22. Bundeswettbewerb Informatik, 2. Runde

Lösungshinweise und Bewertungskriterien

Allgemeines

Es erfüllt die Bewerterinnen und Bewerter immer wieder mit Erstaunen und oft auch Bewunderung, wieviel an Ideen und Wissen, aber auch an Fleiß und Durchhaltevermögen in den Einsendungen zur zweiten Runde eines Bundeswettbewerbs Informatik steckt. Dennoch ist eine Bewertung beim BWINF in den meisten Fällen kritischer Natur. An eine Lösung, die als mängelfrei bewertet werden will, werden recht hohe Anforderungen gestellt. Von daher sind Punktabzüge die Regel und Bewertungen über das Soll hinaus die Ausnahme, insbesondere bei den schwierigen Aufgaben dieses Jahres. Dies wirkt vielleicht entmutigend, aber nur so lassen sich die Allerbesten für die Endrunde bestimmen.

Bewertungsbögen Kein Kreuz in einer Zeile bedeutet, dass der entsprechende Aufgabenteil den Erwartungen entsprechend bearbeitet wurde. Vermerkt wird also in der Regel nur, wenn davon abgewichen wurde – nach oben oder nach unten. Ein Kreuz in der Spalte „+“ bedeutet in der Regel Zusatzpunkte, ein Kreuz unter „ok“ bedeutet eine gute, aber im wesentlichen noch erwartungsgemäße Lösung (also meist ohne Zusatzpunkte; für manche Dinge gab es bestenfalls ein „ok“), und ein Kreuz unter „–“ bedeutet Minuspunkte für Fehlendes oder Unzulängliches.

Termin der 2. Runde Es vermerken immer wieder Teilnehmer, dass die 2. Runde parallel zum Abitur liegt. Das ist uns bekannt und sicher nicht ideal, lässt sich aber leider nicht ändern. In der zweiten Jahreshälfte läuft die 2. Runde des Mathewettbewerbs, dem wir keine Konkurrenz machen wollen. Also bleibt uns nur die erste Jahreshälfte. Und damit liegt der Abgabetermin der 2. Runde immer in der Zeit der Abiturtermine. Aber: Sie haben etwa vier Monate Bearbeitungszeit für die 2. Runde. Rechtzeitig mit der Bearbeitung der Aufgaben zu beginnen ist der beste Weg, Konflikte mit dem Abitur zu vermeiden.

Dokumentation Eine Dokumentation beginnt nicht mit: „Die Prozedur abc übergibt einen Zeiger p auf ein Feld xy, worauf die Funktion f ...“. Wie in den allgemeinen Hinweisen zu den Aufgaben gesagt, soll die Dokumentation mit der Idee beginnen, die Sie zur Lösung der jeweiligen Aufgabe entwickelt haben. Schildern Sie diese Lösungsidee erst grob und gehen dann darauf ein, wie Sie sie in ein abstraktes, computertaugliches Modell (in Form von Algorithmen und Datenstrukturen) umgesetzt haben. Nutzen Sie dazu auch (halb-)formale Möglichkeiten zur Beschreibung Ihres Modells. Die Implementierung dieses Modells als Programm beschreiben Sie anschließend in der Programmdokumentation, die die wichtigsten Funktionen, Variablen, Klassen, Objekte etc. in Bezug auf die Lösungsidee dokumentiert und idealerweise mitteilt, wo diese Komponenten im Quellcode zu finden sind. Beachten Sie: Wer nicht in der Lage ist, Idee und Modell präzise zu formulieren, bekommt auch keine saubere Umsetzung in welche Programmiersprache auch immer hin.

Lösungshinweise Bei den folgenden Erläuterungen handelt es sich um Vorschläge, nicht um die einzigen Lösungswege, die wir gelten ließen. Wir akzeptieren in der Regel alle Ansätze, die die gestellte Aufgabe vernünftig lösen und entsprechend dokumentiert sind. Einige Dinge gibt es allerdings, die – unabhängig vom gewählten Lösungsweg – auf jeden Fall diskutiert werden müssen.

Aufgabe 1: Überdeckungsspiel

Die Lösungsideen werden entlang der Teilaufgaben besprochen.

Teilaufgabe 1: Grafische Ausgabe

Zunächst sollte man sich über folgende Probleme Gedanken machen:

- Über die Eingabe der anzuzeigenden Konfiguration wird in der Aufgabenstellung nicht viel gesagt. Eine Eingabedatei, die die Antennenstandorte, Frequenzen und die Reichweite angibt, genügt. Natürlich sollte es möglich sein, die Ausgaben der in den späteren Aufgabenteilen entwickelten Programme in das Anzeigeprogramm einzufüttern.
- Ein Feld von 1024x1024 Kästchen ist selbst bei einem Pixel pro Kästchen für viele Bildschirme zu groß. Mehrere Kästchen zu einem Pixel zusammenzufassen ist nur dann akzeptabel, wenn eine Zoom-Funktion vorhanden ist. Dazu kann man zum Beispiel Pixel in verschiedenen Farbtönen einfärben, so dass ein Pixel dunkler wird, wenn in den zusammengezogenen Pixeln mehrere Antennen stehen. Bei einer Lösung, bei der man den Bildausschnitt verändern kann, sollte das Programm zu Beginn automatisch den bebauten Teil von Lilliland zeigen, so dass man die Antennen nicht suchen muss.
- Bei den Interferenzen muss man darauf achten, dass wirklich unterscheidbar ist, welche Antenne mit welcher Antenne (potenziell) interferiert. Die Kreise einfach einzufärben reicht nicht, besser sollte man die Kreisbögen von interferierenden Antennen einfärben, eventuell mit mehreren Farben. Eine Linie, die interferierende Antennen verbindet und deren Farbe sich nach der Entfernung richtet, ist auch eine gute Idee, allerdings sieht man hier nicht, wie viel Fläche betroffen ist.

Es ist nicht gefordert, dass die grafische Ausgabe unmittelbar als Bitmap auf dem Bildschirm realisiert wird. Auch andere Formen können sinnvoll sein, z.B. die Ausgabe in ein druckfähiges Format, das mit Hilfe eines Viewers auch am Bildschirm betrachtet werden kann.

Teilaufgabe 2: Konfigurator

Grundsätzliche Gedanken muss man sich über folgende Dinge machen:

Welche Antennen interferieren? Dazu muss man die Strecke zwischen je zwei Antennen berechnen und schauen, ob sie länger oder gleich 2R ist, ob die Strecke zwischen R und 2R lang ist oder ob sie kürzer ist.

Welchen Bereich deckt eine Antenne ab? Man lasse sich nicht von dem Bild in der Aufgabenstellung verwirren. Alle Antennen decken eine Fläche der gleichen Größe ab, die nur von der Reichweite abhängt. Simple Strategie: Maximal können die Felder innerhalb des 2R x 2R großen Rechtecks um die Antenne herum abgedeckt sein. Ob ein Feld

noch abgedeckt ist, kann man überprüfen, indem man mittels Pythagoras überprüft, ob alle vier Ecken noch innerhalb der Reichweite liegen. Aufpassen muss man nur, dass man die Koordinate eines Kästchens auch – so wie vorgeschrieben – in die Mitte des Kästchens setzt, dann haben die Ecken um 0,5 verschobene Koordinaten.

