Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

20.4.2009

1. Hausübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Abgabetermin: Montag, 27.04.2009, 10:00 Uhr

1 Algorithmenbegriff

1. Die „*Shark Investment Group*“ (SIG) droht in Folge riskanter Finanzaktivitäten zahlungsunfähig zu werden. Deshalb wurde das „Sardine Consulting Center“ damit beauftragt, einen Notfallplan zur Sanierung des Unternehmens zu entwickeln.

   Dr. Fluke stellt folgende „Strategie“ vor:

   a) Wähle zufällig ein Unternehmen *u* mit geringerem Marktwert als die SIG. Falls ein solches Unternehmen nicht existiert, breche ab und emigriere auf die Bahamas.
   b) Kaufe *u* auf und entlasse zehn Prozent der Beschäftigten.
   c) Warte bis der Marktwert von *u* um fünf Prozent gestiegen ist.
   d) Verkaufe *u*.
   e) Wiederhole Schritte a) bis d) solange ein solches Unternehmen *u* existiert.

   Handelt es sich hierbei um einen *Algorithmus*? Falls ja, ist er

   - *deterministisch* (im Ablauf)?
   - *determiniert* (im Ergebnis)?
   - *terminierend*?

   Begründen Sie jeweils kurz. Gehen Sie in Ihrer Antwort davon aus, dass die Handlungsanweisungen ausreichend präzise sind.

2. Auch in der Automobilindustrie hat man mit finanziellen Engpässen zu kämpfen. Der Ingenieur „Paul Propellerantrieb“ bekommt folgenden Auftrag:

   Entwerfe ein *schönes* Auto, das allen gefällt.

   Untersuchen Sie diese Vorgabe unter den gleichen Gesichtspunkten wie in der vorigen Aufgabe.

3. Um das Leistungspotential der Mitarbeiter zu erhöhen, soll der Kaffee-Konsum in den Mittagspausen der SIG gesteigert werden. Hierzu soll in jeder Küche eine Anleitung zur Zubereitung von Kaffee ausgehängt werden.

   Entwerfen Sie einen Algorithmus zur Zubereitung von Kaffee, welcher sich in jeder der Eigenschaften *deterministisch*, *determiniert* und *terminierend* von der unter 1. angegebenen „Strategie“ unterscheidet.

(10 Punkte)

1. Ein nichtdeterministischer, indeterminierter, nicht terminierender Algorithmus:

   Nichtdeterministisch: In Schritt a wird Unternehmen *u zufällig* ausgewählt.

   Indeterminiert: Abhängig von der jeweiligen Wahl von *u* in Schritt a kann bei gleicher Ausgangssituation die Strategie zu unterschiedlichen Ergebnissen führen.

   Nicht terminierend: Wird in einer Iteration in Schritt a ein Unternehmen *u* ausgewählt und in Schritt d wieder verkauft, so kann dieses in einer folgenden Iteration wiederholt ausgewählt werden, falls der Marktwert von *u* unter dem von SIG liegt.

   (Wer die Endlichkeit der menschlichen Existenz voraussetzt, kann an dieser Stelle natürlich argumentieren, dass irgendwann kein Unternehmen mehr existiert und der Algorithmus entsprechend terminierend ist.)

   Dazu kommt, dass die Finanzwelt nicht nur durch diese eine Strategie verändert wird. M.a.W., es gibt nebenläufige Prozesse, die die Ausführung dieses Algorithmus nichtdeterministisch, indeterminiert und ggf. nicht terminierend machen.

2. Es ist kein Algorithmus, da die Beschreibung nicht ausreichend präzise die einzelnen Schritte zur Abarbeitung definiert.

3. Individuelle Lösung.

2 Programmieren, Laufzeitmessung, Anagramme

Als *Anagramm* bezeichnet man ein Wort, das aus einem anderen Wort durch Vertauschen der Buchstaben gebildet wurde. So entsteht z.B. das Wort "ANGSTBUDE" aus dem Wort "BUNDESTAG" durch Permutation der Buchstaben. Weitere Beispiele sind FAUL – LAUF oder FERIEN – REIFEN. Die Java-Klasse `Anagramm.java`, die sie sich von der Vorlesungswebseite herunterladen können, berechnet bei Eingabe eines Wortes über die Konsole alle möglichen Anagramme und gibt sie auf dem Bildschirm aus.

