Übungen zu Informatik IIa (Sommersemester 2010)

Oliver Jack
Fachhochschule Jena
Fachbreich Elektrotechink und Informationstechnik

März 2010

1 Organisatorisches

Dieses Dokument beschreibt den Inhalt und das Verfahren der *Alternativen Prüfungsleistung* für die Lehrveranstaltung *Informatik IIa* im Sommersemester 2010.

- Bilden Sie *Projektgruppen* gemäß Set-Einteilung mit einer Sollstärke von *vier Studenten*.
- Wählen Sie pro Projektgruppe eine der in Abschnitt 4 folgenden drei Software-Entwicklungsprojekte aus.
- Die Projektgruppen und die Wahl der Software-Entwicklungsprojekte wird in der zweiten Übung festgelegt.
- Lösen Sie die gewählte Aufgabe in Gruppenarbeit bis zum *vorletzten Übungstermin*.
- *Präsentieren Sie Ihre Lösung.* Stellen Sie in der Präsentation Ihren Entwurf, die Realisierung, die qualitätssichernden Maßnahmen vor und führen Sie Ihre Software vor. Jedes Projektgruppenmitglied präsentiert einen Teil der Lösung. Für die Präsentation stehen pro Gruppe *20 Minuten inklusive Beantwortung von Fragen* zur Verfügung.
- Die Präsentationen finden an den letzten beiden Übungsterminen statt. Die Übungstermine finden Sie unter `http://stundenplanung.fh-jena.de/`.
- Geben Sie pro Projektgruppe *spätestens zum Präsentationstermin* eine *schriftliche Ausarbeitung* Ihrer Projektarbeit ab. Die Abgabe erfolgt *ausschließlich per E-Mail* an oliver.jack@fh-jena.de unter Angabe des Betreffs „Info IIa APL SS

2010“. Im Text der E-Mail geben Sie bitte Ihre Übungsgruppe, Ihre Projektgruppennummer und alle Projektgruppenmitglieder mit Namen und Matrikelnummer an. Bitte geben Sie nur eine Datei im ZIP- oder RAR-Format ab. Die schriftliche Ausarbeitung muss folgende Teile enthalten:

1. Präsentationsfolien im PDF-Format,
2. kommentierten Quell-Code im Textformat (ASCII) oder annotierten Quell-Code im doxgen-html-Format,
3. Testfälle im Tabellendokument (ODS oder Microsoft Excel),
4. Testprotokoll im Tabellendokument (ODS oder Microsoft Excel),
5. Zuordnung der Arbeitspakete zu den einzelnen Projektgruppenmitgliedern.

- Weitere Informationen und Hinweise zu Lehrveranstaltung und der Alternativen Prüfungsleistung finden Sie auf der Website `http://www.et.fh-jena.de/jack/` bzw. `http://www.fh-jena.de/~jack/`.

2 Bewertungskriterien

- Vortrag und Präsentation (35%)
  - Vortrag: Inhalt / Substanz / Qualität (50%)
  - Vortrag: Vermittlung (50%)
- Fachlicher Teil (65%)
  - Entwurf (25%)
  - Implementierung (25%)
  - Funktionsfähigkeit der Lösung (25%)
  - Qualitätssicherung (25%)

3 Hinweise zur Bearbeitung

- Beginnen Sie bei der Bearbeitung der Aufgabe mit der Modellierung der Datenstrukturen (Klassen), nutzen Sie möglichst viele Konzepte der objektorientierten Programmierung (Datenkapselung, Polymorphie, Vererbung etc.).
- Teilen Sie die Bearbeitung der Aufgabe auf die einzelnen Projektgruppenmitglieder auf, nachdem Sie die Klassenstruktur und die Schnittstellen der Klassen (Member-Variablen und -Funktionen) festgelegt haben, so dass die Implementierung der einzelnen Klassen arbeitsteilig erfolgen kann.

4 Software-Entwicklungsprojekte

4.1 Genetische Algorithmen

Such- und Optimierungsalgorithmen, die nach den Gesetzen der natürlichen Auslese und genetischer Fortpflanzung funktionieren. Anwendung auf alle Systeme, deren inneres Verhalten nicht bekannt, bzw. nicht modelliert ist.

