Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Frühjahr
2009

46114

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Algorithmen / Datenstrukturen / Programmiermethoden**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 7

Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

Diese Aufgabe diskutiert die korrekte Abarbeitung einer Folge von Operationen, die untereinander Reihenfolgeabhängigkeiten besitzen. Ein naheliegendes Beispiel liefert der alltägliche Vorgang des Ankleidens: Einen Mantel sollte man erst dann anziehen, wenn man das Sakko bereits angezogen hat.

Etwas genereller betrachten wir eine Menge von n Operationen $O = \{o_1, o_2, \ldots, o_n\}$. Eine Reihenfolgeabhängigkeit zwischen zwei derartigen Operationen notieren wir als $[o_x, o_y]$ wobei $x, y \in \{1, \ldots, n\}, x \neq y$: Operation o_x muss zwingend vor Operation o_y durchgeführt werden.

Mit Bezug auf das obige Beispiel könnten wir O als

$$\{\texttt{Mantel}, \texttt{Sakko}, \texttt{Unterhemd}, \texttt{Unterhose}, \texttt{Strümpfe}, \texttt{Hemd}, \texttt{Krawatte}, \texttt{Hose}, \texttt{Hut}\}$$

festlegen (`Sakko` steht dabei bspw. für die Operation *Sakko anziehen*). Der Alltag lehrt uns, dass folgende Reihenfolgeabhängigkeiten das Ankleiden deutlich erleichtern:

[`Hemd`, `Krawatte`]

[`Sakko`, `Mantel`]

[`Unterhemd`, `Hemd`]

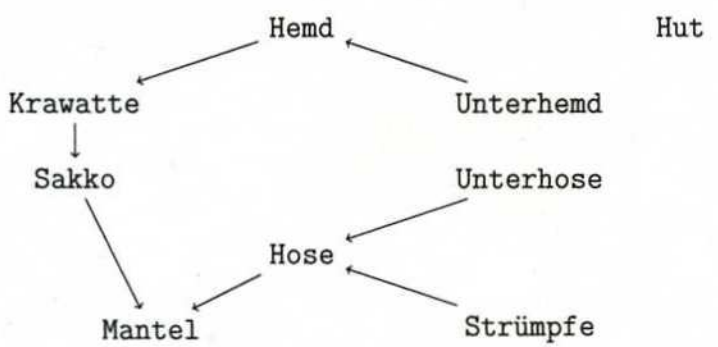
[`Unterhose`, `Hose`]

[`Strümpfe`, `Hose`]

[`Krawatte`, `Sakko`]

[`Hose`, `Mantel`]

Diese Abhängigkeiten werden äquivalent auch durch folgenden gerichteten Graphen beschrieben:



a) Geben Sie für das obige Beispiel mindestens drei verschiedene Folgen der neun Operationen an, wobei jeweils *alle* angegebenen Reihenfolgeabhängigkeiten berücksichtigt werden. Die zuerst auszuführenden Operationen sollen Sie dabei zuerst aufführen.

Fortsetzung nächste Seite!

Notieren Sie die Folgen in folgender Form: [`Unterhemd`, `Hemd`, `Krawatte`, ...].

Skizzieren Sie verbal kurz (max. 100 Worte), welche Systematik Sie bei der Ermittlung der gültigen Folgen anwenden.

Formulieren Sie die folgenden Teilaufgaben in einer prozeduralen Programmiersprache Ihrer Wahl (bspw. Java, C, Pascal, ...).

b) Definieren Sie geeignete Datenstrukturen, die Ihnen die Repräsentation der Operationenmenge O und der Menge von Reihenfolgeabhängigkeiten $[o_x, o_y]$ erlauben. Gehen Sie dabei davon aus, dass die Elemente von O durch Zeichenketten zu repräsentieren sind.

c) Geben Sie eine Prozedur P an, die – basierend auf den in Teil b) definierten Datenstrukturen – eine gültige Folge der Operationen in O als Ausgabe generieren kann (dazu sind die Elemente von O einfach durch ',' getrennt auszugeben).

d) Führen Sie eine Analyse der Laufzeit der in Teil c) definierten Prozedur P durch. Nehmen Sie dabei an, dass $N = |O|$ und M die Anzahl der vorgegebenen Reihenfolgeabhängigkeiten bezeichnet.