Welchen Bereich decken mehrere, angeschaltete Antennen ab ... und wie ändert sich die Überdeckung, wenn man eine evtl. interferierende Antenne an- oder abschaltet? Simple Strategie: Ein Integer-Array für den gesamten Bereich, in dem man eine 0 für jedes nicht überdeckte Kästchen notiert. Wenn eine Antenne angeschaltet wird, muss man den wie oben berechneten Überdeckungskreis auf die Koordinaten der neuen Antenne abbilden (aufpassen bei Antennen am Rand!) und den Wert für die berührten Kästchen erhöhen. Beim Abschalten setzt man den Wert wieder herunter.

In welche Richtungen darf man eigentlich Frequenzen verteilen? Die Aufgabenstellung sagt, es würden „(F+1) TEUR für eine Frequenz, die zur Basisfrequenz um F GulHertz differiert“ berechnet. Am billigsten wird es also, wenn Frequenzen auch nach unten von der Basisfrequenz abweichen. Das macht alles ein bißchen komplizierter, das Problem an sich aber nicht interessanter, daher ist es in Ordnung, wenn man nur in einer Richtung von der Basisfrequenz abweicht.

Was ist eine beste Lösung? Klar, eine Antennenauswahl, die die gewünschte Überdeckung erreicht und deren Kosten minimal sind. Aber: in der Aufgabenstellung steht, dass man bei gleichen Kosten die Lösung mit den wenigsten Antennen bevorzugen muss! In den meisten Verfahren zur Bestimmung einer Lösung wird dieser Punkt allerdings automatisch beachtet.

Es gibt natürlich auch reichlich Spielraum für **falsche Überlegungen**:

Ich lasse nur Frequenzen oberhalb der Basisfrequenz zu, berechne die beste Belegung und erhöhe die Basisfrequenz so lange, bis die Kosten minimal sind; dadurch erreiche ich eine optimale Lösung mit einer Abweichung in beide Richtungen von der Basisfrequenz. Wohl eher nicht. Wenn man nur nach oben abweicht, dann werden sich die Frequenzen so weit wie möglich nach links orientieren, weil das am billigsten wird. Wenn man die Basisfrequenz mitten drin hat, müssen sich die Frequenzen um einen Mittelpunkt herum anordnen. Wenn man die Frequenzen nur verschiebt, bekommt man also keine optimale Lösung für das erweiterte Problem.

Ich berechne die abgedeckte Fläche aus den Einzelflächen und den paarweisen Schnittflächen von Antennen. Eine Antennenfläche ist hier die Fläche, die eine Antenne abdecken würde, wenn nur diese Antenne angeschaltet wäre. Leider funktioniert die Berechnung der Überdeckung auf diese einfache Weise auch nur in einfachen Fällen.

Wenn ich eine Lösung mit N Antennen gefunden habe, brauche ich Lösungen mit mehr Antennen nicht mehr auszuprobieren, weil diese sowieso nicht besser sein können. Vom Prinzip her eine gute Idee, denn: Wenn mehr Antennen günstiger sein sollen, müssen sie weiter auseinander stehen. Dadurch überdecken sie mehr Fläche (auf jeden Fall nicht weniger!), und man kann eine ausschalten, denn die Überdeckung durch N Antennen

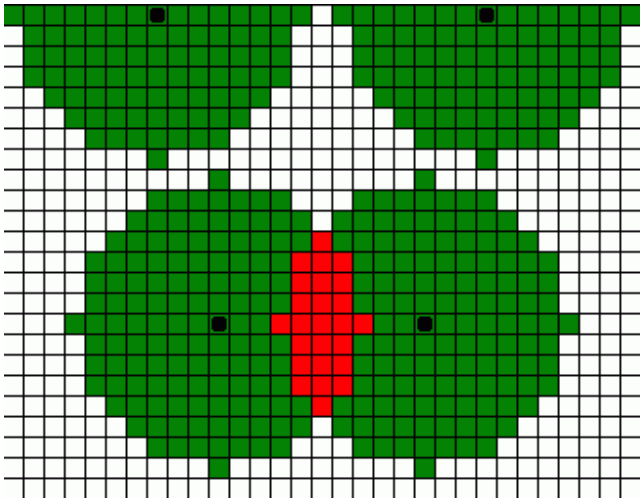

Abbildung 1: Antennen an Lililands Grenzen

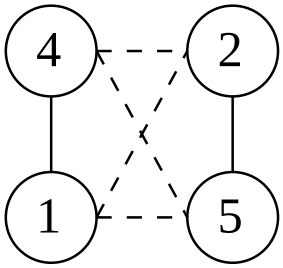

Abbildung 2: Frequenzverteilung: Die Kreise repräsentieren die Antennen, die gestrichelten Linien einen Abstand $< 2R$, die durchgezogenen einen Abstand $< R$, die Zahl in den Kreisen die zugewiesene Frequenz.

reicht ja schon. Sobald jedoch Antennen am Rand stehen, ist dies leider nicht mehr gültig, wie das Beispiel in Abbildung 1 zeigt. Die erforderliche Überdeckung wird erreicht, wenn entweder beide mittigen Antennen senden oder eine der beiden in Kombination mit beiden Antennen am Rand. Dabei braucht man für die Lösung mit drei Antennen nur die Basisfrequenz, wohingegen bei den beiden mittleren Antennen eine zweite Frequenz gekauft werden muss.

Aber: *Sei a die Fläche, die eine freistehende Antenne überdecken kann. Ich kann eine Lösung ausschließen, wenn deren Antennen theoretisch eine Fläche überdecken, die um a größer ist als die Fläche, die die Antennen der bisherigen Minimallösung theoretisch überdecken.* Die neue Lösung könnte nur dann besser sein, wenn die Antennen weiter auseinander stehen als die der Minimallösung. Dadurch gibt es weniger Interferenzen, und von der theoretischen Überdeckung bleibt mehr erhalten. Also erreicht die neue Lösung mehr als a mehr Fläche als die Minimallösung, und man könnte eine der Antennen ausschalten und trotzdem noch eine Lösung erhalten, die genügend Überdeckung hat und nicht teurer wäre.

Kleine Frequenzabstände zuerst: Man kann die Freqenzen immer so anordnen, dass nach zwei Frequenzen mit dem Abstand 3 keine Frequenzen mehr mit dem Abstand 1 kommen (bei optimalen Kosten.) Man könnte sich überlegen, dass die Antennen eine Frequenzkombination benötigen mit einer gewissen Mindestanzahl von 1er- und 3er-Abständen, deren Anordnung egal ist. In dem Fall wäre es am geschicktesten, das untere billigere Frequenzspektrum komplett zu kaufen, um die 1er-Unterschiede zu erhalten, und dann die fehlenden 3er-Frequenzen nach oben hin aufzufüllen, bspw. $1, 2, 3, 6, 9, 12$. Dass dadurch jede optimale Frequenzkombination beschrieben werden kann, wird jedoch schon durch das einfache Beispiel in Abbildung 2 widerlegt.

