Sammlung aller Staatsexamensaufgaben der Prüfungsnummer

46114

Algorithmen / Datenstrukturen / Programmiermethoden (nicht vertieft)

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Herbst 1998

Prüfungsteilnehm.	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____ Kennwort: _____ Arbeitsplatz-Nr.: _____	**Herbst 1998**	**46114**

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (nicht vertieft studiert)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 5

Bitte wenden!

Thema Nr. 1

Sämtliche Teilaufgaben sind zu bearbeiten!

Aufgabe 1:

Eine Geldbörse enthält eine Menge Me von Münzen Mü (mit Wert w(Mü)).
(Die Münzen können verschiedene Werte haben.)

a) Beschreiben Sie einen Algorithmus, der zu einem Geldbetrag g eine Teilmenge T von Münzen aus Me liefert, deren Wert gerade g beträgt (falls es eine solche Teilmenge T gibt)!

b) Ist entscheidbar, ob es zu g eine Teilmenge T mit Wert g gibt? (Begründung!)

c) Formulieren Sie Ihren Algorithmus aus a) in einer Programmiersprache ihrer Wahl!

d) Beschreiben Sie die in a) bzw. c) verwendeten Datenstrukturen!

Aufgabe 2:

Geben Sie die Grammatik $G=(V,T,P,S)$ mit Variablenmenge $V=\{S,A,B\}$, terminalem Zeichenvorrat $T=\{a,b\}$, dem Startsymbol S und der Produktionenmenge $P=\{S\to AB, S\to AaBb, S\to AaaBbb, A\to aaaA, A\to\varepsilon, B\to bbbB, B\to\varepsilon\}$.

a) Welchen Typ hat die Grammatik?

b) Bestimmen Sie die von G erzeugte Sprache L(G)!

c) Ist L(G) regulär?

Aufgabe 3:

Betrachten Sie die rekursiv definierte Wortcodierungsfunktion
$code: \{a,b\}^* \to \mathbb{N}$ mit
$code(\varepsilon)=0, code(a)=1, code(b)=2,$
$code(w\sigma)=2\cdot code(w)+code(\sigma) \quad (w\in\{a,b\}^*, \sigma\in\{a,b\})$.

a) Berechnen Sie code(abb)!

b) Zeigen Sie, daß code bijektiv ist!

c) Definieren Sie die Umkehrfunktion $decode = code^{-1}$, oder geben Sie einen Algorithmus für decode an!

d) Berechnen Sie decode(11)!

Thema Nr. 2

Sämtliche Teilaufgaben sind zu bearbeiten!

Aufgabe 1:

Für einen endlichen nicht-leeren Zeichenvorrat Z sei eine Binärcodierung

$$c : Z \rightarrow \{O, L\}^*$$

gegeben.

1.1 Geben Sie eine Datenstruktur (durch Typdefinitionen) an, mit deren Hilfe ein beliebiger Binärcode c im Hinblick auf die folgende Teilaufgabe 1.2 geeignet dargestellt werden kann!

1.2 Schreiben Sie eine Funktionsprozedur *fano*, mit deren Hilfe festgestellt werden kann, ob ein Binärcode c der Fano-Bedingung genügt! Die Funktionsprozedur *fano* soll dabei folgender Spezifikation genügen:

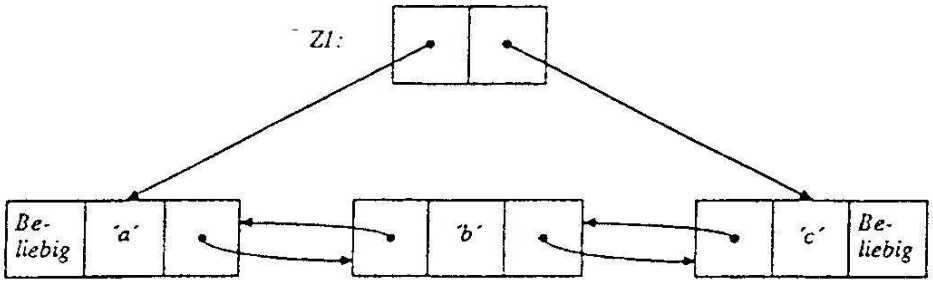
Eingabeparameter: c vom Typ *CODE*

Resultat: Das Resultat ist vom Typ *BOOLEAN* und hat den Wert *TRUE* genau dann, wenn c der Fano-Bedingung genügt.

Hinweise:
- Der Typ *CODE* muß in Ihrer Antwort zu 1.1 definiert werden.
- Ein Code erfüllt die Fano-Bedingung genau dann, wenn in ihm kein Codewort Anfang eines von ihm verschiedenen Codewortes ist.