- Kodierung der Eingaben (Individuen) in Bit-Strings
- Bewertung der Individuen (Ergebnis zu den Eingaben) mit einer Fitness-Funktion
- Selektion der *Besten*
- Auswahl von *Elternpaaren*
- Reproduktion der Population durch *Crossover* und *Mutation*

Schreiben Sie ein C++-Programm, das einen genetischen Algorithmus gemäß nachfolgender Beschreibung implementiert. Halten Sie Ihr Programm flexibel, d. h., sehen Sie eine Konfigurierung der Parameter des genetischen Algorithmus vor (Länge der Bit-Strings, Größe der Population, Mutationswahrscheinlichkeit). Implementieren Sie das System so, dass verschiedene Fitness-Funktionen einsetzbar sind. Implementieren Sie ein Testsystem, z. B. erzeugen Sie eine Tabelle mit ca. 100.000 Zufallszahlen, die Sie als Funktion interpretieren. Wenden Sie Ihr Programm an, um mit einer zufällig gewählten Start-Population den Maximalwert der Funktion zu bestimmen.

Funktionsweise des genetischen Algorithmus

Ziel ist die Optimierung einer Zielfunktion f (Fitness-Funktion). Jedes Individuum wird als ein Bit-String kodiert. Eine Population ist eine Menge von Individuen. Als Beispiel betrachte die (zufallserzeugte) Population

```
01101
11000
01000
10011
```

Im folgenden werden die drei Operationen

1. Reproduktion
2. Crossover
3. Mutation

beschrieben.

Reproduktion ist der Prozess, in dem Individuen gemäß der Fitness-Funktion f behandelt werden. Die Individuen werden so bewertet, dass sie eine um so höhere Wahrscheinlichkeit erhalten, zur folgenden Generation beizutragen, je höher der Wert

ihrer Fitness-Funktion ist. Tabelle 1 zeigt die Werte für die Beispiel-Population. Die Einzelwerte der Fitness-Funktion werden summiert (1170) und den Individuen werden damit Prozentwerte zugewiesen, die die Wahrscheinlichkeit der Auswahl für die folgende Population angeben. Damit ergibt sich für die Auswahl des Strings Nr. 1 für die

Nr.	String	Fitness	%
1	`01101`	169	14,4
2	`11000`	576	49,2
3	`01000`	64	5,5
4	`10011`	361	30,9
Total		1170	100,0

Tabelle 1: Population und Fitness-Werte

folgende Population eine Wahrscheinlichkeit von 0,144. Wird als Beispiel die Funktion $f(x) = x^2$ maximiert, so sind die Werte für die Reproduktion in Tabelle 2 dargestellt. Damit ist die Grundlage für die folgende Population geschaffen.

Nr.	String	x-Wert unsigned int	Fitness $f(x) = x^2$	W'keit $\frac{f(x)}{\sum f(x)}$	E.-Wert $f(x)/\bar{f}$	Rundung (Anz. in neuer Pop.)
1	`01101`	13	169	0,14	0,58	1
2	`11000`	24	576	0,49	1,97	2
3	`01000`	8	64	0,06	0,22	0
4	`10011`	19	361	0,31	1,23	1
Summe			1170	1,00	4,00	4,0
Mittel			293	0,25	1,00	1,0
Max			576	0,49	1,97	2,0

Tabelle 2: Reproduktion und Fitness-Werte

Nach der Reproduktion werden die Individuen der folgenden Population mittels Crossover bestimmt. Dazu werden aus der Reproduktion Paare von Individuen (Eltern) per Zufallsauswahl gebildet. Ein solches Paar kann z. B. aus den Strings Nr. 1 und 2 bestehen. Zu jedem Paar wird per Zufall ein *Crossover-Punkt* bestimmt. Dies ist eine Bit-Position k der Strings zwischen 1 und $l-1$, wobei l die Länge der Strings beschreibt (alle Strings sind gleich lang). Zwei neue Strings werden nun konstruiert, in dem die Bits der Positionen $k+1$ bis l zwischen den beiden Strings eines Paares getauscht werden, z. B. für $k = 2$ (die Teilstrings rechts vom Symbol `|` werden getauscht).

alt	`01|101`	neu	`01|000`
	`11|000`		`11|101`

Tabelle 3 zeigt das Ergebnis des Crossover aus der Reproduktion. Die Menge der ausgewählten Individuen, die zum Crossover zu Paaren zusammengefasst werden, heißt *Mating-pool*. Der Mating-pool besteht aus einmal dem String Nr.1, zweimal dem String

Nr. 2 und einmal dem String Nr. 4 aus der ursprünglichen Population (siehe die letzte Spalte in der Tabelle in Tabelle 2).

