitere Spielvariationen

In den folgenden Tabellen sind die Gewinnwahrscheinlichkeiten bei einigen Spielvariationen dargestellt. Zwar ändern sich die globalen Rangfolgen nicht gravierend, im Detail ergeben sich aber trotzdem Verschiebungen in der Größe einiger Prozent. Damit ist noch einmal verdeutlicht, dass es sehr wichtig ist, die Regeln für einen fairen Vergleich genau zu beachten.

		Würfel Sp. *B*							
		W_1	W_2	W_3	W_4	W_5	W_6	$\overline{p}_A$	Pl.
Würfel Sp. *A*	W_1	50,1	1,5	100,0	65,1	64,7	88,4	64,0	2
	W_2	98,5	49,8	100,0	98,6	96,7	99,0	98,6	1
	W_3	0,0	0,0	-	0,0	0,0	0,0	0,0	6
	W_4	35,1	1,5	100,0	50,3	51,5	79,3	53,5	3
	W_5	35,2	3,3	100,0	48,6	50,1	75,9	52,6	4
	W_6	11,7	1,1	100,0	20,8	23,9	49,8	31,5	5

Tabelle 9: Gewinnwahrscheinlichkeiten von Spieler *A* in Prozent bei Standardregeln für `wuerfel0.txt`.

		Würfel Sp. *B*							
		W_1	W_2	W_3	W_4	W_5	W_6	$\overline{p}_A$	Pl.
Würfel Sp. *A*	W_1	50,0	0,8	100,0	72,0	71,7	91,1	67,1	2
	W_2	99,3	49,9	100,0	99,5	98,8	99,5	99,4	1
	W_3	0,0	0,0	-	0,0	0,0	0,0	0,0	6
	W_4	28,1	0,6	100,0	50,2	53,3	80,7	52,5	3
	W_5	28,2	1,2	100,0	46,7	50,0	76,5	50,5	4
	W_6	8,9	0,5	100,0	19,3	23,2	49,8	30,4	5

Tabelle 10: Gewinnwahrscheinlichkeiten von Spieler *A* in Prozent mit der Zusatzregel Z3 für `wuerfel0.txt`. Spieler priorisieren es nun, die Steine der anderen zu schlagen.

		Würfel Sp. *B*							
		W_1	W_2	W_3	W_4	W_5	W_6	$\overline{p}_A$	Pl.
Würfel Sp. *A*	W_1	50,2	1,0	100,0	69,4	70,8	92,4	66,7	2
	W_2	99,0	49,8	100,0	99,3	98,4	99,6	99,3	1
	W_3	0,0	0,0	-	0,0	0,0	0,0	0,0	6
	W_4	30,3	0,7	100,0	50,2	54,0	83,3	53,7	3
	W_5	29,1	1,6	100,0	46,0	50,0	78,8	51,1	4
	W_6	7,5	0,4	100,0	16,9	20,8	50,0	29,1	5

Tabelle 11: Gewinnwahrscheinlichkeiten von Spieler *A* in Prozent mit der Zusatzregel Z4 für `wuerfel0.txt`. Die Spieler starten nun nicht mehr mit einer Figur auf dem Feld.

		Würfel Sp. *B*							
		W_1	W_2	W_3	W_4	W_5	W_6	$\overline{p}_A$	Pl.
Würfel Sp. *A*	W_1	50,1	1,5	100,0	60,8	56,5	84,1	60,6	2
	W_2	98,5	49,8	100,0	98,3	95,8	98,5	98,2	1
	W_3	0,0	0,0	-	0,0	0,0	0,0	0,0	6
	W_4	39,5	1,8	100,0	50,0	47,3	76,7	53,9	4
	W_5	43,3	4,2	100,0	52,6	49,9	75,8	55,2	3
	W_6	16,0	1,5	100,0	23,5	24,0	50,0	33,0	5

Tabelle 12: Gewinnwahrscheinlichkeiten von Spieler *A* in Prozent mit der Zusatzregel Z5 für `wuerfel0.txt`. Spieler ohne einen Stein auf dem Feld können nun dreimal versuchen, eine 6 zu würfeln.

4.3 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- $[-1]$ **Modellierung ungeeignet**
  Die Darstellung des Spielfeldes und der Würfel sollte so gewählt werden, dass Spiele zwischen 2 Personen gut simuliert werden können. So sollte zum Beispiel ein Vergleich der Positionen zweier Spielfiguren in konstanter Zeit möglich sein.
- $[-1]$ **Regeln nicht richtig umgesetzt**
  Das Spiel muss korrekt nach den Regeln des Herstellers und den ergänzenden BWINF-Regeln simuliert werden. Wenn mindestens eine wichtige Regel nicht beachtet wird, dann führt dies zu Punktabzug. Eine Regel ist wichtig, wenn es bei Nichteinhaltung zu signifikanten Abweichungen in den Gewinnwahrscheinlichkeiten kommt.
  Die Regeln lassen offen, was passiert, wenn kein Spieler mehr einen Zug machen kann, das Spiel also nicht endet, und sind in Sonderfällen nicht eindeutig. Wenn diese Fälle nicht beschrieben oder behandelt werden, führt das nicht zu Punktabzug.
- $[-1]$ **Simulation unzureichend**
  Es muss jeder Würfel mit jedem anderen verglichen werden. Mindestens eine dreistellige Anzahl an Simulationen je Würfelpaar ist gefordert.
- $[-1]$ **Verfahren bzw. Implementierung unnötig aufwendig oder ineffizient**
  Das Programm muss in angemessener Zeit für alle vorgegebenen Beispiele ein Ergebnis liefern können.
- $[-1]$ **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**
  Die Dokumentation soll Ergebnisse zu mindestens 3 der vorgegebenen Beispiele (`wuerfel0.txt` bis `wuerfel3.txt`) enthalten. Als Ergebnis werden sowohl eine Rangliste der Würfel (basierend auf ihrer gemittelten Gewinnwahrscheinlichkeit) als auch Tabellen mit Ergebnissen der paarweisen Vergleiche akzeptiert. Dabei sind ausnahmsweise auch Übersichten außerhalb der Dokumentation akzeptabel, sofern in der Dokumentation darauf verwiesen wird; eine Darstellung insbesondere der paarweisen Ergebnisse in einem Tabellendokument bietet sich bei dieser Aufgabe an. Da sehr viele Faktoren die Ergebnisse beeinflussen können, insbesondere die Anzahl der Simulationen pro Würfelpaar, können die dokumentierten Ergebnisse von den hier präsentierten etwas abweichen. Deutliche Abweichungen deuten allerdings auf Fehler hin.