Aufgabe 2:

Zeichenreihen über dem endlichen Alphabet $A = ('a','b',\ldots,'z')$ sollen durch vorwärts und rückwärts verkettete Listen dargestellt werden, für deren Anfang und Ende Anker (Verweise) vorgesehen sind. Die Darstellung der Zeichenreihe $z1='abc'$ läßt sich also beispielsweise folgendermaßen graphisch veranschaulichen:



Fortsetzung nächste Seite!

2.1 Geben Sie eine Datenstruktur mit Hilfe geeigneter Typdefinitionen an, durch die sich die o.a. Darstellung von Zeichenreihen realisieren läßt!

2.2 Wie stellen Sie die leere Zeichenreihe ϵ dar?

2.3 Als Länge einer Zeichenreihe bezeichnet man die Anzahl der Zeichen, die sie enthält, wobei mehrfach vorkommende Zeichen entsprechend ihrer Häufigkeit gezählt werden. Die Länge der leeren Zeichenreihe ϵ ist 0.
Schreiben Sie eine Funktionsprozedur *laenge*, die die Länge einer Zeichenreihe berechnet, also folgender Spezifikation genügt:

2.3.1 Eingabeparameter: $x1$ vom Typ *ZEICHENREIHE*

Hinweis: *ZEICHENREIHE* muß als Typ in Ihrer Antwort zu 2.1 definiert sein.

2.3.2 Der Funktionswert ist vom Typ *INTEGER* und ist gleich der Länge von $x1$.

Hinweis: Für die leere Zeichenreihe ϵ soll die in Ihrer Antwort zu 2.2 angegebene Darstellung verwendet werden.

2.4 Schreiben Sie zwei Funktionsprozeduren *anfang* und *teil*, die feststellen, ob x Anfangs- bzw. Teilzeichenreihe einer Zeichenreihe y ist! Die entsprechenden Spezifikationen lauten:

2.4.1 Eingabeparameter (für beide Funktionsprozeduren): x, y vom Typ *ZEICHENREIHE*

2.4.2 Der Funktionswert ist jeweils vom Typ *BOOLEAN*. Er hat den Wert *TRUE* für die Funktion *anfang* bzw. *teil* genau dann, wenn x Anfangs- bzw. Teilzeichenreihe von y ist.

Hinweis: Es ist empfehlenswert, erst die Funktionsprozedur *anfang* zu schreiben und in der Funktionsprozedur *teil* die Funktion *anfang* zu verwenden. Die Funktionsprozedur *laenge* darf verwendet werden.

2.5 Ein Palindrom ist eine Zeichenreihe, die mit ihrer rückwärts gelesenen identisch ist. Beispiele sind

'otto', *'reliefpfeiler'*, ϵ (leere Zeichenreihe).

Schreiben Sie eine Funktionsprozedur *palin*, die feststellt, ob eine Zeichenreihe ein Palindrom ist, also folgender Spezifikation genügt:

2.5.1 Eingabeparameter: $x1$ vom Typ *ZEICHENREIHE*

2.5.2 Die Funktion ist total, d.h. sie liefert für jede Eingabe-Zeichenreihe einen eindeutig bestimmten Wert. Der Funktionswert ist vom Typ *BOOLEAN* und ist genau dann gleich *TRUE*, wenn $x1$ ein Palindrom ist.

Hinweis: In der Funktionsprozedur *palin* darf die unter 2.3 spezifizierte Funktion *laenge* verwendet werden.

Fortsetzung nächste Seite!

Aufgabe 3:

Gegeben seien die Standard-Rechenstrukturen $\mathbb{Z}$ der ganzen Zahlen und $\mathbb{B}_2$ der Wahrheitswerte. Wie üblich seien dabei der Typ der Elemente von $\mathbb{Z}$ mit *INTEGER* und der Typ der Elemente von $\mathbb{B}_2$ mit *BOOLEAN* bezeichnet. Als Operationen sind

$+,\ -,\ *,\ /$ (partiell!)	in $\mathbb{Z}$
$\neg,\ \wedge,\ \vee$	in $\mathbb{B}_2$

sowie die Vergleichsoperationen

$<,\ =$	für Elemente aus $\mathbb{Z}$

standardmäßig vorgesehen.

Durch Abstützung auf die Rechenstrukturen $\mathbb{Z}$ und $\mathbb{B}_2$ soll die im folgenden spezifizierte Teil-Rechenstruktur $\mathbb{Q}_1$ der rationalen Zahlen $\mathbb{Q}$ vollständig dargestellt werden.

3.1 Geben Sie eine Datenstruktur an, mit deren Hilfe sich jede Zahl aus $\mathbb{Q}$ im Hinblick auf die folgenden Teilaufgaben geeignet darstellen läßt! Geben Sie außerdem an, welchen Einschränkungen die Objekte der von Ihnen vorgeschlagenen Datenstruktur genügen sollen, damit sie in $\mathbb{Q}$ zulässig sind!