Algorithmen

Die beiden an den Überdeckungskonfigurator gestellen Fragen sind Varianten desselben Problems. Es unterscheiden sich nur die Bedingungen dafür, welche Antennen- bzw. Frequenzkonfiguration eine (optimale) Lösung ist. Ein Suchprozess, der für Frage (a) Konfigurationen mit Überdeckung K findet und darunter die billigste ermittelt, kann durch Änderung seiner Abbruchbedingungen dazu gebracht werden, Konfigurationen zu den vorgegebenen Kosten zu ermitteln und darunter die mit der höchsten Überdeckung zu wählen.

In beiden Teilaufgaben ist also das Problem gleich schwer und wahrscheinlich nicht in Polynomialzeit entscheidbar, und auch die exponenziellen Algorithmen, die optimale Lösungen finden, erfordern mehr Überlegung, als man zunächst erwartet. Näherungslösungen böten sich an, wenn nicht in den vorgegebenen Konfigurationsproblemen die Worte „mindestens“ und „maximal“ klar dagegen sprächen. Zumindest für Teil 2 wird also eine präzise Lösung erwartet. Präzise Ansätze können u.a. auf einer Tiefensuche basieren. Zwei Beispiele: Der erste Ansatz probiert verschiedene Frequenzen für in einer bestimmten Reihenfolge angeordnete Antennen, der andere geht alle Permutationen der Antennen durch und weist ihnen die jeweils besten Frequenzen zu. In beiden Fällen werden Lösungen mit weniger Antennen vor Lösungen mit mehr Antennen betrachtet, so dass diesen Lösungen der Vorzug gegeben wird.

Die zweite Idee betrachten wir näher. Der Graph für die Tiefensuche sieht dabei so aus:

Knoten: Hier repräsentiert ein Knoten eine Belegung von i beliebigen Antennen, zum Beispiel: Antenne 4: Frequenz 0, Antenne 2: Frequenz 3.

Kanten: Eine Belegung mit i Antennen wird durch $N - i$ Kanten mit Belegungen von $i + 1$ Antennen verbunden, wobei sich die Kanten durch die neu belegte Antenne unterscheiden. Dieser Antenne wird dann die kleinstmögliche Frequenz zugewiesen.

Startknoten: Begonnen wird mit der Belegung von 0 Antennen.

Die Abbruchbedingung für die Suche lautet: Sobald alle Antennen belegt worden sind oder die Überdeckung der schon belegten Antennen genügt, werden die Kosten mit den bisherigen Minimalkosten verglichen und diese eventuell aktualisiert. Im schlimmsten Fall hat der Algorithmus mindestens eine Laufzeit von $n!$, mit einem Faktor für das Berechnen der aktuellen Überdeckung etc. Allerdings kann der Suchprozess optimiert werden, und zwar wie folgt:

Zusätzliche Abbruchbedingung Sobald die Kosten höher sind als die bisherigen Minimalkosten, kann abgebrochen werden (der Geschwindigkeitszuwachs ist typischerweise in der Größenordnung Faktor 2).

Äquivalente Zustände Mit einer Hashtabelle kann man verhindern, dass man zweimal einen äquivalenten Zustand betrachtet. Für jeden Knoten, der besucht wird, berechnet man einen Hashwert und speichert ihn in einer Hashtabelle. Gelangt der Suchprozess bei einem Knoten an, dessen Hashwert bereits in der Tabelle steht, kann man abbrechen. Problematisch ist hierbei, die Äquivalenz zweier Zustände sicherzustellen; ein möglicher Weg ist, den Hashwert aus den bisher verwendeten Frequenzen, der erreichten Abdeckung und der Frequenzzuweisung an alle belegten Antennen zu berechnen. Weiter kann man hierbei alle Antennen vernachlässigen, die keine Interferenz mit einer nicht belegten Antenne haben. Selbst dieses einfache Verfahren bringt einen gewaltigen Geschwindigkeitszuwachs, und es gibt noch umfangreiche Erweiterungsmöglichkeiten. Der Gewinn hängt stark von der Anzahl der Komponenten (also den Gruppen von Antennen, die nicht miteinander interferieren) ab; in einem komplexen Beispiel mit 11 Antennen in einer Komponente konnte die Anzahl der zu betrachtenden Zustände von ca. 100000 auf weniger als 4000 reduziert werden.

Bitoperationen Allgemein kann man durch die Verwendung von Bitoperationen (z.B. beim Hashen) deutlich beschleunigen, meistens führt das aber dazu, dass man nur noch 32 Antennen oder Frequenzen von 0-31 behandeln kann. Größere Instanzen sind aber auch nur in trivialen Fällen noch schnell lösbar, weshalb dies keine echte Einschränkung darstellt.

Verändern der Reihenfolge Den Abbruchbedingungen entsprechend kann es sinnvoll sein, die Reihenfolge, in der die Kanten abgegangen werden, zu verändern, so dass man früher niedrige Kosten für die Abdeckung findet und in Zukunft schneller abbrechen kann.

Es gibt auch einen völlig anderen Ansatz, der aber wohl deutlich ineffizienter ist. So könnte man für alle möglichen Kosten C bei 1 beginnend versuchen, ob es möglich ist, die nötige Abdeckung zu erreichen; hierzu generiert man alle Frequenzkombinationen, die mit genau C möglich sind (ca. C^3 bis C^4 verschiedene) und alle Teilmengen von Antennen (mit ein paar Optimierungen muss man hier maximal ca. $\binom{n}{n/2}$ betrachten). Nun versucht man für jede der Teilmengen die Frequenzen zu verteilen und bricht ab, sobald eine interferenzfreie Verteilung gefunden ist. Da es sich beim letzten Schritt um eine Art Färbungsproblem handelt, das man wahrscheinlich nur in Exponenzialzeit lösen kann, fällt die Laufzeit insgesamt deutlich schlechter aus als bei dem vorgestellten Verfahren.

Teilaufgabe 3: Weitere Fragen

Dieser Aufgabenteil verlangt explizit weitere Fragen, nicht unbedingt eine Erweiterung der Vorgaben. Dafür gibt es (unter anderem) folgende zwei Möglichkeiten:

Gegebene Größen verändern Man kann prinzipiell beliebige Parameter vorgeben und andere ermitteln lassen. Zum Beispiel: „Wenn ein Etat von E TEUR verfügbar ist und man eine Überdeckung von K erreichen soll, wie viele Antennen benötigt man dazu mindestens?“. Es ist auch möglich, Größen in Abhängigkeit zueinander zu stellen, so z.B.: „Wenn eine Abdeckung von mindestens K erreicht werden soll, wann ist dann das Verhältnis von Frequenzkosten zur Abdeckung optimal?“ Oder es werden beliebige andere Einschränkungen vorgegeben; z.B. in einer Variante von 2.a mit bestimmten Frequenzen, die für wichtige Zwecke reserviert sind und deswegen nicht genutzt werden können.

Je nachdem, welche Größen man fest wählt, benötigt man nur kleine Erweiterungen des Algorithmus oder völlig neue Verfahren. Ein Beispiel für Letzteres: *Wenn die Reichweite der Antennen, die Anzahl der Antennen, die geforderte Überdeckung und die maximalen Kosten gegeben sind, wo platziert man dann am besten die Antennen?* Im Gegensatz zu Frage 2.b ist diese Frage keine Variation des ursprünglichen Algorithmus', obwohl sie nahe liegt.