Aufgabe 5: Marktwaage

5.1 Lösungsidee

Dieses Problem lässt sich mithilfe von *Backtracking* lösen. Für jedes einzelne Gewichtsstück gibt es 3 Möglichkeiten, es auf die Waage zu legen: links (mit dem zu wiegenden Objekt), rechts oder es wird nicht verwendet. Indem wir alle möglichen Kombinationen von Belegungen der Waage testen, können wir alle abmessbaren Gewichte bestimmen.

Unser Algorithmus liest die Gewichte zunächst ein und speichert in zwei Arrays jeweils das Gewicht bzw. die Anzahl der vorkommenden Gewichtsstücke mit diesem Gewicht. Dann verhält sich der Algorithmus wie folgt:

Algorithmus 4 Backtracking(Gewichte, Anzahl)

```
 1: Kombinationen = [ ]                                  ▷ Eine Liste von Kombinationen
 2: Kombination = [ ]
 3: FindeKombinationen(0, 0)
 4: Return Sortiere(Kombinationen)

 5: procedure FINDEKOMBINATIONEN(tiefe, summe)
 6:     if tiefe < Anzahl der unterschiedlichen Gewichte then
 7:         for i = − Anzahl[tiefe] to Anzahl[tiefe] do          ▷ Inklusive Grenzen
 8:             Füge (Gewicht[tiefe], i) zur Kombination hinzu
 9:             FindeKombinationen(tiefe + 1, summe + Gewichte[tiefe] * i)
10:             Entferne (Gewicht[tiefe], i) aus der Kombination
11:         end for
12:     else
13:         Füge (summe, Kombination) zu Kombinationen hinzu
14:     end if
15: end procedure
```

Für jedes Gewicht werden alle Möglichkeiten durchprobiert, von der Kombination, bei der alle Gewichtsstücke dieses Gewichts links zusammen mit dem zu wiegenden Objekt, bis zur Kombination, bei der alle auf der rechten Seite der Waage liegen. Liegt ein Gewichtsstück auf der rechten Seite der Waage, so wird sein Gewicht zur Gesamtsumme addiert, und liegt es auf der linken Seite, so wird sein Gewicht von der Gesamtsumme abgezogen. Hierzu werden die Gewichte jeweils mit einem Faktor multipliziert, der beschreibt, wie viele Gewichtsstücke dieses Gewichts auf eine Seite kommen, wobei negative Faktoren das Ablegen auf die linke Seite simulieren. Die Kombinationen werden dabei zusammen mit der jeweiligen Summe in einer Liste gespeichert.

Die Liste der Kombinationen wird nun nach aufsteigender Summe sortiert. Für jeden der gesuchten Werte wird eine Kombination ausgegeben, welche möglichst nah an dem gesuchten Wert liegt. Um diese Kombination effizient zu finden, wird eine Binärsuche in der Liste durchgeführt.

5.2 Beispielergebnis

Wenn wir die Gewichte $5\,\mathrm{g}, 5\,\mathrm{g}, 20\,\mathrm{g}$ haben, so findet unser Programm die folgenden Kombinationen und messbaren Gewichte:

Faktor 5 g	Faktor 20 g	Summe in g
-2	-1	-30
-1	-1	-25
0	-1	-20
1	-1	-15
2	-1	-10
-2	0	-10
-1	0	-5
0	0	0
1	0	5
2	0	10
-2	1	10
-1	1	15
0	1	20
1	1	25
2	1	30

Bei diesem Beispiel werden 15 verschiedene Kombinationen getestet. Wie die Anzahl der Kombinationen berechnet werden kann, wird in der Laufzeitanalyse behandelt. Wir stellen fest, dass es zu jeder negativen Summe die „gespiegelte“ Kombination mit entsprechender positiver Summe gibt, die negativen Summen also ignoriert werden können. Zu beachten ist auch, dass der Wert $10\,\mathrm{g}$ (und entsprechend $-10\,\mathrm{g}$) mehrere Lösungen hat. Wenn dies der Fall ist, muss eine Wahl für eine davon getroffen werden. Unser Programm entscheidet sich jeweils für die Lösung, welche die geringste Anzahl an Gewichtsstücken verwendet.

5.3 Laufzeitanalyse

Sei N die Anzahl der unterschiedlichen Gewichte und a_i die Anzahl der Gewichtsstücke mit dem i-ten Gewicht.

Da jedes Gewicht bis zu a_i mal auf der linken Seite, bis zu a_i mal auf der rechten Seite, oder gar nicht auf der Waage liegen kann, gibt es $2 \cdot a_i + 1$ Möglichkeiten pro Gewicht. Daher ist die Anzahl der Kombinationen

$$K = (2 \cdot a_1 + 1)(2 \cdot a_2 + 1) \cdots (2 \cdot a_N + 1).$$

Mit $a := \max(a_1, \ldots, a_n)$ gilt insbesondere

$$K \leq (2 \cdot a + 1)^N.$$

Ein Algorithmus, welcher für ein gegebenes Zielgewicht alle Kombinationen testet, hat also für jedes Zielgewicht eine exponentielle Laufzeit in Abhängigkeit der Anzahl der Gewichte:

$$T(a,N) = \mathcal{O}((2 \cdot a + 1)^N)$$

Ein Algorithmus, welcher hingegen zuerst alle Kombinationen berechnet und dann für jedes Zielgewicht mit einer Binärsuche nach der besten Kombination sucht, hat *insgesamt* die folgen-

de, bessere Laufzeit (wobei b die Anzahl der Zielgewichte bezeichnet):

$$\begin{aligned} T(a,b,N) &= \mathcal{O}((2\cdot a+1)^N + b\cdot\log((2\cdot a+1)^N)) \\ &= \mathcal{O}((2\cdot a+1)^N + b\cdot N\cdot\log(2\cdot a+1)) \end{aligned}$$

5.4 Dynamischer Ansatz

Ein dynamischer Ansatz (im Sinne *Dynamischer Programmierung*) zu diesem Problem würde in einer geeigneten Datenstruktur die Menge aller erreichbaren Summen abspeichern, die sich mit einer Teilmenge der Gewichtsstücke erzeugen lässt. Es wird begonnen mit der Menge, die nur den Wert 0 enthält, und dann werden auf alle Werte in dieser Menge nacheinander die Gewichtsstücke aufaddiert, bis alle Gewichtsstücke verwendet werden. Die Laufzeit dieses Ansatzes lässt sich nach oben abschätzen durch die Gesamtanzahl der Gewichte multipliziert mit der Summe aller Gewichte. Der Backtrackingansatz ist jedoch etwas effizienter bei den gegebenen Beispieldateien. Ein Satz von Gewichten, bei dem dieser Dynamische Ansatz effizienter ist, könnte folgendermaßen aussehen:

Gewicht	Anzahl
10	1
20	1
30	1
⋮	⋮
190	1
200	1

Die Anzahl der Kombinationen ist hier $3^{20} = 3486784401$ und die Summe aller Gewichte multipliziert mit der doppelten Anzahl aller Gewichte ist $20\cdot 2\cdot\frac{20\cdot 21}{2} = 8400$. Der dynamische Ansatz hätte hier eine bessere Laufzeit als der Backtrackingansatz. Er würde auch alle Beispieldateien in angemessener Zeit lösen.