3.2 Schreiben Sie Prozeduren für die Multiplikation (*mult*) und die Division (*div*) in $\mathbb{Q}$. Diese Prozeduren sollen den folgenden Spezifikationen genügen:

3.2.1 Eingabeparameter: a, b: *RATIONAL*
Dabei dürfen Sie unterstellen, daß die von Ihnen in Ihrer Antwort zu 3.1 angegebenen Einschränkungen für in $\mathbb{Q}$ zulässige Objekte von a und b erfüllt sind! Bei *div* dürfen Sie außerdem unterstellen, daß b nicht die Null darstellt!

3.2.2 Resultatparameter: VAR c: *RATIONAL*
Der Wert von c muß nach Ausführung von *mult* gleich $a * b$ und nach Ausführung von *div* gleich a/b sein. Dabei muß dieser Wert jeweils den Zulässigkeitseinschränkungen, die Sie in Ihrer Antwort zu 3.1 formuliert haben, genügen!

3.3 Schreiben Sie eine Funktionsprozedur *gr*, die feststellt, ob eine Zahl a aus $\mathbb{Q}$ größer ist als eine Zahl b aus $\mathbb{Q}$! Dabei soll *gr* folgender Spezifikation genügen:

3.3.1 Eingabeparameter: a, b: *RATIONAL*
Sie dürfen wieder unterstellen, daß a und b den Zulässigkeitseinschränkungen gemäß 3.1 genügen!

3.3.2 Das Resultat ist vom Typ *BOOLEAN* und hat den Wert *TRUE* genau dann, wenn $a > b$ ist.

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Frühjahr 2000

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Frühjahr**	
Kennwort: _____	**2000**	**46114**
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (nicht vertieft studiert)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 5

Bitte wenden!

Thema Nr. 1

Sämtliche Teilaufgaben sind zu bearbeiten!

Teilaufgabe 1:

Realisieren Sie in einer imperativen Sprache Ihrer Wahl den Typ der ganzen Zahlen als 2-Tupel ihres absoluten Betrages (Typ CARDINAL für IN_0) und einer BOOLEAN-Komponente, die das Vorzeichen angibt!

Geben Sie für die so dargestellten ganzen Zahlen Prozeduren für die Grundrechenarten Addition, Subtraktion und Multiplikation an, die die entsprechenden Operationen auf CARDINAL verwenden!

Teilaufgabe 2:

Jemand gibt nachfolgenden Algorithmus vor.

Analysieren Sie dessen genaue, teilweise „unerwartete" Wirkung!

```
TYPE Liste = POINTER TO Elem;
     Elem  = RECORD obj    : CARDINAL;
                    nachf  : Liste  END;

PROCEDURE Unbekannt(l:Liste; a:CARDINAL);
VAR y : Liste;
BEGIN
  IF l = NIL THEN NEW(y); y^.obj := a;
                          y^.nachf := NIL;
                          l := y
  ELSE IF (a<l^.obj) THEN NEW(y); y^.obj  := a;
                                  y^.nachf := l;
                                  l := y
       ELSE Unbekannt(l^.nachf,a)
       END
  END
END Unbekannt
```

Teilaufgabe 3:

Schreiben Sie ein Programm in einer imperativen Sprache Ihrer Wahl, das die Inverse einer (einzulesenden) 2x2-Matrix reeller Zahlen berechnet (und ausgibt), falls die Matrix invertierbar ist und ansonsten eine Fehlermeldung ausgibt! Eine Matrix der Form $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ ist invertierbar, falls $ad - bc \neq 0$ und die Inverse lautet dann $\begin{pmatrix} e & f \\ g & h \end{pmatrix}$, wobei mit $D = ad - bc$ (Determinante) gilt:

- $e = \frac{d}{D}$
- $f = \frac{-b}{D}$
- $g = \frac{-c}{D}$
- $h = \frac{a}{D}$

Thema Nr. 2

Sämtliche Teilaufgaben sind zu bearbeiten!

Teilaufgabe 1: Programmiermethodik

(1a) Ein typisches Strukturierungsmodell bei der Programmentwicklung ist die schrittweise Verfeinerung. Was versteht man darunter?

(1b) Eine Form, die schrittweise Verfeinerung in der Praxis auszunutzen, besteht in der Verwendung der Struktogramme (Nassi-Shneidermann-Diagramme). Aus welchen Elementen sind diese zusammengesetzt?

(1c) Die gängigen Programmiersprachen unterscheiden drei Typen von Schleifen:
- die indizierende Schleife: `for Var := Expr1 to Expr2 do ...`
- die abweisende iterative Schleife: `while LogExpr do ...`
- die nichtabweisende iterative Schleife: `repeat ... until LogExpr`

Wie unterscheiden sich diese Schleifentypen? (Erläutern Sie die Abarbeitungsweise und geben Sie typische Randbedingungen für den Einsatz der unterschiedlichen Typen an.)

