Einführung in Foundations of Computing
Vorlesungsbegleitende Prüfungsaufgabe
– Teil 1 –

Technische Universität Darmstadt

Wintersemester 2005/06

Prof. Dr. Karsten Weihe

Empfohlene Bearbeitungszeit: bis 23.12.2005

Dies ist kein Pflichttermin, aber Sie liegen goldrichtig in der Zeit, wenn Sie diesen Termin einhalten. Die zweite Aufgabe wird voraussichtlich schon vor Weihnachten herauskommen.

Allgemeiner Hinweis:

1. Diese Übungsaufgaben dienen einem einzigen Ziel: vertraut zu werden mit der abstrakten, mehrere Ebenen überspannenden Denkweise der Vorlesung ("den Knoten im Gehirn zum Platzen bringen"). Ihre wesentliche Leistung ist nicht Programmieren, sondern verstehen. Unsere Erfahrung ist, dass diese Herausforderung leicht unterschätzt wird.

2. Dieses Verständnis ist auch notwendig zum Bestehen der Abschlussklausur.[1] Sie tun sich also absolut keinen Gefallen damit, wenn Sie es doch schaffen, sich die Punkte aus den Übungen durch "Abschreiben" zu ergaunern und damit diese Gelegenheit verpassen, die Grundlagen für die notwendigen ca. 28% der Klausurpunkte zu legen.

3. Diese Zielsetzung ist auch diese Motivation hinter diesem speziellen Softwaredesign. Es ist für uns kein Thema, ob dieses Design nun für "echte" Software sinnvoll ist oder nicht.

Achtung: Wie schon in der Vorlesung gesagt, wird Java empfohlen wegen der graphischen Anteile. Wer lieber C oder C++ nehmen will, mag das tun und muss sich dann eben eine Graphikbibliothek suchen. Die Wahl der Bibliothek sprechen Sie bitte mit mir ab. Wer eine ganz andere Programmiersprache nehmen möchte, kontaktiert mich bitte rechtzeitig vorher. Die Spezifikation der Aufgaben wird sich auf Java beziehen und muss bei Wahl einer anderen Programmiersprache sinngemäß interpretiert werden.

Achtung: Korrekturen und Erläuterungen meinerseits im Forum verstehen sich als Bestandteil der Aufgabe.

Teil (a)

Zulässige Lösungen und Änderungsinformationen erscheinen auf der Abstraktionsebene, die in Teil (a) zugrundegelegt ist, generell unter dem statischen Typ `java.lang.Object`.

[1] Achtung: Daraus folgt **nicht** logisch zwingend, dass die Klausuraufgaben Ähnlichkeit mit diesen Übungsaufgaben haben!

Sie benötigen folgende Interfaces und Klassen:

1. Ein Interface, das nichtmodifizierbare Prototypen von zulässigen Lösungen repräsentiert. Ein Prototyp ist ein Paar bestehend aus einer (anderen) zulässigen Lösung und einer Änderungsinformation. Das Interface hat zwei Methoden, um beide Bestandteile einzeln zurückzuliefern, sowie eine Methode, die die implizit repräsentierte zulässige Lösung explizit konstruiert und zurückliefert.
2. Ein Iterator-Interface in Anlehnung an `java.util.Enumeration`, das allerdings kein `java.lang.Object` zurückliefert, sondern das obige Prototyp-Interface. Zudem soll dieser Iterator zyklisch sein, d.h. wenn er alle Elemente durchlaufen hat, fängt er wieder von vorne an. Er hat aber eine Methode, mit der man abfragen kann,m ob man gerade am Ende angekommen ist.
3. Ein Interface für die Konstruktion der Nachbarschaft einer zulässigen Lösung. Dieses Interface hat eine Methode, deren Implementation für eine gegebene zulässige Lösung ein Objekt des oben spezifizierten Iterator-Interface zurückliefern, das über die Nachbarn der gegebenen zulässigen Lösung läuft.
4. Zwei Interfaces für die Kostenwerte von zulässigen Lösungen. Jedes Interface hat eine Methode, deren Implementationen für eine gegebene zulässige Lösung den Kostenwert berechnet. Der Unterschied liegt in der Repräsentation der zulässigen Lösung: In einem Fall ist die zulässige Lösung explizit gegeben, im anderen Fall in Form eines Prototyps wie oben beschrieben.