5.5 Dokumentation der Beispiele

Es folgen die Ergebnisse für alle Beispieldateien der BWINF-Website. Die Ergebnisse werden in Tabellenform dargestellt, wobei für jede Zielsumme die Platzierung der einzelnen Gewichtsstücke mithilfe eines Faktors dargestellt wird. Ist der Faktor negativ, so werden diese Gewichtsstücke auf die linke Seite gelegt, ist er positiv, werden sie auf die rechte Seite gelegt. Alle Summen, die die Zielsumme exakt erreichen, werden **fett** geschrieben.

gewichtsstuecke0.txt

Anzahl der Kombinationen: 36015
Summe aller Gewichtsstücke: 9930
Exakte Lösungen: 993
Zeit in ms: 10

Zielsumme in *g*	10	50	100	500	1000	5000	Summe in *g*
10	1	0	0	0	0	0	**10**
20	2	0	0	0	0	0	**20**
30	-2	1	0	0	0	0	**30**
40	-1	1	0	0	0	0	**40**
50	0	1	0	0	0	0	**50**
60	1	1	0	0	0	0	**60**
70	2	1	0	0	0	0	**70**
80	-2	0	1	0	0	0	**80**
90	-1	0	1	0	0	0	**90**
100	0	0	1	0	0	0	**100**
110	1	0	1	0	0	0	**110**
120	2	0	1	0	0	0	**120**
130	-2	1	1	0	0	0	**130**
140	-1	1	1	0	0	0	**140**
150	0	1	1	0	0	0	**150**
160	1	1	1	0	0	0	**160**
170	2	1	1	0	0	0	**170**
180	-2	0	2	0	0	0	**180**
190	-1	0	2	0	0	0	**190**
200	0	0	2	0	0	0	**200**
210	1	0	2	0	0	0	**210**
220	2	0	2	0	0	0	**220**
230	-2	1	2	0	0	0	**230**
240	-1	1	2	0	0	0	**240**
250	0	1	2	0	0	0	**250**
260	1	1	2	0	0	0	**260**
270	2	1	2	0	0	0	**270**
280	-2	0	-2	1	0	0	**280**
290	-1	0	-2	1	0	0	**290**
300	0	0	-2	1	0	0	**300**
310	1	0	-2	1	0	0	**310**
320	2	0	-2	1	0	0	**320**
330	-2	-1	-1	1	0	0	**330**
340	-1	-1	-1	1	0	0	**340**
350	0	-1	-1	1	0	0	**350**
360	1	-1	-1	1	0	0	**360**
370	-3	0	-1	1	0	0	**370**
380	-2	0	-1	1	0	0	**380**
390	-1	0	-1	1	0	0	**390**
400	0	0	-1	1	0	0	**400**
410	1	0	-1	1	0	0	**410**
420	2	0	-1	1	0	0	**420**
430	-2	-1	0	1	0	0	**430**
440	-1	-1	0	1	0	0	**440**
450	0	-1	0	1	0	0	**450**
460	1	-1	0	1	0	0	**460**
470	-3	0	0	1	0	0	**470**
480	-2	0	0	1	0	0	**480**
490	-1	0	0	1	0	0	**490**
500	0	0	0	1	0	0	**500**

Zielsumme in *g*	10	50	100	500	1000	5000	Summe in *g*
9500	0	0	0	3	3	1	**9500**
9510	1	0	0	3	3	1	**9510**
9520	2	0	0	3	3	1	**9520**
9530	-2	1	0	3	3	1	**9530**
9540	-1	1	0	3	3	1	**9540**
9550	0	1	0	3	3	1	**9550**
9560	1	1	0	3	3	1	**9560**
9570	2	1	0	3	3	1	**9570**
9580	-2	0	1	3	3	1	**9580**
9590	-1	0	1	3	3	1	**9590**
9600	0	0	1	3	3	1	**9600**
9610	1	0	1	3	3	1	**9610**
9620	2	0	1	3	3	1	**9620**
9630	-2	1	1	3	3	1	**9630**
9640	-1	1	1	3	3	1	**9640**
9650	0	1	1	3	3	1	**9650**
9660	1	1	1	3	3	1	**9660**
9670	2	1	1	3	3	1	**9670**
9680	-2	0	2	3	3	1	**9680**
9690	-1	0	2	3	3	1	**9690**
9700	0	0	2	3	3	1	**9700**
9710	1	0	2	3	3	1	**9710**
9720	2	0	2	3	3	1	**9720**
9730	-2	1	2	3	3	1	**9730**
9740	-1	1	2	3	3	1	**9740**
9750	0	1	2	3	3	1	**9750**
9760	1	1	2	3	3	1	**9760**
9770	2	1	2	3	3	1	**9770**
9780	-2	0	3	3	3	1	**9780**
9790	-1	0	3	3	3	1	**9790**
9800	0	0	3	3	3	1	**9800**
9810	1	0	3	3	3	1	**9810**
9820	2	0	3	3	3	1	**9820**
9830	-2	1	3	3	3	1	**9830**
9840	-1	1	3	3	3	1	**9840**
9850	0	1	3	3	3	1	**9850**
9860	1	1	3	3	3	1	**9860**
9870	2	1	3	3	3	1	**9870**
9880	-2	2	3	3	3	1	**9880**
9890	-1	2	3	3	3	1	**9890**
9900	0	2	3	3	3	1	**9900**
9910	1	2	3	3	3	1	**9910**
9920	2	2	3	3	3	1	**9920**
9930	3	2	3	3	3	1	**9930**
9940	3	2	3	3	3	1	9930
9950	3	2	3	3	3	1	9930
9960	3	2	3	3	3	1	9930
9970	3	2	3	3	3	1	9930
9980	3	2	3	3	3	1	9930
9990	3	2	3	3	3	1	9930
10000	3	2	3	3	3	1	9930

gewichtsstuecke1.txt

Anzahl der Kombinationen: 7007
Summe aller Gewichtsstücke: 12733
Exakte Lösungen: 300
Zeit in ms: 3