- Passen Sie die Klasse wie folgt an: Bis zu einer Wortlänge von 8 Zeichen sollen die Anagramme ausgegeben werden; enthält die Eingabe mehr Zeichen, soll keine Ausgabe erfolgen. Die Ausgabe des Programms soll also im ersten Fall so aussehen:

```
Bitte ein Wort eingeben: eis
Eingabelänge 3 <= 8, berechnete Anagramme werden angezeigt:
1 eis
2 esi
3 ise
4 ies
5 sei
6 sie
Gesamtanzahl der Anagramme: 6
```

Im zweiten Fall (Eingabelänge mehr als 8 Zeichen) soll die Ausgabe so aussehen:

```
Bitte ein Wort eingeben: bundestag
Eingabelänge 9 > 8, berechnete Anagramme werden nicht angezeigt.
Berechnung der Anzahl läuft...
Gesamtanzahl der Anagramme: XXX
```

(10 Punkte)

- Die Funktion `System.currentTimeMillis()` liefert die aktuelle Systemzeit in Milisekunden zurück. Verwenden Sie diese Funktion, um eine Laufzeitmessung der Anagramm-Berechnung durchzuführen. Messen Sie hierbei die Laufzeit abhängig von der Länge der Eingabe; verwenden Sie hierzu Eingaben der Länge 2 bis 12. Passen Sie die Klasse `Anagramm.java` dazu so an, dass die Berechnungszeit mit ausgegeben wird:

  ```
  Bitte ein Wort eingeben: longinput
  Eingabelänge 9 > 8, berechnete Anagramme werden nicht angezeigt.
  Berechnung der Anzahl läuft...
  Gesamtanzahl der Anagramme: XXX
  Berechnungszeit: XXX ms.
  ```

  Fassen Sie Ihre Ergebnisse in einer Tabelle der Form

Eingabelänge	2	3	...	12
Anzahl Anagramme				
Laufzeit				

  zusammen. (5 Punkte)

- Wie könnten Sie die Berechnung der Anzahl der Anagramme beschleunigen? Beschreiben Sie ihre Idee und modifizieren/erweitern Sie die Java-Klasse dementsprechend. Sie können vereinfachend annehmen, dass die Eingabe immer aus einem Wort besteht, in dem sich kein Buchstabe wiederholt.

  Ein Wort, in dem jeder Buchstabe gleich oft (z.B. ein Mal) einmal vorkommt, nennt man *Isogramm*. Das längste deutsche (Fantasie-)Isogramm ist *Heizölrückstoßabdämpfung* (24 Buchstaben). Nehmen Sie an, die Berechnung eines seiner Anagramme würde 0.000001 Sekunden (also eine Millionstel Sekunde) dauern; wie lange würde die Berechnung aller Anagramme für dieses Isogramm dauern? (5 Punkte)

- Überlegen Sie sich ein Verfahren, um in einem Text Anagramme zu finden. Das Ziel hierbei ist es, in einem Text alle Wortpaare zu finden, die gegenseitig Anagramme sind. Vereinfachend betrachten wir Texte ohne Satzzeichen in denen sämtliche Wörter kleingeschrieben sind. Im folgenden Satz:

```
in den ferien kaufte papa neue reifen f"ur sein auto
```

sollte das Verfahren also `ferien - reifen` entdecken. Versuchen Sie hierbei, ein möglichst effizientes Verfahren zu finden. Sie müssen das Verfahren nicht in Java implementieren; eine detaillierte Beschreibung der einzelnen Schritte genügt.