Teil (b)

Sie erstellen Implementationen für die obigen Interfaces für zwei Optimierungsprobleme, die Sie in der Vorlesung kennengelernt haben: TSP (Folie 6) und Bipartition (Folie 55). Dazu müssen Sie als Basis konkrete Repräsentationen von Instanzen und zulässigen Lösungen entwickeln. Genauer gesagt, repräsentieren Sie zulässige Lösungen des TSP in zwei verschiedenen Formen: zum einen als Permutationen (Folie 6), zum anderen als Mengen von Kanten (Folie 34 ff.). Bei Bipartition ist eigentlich nur eine Form naheliegend und sinnvoll: Sie repräsentieren eine zulässige Lösung mit einer geeigneten Datenstruktur für Collections als Teilmenge der Gesamtmenge. Das machen Sie aber in zwei leicht verschiedenen Formen: In der einen Form wird die Summe der Werte $a[\cdot]$ in der Teilmenge explizit mitgespeichert (und kann für die Berechnung des Kostenwertes daher umstandlos hergenommen werden), in der anderen Form wird dieser Wert nicht mitgespeichert.

Auf dieser Basis erstellen Sie konkret:

1. Prototypenklassen für verschiedene Definitionen von Nachbarschaft: 2-OPT und 3-OPT für TSP bzw. je eine Klasse für jede der beiden Nachbarschaften für Bipartition auf Folie 55-56. Die Änderungsinformation bei 2-OPT bzw. 3-OPT würde aus den beiden bzw. drei zu entfernenden Kanten bestehen, bei den Nachbarschaften für Bipartition überlegen Sie sich analog eine geeignete Repräsentation.
2. Für jede dieser Nachbarschaften brauchen Sie eine Iteratorklasse gemäß Punkt 2 in Teil (a). Für 2-OPT und 3-OPT beim TSP müssen Sie sich überlegen, wie Sie es hinbekommen, jedes Paar bzw. Tripel von Kanten genau einmal zu durchlaufen.
   (a) Für 2-OPT durchläuft Ihr Iterator die Kanten in lexikographischer Ordnung, d.h. eine Kante (i_1, j_1) kommt vor eine Kante (i_2, j_2) "dran", wenn entweder $i_1 < i_2$ gilt, oder wenn $i_1 = i_2$ und $j_1 < j_2$ gilt.

(b) Für 3-OPT durchläuft der Iterator die Tripel von Kanten in zufälliger Reihenfolge. Dazu werden alle Tripel, die schon durchlaufen worden sind, geeignet abgespeichert (z.B. in einer Hashtabelle). Um das nächste Tripel zu berechnen, werden solange neue Tripel zufällig generiert, bis ein Tripel entsteht, das noch nicht als "schon durchlaufen" abgespeichert ist. Wenn alle Elemente abgespeichert sind, wird der Speicher wieder geleert.

(c) Für Bipartition implementieren Sie einen Iterator analog zu TSP/2-OPT.

3. Ebenso brauchen Sie für jede dieser Nachbarschaften eine Klasse für die Konstruktion der Nachbarschaft gemäß Punkt 3 in Teil (a).
4. Sie brauchen für jedes der beiden algorithmischen Probleme ein Interface zur Berechnung von Kostenwerten für explizit gegebene Lösungen und für jede Nachbarschaft ein Interface für Prototypen,

Teil (c)

Basierend auf Teil (a) implementieren Sie lokale Suche, Simulated Annealing, Tabusuche und Evolutionsstrategien.

Einige dieser Algorithmen erwarten ein zufällig gewähltes Element der Nachbarschaft. Das verallgemeinern wir zu "das Element, das vom Nachbarschaftsiterator halt als nächstes zurückgeliefert wird".

Teil (d)

Für TSP / 2-OPT schreiben Sie eine Graphikanbindung für eine der beiden Repräsentationen der Problemstellung. In einem Bildschirmfenster wird rein zufällig eine Menge an Bildschirmpunkten gewählt, die geeignet groß für eine anschauliche Visualisierung ist (bei jedem Lauf eine andere Zufallsmenge). Darauf wenden Sie die von Ihnen implementierten Algorithmen an und zeigen die Veränderungen an der Rundtour in einer angemessenen Geschwindigkeit.

Details können Sie nach eigenen Vorstellungen ausgestalten, für Fleiß oder Kreativität bei der Graphik gibt es aber keine Extrapunkte.

Einführung in Foundations of Computing
Vorlesungsbegleitende Prüfungsaufgabe
– Teil 2 –

Technische Universität Darmstadt
Wintersemester 2005/06
Prof. Dr. Karsten Weihe

Achtung:

Während es im ersten Teil der Aufgabe durchaus ok war, sich wie im Forum sehr allgemein auszutauschen, wird in diesem zweiten Teil **strikte Alleinarbeit** gefordert. Das bedeutet, dass jede aufgefallene Ähnlichkeit zwischen verschiedenen Abgaben auf jeden Fall eine ernste Befragung der Beteiligten nach sich ziehen wird, die ggf. auch zu Konsequenzen führen wird.