Zielsumme in *g*	42	127	371	2000	Summe in *g*
10	0	3	-1	0	**10**
20	-3	0	-5	1	19
30	-3	3	-6	1	29
40	1	0	0	0	42
50	-2	-3	-4	1	51
60	1	-1	-5	1	**60**
70	1	2	-6	1	**70**
80	2	0	0	0	84
90	-1	-3	-4	1	93
100	-3	0	6	-1	**100**
110	-3	3	5	-1	**110**
120	-3	-1	1	0	118
130	-3	2	0	0	128
140	1	-1	6	-1	141
150	1	2	5	-1	151
160	-2	-1	1	0	**160**
170	-2	2	0	0	**170**
180	-2	-2	-4	1	178
190	-2	1	-5	1	188
200	2	-2	1	0	201
210	2	1	0	0	211
220	-1	-2	-4	1	**220**
230	-1	1	-5	1	**230**
240	2	3	-6	1	239
250	3	1	0	0	253
260	3	-3	-4	1	261
270	-2	1	6	-1	269
280	3	3	-6	1	281
290	-2	0	1	0	287
300	-2	3	0	0	297
310	2	0	6	-1	**310**
320	2	3	5	-1	**320**
330	-1	0	1	0	329
340	-1	3	0	0	339
350	3	0	6	-1	352
360	0	-3	2	0	361
370	3	-1	1	0	**370**
380	3	2	0	0	**380**
390	-3	0	-4	1	**390**
400	-3	3	-5	1	**400**
410	1	0	1	0	413
420	-2	-3	-3	1	422
430	1	-1	-4	1	431
440	1	2	-5	1	441
450	-1	-2	2	0	446
460	-1	1	1	0	456
470	2	-1	-4	1	473
480	0	2	6	-1	**480**

490	-3	-1	2	0	489
500	-3	2	1	0	499

Zielsumme in *g*	42	127	371	2000	Summe in *g*
9500	3	-2	-1	5	9501
9510	3	1	-2	5	9511
9520	-2	-2	5	4	9517
9530	-2	1	4	4	9527
9540	1	3	3	4	9536
9550	1	2	-2	5	9554
9560	-1	-2	5	4	9559
9570	-1	1	4	4	9569
9580	2	3	3	4	9578
9590	-1	0	-1	5	9587
9600	3	-3	5	4	**9600**
9610	3	0	4	4	**9610**
9620	-3	-2	0	5	**9620**
9630	-3	1	-1	5	**9630**
9640	0	3	-2	5	9639
9650	1	1	4	4	9653
9660	1	-3	0	5	9661
9670	1	0	-1	5	9671
9680	1	3	-2	5	9681
9690	-1	-1	5	4	9686
9700	2	-3	0	5	9703
9710	2	0	-1	5	9713
9720	-3	-3	6	4	9719
9730	-3	0	5	4	9729
9740	-3	3	4	4	9739
9750	-3	-1	0	5	9747
9760	-2	-3	6	4	9761
9770	1	-1	5	4	**9770**
9780	1	2	4	4	**9780**
9790	-2	-1	0	5	9789
9800	-2	2	-1	5	9799
9810	2	-1	5	4	9812
9820	2	2	4	4	9822
9830	2	-2	0	5	**9830**
9840	2	1	-1	5	**9840**
9850	-3	-2	6	4	9846
9860	-3	1	5	4	9856
9870	3	-2	0	5	9872
9880	3	1	-1	5	9882
9890	-2	-2	6	4	9888
9900	-2	1	5	4	9898
9910	1	3	4	4	9907
9920	-2	0	0	5	9916
9930	-1	-2	6	4	**9930**
9940	-1	1	5	4	**9940**
9950	2	3	4	4	9949
9960	-1	0	0	5	9958
9970	3	-3	6	4	9971

9980	3	0	5	4	9981
9990	0	-3	1	5	**9990**
10000	0	0	0	5	**10000**

gewichtsstuecke2.txt

Anzahl der Kombinationen: 59049
Summe aller Gewichtsstücke: 10230
Exakte Lösungen: 1000
Zeit in ms: 2

Zielsumme in *g*	10	20	40	80	160	320	640	1280	2560	5120	Summe in *g*
10	1	0	0	0	0	0	0	0	0	0	**10**
20	0	1	0	0	0	0	0	0	0	0	**20**
30	-1	0	1	0	0	0	0	0	0	0	**30**
40	0	0	1	0	0	0	0	0	0	0	**40**
50	1	0	1	0	0	0	0	0	0	0	**50**
60	0	-1	0	1	0	0	0	0	0	0	**60**
70	-1	0	0	1	0	0	0	0	0	0	**70**
80	0	0	0	1	0	0	0	0	0	0	**80**
90	1	0	0	1	0	0	0	0	0	0	**90**
100	0	1	0	1	0	0	0	0	0	0	**100**
110	-1	0	-1	0	1	0	0	0	0	0	**110**
120	0	0	-1	0	1	0	0	0	0	0	**120**
130	-1	-1	0	0	1	0	0	0	0	0	**130**
140	0	-1	0	0	1	0	0	0	0	0	**140**
150	-1	0	0	0	1	0	0	0	0	0	**150**
160	0	0	0	0	1	0	0	0	0	0	**160**
170	1	0	0	0	1	0	0	0	0	0	**170**
180	0	1	0	0	1	0	0	0	0	0	**180**
190	-1	0	1	0	1	0	0	0	0	0	**190**
200	0	0	1	0	1	0	0	0	0	0	**200**
210	1	0	1	0	1	0	0	0	0	0	**210**
220	0	-1	0	-1	0	1	0	0	0	0	**220**
230	-1	0	0	-1	0	1	0	0	0	0	**230**
240	0	0	0	-1	0	1	0	0	0	0	**240**
250	1	0	0	-1	0	1	0	0	0	0	**250**
260	0	-1	-1	0	0	1	0	0	0	0	**260**
270	-1	0	-1	0	0	1	0	0	0	0	**270**
280	0	0	-1	0	0	1	0	0	0	0	**280**
290	-1	-1	0	0	0	1	0	0	0	0	**290**
300	0	-1	0	0	0	1	0	0	0	0	**300**
310	-1	0	0	0	0	1	0	0	0	0	**310**
320	0	0	0	0	0	1	0	0	0	0	**320**
330	1	0	0	0	0	1	0	0	0	0	**330**
340	0	1	0	0	0	1	0	0	0	0	**340**
350	-1	0	1	0	0	1	0	0	0	0	**350**
360	0	0	1	0	0	1	0	0	0	0	**360**
370	1	0	1	0	0	1	0	0	0	0	**370**
380	0	-1	0	1	0	1	0	0	0	0	**380**
390	-1	0	0	1	0	1	0	0	0	0	**390**
400	0	0	0	1	0	1	0	0	0	0	**400**