Heuristiken Die Forderung nach Optimalität in den Fragen kann aufgeweicht werden. Auch hierzu ein Beispiel: *Wenn Antennenstandorte, gewünschte Überdeckung und Reichweite gegeben sind, wie hoch werden die Sendefrequenzkosten ungefähr sein?* Als Konsequenz wird es möglich, sich mit Heuristiken zu versuchen, die sich durchaus als Alternativen zu den langsamen Algorithmen für optimale Lösungen anbieten. Damit kann man auch für eine große Menge von Antennen einen ungefähren Kostenwert ermitteln.

Erweiterungen

Mögliche Erweiterungen sind:

- Andere Kartenformen: Weniger spannend. Lililand ist vorgegeben, und andere Karten sind nicht schwieriger zu bearbeiten. Einzige Ausnahme: Deutlich größere Karten wären eine Herausforderung.
- Interaktiver Modus: Setzen und Verschieben von Antennen on-the-fly, während eine Heuristik eine ungefähre Kostenabschätzung macht.
- Besondere Gebiete: dürfen entweder nicht bestrahlt werden (Naturschutzgebiet) oder müssen bestrahlt werden (Innenstadt).
- Unterschiedliche Antennen, zum Beispiel Richtantennen.
- Sendeleistung: In der Wirklichkeit nimmt die Sendeleistung von Antennen mit der Entfernung vom Antennenstandort ab. Man könnte eine quadratische Abnahme annehmen und verlangen, dass die überdeckte Fläche auch mit einer Sendeleistung von 80 Prozent überdeckt ist. Interferenzen können in diesem Fall positiv sein.

Bewertungskriterien

Folgende Punkte wurden bei der Bewertung besonders beachtet:

Teilaufgabe 1: Grafische Ausgabe (Visualisierung)

- Die Eingabe einer anzuzeigenden Konfiguration muss sinnvoll gelöst sein, idealerweise mit einem eigenen Beschreibungsformat für Konfigurationen, das natürlich auch die Beschreibung von Problemstellungen, also Antennenstandorten ohne Frequenzzuweisung, erlauben muss.
- Auch die Darstellung von Konfigurationen (und Problemstellungen) muss angesichts der Größe Lililands geeignet realisiert werden.
- (Potenzielle) Interferenzen sollten bei der Darstellung einer Konfiguration nachvollziehbar abgebildet werden.

Teilaufgabe 2: Konfigurator

- Bei der Berechnung von Interferenzen, Überdeckungen und bester Lösungen darf es keine Fehler geben.
- Die Verteilung der Frequenzen sollte möglichst kostengünstig sein, wobei ein nur einseitiges Abweichen von der Basisfrequenz akzeptiert wird.
- Die anerkannt aufwändige Suche nach einer präzisen Lösung darf nicht unnötig ineffizient sein; Optimierungen von Suchprozessen sind unabdingbar.
- Die Verwendung eines Näherungsverfahrens wird mit Punktabzug bewertet. Bei gut durchdachten Methoden, die offensichtlich gute Lösungen auch für umfangreichere Testdaten berechnen konnten, haben die Bewerter zum Ausgleich Pluspunkte vergeben.

Teilaufgabe 3: Weitere Fragen Die mindestens zwei weiteren Fragen müssen Konfigurationsprobleme darstellen. Wenn zur Beantwortung der Frage neue Verfahren (im Vergleich mit Teil 2) realisiert werden mussten, gab es Bonuspunkte.

Aufgabe 2: Handlungsreisegruppe

Lösungsidee

Das Problem ist eine Kombination aus einem Travelling Salesman Problem (TSP) und einem Partitionierungsproblem, beides für planare Graphen.

Da keine Einschränkung für die Verbindungen zwischen den Städten vorgegeben ist, kann man davon ausgehen, dass jede Stadt mit jeder anderen verbunden ist und die Entfernungen jeweils der Luftlinie zwischen den beiden Städten entsprechen. Es ist aber auch möglich, das Fehlen von Einschränkungen anders zu verstehen und selbst festzulegen, zwischen welchen Städten es Wege gibt (und wie lang diese sind). Das entspricht der realistischen Annahme, dass die Städte untereinander durch Straßen verbunden sind, die nicht notwendigerweise von jeder Stadt zur jeder anderen führen (und nicht immer geradlinig verlaufen). Allerdings darf das Problem durch eigene Angaben nicht grundsätzlich vereinfacht werden, z.B. indem Städte und Straßen immer nur eine Baumstruktur ergeben.

In beiden Fällen liegt ein planarer Graph mit gewichteten, nicht notwendigerweise kreuzungsfreien Kanten vor. Im ersten Fall gilt (wie üblich sei V die Menge der Knoten/Städte und E die Menge der Kanten/Straßen) $|E| = |V| * (|V| - 1)$, im zweiten Fall liegt $|E|$ in den Grenzen $|V - 1|$ und $|V| * (|V| - 1)$[1].

Auf jeden Fall ist schnell ersichtlich, dass man mit bloßem Durchprobieren aller Kombinationen selbst auf einem schnellen Rechner nicht weiterkommt. Im Fall der paarweise verbundenen Städte kommt man auf $|V|!$ verschiedene mögliche Wege für einen Handlungsreisenden. Dabei ist dann noch die Aufteilung auf drei Personen zu berücksichtigen, die man aus zwei Teilungen der entstandenen Route an beliebigen Stellen betrachten kann. Die Komplexität dieser Teilungsoperation beläuft sich auf $|V|^2$ und muß für jeden möglichen Weg ausgeführt werden. Die Gesamtkapazität beläuft sich also auf $|V|! * |V|^2$. Auf Grund der Fakultät wächst dieser Ausdruck extrem schnell. Für 25 Städte ergibt sich schon ein Aufwand von ungefähr 10.000.000.000.000 Milliarden Tests, ob die aktuelle Kombination die günstigste ist. Die Lösung des Problems kann daher noch über approximative Algorithmen erfolgen. Welcher Algorithmus im einzelnen gewählt wird, hängt ganz speziell auch von der Wahl des Kriteriums (Teilaufgabe 2) ab. Daher sollen zunächst mögliche Gütekriterien betrachtet werden.

1. Ein Kriterium ist, dass die Dauer der Reise der drei Personen möglichst kurz ist, also die längste Reise einer Einzelperson möglichst kurz ist. Dies führt gleichzeitig dazu, dass die Arbeitsauslastung aller drei Reisender möglichst gleich ist.

2. Alternativ kann nach Kosten optimiert werden – wie beim klassischen TSP also. Wenn man lange Strecken mit hohen Kosten gleichsetzt, sollte die von allen zusammen zurückgelegte Strecke möglichst kurz sein. Der Nachteil ist, dass die drei Reisenden dann unter Umständen sehr unterschiedliche Reisedauern haben.