Aufgabe 2:

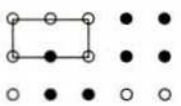
Stellen Sie sich den Sitzplan eines Kinosaales vor. Der Kinosaal hat R Reihen mit jeweils C Sitzen (also eine rechteckige Sitzanordnung). Auf dem Sitzplan sei ein freier Sitz jeweils mit ∘, ein bereits reservierter Sitz mittels • markiert. Für $R = 3, C = 5$ könnte sich bspw. also der folgende Sitzplan ergeben:

∘ ∘ ∘ • •
∘ • ∘ • •
∘ • • ∘ ∘

In diesem Kinosaal möchten K Freunde gemeinsam einen Film anschauen und dazu Sitzplätze reservieren. Die K Freunde möchten gern so nah wie möglich zusammensitzen: Die *Verteilung der Sitze* im Saal soll minimal sein. In dieser Aufgabe messen wir die *Verteilung von K Sitzen wie folgt*:

die Fläche des kleinsten Rechtecks im Sitzplan, das mindestens K freie Sitze (∘) enthält.

Nehmen wir bspw. $K = 5$ an. Für das obige Beispiel ergibt sich dann eine Verteilung von 6:



Formulieren Sie die folgenden Teilaufgaben in einer prozeduralen Programmiersprache Ihrer Wahl (bspw. Java, C, Pascal, ...).

a) Definieren Sie eine geeignete Datenstruktur, die einen rechteckigen Sitzplan mit R Reihen und C Spalten repräsentieren kann. R und C seien als Konstanten vorgegeben.

Überlegen Sie sich dazu, wie Sie einen einzelnen Sitz und seinen Reservierungsstatus (frei/belegt) geeignet darstellen. Möglichkeiten sind etwa ein Boolescher Wert oder eine ganze Zahl (0: belegt, 1: frei) pro Sitz.

Fortsetzung nächste Seite!

b) Gegeben sei eine ganze Zahl $1 \leqslant K \leqslant RC$. Definieren Sie eine Funktion *Verteilung*, die – basierend auf der in Teil a) entworfenen Datenstruktur – die Verteilung von K Sitzen, wie oben definiert, bestimmt (die Ausgabe der Funktion ist also lediglich eine ganze Zahl). Sollte der Sitzplan eine Reservierung von K Sitzen nicht mehr zulassen, soll die Funktion den Wert 0 liefern.

c) Führen Sie eine Analyse der Laufzeit der Funktion *Verteilung* durch.

d) Diskutieren Sie eine alternative Repräsentation der Datenstruktur aus Teil a) unter der Annahme, dass das Produkt RC zu groß wird, um alle Sitze gleichzeitig im Hauptspeicher Ihres Rechners zu halten.

- Welche Komprimierungstechniken lassen sich auf den Sitzplan anwenden?
- Welche Teile des Sitzplans müssen jeweils tatsächlich im Hauptspeicher vorhanden sein, um die Funktion *Verteilung* zu berechnen?
- Sei ein konkretes K gegeben. Wie kann diese Information genutzt werden, um Teile des Sitzplans als nicht relevant zu identifizieren?

Thema Nr. 2
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

Gegeben ist die folgende rekursive Funktionsdefinition zur Berechnung der Fibonacci Zahlen:

```
function fib(n: nat): nat;
  if n = 0 then 0
  else if n = 1 then 1
       else fib(n-1) + fib(n-2) endif
  endif
```

a) Geben Sie alle Paare `(n, fib(n))` an, für die gilt `fib(n) < 30`.

b) Die Fibonacci Funktion soll nun durch einen nicht-rekursiven Algorithmus realisiert werden. Implementieren Sie dazu in der Programmiersprache `Java` eine nicht-rekursive Methode mit Kopf `int fibJava(int n)`, die für jedes $n \geq 0$ die Fibonacci Zahl `fib(n)` als Ergebnis liefert.

Aufgabe 2:

Wir betrachten den Anwendungsbereich des Lottospiels. Die folgenden Aufgaben sind in der Programmiersprache `Java` zu lösen. Dabei ist jeweils das Geheimnisprinzip zu befolgen, d.h. alle verwendeten Instanzvariablen sollen privat sein.

a) Ein Lottotipp besteht aus 6 Zahlen zwischen 1 und 49. Im folgenden werden Lottotipps durch Objekte der Klasse `Lottotipp` repräsentiert, wobei die Instanzvariable `zahlen` die 6 Zahlen des Tipps beinhaltet. Es kann immer angenommen werden, dass `zahlen` nur voneinander verschiedene Werte zwischen 1 und 49 enthält.

```
class Lottotipp {
  private int[] zahlen = new int[6];
}
```

Erweitern Sie die Klasse `Lottotipp` um eine Methode mit Kopf
`public boolean kommtVor(int x)`, die feststellt, ob die Zahl `x` unter den Zahlen eines Lottotipps vorkommt.