410	1	0	0	1	0	1	0	0	0	0	**410**
420	0	1	0	1	0	1	0	0	0	0	**420**
430	-1	0	-1	0	-1	0	1	0	0	0	**430**
440	0	0	-1	0	-1	0	1	0	0	0	**440**
450	-1	-1	0	0	-1	0	1	0	0	0	**450**
460	0	-1	0	0	-1	0	1	0	0	0	**460**
470	-1	0	0	0	-1	0	1	0	0	0	**470**
480	0	0	0	0	-1	0	1	0	0	0	**480**
490	1	0	0	0	-1	0	1	0	0	0	**490**
500	0	1	0	0	-1	0	1	0	0	0	**500**

Zielsumme in *g*	10	20	40	80	160	320	640	1280	2560	5120	Summe in *g*
9500	0	-1	0	-1	0	0	1	1	1	1	**9500**
9510	-1	0	0	-1	0	0	1	1	1	1	**9510**
9520	0	0	0	-1	0	0	1	1	1	1	**9520**
9530	1	0	0	-1	0	0	1	1	1	1	**9530**
9540	0	-1	-1	0	0	0	1	1	1	1	**9540**
9550	-1	0	-1	0	0	0	1	1	1	1	**9550**
9560	0	0	-1	0	0	0	1	1	1	1	**9560**
9570	-1	-1	0	0	0	0	1	1	1	1	**9570**
9580	0	-1	0	0	0	0	1	1	1	1	**9580**
9590	-1	0	0	0	0	0	1	1	1	1	**9590**
9600	0	0	0	0	0	0	1	1	1	1	**9600**
9610	1	0	0	0	0	0	1	1	1	1	**9610**
9620	0	1	0	0	0	0	1	1	1	1	**9620**
9630	-1	0	1	0	0	0	1	1	1	1	**9630**
9640	0	0	1	0	0	0	1	1	1	1	**9640**
9650	1	0	1	0	0	0	1	1	1	1	**9650**
9660	0	-1	0	1	0	0	1	1	1	1	**9660**
9670	-1	0	0	1	0	0	1	1	1	1	**9670**
9680	0	0	0	1	0	0	1	1	1	1	**9680**
9690	1	0	0	1	0	0	1	1	1	1	**9690**
9700	0	1	0	1	0	0	1	1	1	1	**9700**
9710	-1	0	-1	0	1	0	1	1	1	1	**9710**
9720	0	0	-1	0	1	0	1	1	1	1	**9720**
9730	-1	-1	0	0	1	0	1	1	1	1	**9730**
9740	0	-1	0	0	1	0	1	1	1	1	**9740**
9750	-1	0	0	0	1	0	1	1	1	1	**9750**
9760	0	0	0	0	1	0	1	1	1	1	**9760**
9770	1	0	0	0	1	0	1	1	1	1	**9770**
9780	0	1	0	0	1	0	1	1	1	1	**9780**
9790	-1	0	1	0	1	0	1	1	1	1	**9790**
9800	0	0	1	0	1	0	1	1	1	1	**9800**
9810	1	0	1	0	1	0	1	1	1	1	**9810**
9820	0	-1	0	-1	0	1	1	1	1	1	**9820**
9830	-1	0	0	-1	0	1	1	1	1	1	**9830**
9840	0	0	0	-1	0	1	1	1	1	1	**9840**
9850	1	0	0	-1	0	1	1	1	1	1	**9850**
9860	0	-1	-1	0	0	1	1	1	1	1	**9860**
9870	-1	0	-1	0	0	1	1	1	1	1	**9870**
9880	0	0	-1	0	0	1	1	1	1	1	**9880**
9890	-1	-1	0	0	0	1	1	1	1	1	**9890**

9900	0	-1	0	0	0	1	1	1	1	1	**9900**
9910	-1	0	0	0	0	1	1	1	1	1	**9910**
9920	0	0	0	0	0	1	1	1	1	1	**9920**
9930	1	0	0	0	0	1	1	1	1	1	**9930**
9940	0	1	0	0	0	1	1	1	1	1	**9940**
9950	-1	0	1	0	0	1	1	1	1	1	**9950**
9960	0	0	1	0	0	1	1	1	1	1	**9960**
9970	1	0	1	0	0	1	1	1	1	1	**9970**
9980	0	-1	0	1	0	1	1	1	1	1	**9980**
9990	-1	0	0	1	0	1	1	1	1	1	**9990**
10000	0	0	0	1	0	1	1	1	1	1	**10000**

gewichtsstuecke3.txt

Anzahl der Kombinationen: 2187
Summe aller Gewichtsstücke: 10930
Exakte Lösungen: 1000
Zeit in ms: 2

Zielsumme in *g*	10	30	90	270	810	2430	7290	Summe in *g*
10	1	0	0	0	0	0	0	**10**
20	-1	1	0	0	0	0	0	**20**
30	0	1	0	0	0	0	0	**30**
40	1	1	0	0	0	0	0	**40**
50	-1	-1	1	0	0	0	0	**50**
60	0	-1	1	0	0	0	0	**60**
70	1	-1	1	0	0	0	0	**70**
80	-1	0	1	0	0	0	0	**80**
90	0	0	1	0	0	0	0	**90**
100	1	0	1	0	0	0	0	**100**
110	-1	1	1	0	0	0	0	**110**
120	0	1	1	0	0	0	0	**120**
130	1	1	1	0	0	0	0	**130**
140	-1	-1	-1	1	0	0	0	**140**
150	0	-1	-1	1	0	0	0	**150**
160	1	-1	-1	1	0	0	0	**160**
170	-1	0	-1	1	0	0	0	**170**
180	0	0	-1	1	0	0	0	**180**
190	1	0	-1	1	0	0	0	**190**
200	-1	1	-1	1	0	0	0	**200**
210	0	1	-1	1	0	0	0	**210**
220	1	1	-1	1	0	0	0	**220**
230	-1	-1	0	1	0	0	0	**230**
240	0	-1	0	1	0	0	0	**240**
250	1	-1	0	1	0	0	0	**250**
260	-1	0	0	1	0	0	0	**260**
270	0	0	0	1	0	0	0	**270**
280	1	0	0	1	0	0	0	**280**
290	-1	1	0	1	0	0	0	**290**
300	0	1	0	1	0	0	0	**300**
310	1	1	0	1	0	0	0	**310**
320	-1	-1	1	1	0	0	0	**320**

330	0	-1	1	1	0	0	0	**330**
340	1	-1	1	1	0	0	0	**340**
350	-1	0	1	1	0	0	0	**350**
360	0	0	1	1	0	0	0	**360**
370	1	0	1	1	0	0	0	**370**
380	-1	1	1	1	0	0	0	**380**
390	0	1	1	1	0	0	0	**390**
400	1	1	1	1	0	0	0	**400**
410	-1	-1	-1	-1	1	0	0	**410**
420	0	-1	-1	-1	1	0	0	**420**
430	1	-1	-1	-1	1	0	0	**430**
440	-1	0	-1	-1	1	0	0	**440**
450	0	0	-1	-1	1	0	0	**450**
460	1	0	-1	-1	1	0	0	**460**
470	-1	1	-1	-1	1	0	0	**470**
480	0	1	-1	-1	1	0	0	**480**
490	1	1	-1	-1	1	0	0	**490**
500	-1	-1	0	-1	1	0	0	**500**