(10 Punkte)

3 Hinweise zu den Programmieraufgaben und zur Abgabe

Um zu der Prüfung zur Lehrveranstaltung „*Algorithmen und Datenstrukturen*“ zugelassen zu werden, müssen über alle Hausaufgaben summiert am Ende des Semesters 50% der Punkte erreicht werden und zu jedem bis auf einem Aufgabenblatt eine „vernünftige“ Lösung *fristgerecht* abgebeben werden. In begründeten Ausnahmefällen ist eine entsprechende „Abmeldung“ (Krankenschein, etc.) bei uns im Sekretariat zeitnah abzuliefern.
Eine weitere „verpasste“ Abgabe kann durch Bearbeitung eines zusätzlichen Aufgabenblatts ausgeglichen werden, welches im Laufe des Semesters ausgeteilt wird.
Grundsätzlich ist jeder Teilnehmer der Lehrveranstaltung angehalten, im Rahmen der Hausaufgaben eigene Leistung zu erbringen und sich mit jeder Aufgabe auseinanderzusetzen. Häufig ist es jedoch sinnvoll, über Probleme zu diskutieren und gemeinschaftlich Lösungen zu erarbeiten. Deshalb ist es möglich, Aufgaben als Gruppe zu *zweit* abzugeben, sofern folgendes Verfahren eingehalten wird:

- Ein Team-Mitglied reicht die Lösung über das Abgabesystem ein.
- Das gleiche Team-Mitglied schickt an Herrn Eisterlehner eine E-Mail mit dem Betreff *###AlgoDS:<Nr. des Aufgabenblattes>:<Eigene Matrikel>:<Matrikel des Partners>*
- Beispiel
  - Aufgabe: 1
  - Matrikelnr. 1: 170012
  - Matrikelnr. 2: 250098
  - Betreff: *###AlgoDS:01:170012:250098*
- Das zweite Team-Mitglied sieht sein Ergebnis am Montag der Folgewoche (also bis zu eine Woche nach dem ersten Team-Mitglied) im Abgabesystem.
- Bitte überprüfen Sie das Ergebnis innerhalb von zwei Tagen.

Wird dagegen von einem Teilnehmer eine fremde Lösung kopiert oder lediglich leicht modifiziert, so wird dies als *Täuschungsversuch* gewertet und per E-Mail entsprechend mitgeteilt. Ein zweiter Täuschungsversuch führt zum Ausschluss von der Prüfung. Ebenfalls als Täuschungsversuch wird gewertet, wenn eine Abgabe offensichtlich nicht ausreichend umfangreich bearbeitet wurde oder mutwillig beschädigte bzw. ungültige Dateien abgegeben werden.

Grundsätzlich werden (abhängig von der Aufgabenstellung) nur folgende Dateiformate akzeptiert:

- `.java` — für JAVA-Quellcode
- `.txt`,`.pdf` — für Text
- `.jpg`, `.png` — für Bilder/Zeichnungen

Geben Sie Ihre Lösungen stets als *jar*-Archiv ab mit möglichst flacher Dateihierarchie (vermeiden Sie Unterverzeichnisse).
Programmieraufgaben sind stets in Form von *syntaktisch korrekten* JAVA-Quelldateien zu lösen. Diese müssen sich mit dem JAVA-Kommandozeilen-Compiler von SUN in der Version 1.5 direkt übersetzen lassen und in der JAVA-Laufzeitumgebung von SUN (ebenfalls Version 1.5) lauffähig sein. Geben Sie mit an, wie Ihr Programm zu starten ist.
Wie in der Vorlesung „Einführung in die Programmierung“ auch, ist ein konsistenter *Programmierstil* einzuhalten [1]. Grobe Inkonsistenzen und unleserliche bzw. ungenügend kommentierte Programme führen zu Punkteabzug.
Bei Fragen bzw. Anmerkungen zu konkreten Korrekturen, wenden Sie sich bitte direkt an den jeweiligen Korrektoren (dieser wird zufällig zugeteilt). Für allgemeine Fragen wenden Sie sich bitte an Dominik Benz (`benz@cs.uni-kassel.de`) oder Folke Eisterlehner (`eisterlehner@cs.uni-kassel.de`).

Viel Spaß und viel Erfolg!

[1] siehe `http://www.plm.eecs.uni-kassel.de/plm/fileadmin/pm/courses/einfProg08/einfProgAufg2.pdf`