(1d) In der Hoare-Semantik kann die Wirkung der abweisenden iterativen Schleife durch folgende Formel beschrieben werden:

$$\frac{B \wedge P \qquad \{S\} \qquad P}{P \qquad \{\texttt{while}\ B\ \texttt{do}\ S\} \quad P \wedge \neg B}$$

Erläutern Sie diese Formel anschaulich, und entwickeln Sie eine entsprechende Formel für die nichtabweisende iterative Schleife!

Teilaufgabe 2: Systementwurf

Ein aktuelles Thema in der Diskussion um die Strukturierung von Softwaresystemen ist die Objektorientierung.

(2a) Erläutern Sie die Begriffe Klasse, Objekt und Methode unter Zuhilfenahme des folgenden (in einer Phantasiesprache formulierten) Beispiels:

```
class konto is
   eroeffne_konto (kunde);
   einzahlen (betrag);
   auszahlen (betrag);
   kontostand () returns betrag;
end konto;
```

(`betrag` und `kunde` seien geeignet definierte Klassen.) Was versteht man unter Instanzenvariablen?

Fortsetzung nächste Seite!

(2b) Das Beispiel werde durch die folgenden Definitionen ergänzt:

```
class girokonto inherits from konto
   setze_dispo_limit (betrag);
   dispo_limit () returns betrag;
end girokonto;
class sparkonto inherits from konto
   setze_zinssatz (real);
   schreibe_zinsen_gut ();
end sparkonto;
```

Erläutern Sie den Begriff der Vererbung, und geben Sie die vollständige Liste der Methoden an, die für Objekte der Klassen Girokonto und Sparkonto definiert sind!

(2c) Was versteht man unter einer virtuellen Funktion? (Hinweis: In unserem Beispiel könnte man `auszahlen` als solche auffassen, da sie bei Girokonten zu einem negativen Saldo führen darf, bei Sparkonten nicht.)

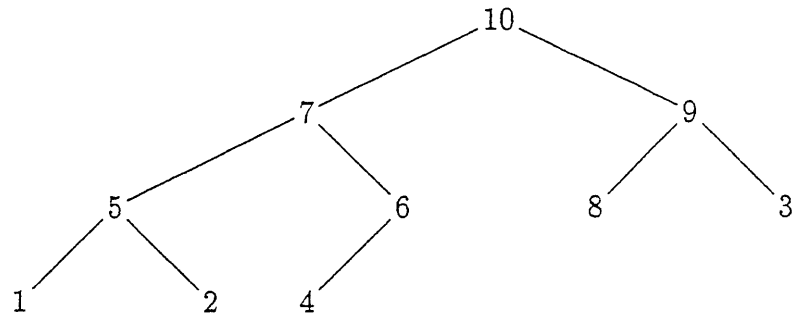
(2d) Erläutern Sie an Hand der virtuellen Funktionen, was man unter „dynamischem Binden" versteht, und dessen Vorteil bei der Systementwicklung!

Teilaufgabe 3: Algorithmen und Datenstrukturen

Beim Heapsort-Verfahren stellt man sich vor, die in einer Reihung (array) gespeicherten Daten seien als Baum organisiert. Beispielsweise denkt man sich die Reihung

$$10, 7, 9, 5, 6, 8, 3, 1, 2, 4$$

in der folgenden Form gegeben:



(3a) Wann nennt man einen Baum einen „Heap"?

(3b) Beschreiben Sie das Einfügen eines neuen Elementes in den Heap!

(3c) Wie ist der Heap zu reorganisieren, wenn das größte Element (d.h. das Element an der Wurzel) entfernt wird?

Fortsetzung nächste Seite!

Teilaufgabe 4: Anwendungen

Verwenden Sie die dem Heapsort zugrundeliegende Idee, um eine Liste von einzuhaltenden Fristen (Terminen) effizient zu verwalten:

- Der Zugriff auf den als nächstes zu bearbeitenden Vorgang (d.h. die nächste ablaufende Frist) soll in konstanter Zeit möglich sein.
- Das Streichen einer abgelaufenen (bearbeiteten) Frist und das Neueintragen sollen in logarithmischer Zeit möglich sein, also nicht das Durchsuchen der gesamten Liste erfordern.

(4a) Begründen Sie, warum die Heap-Struktur diese Anforderungen erfüllt, wobei als Sortierkriterium das Datum des Fristablaufs verwandt wird!

(4b) Es kann vorkommen, dass sich eine Frist vorzeitig erledigt. Welchen Aufwand erfordert es, eine solche Frist zu streichen? (Beachten Sie, dass Sie sie zuerst suchen müssen, wobei Sie selbstverständlich nur das Aktenzeichen kennen!)

(4c) Beschreiben Sie eine andere Datenstruktur, bei der das Suchen und Eintragen von Vorgängen (Sortierkriterium: Aktenzeichen) in logarithmischer Zeit möglich ist! (Es gibt verschiedene Lösungen.)