Zielsumme in *g*	10	30	90	270	810	2430	7290	Summe in *g*
9500	-1	-1	1	-1	0	1	1	**9500**
9510	0	-1	1	-1	0	1	1	**9510**
9520	1	-1	1	-1	0	1	1	**9520**
9530	-1	0	1	-1	0	1	1	**9530**
9540	0	0	1	-1	0	1	1	**9540**
9550	1	0	1	-1	0	1	1	**9550**
9560	-1	1	1	-1	0	1	1	**9560**
9570	0	1	1	-1	0	1	1	**9570**
9580	1	1	1	-1	0	1	1	**9580**
9590	-1	-1	-1	0	0	1	1	**9590**
9600	0	-1	-1	0	0	1	1	**9600**
9610	1	-1	-1	0	0	1	1	**9610**
9620	-1	0	-1	0	0	1	1	**9620**
9630	0	0	-1	0	0	1	1	**9630**
9640	1	0	-1	0	0	1	1	**9640**
9650	-1	1	-1	0	0	1	1	**9650**
9660	0	1	-1	0	0	1	1	**9660**
9670	1	1	-1	0	0	1	1	**9670**
9680	-1	-1	0	0	0	1	1	**9680**
9690	0	-1	0	0	0	1	1	**9690**
9700	1	-1	0	0	0	1	1	**9700**
9710	-1	0	0	0	0	1	1	**9710**
9720	0	0	0	0	0	1	1	**9720**
9730	1	0	0	0	0	1	1	**9730**
9740	-1	1	0	0	0	1	1	**9740**
9750	0	1	0	0	0	1	1	**9750**
9760	1	1	0	0	0	1	1	**9760**
9770	-1	-1	1	0	0	1	1	**9770**
9780	0	-1	1	0	0	1	1	**9780**
9790	1	-1	1	0	0	1	1	**9790**
9800	-1	0	1	0	0	1	1	**9800**
9810	0	0	1	0	0	1	1	**9810**

9820	1	0	1	0	0	1	1	**9820**
9830	-1	1	1	0	0	1	1	**9830**
9840	0	1	1	0	0	1	1	**9840**
9850	1	1	1	0	0	1	1	**9850**
9860	-1	-1	-1	1	0	1	1	**9860**
9870	0	-1	-1	1	0	1	1	**9870**
9880	1	-1	-1	1	0	1	1	**9880**
9890	-1	0	-1	1	0	1	1	**9890**
9900	0	0	-1	1	0	1	1	**9900**
9910	1	0	-1	1	0	1	1	**9910**
9920	-1	1	-1	1	0	1	1	**9920**
9930	0	1	-1	1	0	1	1	**9930**
9940	1	1	-1	1	0	1	1	**9940**
9950	-1	-1	0	1	0	1	1	**9950**
9960	0	-1	0	1	0	1	1	**9960**
9970	1	-1	0	1	0	1	1	**9970**
9980	-1	0	0	1	0	1	1	**9980**
9990	0	0	0	1	0	1	1	**9990**
10000	1	0	0	1	0	1	1	**10000**

gewichtsstuecke4.txt

Anzahl der Kombinationen: 27783
Summe aller Gewichtsstücke: 10079
Exakte Lösungen: 610
Zeit in ms: 2

Zielsumme in *g*	5	21	29	259	287	399	2993	Summe in *g*
10	-1	-2	1	-1	1	0	0	**10**
20	0	1	-1	-1	1	0	0	**20**
30	0	0	2	1	-1	0	0	**30**
40	-1	-2	3	0	0	0	0	**40**
50	0	1	1	0	0	0	0	**50**
60	-1	-1	2	-1	1	0	0	**60**
70	0	-2	0	0	-1	1	0	**70**
80	0	1	3	1	-1	0	0	**80**
90	0	-1	-1	-1	0	1	0	**90**
100	0	2	2	0	0	0	0	**100**
110	-1	0	3	-1	1	0	0	**110**
120	0	-1	1	0	-1	1	0	**120**
130	0	-2	-3	1	0	0	0	**130**
140	0	0	0	-1	0	1	0	**140**
150	-1	-1	1	1	1	-1	0	**150**
160	0	2	-1	1	1	-1	0	**160**
170	0	0	2	0	-1	1	0	**170**
180	0	-1	-2	1	0	0	0	**180**
190	0	1	1	-1	0	1	0	**190**
200	0	0	-3	0	1	0	0	**200**
210	0	3	0	1	1	-1	0	**210**
220	-1	-3	1	1	0	0	0	**220**
230	0	0	-1	1	0	0	0	**230**
240	-1	-2	0	0	1	0	0	**240**

250	0	1	-2	0	1	0	0	**250**
260	1	1	3	1	1	-1	0	**260**
270	-1	-2	2	1	0	0	0	**270**
280	0	1	0	1	0	0	0	**280**
290	-1	-1	1	0	1	0	0	**290**
300	0	2	-1	0	1	0	0	**300**
310	1	1	-3	1	-1	1	0	**310**
320	0	-1	-2	0	0	1	0	**320**
330	0	2	1	1	0	0	0	**330**
340	-1	0	2	0	1	0	0	**340**
350	0	-1	0	1	-1	1	0	**350**
360	-1	-3	1	0	0	1	0	**360**
370	0	0	-1	0	0	1	0	**370**
380	-1	-2	0	-1	1	1	0	**380**
390	-1	1	3	0	1	0	0	**390**
400	0	0	1	1	-1	1	0	**400**
410	-1	-2	2	0	0	1	0	**410**
420	0	1	0	0	0	1	0	**420**
430	-1	-1	1	-1	1	1	0	**430**
440	0	2	-1	-1	1	1	0	**440**
450	0	1	2	1	-1	1	0	**450**
460	-1	-1	3	0	0	1	0	**460**
470	0	2	1	0	0	1	0	**470**
480	-1	0	2	-1	1	1	0	**480**
490	0	3	0	-1	1	1	0	**490**
500	0	2	3	1	-1	1	0	**500**