[1] Falls jede Stadt von jeder anderen auf irgendeinem Weg, notfalls auch über andere Städte, erreichbar ist. Davon kann man aber ausgehen, da der Handlungsreisende den Weg früher alleine erledigen musste.

3. Es ist denkbar (als Erweiterung der Aufgabe), den einzelnen Städten Prioritäten zuzuordnen, die angeben, wie wichtig es ist, dass der Handlungsreisende selbst dort erscheint. (Ob diese Kriterien aber nach Kundenvorkommen, -interesse oder persönlichen Vorlieben des Reisenden gesetzt werden, soll hier offen bleiben.) Damit kann man eine Wert-Optimierung der Reise betreiben und Kriterium 1 oder 2 als Nebenbedingung einführen.

Für das zweite Kriterium fällt eine elegante Lösung auf: Wenn man das Travelling-Salesman-Problem über den minimalen Spannbaum des Graphen approximiert, kann man leicht die drei Teilgebiete durch Entfernen der schwersten Kanten im Spannbaum erzeugen. Dieser Ansatz soll im Folgenden näher beschrieben werden.

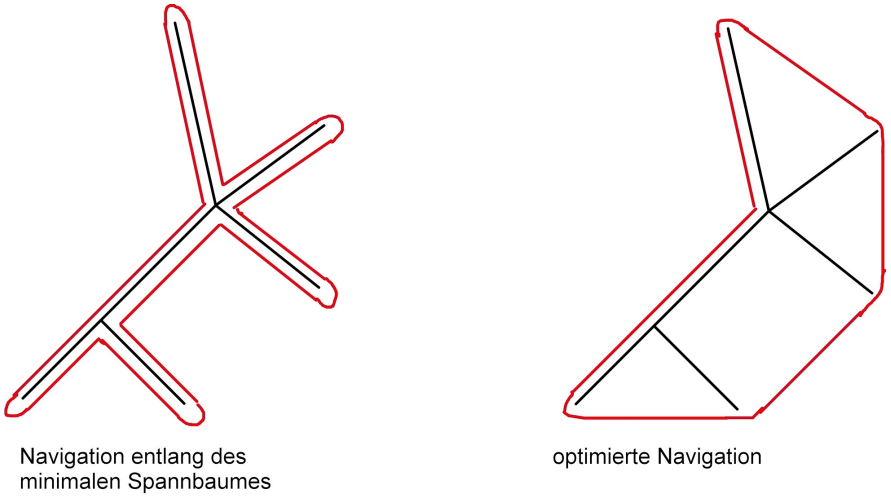

Abbildung 3: Spannbaum-Tour und optimale Tour

Ein Spannbaum eines Graphen besitzt die gleichen Knoten wie der Graph selbst, ist jedoch auf jeden Fall kreisfrei. Es gibt also auf keinen Fall zwei verschiedene Möglichkeiten, von einem Knoten zum anderen zu gelangen. Der minimale Spannbaum ist derjenige Spannbaum, bei dem die Summe der Kantengewichte minimal ist. Ein solcher minimaler Spannbaum lässt sich einfach und elegant berechnen. Die Kanten eines Graphen werden dazu aufsteigend sortiert. Man fängt mit der kleinsten Kante an und fügt diese in den neuen Graphen ein. Nun nimmt man jeweils die nächst kleinere Kante und prüft, ob diese eingefügt werden kann, ohne dass sich ein Kreis bildet. Dies wiederholt man, bis alle Knoten des Graphen miteinander verbunden sind. Diese Verfahren hat eine polynomielle Laufzeit von n^3. Zur Betrachtung der Approximation gehen wir zunächst von nur einem Handlungsreisenden aus, der den gesamten Weg zurücklegen muss. Der Handlungsreisende bewegt sich dabei nur entlang des Spannbaums. Jede Kante wird genau zweimal benutzt. An welchem Knoten er startet, ist hierbei zu vernachlässigen, da der Weg eine Schleife bildet. Ein Beispiel für den Weg entlang eines Spannbaums ist in Abb. 3 zu sehen. Es lässt sich recht einfach zeigen, dass der so gefundene Weg entlang des minimalen Spannbaums nicht mehr als doppelt so lang ist wie der optimale Weg für den Handlungsreisenden.

Teilen sich nun drei Reisende das Gebiet, so kann der Spannbaum in drei Teilbäume zerlegt werden. Soll das zweite Kriterium angewendet werden, so macht es Sinn, die teuersten Kanten aus dem Spannbaum zu löschen. Nach Entfernen zweier Kanten ist der Spannbaum in drei Teilbäume zerfallen. Es kann zwar sein, dass einzelne Teilbäume lediglich einen Knoten und keine Kante besitzen, jedoch sind die Reisekosten, die nun insgesamt entstehen, minimiert.

Nun führt jeder der drei Reisenden eine Rundreise auf einem der Teilbäume durch. Für jede Rundreise gilt natürlich auch wieder, dass sie nicht mehr als doppelt so lang ist, wie die kürzeste mögliche Rundreise im jeweiligen Teilbaum. Es gilt sogar, dass die Summe der drei Reisen auch nicht mehr als doppelt so lang ist wie die optimale Summe. Die Partitionierung verschlechtert das Ergebnis also nicht!

Wenn nur dieser Algorithmus angewendet würde, könnte man lediglich einen maximalen Umweg von 100% angeben. Es gibt jedoch noch einige Vereinfachungen, die der Handlungsreisende auf seinem Weg machen kann. So muss er sich nicht zwangsläufig an den Spannbaum halten, sondern kann evtl. abkürzen. Eine einfache Möglichkeit der Abkürzung sieht folgendermaßen aus: Durch den minimalen Spannbaum lässt sich der Handlungsreisende lediglich eine Reihenfolge der zu besuchenden Städte angeben. In dieser Liste stehen viele Städte mehrfach, müssen aber nur einmal besucht werden. Die wiederholten Vorkommen der Städte können also einfach gestrichen werden. Der Reisende fährt die verbleibende Stadtliste ab und nutzt dabei jeweils die kürzeste mögliche Verbindung, die er anhand des vollständigen Graphen und einem Algorithmus für kürzeste Wege berechnet (mit Dijkstra oder ähnlich). Diese Entfernung ist auf jeden Fall nicht größer als die Länge des Weges, den der Reisende über den minimalen Spannbaum genommen hätte.

Für die Angabe des Umweges geht man nun wie folgt vor: Man vergleicht die Länge des so berechneten Weges mit der des Weges entlang des minimalen Spannbaums. Von letzterer weiß man, dass sie nicht länger als das Doppelte des optimalen Weges ist. Ist also die neue Route mit den Abkürzungen zum Beispiel 30% günstiger, als der ursprüngliche Weg, so entspricht dies 140% der optimalen Route, der maximale Umweg beträgt nur noch 40%.

Eine weitere Präzisierung der Angabe für den Umweg lässt sich durch genaueres Betrachten der Approximation über den minimalen Spannbaum angeben. Ist ein Graph kreisfrei, so liefert die Approximation das optimale Ergebnis. Man könnte also nun alle Abschnitte des minimalen Spannbaums daraufhin untersuchen, ob sie kreisfreien Abschnitten im Graphen entsprechen. In diesem Fall ist der entsprechende Abschnitt bereits optimal. Nur die nicht optimalen Abschnitte müssen für die Berechnung des Umweges berücksichtigt werden.