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Herbst 2000

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Herbst**	
Kennwort: _____	**2000**	**46114**
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (nicht vertieft studiert)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 5

Bitte wenden!

Thema Nr. 1

1. Algorithmen und Datenstrukturen

Ein binärer Baum ist entweder leer, oder er besteht aus einer Wurzel und zwei binären Bäumen als linkem und rechtem Unterbaum.

Ein binärer Suchbaum für die Schlüsselmenge $S = \{s_1, s_2, \ldots, s_n\}$ ist ein binärer Baum mit n Knoten $\{ v_1, v_2, \ldots, v_n\}$ und einer bijektiven Beschriftung $c(v)$: $V \rightarrow S$ der Knoten, so dass für jeden Knoten v folgende Eigenschaft erfüllt ist: Für alle Knoten v' im linken Unterbaum und alle Knoten v'' im rechten Unterbaum gilt: $c(v') < c(v) < c(v'')$.

(1a) Beschreiben Sie verbal, ggf. unter Zuhilfenahme einer Skizze, das Suchen nach einem Knoten mit vorgegebenem Schlüssel!
Woran erkennt der Algorithmus, dass kein Element mit diesem Schlüssel im Baum enthalten ist?
Welche charakteristische Größe des Baumes bestimmt die Suchzeit?

(1b) Beschreiben Sie genauso das Einfügen eines neuen Knotens, der durch einen bisher nicht benutzten Schlüssel charakterisiert ist!

(1c) Die Reihenfolge, in der Knoten in einen ursprünglich leeren Suchbaum eingetragen werden, kann wesentlich die spätere Suchzeit beeinflussen.
Welche beiden Extremfälle können auftreten, und was kann man in beiden Fällen über die Suchzeit sagen?

(1d) Soll man einen Knoten mit vorgegebenem Schlüssel aus dem Baum entfernen, so wird er zuerst gesucht. Handelt es sich um einen Knoten mit maximal einem nichtleeren Unterbaum, so kann er einfach entfernt werden.
Was ist zu tun, wenn der zu entfernende Knoten zwei nichtleere Unterbäume besitzt?

2. Programmiermethodik

Mit folgender Datenstruktur können in C binäre Suchbäume dargestellt werden:

```
struct Node {
  struct Node *left, *right;  /* linker bzw. rechter Unterbaum */
  int    key;                 /* Sortierschluessel */
  info   data;                /* zugehoerige Information */
};
```

oder in PASCAL:

```
TYPE nodepointer = ^node;
     node = RECORD left, right: nodepointer;
                   key:         integer;
                   data:        info
            END;
```

(Der Datentyp `info` sei geeignet definiert.)

Fortsetzung nächste Seite!

(2a) Ein typisches Strukturierungsmodell bei der Programmentwicklung im Kleinen ist die Rekursion.
Erläutern Sie diese verbal anhand der Suche nach einem Knoten mit vorgegebenem Schlüssel in einem binären Suchbaum!

(2b) Schreiben Sie diesen Suchvorgang in C oder PASCAL auf!
(Bemerkung: Sie dürfen C durch C++ oder Java, PASCAL durch MODULA oder OBERON ersetzen, müssen dann aber die oben angegebene Datenstruktur neu definieren.)

(2c) Wenn Sie das Suchprogramm günstig geschrieben haben, ist es ein Beispiel für die sogenannte Endrekursion.
Was versteht man darunter, und was ist ihr Vorteil?

(2d) Schreiben Sie (in der von Ihnen gewählten) Programmiersprache eine rekursive Prozedur, die die im Baum auftretenden Schlüssel als Reihung in aufsteigender Folge abspeichert!
Leiten Sie das Konzept Ihres Programmes zunächst aus der Definition des binären Suchbaumes her!

3. Systementwurf

Wir betrachten ein einfaches Beispiel für einen datenstrukturorientierten Programmentwurf.

(3a) Die folgende formale Grammatik in Backus-Naur-Form (BNF) beschreibt das prinzipielle Aussehen einer Rechnung:

Rechnung	::=	*Datum* *Rumpf* *Gesamtsumme*
Rumpf	::=	*Position* | *Rumpf* *Position*
Position	::=	*Menge* *Bezeichnung* *Einzelpreis* *Gesamtpreis*
	|	1 *Bezeichnung* *Gesamtpreis*

Erläutern Sie die verwendeten Elemente der BNF!
Woran liegt es, dass man mit einer endlichen Beschreibung einen unendlichen Vorrat an Rechnungen beschreiben kann?

(3b) Die erweiterte Backus-Naur-Form (EBNF) kennt die Stern-Operation.
Verwenden Sie diese, um aus der zweiten Zeile des Beispiels die Rekursion zu entfernen!

(3c) Entwickeln Sie aus der EBNF-Beschreibung des Beispiels unmittelbar eine Programmstruktur, z.B. als Struktogramm!