Vorbemerkung

Dieser zweite Aufgabenteil trägt der allgemeinen Tatsache Rechnung, dass Informatiker – insbesondere Absolventen von Universitäten – in erster Linie mit anderen Menschen und weniger mit Computern kommunizieren.[1]

Das Ergebnis dieses Aufgabenblattes soll eine PDF-Datei sein. Der Inhalt kann deutsch oder englisch sein. Mit welchem Textprogramm Sie diese Datei erstellen (Word, LaTeX, ...), bleibt Ihnen überlassen. Falls Sie Interesse haben, bei dieser Gelegenheit eine neue Seite der Informatik kennenzulernen, können Sie auch folgendes ausprobieren: `www.literateprogramming.com`.

Zielgruppe Ihres Textes sind Leute, die die Inhalte der Vorlesung gut kennen und die die von Ihnen gewählte Programmiersprache gut beherrschen, so dass Sie zu beidem keine Erläuterungen geben müssen.

Wählen Sie für die Bilder bitte ein platzsparendes Format, so dass ein einzelnes Bild möglichst nicht mehr als 100 KB groß ist, wenn möglich deutlich unter 100 KB.

Konkrete Aufgabe

Betten Sie Ihre Quelltexte aus Teil (a)-(c) des ersten Aufgabenblatts in einen erläuternden deutschen oder englischen Prosatext ein. Das heißt, Sie zerlegen Ihre Quellen in einzelne Bestandteile und schreiben Ihren Text um diese Bestandteile herum, wobei Sie rein technische Bestandteile (z.B. `import`-Anweisungen in Java) natürlich weglassen können und sollten. Die Granularität der Zerlegung (ob ganze Klassen, einzelne Methoden, Teile von Methoden usw.) und Reihenfolge der Bestandteile bleibt Ihnen überlassen.[2] Der Text soll aus Ihrer Sicht

[1] Damit ist natürlich nicht nur die schriftliche Kommunikation zwischen Student und Prüfer bei Klausuren gemeint...

[2] Es bietet sich natürlich an, hierarchisch zu strukturieren, beispielsweise in der Form, dass jede Klasse ein Oberkapitel bekommt und jede Methode der Klasse ein Unterkapitel. Aber das bleibt Ihnen überlassen.

erläutern, welche Funktion die einzelnen Bestandteile haben, einerseits in Ihrer Software, andererseits auch im Zusammenhang mit den Inhalten der Vorlesung. Machen Sie ausgiebig Gebrauch von Querverweisen auf Folien aus der Vorlesung – aber erläutern Sie Ihre Querverweise auch.

Sofern Ihr Design oder Ihre Algorithmen von den Vorgaben auf dem ersten Aufgabenblatt abweichen, schreiben Sie auch jeweils eine Erläuterung und Begründung in den Text hinein.

Die Quelltexte Ihrer GUI fügen Sie bitte nicht ein, aber eine kurze Beschreibung in Ihren eigenen Worten, wie und mit welchen technischen Mitteln Sie die GUI erstellt haben. Fügen Sie eine reichliche Zahl von aussagekräftigen „Schnappschüssen“ Ihres Programms in den Text ein und benutzen Sie diese zur Erläuterung Ihrer Software.

Umfang des Textes

Hängt naturgemäß von Ihrer Herangehensweise und Ihrem Stil ab.

Grobe Faustregel: Pro Klasse inkl. aller Methoden dürften 20 Zeilen das Minimum, dazu noch einmal vielleicht 1-2 Seiten insgesamt für übergreifenden Text inkl. Beschreibung der GUI. Hinzu kommen die Bildunterschriften.

Bewertung

Da die Texte der verschiedenen Übungsteilnehmer sehr unterschiedlich sein können, verbietet sich natürlich ein allzu enges Bewertungsschema. Daher werden nur folgende allgemeine Punkte in die Bewertung einbezogen:

- Entspricht der eingebettete Code prinzipiell der Aufgabenstellung vom ersten Aufgabenblatt.

  Falls es Abweichungen von den Vorgaben des ersten Aufgabenblattes gibt: Sind sie nachvollziehbar begründet.[3]

- Ist der gesamte Text semantisch korrekt, in sich schlüssig und mit dem Code und der Vorlesung konsistent.

- Werden die einzelnen Informationen korrekt, verständlich und formulierungsscharf auf den Punkt gebracht.

[3] Das muss nicht bedeuten, dass ich Ihrer Abweichung oder Ihrer Begründung zustimme. Ich würde selbstverständlich auch Abweichungen bzw. Begründungen auch dann schon positiv bewerten, wenn ich zwar nicht Ihrer Meinung bin, Ihre Begründung aber in sich schlüssig und nachvollziehbar ist und auch nicht den Boden der Realität verlässt.