Soweit eine einfache Lösung über den minimalen Spannbaum. Diese Lösung funktioniert sehr gut für das Kriterium 2. Für das erste Kriterium ist die gezeigte Aufteilung hingegen recht ungeeignet. Unter Umständen kann dabei herauskommen, dass zwei Personen jeweils nur eine Stadt besuchen (Reisekosten = 0) und eine Person den gesamten Rest des Gebietes besucht. Für das erste Kriterium sollte der Baum also in drei Teilbäume zerlegt werden, die ungefähr gleich groß sind. Die Angaben zur Umwegberechnung gelten natürlich weiterhin für jeden Teilbereich, der gesamte Umweg kann jedoch höher sein, da die Aufteilung nicht notwendigerweise optimal ist.

Erweiterungen

Echte, interessante Erweiterungen sind nicht offensichtlich – immerhin ist das Problem schon NP-vollständig. Natürlich lässt sich das Drei-Rundreisen-Problem auf ein N-Rundreisen-Problem verallgemeinern. Es ist positiv, wenn dies erkannt wurde, bedeutet aber keine besondere Steigerung der Schwierigkeit. Mehr als zwei Kriterien für die Güte einer Lösung zu nennen, ist natürlich keine Erweiterung des Problems. Denkbar ist, Bedingungen einzuführen, die bei der Suche nach der Lösung berücksichtigt werden müssen (wie die bei Kriterium 3 genannten Prioritäten), aber auch das Umsetzen solcher Bedingungen wird in der Regel nicht allzu schwierig sein.

Bewertungskriterien

Folgende Punkte wurden bei der Bewertung besonders beachtet:

Zwei Programmteile waren zu entwickeln, und zwar die Visualisierung und Auswertung von Drei-Rundreisen-Lösungen (Teile 1 und 2) und die Berechnung dieser Lösungen (Teil 3). Um beide Teile separat testen zu können, ist es wichtig, sich ein Dateiformat für Städtepositionen und Routen zu überlegen und entsprechend in allen Teilen die Möglichkeit vorzusehen, dass die Eingabedaten aus einer Datei eingelesen werden.

Teilaufgabe 1: Grafische Darstellung Die graphische Oberfläche muss die Städte und die Verbindungen zwischen den Städten darstellen können. Ist die Darstellung besonders übersichtlich (mehrfarbig für Städte, Straßen und optimale Verbindungen, scrollbar, ...), wird das honoriert.

Teilaufgabe 2: Kriterien Mindestens zwei vernünftige Kriterien für die Güte einer Drei-Rundreisen-Lösung sollten vernünftig dokumentiert und begründet sein.

Teilaufgabe 3: Drei-Rundreisen-Lösung

- Hier müssen zunächst das Problem des Handlungsreisenden identifiziert und die Komplexität (NP-vollständig) erkannt werden. Die Verwendung von Näherungsverfahren war bei dieser Aufgabe möglich („möglichst gut“) und sinnvoll, im Gegensatz zu uninformierten Suchverfahren.
- Das für die Berechnung von Lösungen verwendete (Näherungs-)Verfahren sollte insbesondere im Hinblick auf die Aufteilung in drei Rundreisen geeignet sein. Wichtig ist außerdem, dass ein Lösungsweg beschritten wird, der es ermöglicht, die Abweichung vom optimalen Pfad zu bestimmen.

- Wichtig ist auch die Erkenntnis, dass die Aufteilung auf drei Personen zwar für den Handlungsreisenden eine Erleichterung darstellt, das Problem aber komplexer macht.
- Die Partionierung sollte ein Ergebnis ermöglichen, das dem in Aufgabenteil 2 gewählten Kriterium entspricht.
- Schließlich sollte der maximale Umweg korrekt berechnet sein. Das bedeutet zum einen, dass ein solcher Umweg überhaupt berechnet und ausgegeben wird, zum anderen aber auch, dass die Güte des verwendeten Algorithmus korrekt bestimmt ist. Auf der anderen Seite wird honoriert, wenn der Umweg sehr genau angegeben wird.

Aufgabe 3: Synonymie

Diese Aufgabe unterscheidet sich deutlich von den beiden anderen in dieser Runde gestellten Aufgaben. Während beim Überdeckungsspiel und ganz besonders bei der Handlungsreisegruppe algorithmische Fragen von zentraler Bedeutung sind, sind hier Datenmodellierung und Systementwurf wichtig. Wem das auf den ersten Blick einfach vorkam, hat sich möglicherweise dazu verführen lassen, nicht gründlich über die Problemstellung nachzudenken und entscheidende Dinge zu übersehen. Bei einer Aufgabe wie dieser sind jedenfalls Begründungen für Modellierungs- und Implementationsentscheidungen besonders wichtig, denn nur anhand dieser Begründungen kann entschieden werden, ob die Bearbeitung durchdacht ist oder aus dem Ärmel geschüttelt wurde.

Die weiteren Lösungsideen werden entlang der Teilaufgaben besprochen.

Teilaufgabe 1: Datenstrukturen und Operationen

Die Aufgabenstellung fordert sehr explizit und konkret ein informatisches Modell ein, und zwar das eines Wörterbuchs, das über die Worteinträge hinaus auch Angaben über Synonymbeziehungen zwischen den Wörtern enthält. Es ist klar vorgegeben, dass ein Wort als ein abstrakter Datentyp mit verschiedenen Attributen aufgefasst werden soll. Dabei ist das Attribut „Schriftform“ vom Typ „Zeichenkette“ vorgegeben. Darüber hinaus gibt es noch das Wörterbuch und die Synonymbeziehung.

Für alle diese Einheiten ist eine Implementierung zu bestimmen. Dazu muss man sich aber vorher über deren Eigenschaften Klarheit verschaffen. Hier einige Punkte:

- Ein Wort ist nicht mit seiner Schriftform (also der Folge von Buchstaben, die das Wort in Schriftstücken repräsentiert) zu identifizieren. Das ist zwar in der Aufgabenstellung vorgegeben, aber nicht direkt einsichtig. Für diese Aufgabe ist es essenziell, dass ein Wort außer der Schriftform weitere Eigenschaften hat, auf die der Synonymisator zurückgreifen kann. Es ist aber akzeptabel, wenn die Schriftform als Schlüsselattribut für ein Wort dient. Daraus folgt allerdings, dass nicht mehrere Worte mit gleicher Schriftform, also Homonyme, abgespeichert werden können.
- Die Datenstrukturen für Wörter und Wörterbuch müssen flexibel genug sein, Änderungen der (für alle Wörter gleichen) Attribute zu erlauben.
- Die Datenstrukturen für Wörterbuch, Attribute und Synonymbeziehungen sollen eine effiziente Implementation der eingeforderten Operationen erlauben.