(3d) Fassen Sie allgemein zusammen, wie man aus einer mit EBNF beschriebenen Datenstruktur unmittelbar die Programmstruktur erzeugen kann! (Geben Sie zu jeder Konstituenten der EBNF die entsprechende programmiersprachliche Konstruktion an!)

Thema Nr. 2

Für die Realisierung der Programme in den folgenden Aufgaben können Sie eine beliebige höhere Programmiersprache wählen. Geben Sie die gewählte Programmiersprache an! Für alle Aufgaben ist die gleiche Programmiersprache zu wählen. Kommentieren Sie Ihre Programme so gut wie möglich!

Aufgabe 1

a) Erläutern Sie die Parameterübergabemechanismen Call-by-value und Call-by-reference!

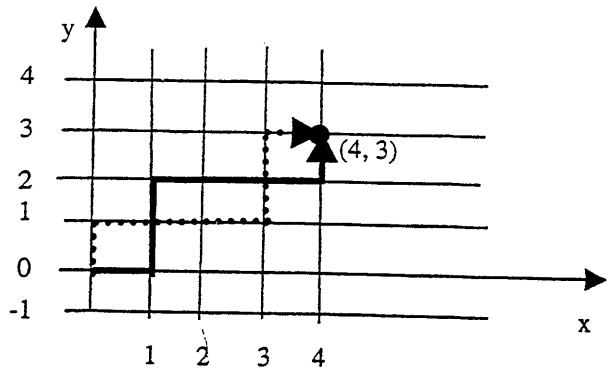
b) Was sind programmiermethodisch die Vor- und Nachteile dieser Parameterübergabemechanismen für die Übergabe von Grunddatentypen wie `INTEGER` und von strukturierten Datentypen wie z.B. Felder und Verbunde?

c) Gegeben sei eine Operation f mit einem formalen Parameter a vom Typ `ARRAY OF INTEGER`. Erklären Sie, ob und wie die Wahl des Parameterübergabemechanismus Einfluss auf die Zeit- und Speicherplatzkomplexität der Implementierung von f haben kann!

Aufgabe 2

a) Erklären Sie die Begriffe rekursive Funktion und rekursiver Datentyp!

b) Gegeben sei ein zweidimensionales Raster mit ganzzahligen Koordinaten. Ein Weg zwischen zwei Rasterpunkten verlaufe nur auf dem Raster und zwar für jedes Teilstück in Laufrichtung von links nach rechts oder von unten nach oben.



Die Abbildung zeigt zwei mögliche Wege vom Ursprung (0, 0) zum Punkt (4, 3).
Schreiben Sie in einer höheren Programmiersprache ein rekursives Programm

```
alleWege(i: INTEGER, j: INTEGER): INTEGER
```

das die Anzahl aller möglichen Wege vom Ursprung (0, 0) zum Punkt (i, j) berechnet!

c) Beweisen Sie, dass Ihr Programm für alle ganzen Zahlen i, j terminiert!

Fortsetzung nächste Seite!

Aufgabe 3

a) Definieren Sie in einer höheren Programmiersprache einen Datentyp `Farbpunkt` für farbige Punkte im zweidimensionalen Raum! Ein farbiger Punkt besitze die Koordinaten `x_coord`, `y_coord` und eine Farbe. Als Farben seien nur `rot` und `schwarz` zugelassen.

b) Schreiben Sie ein Programm für die Operation

```
kleiner(p: Farbpunkt, q: Farbpunkt): Boolean
```

das genau dann *true* liefert, wenn p einen kleineren (euklidischen) Abstand vom Ursprung hat als q!

c) Schreiben Sie ein Programm für die Operation

```
minSchwarz (a: ARRAY OF Farbpunkt): Farbpunkt
```

das als Resultat einen kleinsten schwarzen Punkt aus a liefert, sofern a mindestens einen schwarzen Punkt enthält!

d) Welche Möglichkeiten kennen Sie, um `minSchwarz` robust zu gestalten, so dass `minSchwarz` auch dann ein aussagekräftiges Ergebnis liefert, wenn a keinen schwarzen Punkt enthält?

Aufgabe 4

Ein Autokonzern investiert für die Entwicklung eines neuen Modells einen Betrag *I*. Der Konzern möchte berechnen, nach wie vielen Jahren sich der Investitionsbetrag amortisiert hat. Dabei wird von folgenden fest gegebenen Daten ausgegangen:

- Herstellungskosten für ein Auto im Jahr: 22500,00 DM,
- Erhöhung der Herstellungskosten nach jedem Jahr um 1,5 %,
- Verkaufspreis für ein Auto in den ersten drei Jahren: 25000,00 DM,
- Erhöhung des Verkaufspreises nach jeweils drei Jahren um 7,0 %.