b) Erweitern Sie die Klasse `Lottotipp` um eine Methode mit Kopf
`public int richtige(Lottotipp t)`, die berechnet, wieviele Zahlen eines Lottotipps auch in einem anderen Lottotipp `t` vorkommen.

c) Auf jedem Lottoschein sollen neben der Scheinnummer genau ein Tipp sowie der Name und die Adresse des Lottospielers angegeben sein. Deklarieren Sie eine geeignete Klasse `Lottoschein`, deren Objekte Lottoscheine repräsentieren. Für den lesenden Zugriff auf die Instanzvariablen der Klasse `Lottoschein` sind entsprechende, öffentliche Zugriffs-Methoden zu implementieren.

d) Bei einer Lottoziehung wurden 6 Gewinnzahlen und eine Zusatzzahl gezogen. Deklarieren Sie eine geeignete Klasse `Lottoziehung`, deren Objekte Lottoziehungen repräsentieren. Für

Fortsetzung nächste Seite!

die Gewinnzahlen kann dabei der Klassentyp `Lottotipp` verwendet werden. Für den lesenden Zugriff auf die Instanzvariablen der Klasse `Lottoziehung` sind entsprechende, öffentliche Zugriffs-Methoden zu implementieren.

e) Gegeben sei die folgende Klasse `Lotto`.

```
class Lotto {
  public static gewinner(Lottoschein[] liste, Lottoziehung z)
  { ... }
}
```

Gewinner einer Lottoziehung sind alle Spieler, die einen Lottoschein vorgelegt haben, auf dem mindestens drei Gewinnzahlen richtig getippt wurden. Implementieren Sie den Rumpf der oben angegebenen Methode `gewinner`, so dass die Methode für eine gegebene `liste` von Lottoscheinen und eine gegebene Lottoziehung `z` den Namen jedes Gewinners, die Anzahl der von ihm/ihr richtig getippten Gewinnzahlen und eine Information, ob die Zusatzzahl getippt wurde, ausgibt. Für die Ausgabe von Strings kann der Befehl `System.out.print(..);` bzw. `System.out.println(..);` verwendet werden. Zur Umwandlung einer Zahl `z` des Typs `int` in einen String kann der Befehl `Integer.toString(z);` verwendet werden.

Aufgabe 3:

Eine Bibliothek hat einen Namen und besitzt eine Menge von Büchern und einen Kundenstamm. Ein Buch hat einen Titel, den Namen seines Autors und eine Inventarnummer. Es kann an einen Kunden ausgeliehen sein. Ein Kunde hat einen Namen und eine Adresse.

a) Geben Sie ein UML Klassendiagramm mit geeigneten Klassen und Assoziationen an, das den oben beschriebenen Sachverhalt modelliert. Die Assoziationen sollen unidirektional ausgerichtet sein und an ihren ausgerichteten Enden mit Multiplizitäten und Rollennamen versehen sein. Für jede Klasse sind die Attribute anzugeben, wobei jedem Attribut ein geeigneter Typ, entweder `String` oder `Integer`, zuzuordnen ist.

b) Implementieren Sie das in Teil a) angegebene Klassendiagramm in Java. Zur Speicherung von mehreren Büchern oder Kunden kann die Klasse `Vector` verwendet werden. Alle Instanzvariablen sollen privat sein. Für jede Klasse soll es einen Konstruktor geben mit geeigneten Parametern zur Initialisierung der in dem Klassendiagramm angegebenen Attribute der jeweiligen Klasse.

Ein Buch soll folgende Operationen durchführen können, die in Java zu realisieren sind:

c) Den Namen seines Autors liefern.

d) Den Kunden liefern, der das Buch entliehen hat. (Falls das Buch nicht verliehen ist, wird der Wert `null` zurückgegeben.)

e) An einen Kunden ausgeliehen werden.

f) Einen Booleschen Wert liefern, der anzeigt, ob das Buch verliehen ist.

Eine Bibliothek soll folgende Operationen durchführen können, die in Java zu realisieren sind:

g) Für einen gegebenen Namen eines Autors ein nicht verliehenes Buch dieses Autors zurückgeben. Falls kein solches Buch existiert, wird der Wert `null` zurückgegeben.

Fortsetzung nächste Seite!

h) Für einen gegebenen Kunden und Autornamen ein nicht verliehenes Buch dieses Autors an den Kunden entleihen und den Booleschen Wert `true` zurückgeben. Falls kein Buch dieses Autors frei ist, wird der Wert `false` zurückgegeben.

Hinweis: Für den Zugriff auf Vektoren können die Methoden `int size()` (liefert die Größe eines Vektors) und `Object get(int i)` (liefert das i-te Element eines Vektors) verwendet werden.