Besondere Aufmerksamkeit verdienen die Eigenschaften der Synonymbeziehung. Vorgegeben ist, dass es sich um eine Paarrelation über der Menge der Wörter handelt – keinesfalls um eine Funktion, denn ein Wort kann mehrere Synonyme haben, und nicht jedes Wort ist Synonym eines anderen. Ein Wort ist mit sich selbst natürlich gleichbedeutend, doch ist es nicht sinnvoll, es als ein Synonym von sich selbst aufzufassen (es sei denn, dies erleichterte die Realisierung

des Synonymisators). Die Symmetrie der Synonymrelation (A ist synonym zu B genau dann, wenn B synonym zu A) sollte hingegen bedacht werden. Einen wichtigen Einfluss auf die Synonymrelation hat die Behandlung von Homonymen, also von „Teekesselchen“[2]. Wenn nur die Schriftform betrachtet wird, ist „Bank“ synonym mit „Sitzgelegenheit“ und „Geldinstitut“, während letztere nicht synonym zueinander sind. Dieses Problem sollte erkannt werden; außerdem muss klar werden, ob und wie es behandelt wird. Zum einen kann man mehrere Worte mit der gleichen Schriftform erlauben, die, was durch ihre Synonyme klar werden kann, unterschiedliche Bedeutung haben. Zum anderen kann man sich Gedanken darüber machen, ob die Symmetrie der Synonymie auch durch den Synonymisator in Teil 3 ausgenutzt werden kann. Eine Möglichkeit ist, im Hinblick auf den Synonymisator für Synonympaare ggf. eine Ersetzungsrichtung anzugeben: „Geldinstitut“ kann durch „Bank“ ersetzt werden, aber nicht umgekehrt.

Für die Implementation von Wörterbuch und Synonymbeziehung gibt es eine ganze Reihe mehr oder weniger guter Möglichkeiten. Dabei reicht es nicht, sich z.B. für eine objektorientierte Realisierung zu entscheiden, denn damit sind weder die Datenstruktur für das Wörterbuch noch die Funktionsweise der zu definierenden Operationen gegeben. Eine naheliegende Möglichkeit ist die Verwendung einer relationalen Datenbank. Ein Wörterbuch wird als Tabelle realisiert, in einer weiteren Tabelle können die Attribute verwaltet werden, und auch die Synonymbeziehung ist eine Relation, die als Tabelle implementiert werden kann. Eine ebenfalls naheliegende Alternative sind die aus der Algorithmenlehre bekannten Datenstrukturen für Wörterbücher wie Binärbäume, Hashtabellen oder andere effiziente Suchstrukturen. Hierzu kann man z.B. bei Sedgewick[3] mehrere Kapitel finden.

Die Menge der zu realisierenden Operationen ist relativ offensichtlich: Wörter einschließlich ihrer Attributwerte, Attribute (evtl. einschließlich einer Angabe zu ihrer Wertemenge) und Synonympaare müssen hinzugefügt und entfernt werden können. Änderungen können theoretisch als Kombination aus Entfernung und Hinzufügen aufgefasst werden, es ist aber sinnvoll, eigene Operationen dafür zu definieren.

Teilaufgabe 2: Benutzungsoberfläche

Auf den in Teil 1 definierten Datenstrukturen und Operationen soll nun eine Benutzungsoberfläche aufgesetzt werden. Das Wort „einfach“ hatte klarmachen sollen, dass nicht besondere Energie auf die äußere Gestaltung zu verwenden ist. Entscheidend ist, dass die Oberfläche „eine effektive Arbeit mit dem Wörterbuch ermöglicht“, also die möglichst sichere Eingabe neuer Wörter und Synonymbeziehungen erlaubt. Die Aufgabenstellung fordert die Nennung von Kriterien (Plural!) für die entsprechende Gestaltung der Oberfläche ein. Dabei wird keine besondere Erfahrung im Themenbereich Mensch-Computer-Interaktion vorausgesetzt, aber ein „muss intuitiv bedienbar sein“ reicht nicht. Leichte Erlernbarkeit durch Neulinge und effektive Bedienbarkeit durch erfahrene Benutzer können zu verschiedenen Gestaltungskriterien

[2] Siehe dazu auch http://de.wikipedia.org/wiki/Homonym.
[3] Robert Sedgewick: Algorithmen. Pearson Studium, ISBN 3-8273-7032-9.

führen. Effektivität bedeutet insbesondere die Fehlerfreiheit der Interaktion; bei dieser Anwendung kann dies z.B. durch die inhaltliche Unterstützung beim Ausbau des Wörterbuchs gefördert werden. Neben Effektivität ist auch Effizienz wichtig, also die möglichst schnelle Bearbeitung des Wörterbuchs. Da die Kombination von beidem ideal ist, sind auch Effizienzkriterien von Interesse.

Typisch für die schnelle Eingabe von Datenbankinhalten sind Eingabemasken, die allein mit der Tastatur bedient werden können (z.B. durch Verwendung der Tabulatortaste zur Navigation zwischen den Eingabefeldern). Evtl. vorhandene Standardwerte für die Attribute sollten eingeblendet werden, damit direkt klar ist, dass die entsprechenden Felder ggf. übersprungen werden können. Bei der Eingabe von Synonymen kann die Oberfläche nach Eingabe der ersten Buchstaben Vervollständigungen aus dem Wörterbuch anbieten. Auch das Erarbeiten von Wörterbuchdaten mittels eines guten Texteditors und das Einlesen der so entstandenen Datei(en) kann schneller sein als die Eingabe über eine Maske. Eine effiziente und effektive Benutzungsoberfläche kann durchaus auch Kommandozeilenbefehle enthalten.

Inhaltlich kann der Aufbau des Wörterbuchs unterstützt werden, wenn z.B. bei der Eingabe eines Synonympaares schon vorhandene Synonyme der beiden Wörter angezeigt und für den Eintrag weiterer Paare angeboten werden (falls die Transitivität der Synonymie, um die es dabei geht, nicht anderweitig behandelt wird).

Es ist zu begründen, wie und warum die realisierte Benutzungsoberfläche die angegebenen Kriterien umsetzt. Schön ist, wenn auf der Grundlage der eigenen Erfahrungen beim Erstellen des eigenen Wörterbuchs gezeigt wird, wie effektiv bzw. effizient die Arbeit mit der Oberfläche nun vonstatten gegangen ist.

Nicht vorgegeben war die Sprache, der die Inhalte des Wörterbuchs entstammen. Die Verwendung englischer Wörter kann die Arbeit des Synonymisators erleichtern, je nachdem wie komplex dieser ausgelegt wurde. Ob Englisch oder Deutsch: Ganz genau 100 Wörter müssen es nicht sein, aber doch annähernd so viele. Fünf Attribute sollten es hingegen schon sein. Neben der Schriftform enthält die Aufgabenstellung zwei weitere Vorschläge, nämlich „Grad der Förmlichkeit“ und „Wortlänge“, wobei letzteres automatisch ermittelt werden kann. So müssen letztlich nur noch zwei weitere Attribute selbstständig bestimmt werden.