Der Reingewinn eines verkauften Autos ist die Differenz des Verkaufspreises und der Herstellungskosten (im aktuell betrachteten Jahr). Der Konzern geht davon aus, dass jedes Jahr eine gleich bleibende Anzahl *Anz* von Autos verkauft wird.

Schreiben Sie ein Programm, das den Investitionsbetrag *I* und die Anzahl *Anz* der pro Jahr verkauften Autos einliest und daraufhin berechnet und ausgibt, nach wie vielen Jahren der durch den Verkauf des neuen Modells insgesamt entstandene Reingewinn den Investitionsbetrag erreicht oder übertroffen hat!

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Frühjahr 2001

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Frühjahr**	
Kennwort: _____	**2001**	**46114**
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (nicht vertieft studiert)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 6

Bitte wenden!

Thema Nr. 1

Vorbemerkung:

Die in der folgenden Aufgabe geforderten Programmierungsteile dürfen Sie wahlweise entweder in *Java* oder in *Pascal* ausführen. Dabei soll die Kartei, auf die sich die Programme beziehen, als ein globaler Parameter behandelt werden.

Aufgabe:

Es soll eine einfache elektronische Bibliothekskartei erstellt und mit den wichtigsten Funktionen ausgestattet werden. Für jedes Buch sind dabei die folgenden vier Daten vorzusehen:

- Familienname des Autors
- Vorname des Autors
- Buchtitel
- Signatur für den Aufbewahrungsort

Alle Daten sollen als *strings* dargestellt werden. Wenn Sie mit *Pascal* arbeiten, dürfen Sie annehmen, dass für diese Daten jeweils 50 Zeichen genügen; in *Java* ist hierfür keine Einschränkung erforderlich.

Die Zahl der in der Kartei jeweils erfassten Titel sei n.

Die Einträge in der Kartei seien eindeutig durch die in ihr erfassten Buchtitel, d.h. zu einem Buchtitel sei jeweils höchstens ein Eintrag in der Kartei enthalten.

Im Einzelnen sind folgende Teilaufgaben zu bearbeiten:

1. Beschreiben Sie genau, mit welcher Datenstruktur Sie die Kartei darstellen werden, wenn der Zugriff primär über den Familiennamen des Autors erfolgen soll, und zwar mit einer Komplexität, deren Ordnung im allgemeinen $O(\log n)$ und nur im schlechtesten Fall $O(n)$ sei! Beachten Sie dabei aber, dass der Name eines Autors im Allgemeinen mehrfach auftritt und ein Zugriff nur in Verbindung mit dem Buchtitel eindeutig ist (s.o.)!

2. Programmieren Sie einschließlich der für die Aufgabe erforderlichen Datenstruktur (nach 1.) die folgenden Prozeduren (*Pascal*) bzw. Methoden (*Java*):

   a) Eine Prozedur (*Pascal*) bzw. eine Instanz-Methode (*Java*) ***eintragen*** mit 4 *string*-Parametern für den Familiennamen des Autors, den Vornamen des Autors, den Buchtitel und die Signatur. Falls dieser Titel in der Kartei noch nicht enthalten ist, soll er eingefügt werden. Andernfalls bleibt die Kartei unverändert.

Fortsetzung nächste Seite!

b) Eine Funktionsprozedur (*Pascal*) bzw. eine Instanz-Methode (*Java*) ***suche*** mit 2 *string*-Parametern für den Familiennamen des Autors und den Buchtitel. Als Ergebnis soll die zugehörige Signatur geliefert werden, wenn das gesuchte Buch in der Kartei erfasst ist. Andernfalls soll das Ergebnis der string *„nicht vorhanden"* sein.

3. Die Datenstruktur soll nun so ergänzt werden, dass auch ein Zugriff über den Buchtitel allein, der ja eindeutig ist, möglich wird. Auch für die Komplexitätsordnung dieses sekundären Zugriffs wird gefordert, dass sie im Allgemeinen $O(\log n)$ und nur im schlechtesten Fall $O(n)$ sei. Beschreiben Sie genau, wie Sie diese ergänzende Datenstruktur darstellen!

4. Beschreiben Sie genau, wie das Programm ***eintragen*** (2.a) erweitert werden muss, damit die durch die sekundäre Zugriffsstruktur ergänzte Datenstruktur jederzeit konsistent bleibt!

5. Programmieren Sie die in Teilaufgabe 4 beschriebene Erweiterung für die Prozedur (*Pascal*) bzw. die Instanz-Methode (*Java*) ***eintragen***!