Mating-pool	Crossover-Punkt	Neue Pop.	x-Wert	$f(x) = x^2$
`0110|1`	4	`01100`	12	144
`1100|0`	4	`11001`	25	625
`11|000`	2	`11011`	27	729
`10|011`	2	`10000`	16	256
Summe				1754
Mittel				439
Max				729

Tabelle 3: Crossover und neue Population

Es ist zu erkennen, dass die neue Population ein besseres Ergebnis bzgl. der Zielfunktion $f(x)$ liefert, sowohl der Mittelwert (Steigerung von 293 auf 439) als auch der Maximalwert (Steigerung von 576 auf 729) ist verbessert.

Das Verfahren wird schließlich noch um die Mutation erweitert. Das bedeutet, dass für einen (großen) Wert n jede n-te Bit-Stelle der neuen Population invertiert wird (von 0 nach 1 bzw. von 1 nach 0). Dies geschieht nach dem Zufallsprinzip. Ist etwa $n = 1000$, so ist für jedes Bit der Population die Wahrscheinlichkeit der Invertierung $0,001$. Im Beispiel besteht die Population aus $4 \cdot 5 = 20$ Bits, also ist zu erwarten, dass $20 \cdot 0,001 = 0,02$ Bits invertiert werden. Im obigen Beispiel tritt demnach keine Mutation auf.

4.2 Flughafensimulation

Implementieren Sie die Simulation des Betriebs eines Flughafens gemäß nachfolgender Beschreibung.

Durch Zufallserzeugung erscheinen Flugzeuge, die einen Service des Flughafens in Anspruch nehmen wollen (Starten und Landen). Die *Air Control* ist für Flugzeuge, die sich in der Luft befinden zuständig, sie leitet auf Landung wartende Flugzeuge über dem Flughafen um. *Ground Control* ist für Flugzeuge zuständig, die sich am Boden befinden und leitet diese Flugzeuge um. *Airport* verwaltet die Warteschlangen und Kontrollen des Verkehrs in der Luft und am Boden. Ihr unterliegt die Kontrolle der Rollbahn. Die Zeit der Flugzeiten und Verzögerungen werden durch eine globale Zeit simuliert. Das System ist als Scheduling-Problem mit Warteschlangen zu implementieren. Sehen Sie für die Flugzeuge Flugnummern vor, die gemäß ihrem zufälligen Auftreten jeweils in zwei Warteschlangen für startende und landende Flugzeuge eingetragen werden. Sehen Sie einen Maximalwert der Länge der Warteschlangen vor (z. B. 50). Sehen Sie für landende Flugzeuge Maximalzeiten vor, sie diese Flugzeuge in der Luft bleiben können. Falls die maximale Wartezeit überschritten ist, stürzen die Flugzeuge ab. Sehen sie folgende Ausgaben der Simulation vor

- „Flug ... kommt zur Landung an.“

  Fluzeug kann in die Lande-Warteschlange aufgenommen werden.

- „Flug ... umgeleitet.“

  Flugzeug kann nicht in die Lande-Warteschlange aufgenommen werden, da diese voll ist.

- „Flug ... Landung verzögert.“

  Flugzeug ist am Anfang der Lande-Warteschlange, kann aber nicht landen, da die Rollbahn durch ein startendes Flugzeug blockiert ist.

- Flug ... ist abgestürzt.“

  Flugzeug ist am Anfang der Lande-Warteschlange, hat aber die Maximalzeit in der Luft überschritten.

- „Flug ... landet.“

  Flugzeug ist am Anfang der Lande-Warteschleife und die Rollbahn ist frei oder Flugzeug war lande-verzögert und hat die Maximalzeit in der Luft nicht überschritten.