Teilaufgabe 3: Synonymisator

Der Synonymisator soll anhand von Regeln, die sich auf die Attribute der Wörter beziehen, Wörter einer Wortfolge durch Synonyme ersetzen. Dabei gibt es zwei wichtige Punkte:

Die Abarbeitung der Wortfolge Die Aufgabenstellung legt eine Abarbeitung Wort für Wort nahe. Bei natürlichsprachlichen Beispielen ist es aber hilfreich, ein wenig „Sprachverarbeitung“ zu betreiben. Wird ein Wort nicht im Wörterbuch gefunden, kann man z.B. versuchen, durch Grundformbildung (im Englischen relativ leicht: Plural-s abtrennen, Verlaufsform oder Past Tense von Verben in Grundform umwandeln durch Abtrennung von „ing“ bzw. „ed“)

noch etwas zu erreichen. Eine Bearbeitung sollte diese Problematik zumindest beachten, auch wenn letztlich eine einfache Wort-für-Wort-Abarbeitung realisiert ist.

Die Form der Regeln Die Aufgabenstellung legt mit den Worten „programmiere sie“ eine feste Implementierung der gewählten Regeln im Quellcode nahe. Schöner ist natürlich, wenn ein Mechanismus entwickelt wird, der mehr oder weniger flexibel die Definition von Regeln erlaubt; dies stellt allerdings eine Erweiterung der Aufgabenstellung dar (s.u.). Als Abstraktionsleistung ist aber zu verlangen, dass die Gemeinsamkeiten aller möglichen Regeln erkannt und besprochen werden; diese Diskussion führt dann idealerweise zu einem eigenen Datentyp für Regeln. Bedingungen innerhalb der Regeln können sich beziehen

- auf die Auswahl der zu ersetzenden Wörter. Beispiel: Nur Wörter einer bestimmten Länge sollen überhaupt ersetzt werden.
- auf die Auswahl der ersetzenden Synonyme. Beispiel: Ein möglichst förmliches Synonym soll gewählt werden.
- auf zu ersetzendes Wort und ersetzendes Synonym gleichzeitig. Beispiel: Ein Wort soll durch ein längeres Synonym ersetzt werden.

Die Beispielregel der Aufgabenstellung enthält nur eine Bedingung bzgl. des ersetzenden Synonyms. Es ist ausreichend, nur derartige Bedingungen zu realisieren, mehr ist aber besser.

Wenn bei der Abarbeitung der Wortfolge ein Wort kein Synonym hat oder gar nicht im Wörterbuch enthalten ist, kann es einfach ignoriert werden. Das erlaubt den Einsatz natürlichsprachlicher Beispiele auch bei einem mit 100 Wörtern sehr kleinen Wörterbuch.

Die Anzahl der zu erfindenden Synonymisationsregeln ist nicht vorgegeben. Aber aller guten Dinge sind traditionell auch beim BWINF drei, so dass insgesamt mindestens drei mal zwei Beispiele vorhanden sein sollten.

Erweiterungen

Folgende zusätzliche Leistungsmerkmale stellen Erweiterungen dar:

Sprachverarbeitung Linguistische Vorverarbeitung der Eingabewortfolge, z.B. zur Grundformbildung oder zur Bestimmung von Wortgruppen. Damit muss der Synonymisator nicht nur wortweise agieren.

Flexibler Regelmechanismus Möglichkeit der Deklaration von Synonymisationsregeln, von logischen Verknüpfungen zwischen Regelbedingungen usw. Dies erfordert eine Art Regelinterpreter.

Geschickte technische Realisierung Eine Client-Server-Lösung zum Beispiel könnte das gemeinsame Führen eines zentralen Wörterbuchs erlauben.

Bewertungskriterien

Folgende Punkte wurden bei der Bewertung besonders beachtet:

Teilaufgabe 1: Datenstrukturen und Operationen

- Modellierung und Umsetzung der wichtigsten Einheiten wie Wort, Attribut, Wörterbuch und Synonymrelation müssen geeignet sein, entsprechend der obigen Hinweise. Die Modellierung sollte nicht nur in natürlicher Sprache, sondern zumindest auch halbformal dokumentiert werden.
- Die Menge der Attribute muss verändert werden können.
- Die Synonymbeziehung muss analysiert und das Homonymproblem zumindest erkannt werden. Eine sinnvolle Behandlung des Homonymproblems führt zu Pluspunkten.
- Die Operationen müssen sinnvoll definiert sein, wie oben beschrieben, und effizient realisiert sein, ob mit fremden oder eigenen Mitteln.

Teilaufgabe 2: Benutzungsoberfläche

- Die Beachtung der genannten Kriterien sollte wirklich zu einer effektiven (und/oder effizienten) Bedienung führen.
- Die gewählte Realisierung muss die genannten Kriterien auch umsetzen.
- Die Effektivität und Effizienz der Benutzungsoberfläche kann nur mit einem ausreichend großen Wörterbuch und einer vernünftigen Zahl von Attributen sinnvoll demonstriert werden.

Teilaufgabe 3: Synonymisator

- Es muss zumindest erwähnt sein, dass eine ordentliche Synonymisierung eigentlich Methoden der Sprachverarbeitung erforderte.
- Von den realisierten Regeln soll auf deren Gemeinsamkeiten abstrahiert werden. Ideal ist die Implementation eines eigenen Datentyps für Regeln.

Perlen der Informatik – aus den Einsendungen

Allgemeines

Komplexibilität

Der Fakt ist nämlich der, dass man jemand möglichst Unkundigen braucht, der das Programm testet. … Also: NN *(Name der Redaktion bekannt)* ist ein eins a Programmtester.

Aufgabe 1

Indifferenzen

… wird die Rekursion durch einen Stack (der eigentlich wie eine Schlange arbeitet) aufgelöst.

Die benutzte Frequenz wird mit weißer Schrift in dieses Kästchen hinein geschrieben. … Für die Größe der Kästchen hat sich dabei eine Kantenlänge von einem Pixel als zweckmäßig herausgestellt.

Aufgabe 2

Reisgruppe

Trading Salesman Problem

… Abweichung der Drei-Rundreisen-Lösung von einer idealen Drei-Rundreisen-Lösung für eine vorgegebene Antennenkonfiguration zu berechnen.

Es ist ja vollkommen sinnlos, auf 5 Städte einen Algorithmus von 330 Codezeilen Länge anzuwenden.

Meinen ersten Parameter habe ich „Tour de France“-Parameter getauft.

Aufgabe 3

Symonoymwörterbuch

Hier sehen wir als Beispiel die Anwendung der Regel „Gedicht“. Ich habe ein Gedicht …, welches mein Leben verändert hat wie kein zweites Kunstwerk (OK, abgesehen von … und …, ja, na gut, … und … – eigentlich hat jedes Kunstwerk mein Leben mehr verändert als dieses, aber das tut jetzt nichts zur Sache.

Beispielregeln:

Boston T-Party: Jedes vorkommende Wort soll durch ein Synonym ersetzt werden, dessen Schriftform nicht den Buchstaben ’t’ enthält. Bei dieser Umsetzung der Beispielregel geht wieder ’T’ über Bord.

Ni-Hao: Jedes vorkommende Wort, das ein gesprochenes ’r’ enthält, soll durch ein Synonym ersetzt werden, das kein gesprochenes ’r’ enthält. Diese Regel folgt der Volksmeinung, dass Chinesen nicht in der Lage sind, ein ’r’ auszusprechen; ihnen kann hiermit ein Stück weit geholfen werden.