Zielsumme in *g*	5	21	29	259	287	399	2993	Summe in *g*
9500	0	3	3	1	-1	1	3	**9500**
9510	-1	-3	-3	0	1	1	3	**9510**
9520	-1	0	0	1	1	0	3	**9520**
9530	1	0	0	1	1	0	3	**9530**
9540	-1	-3	-1	1	0	1	3	**9540**
9550	0	0	-3	1	0	1	3	**9550**
9560	-1	-2	-2	0	1	1	3	**9560**
9570	-1	1	1	1	1	0	3	**9570**
9580	1	1	1	1	1	0	3	**9580**
9590	-1	-2	0	1	0	1	3	**9590**
9600	0	1	-2	1	0	1	3	**9600**
9610	-1	-1	-1	0	1	1	3	**9610**
9620	1	-1	-1	0	1	1	3	**9620**
9630	1	2	2	1	1	0	3	**9630**
9640	-1	-1	1	1	0	1	3	**9640**
9650	0	2	-1	1	0	1	3	**9650**
9660	-1	0	0	0	1	1	3	**9660**
9670	1	0	0	0	1	1	3	**9670**
9680	1	3	3	1	1	0	3	**9680**
9690	-1	0	2	1	0	1	3	**9690**
9700	0	3	0	1	0	1	3	**9700**
9710	-1	1	1	0	1	1	3	**9710**
9720	1	1	1	0	1	1	3	**9720**
9730	0	3	1	1	0	1	3	9729

9740	-1	1	3	1	0	1	3	**9740**
9750	1	1	3	1	0	1	3	**9750**
9760	-1	2	2	0	1	1	3	**9760**
9770	1	2	2	0	1	1	3	**9770**
9780	1	-3	-3	1	1	1	3	9779
9790	-1	-2	-3	1	1	1	3	**9790**
9800	1	-2	-3	1	1	1	3	**9800**
9810	-1	3	3	0	1	1	3	**9810**
9820	1	3	3	0	1	1	3	**9820**
9830	1	-2	-2	1	1	1	3	9829
9840	-1	-1	-2	1	1	1	3	**9840**
9850	1	-1	-2	1	1	1	3	**9850**
9860	-1	0	-2	1	1	1	3	9861
9870	-1	-1	-1	1	1	1	3	9869
9880	1	-1	-1	1	1	1	3	9879
9890	-1	0	-1	1	1	1	3	**9890**
9900	1	0	-1	1	1	1	3	**9900**
9910	-1	1	-1	1	1	1	3	9911
9920	-1	0	0	1	1	1	3	9919
9930	1	0	0	1	1	1	3	9929
9940	-1	1	0	1	1	1	3	**9940**
9950	1	1	0	1	1	1	3	**9950**
9960	-1	2	0	1	1	1	3	9961
9970	-1	1	1	1	1	1	3	9969
9980	1	1	1	1	1	1	3	9979
9990	-1	2	1	1	1	1	3	**9990**
10000	1	2	1	1	1	1	3	**10000**

gewichtsstuecke5.txt

Anzahl der Kombinationen: 101269035
Summe aller Gewichtsstücke: 4636393543
Exakte Lösungen: 84
Zeit in ms: 2038
Um die Breite der letzten Tabelle zu verringern werden folgende Buchstaben als Abkürzung der Gewichtsstücke verwendet.

Gewicht	Anzahl	Zeichen
11	1	*a*
99480	1	*b*
99511	1	*c*
299836	2	*d*
599761	1	*e*
4497786	3	*f*
1499171	1	*g*
10499654	4	*h*
41999427	1	*i*
94499810	3	*j*
283501867	1	*k*
661499326	3	*l*
1984505261	1	*m*

Zielsumme in g	a	b	c	d	e	f	g	h	i	j	k	l	m	Summe in g
10	1	0	0	0	0	0	0	0	0	0	0	0	0	11
20	-1	-1	1	0	0	0	0	0	0	0	0	0	0	**20**
30	0	-1	1	0	0	0	0	0	0	0	0	0	0	31
40	1	-1	1	0	0	0	0	0	0	0	0	0	0	42
50	-1	1	-1	-2	1	0	0	0	0	0	0	0	0	47
60	0	1	-1	-2	1	0	0	0	0	0	0	0	0	58
70	1	1	-1	-2	1	0	0	0	0	0	0	0	0	69
80	-1	0	0	-2	1	0	0	0	0	0	0	0	0	78
90	0	0	0	-2	1	0	0	0	0	0	0	0	0	89
100	1	0	0	-2	1	0	0	0	0	0	0	0	0	**100**
110	-1	-1	1	-2	1	0	0	0	0	0	0	0	0	109
120	0	-1	1	-2	1	0	0	0	0	0	0	0	0	**120**
130	1	-1	1	-2	1	0	0	0	0	0	0	0	0	131
140	0	1	-1	2	-1	2	1	4	1	-1	0	-3	1	139
150	1	1	-1	2	-1	2	1	4	1	-1	0	-3	1	**150**
160	-1	0	0	2	-1	2	1	4	1	-1	0	-3	1	159
170	0	0	0	2	-1	2	1	4	1	-1	0	-3	1	**170**
180	1	0	0	2	-1	2	1	4	1	-1	0	-3	1	181
190	-1	-1	1	2	-1	2	1	4	1	-1	0	-3	1	**190**
200	0	-1	1	2	-1	2	1	4	1	-1	0	-3	1	201
210	1	-1	1	2	-1	2	1	4	1	-1	0	-3	1	212
220	-1	1	-1	0	0	2	1	4	1	-1	0	-3	1	217
230	0	1	-1	0	0	2	1	4	1	-1	0	-3	1	228
240	1	1	-1	0	0	2	1	4	1	-1	0	-3	1	239
250	-1	0	0	0	0	2	1	4	1	-1	0	-3	1	248
260	0	0	0	0	0	2	1	4	1	-1	0	-3	1	259
270	1	0	0	0	0	2	1	4	1	-1	0	-3	1	**270**
280	-1	-1	1	0	0	2	1	4	1	-1	0	-3	1	279
290	0	-1	1	0	0	2	1	4	1	-1	0	-3	1	**290**
300	1	-1	1	0	0	2	1	4	1	-1	0	-3	1	301
310	-1	1	-1	-2	1	2	1	4	1	-1	0	-3	1	306
320	0	1	-1	-2	1	2	1	4	1	-1	0	-3	1	317
330	1	1	-1	-2	1	2	1	4	1	-1	0	-3	1	328
340	-1	0	0	-2	1	2	1	4	1	-1	0	-3	1	337
350	0	0	0	-2	1	2	1	4	1	-1	0	-3	1	348
360	1	0	0	-2	1	2	1	4	1	-1	0	-3	1	359
370	-1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	368
380	0	-1	1	-2	1	2	1	4	1	-1	0	-3	1	379
390	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	**390**
400	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
410	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
420	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
430	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
440	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
450	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
460	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
470	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
480	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
490	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390
500	1	-1	1	-2	1	2	1	4	1	-1	0	-3	1	390