- „Flug ... verlässt Gate.“

  Flugzeug kann in die Start-Warteschlange aufgenommen werden.

- „Flug ... Start verzögert.“

  Flugzeug ist am Anfang der Start-Warteschlange, kann aber nicht starten, da die Rollbahn durch ein landendes Flugzeug blockiert ist.

- „Flug ... startet.“

  Flugzeug ist am Anfang der Start-Warteschlange und die Rollbahn ist frei oder Flugzeug war start-verzögert und die Rollbahn ist frei.

Sehen Sie Zeiten vor, die für das Starten und das Landen benötigt werden. Sehen eine Zeit vor, die für die Abfertigung eines Flugzeuges am Boden benötigt wird, bevor es zum Start freigegeben wird.

Sehen Sie eine Maximalzeit für die Öffnung des Flughafens vor. Nach verstreichen dieser Zeit werden keine Flugzeuge mehr in die Warteschlangen aufgenommen und die verbleibenden Flugzeuge werden abgearbeitet, danach schließt der Flughafen (Ende des Programms).

Führen Sie Versuche mit Ihrem Programm mit verschiedenen Werten für die Länge der Warteschlangen, Häufigkeit des Auftauchen neuer Flugzeuge, Maximalzeiten in der Luft und Abfertigungszeiten durch. Beobachten Sie das Verhalten des Systems und versuchen Sie Parameter zu finden, so dass möglichst wenig Abstürze vorkommen.

4.3 System zur algebraischen Manipulation von Matrizen

Implementieren Sie ein generisches System zur algebraischen Behandlung von Matrizen. Folgende Operationen sollen realisiert werden:

- Matrix-Addition, -Subtraktion und -Multiplikation,
- Skalarmultiplikation,
- Vektor-Multiplikation,
- Gauß-Elimination (Lösung von linearen Gleichungssystemen),
- Determinantenberechnung,
- Matrix-Inversion.

Implementieren Sie das System so, dass Matrizen verschiedener Dimension, auch nichtquadratische behandelt werden können (für die Determinantenberechnung und die Matrixinversion sind die Matrizen als quadratisch vorausgesetzt).

Implementieren Sie das System als Container-Klasse, d. h. unabhängig von den Datentypen der Matrixelemente, der Vektorelemente und der Saklare. Sie können dabei auch auf die Standard-Template-Library zurückgreifen (z. B. vector).

Implementieren Sie die Funktionen der Matrix-Addition, -Subtraktion und -Multiplikation als überladene Operatoren $+$, $-$ und $*$

Implementieren Sie die Ein- und Ausgabefunktionen für Matrizen und Vekroren durch Überladen der Stream-Operatoren $>>$ und $<<$.

Ihr System muss mit den elementaren Datentypen `float` und `double` und den in der Vorlesung und im Praktikum behandelten Klassen der *Komplexen Zahlen* und der *Brüche* funktionieren. Ein einfacher Beispiel-Test sähe z. B. folgendermaßen aus:

```
#include <iostream>
#include "Matrix.h"  // Ihre Matrix-Klasse
#include "Complex.h" // Komplexe Zahlen
#include "Bruch.h"   // Brüche

using namespace std;

int main() { // Test
  // Fließkomma-Matrizen
  float f;
  // 3x4-, 4x2-, 3x3-Matrizen
  Matrix<float> mf1(3,4), mf2(4,2), mf3;
  cin >> f >> mf1 >> mf2;
  cout << mf1 * f * mf2 << endl; // (Skalar)Multiplikation
  cout << mf3.det() << endl; // Determinante
  cout << mf3.hls() << endl; // Lösung des homogenen LGS
```

```
  // Komplex-Matrizen
  Complex c;
  // 3x4-, 4x2-, 3x3-Matrizen
  Matrix<Complex> mc1(3,4), mc2(4,2), mc3;
  cin >> c >> mc1 >> mc2;
  cout << mc1 * c * mc2; // (Skalar)Multiplikation
  cout << mc3.det() // Determinante
  cout << mc3.hls() // Lösung des homogenen LGS

  // Bruch-Matrizen
  // ...
}
```

Die hier gemachten Annahmen über die Konstruktoren und Member-Funktionen der Matrix-Klasse können Sie an Anregung für Ihre Implementierung auffassen.