Zielsumme in g	a	b	c	d	e	f	g	h	i	j	k	l	m	Summe in g
9500	1	-1	1	0	0	-2	-1	-4	-1	-2	1	0	0	9503
9510	-1	1	-1	-2	1	-2	-1	-4	-1	-2	1	0	0	9508
9520	0	1	-1	-2	1	-2	-1	-4	-1	-2	1	0	0	9519
9530	1	1	-1	-2	1	-2	-1	-4	-1	-2	1	0	0	**9530**
9540	-1	0	0	-2	1	-2	-1	-4	-1	-2	1	0	0	9539
9550	0	0	0	-2	1	-2	-1	-4	-1	-2	1	0	0	**9550**
9560	1	0	0	-2	1	-2	-1	-4	-1	-2	1	0	0	9561
9570	-1	-1	1	-2	1	-2	-1	-4	-1	-2	1	0	0	**9570**
9580	0	-1	1	-2	1	-2	-1	-4	-1	-2	1	0	0	9581
9590	-1	1	-1	2	-1	0	0	0	0	-3	1	-3	1	9589
9600	0	1	-1	2	-1	0	0	0	0	-3	1	-3	1	**9600**
9610	1	1	-1	2	-1	0	0	0	0	-3	1	-3	1	9611
9620	-1	0	0	2	-1	0	0	0	0	-3	1	-3	1	**9620**
9630	0	0	0	2	-1	0	0	0	0	-3	1	-3	1	9631
9640	1	0	0	2	-1	0	0	0	0	-3	1	-3	1	9642
9650	-1	-1	1	2	-1	0	0	0	0	-3	1	-3	1	9651
9660	0	-1	1	2	-1	0	0	0	0	-3	1	-3	1	9662
9670	0	0	0	2	-1	-2	-1	1	0	3	-1	-3	1	9668
9680	1	0	0	2	-1	-2	-1	1	0	3	-1	-3	1	9679
9690	0	1	-1	0	0	0	0	0	0	-3	1	-3	1	9689
9700	1	1	-1	0	0	0	0	0	0	-3	1	-3	1	**9700**
9710	1	-1	1	2	-1	-2	-1	1	0	3	-1	-3	1	**9710**
9720	0	0	0	0	0	0	0	0	0	-3	1	-3	1	**9720**
9730	1	0	0	0	0	0	0	0	0	-3	1	-3	1	9731
9740	-1	-1	1	0	0	0	0	0	0	-3	1	-3	1	**9740**
9750	0	-1	1	0	0	0	0	0	0	-3	1	-3	1	9751
9760	1	-1	1	0	0	0	0	0	0	-3	1	-3	1	9762
9770	1	0	0	0	0	-2	-1	1	0	3	-1	-3	1	9768
9780	0	1	-1	-2	1	0	0	0	0	-3	1	-3	1	9778
9790	1	1	-1	-2	1	0	0	0	0	-3	1	-3	1	9789
9800	1	-1	1	0	0	-2	-1	1	0	3	-1	-3	1	9799
9810	0	0	0	-2	1	0	0	0	0	-3	1	-3	1	9809
9820	1	0	0	-2	1	0	0	0	0	-3	1	-3	1	**9820**
9830	-1	-1	1	-2	1	0	0	0	0	-3	1	-3	1	9829
9840	0	-1	1	-2	1	0	0	0	0	-3	1	-3	1	**9840**
9850	1	-1	1	-2	1	0	0	0	0	-3	1	-3	1	9851
9860	1	0	0	-2	1	-2	-1	1	0	3	-1	-3	1	9857
9870	-1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9866
9880	0	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9877
9890	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9900	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9910	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9920	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9930	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9940	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9950	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9960	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9970	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9980	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
9990	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888
10000	1	-1	1	-2	1	-2	-1	1	0	3	-1	-3	1	9888

5.6 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- [−1] **Lösungsverfahren fehlerhaft**
  Das Lösungsverfahren muss stets eine korrekte Lösung, also eine Verteilung der Gewichtsstücke auf die Waage, mit der das Zielgewicht gemessen werden kann, finden, wenn es eine gibt. Ansonsten muss es herausfinden, wie groß die minimale Differenz eines messbaren Gewichts zum Zielgewicht ist.
- [−1] **Gewichte unnötig kombiniert**
  Es wird erwartet, dass alle Gewichtsstücke des gleichen Gewichts als ununterscheidbar behandelt werden. Weiterhin sollten unnötige Kombinationen (wie 10 g links und 10 g rechts) von vorn herein ausgeschlossen werden.
- [−1] **Verfahren bzw. Implementierung unnötig aufwendig / ineffizient**
  Das Verfahren sollte für eine Liste von Gewichtsstücken und alle Zielgewichte (in 10 g-Schritten zwischen 10 g und 10 kg) in angemessener Zeit je eine Lösung (also eine passende Platzierung von Gewichtsstücken auf der Waage) bzw. die Differenz zum nächstgelegenen messbaren Gewicht bestimmen. Dazu sollten bereits berechnete Gewichtskombinationen gesichert bzw. wiederverwendet werden – ob durch Vorberechnung wie in der Beispiellösung oder auf andere Art und Weise. Die Vorberechnung ist nur sinnvoll, wenn auf die so ermittelten Ergebnisse auch effizient zugegriffen wird. Das bedeutet insbesondere, dass es auch dann Abzug gibt, wenn alle Kombinationen in einer Liste gespeichert werden, in der die Zielsummen dann später linear gesucht werden. Ein erneutes Testen aller möglichen Kombinationen bei jedem Zielgewicht ist nicht akzeptabel.
- [−1] **Ergebnisse schlecht nachvollziehbar**
  Für jedes Beispiel ist für alle Zielgewichte (in 10 g-Schritten zwischen 10 g und 500 g sowie zwischen 9500 g und 10 kg) eine Verteilung von Gewichtsstücken bzw. die Differenz zum nächstgelegenen messbaren Gewicht auszugeben. Wenn die in der Dokumentation enthaltenen Listen knapper ausfallen, ist das akzeptabel, solange man sich insgesamt gut einen Eindruck vom Funktionieren des Verfahrens verschaffen kann. Die Ausgabe des nächstgelegenen messbaren Gewichts statt der Differenz zum Zielgewicht ist ausreichend. Die Ausgabe einer Verteilung der Gewichtsstücke ist nur gefordert, wenn das Zielgewicht exakt erreicht werden kann.
- [−1] **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**
  Die Dokumentation soll Ergebnisse zu mindestens 3 der vorgegebenen Beispiele (`gewichtsstuecke0.txt` bis `gewichtsstuecke5.txt`) enthalten. Die Ergebnisse müssen in der Dokumentation nicht vollständig enthalten sein, Auszüge genügen. Ausnahmsweise sind auch Übersichten bzw. ausreichend viele Ergebnis-Listings außerhalb der Dokumentation akzeptabel, sofern in der Dokumentation darauf verwiesen wird.