Thema Nr. 2

Aufgabe 1: *(Systementwurf und Programmiermethodik)*

Eine Galaxie besteht aus mehreren Himmelskörpern. Es gibt hier nur zwei Arten von Himmelskörpern: Sterne und Planeten. Jeder Himmelskörper wird durch einen Namen (als Zeichenkette) identifiziert. Für Sterne wird zusätzlich ihre Helligkeitsstufe (durch eine ganze Zahl gegeben) gespeichert, für Planeten wird zusätzlich die Eigenschaft gespeichert, ob es dort Wasser gibt. Ein Planet kreist direkt um genau einen Himmelskörper und „kennt" nur den Himmelskörper, um den er direkt kreist. Jeder Planet kreist indirekt um genau einen Stern, d.h. dass für jeden Planeten die transitive Hülle der „Umkreist"-Relation immer genau einen Stern enthält.

a) Geben Sie ein passendes Klassendiagramm zur Modellierung von Galaxien an, mit den entsprechenden Klassen (d.h. `Galaxie`, `Himmelskoerper`, `Stern`, `Planet`), Aggregationen, Assoziationen und ihren Multiplizitäten!

b) Übersetzen Sie das Klassendiagramm nach Java!

c) Implementieren Sie die folgende Methode in der Klasse `Planet`!

```
public String starName() {...}
/** gibt einen String mit dem Namen des Sterns zurück, um den der Planet
(this) kreist. */
```

Fortsetzung nächste Seite!

Aufgabe 2: *(Datenstrukturen und Algorithmen)*

Die folgende Java-Klasse `ListIntElem` beschreibt die Struktur der Listenelemente einer verketteten Liste von ganzen Zahlen:

```
public class ListIntElem
{ private int data;
  private ListIntElem next;

  public int getData()
  { return data;
  }

  public ListIntElem getNext()
  { return next;
  }

  public void chainTo(ListIntElem o)
  { next = o;
  }
}
```

Die folgende unvollständige Java-Klasse soll einfach verkettete Listen implementieren:

```
public class LinkedIntList
{ private ListIntElem head;

  public int max()
  {
    ...
  }

  public void reduce()
  { ...
  }

  public LinkedIntList reverse()
  { ...
  }
}
```

a) Implementieren Sie die Methode `max`, die für jede nicht leere Liste `l` (d.h. `l.head != null`) die größte Zahl findet, die Attributwert eines Elements von `l` ist!

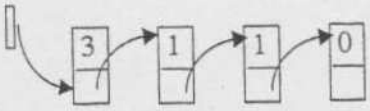
b) Eine Liste heißt aufsteigend geordnet, wenn jedes Element kleiner oder gleich seinem Nachfolger ist (sofern es den Nachfolger gibt). Implementieren Sie die Methode `reduce`, die aus einer aufsteigend geordneten Liste alle Wiederholungen entfernt!

Fortsetzung nächste Seite!

c) Implementieren Sie die Methode `reverse`, die eine Liste umdreht, d.h. eine neue Liste erzeugt, deren Elemente in umgekehrter Reihenfolge angeordnet sind!

Beispiel: Für ein Argument der Form

liefert die Methode `reverse`:



Aufgabe 3: *(Zeitkomplexität)*

Die folgende Java-Methode sortiert einen Array von ganzen Zahlen, indem durch Vertauschen benachbarter Elemente die jeweils größeren nach hinten wandern.

```
static void sort(int[] a)
{
  boolean stop = false;          //fuer den Abbruch der aeusseren Schleife
  int bound = a.length-1;          //obere Grenze fuer die innere Schleife

  while (!stop & bound>0)
  {
    stop = true;                 //stop==true bedeutet, dass die innere Schleife
                                 //(noch) keine Elemente vertauscht hat
    for (int j=0;j<bound;j++)
    {
      if (a[j] > a[j+1])         //a[j] ist groesser als a[j+1]
                                     //also vertausche
      { int h = a[j]; a[j] = a[j+1]; a[j+1] = h;
        stop = false;
      }
    }
    bound--;
  }
}
```

a) Erklären Sie die Begriffe von Best-Case- und Worst-Case-Zeitkomplexität! Erklären Sie die O-Notation, die benutzt wird, um die Zeitkomplexität von Algorithmen qualitativ zu erfassen!

Geben Sie in den folgenden Teilaufgaben b) und c) jeweils eine kurze Begründung an!

b) Für welche Eingaben tritt der Best-Case auf? Welche Best-Case-Zeitkomplexität hat der Algorithmus?

c) Für welche Eingaben tritt der Worst-Case auf? Welche Worst-Case-Zeitkomplexität hat der Algorithmus?

Fortsetzung nächste Seite!

Aufgabe 4: *(Programmierung)*

Programmieren Sie in einer von Ihnen gewählten Programmiersprache eine Funktion `potenz5`, die überprüft, ob eine natürliche Zahl $i \in N$ Potenz von 5 ist, d.h. ob es ein $x \in N$ gibt, so dass $5^x = i$. Geben Sie an, welche Programmiersprache Sie gewählt haben!

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Herbst 2001

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Herbst**	
Kennwort: _____	**2001**	**46114**
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (nicht vertieft studiert)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 6

Bitte wenden!

Thema Nr. 1

Vorbemerkung:

Als Programmiersprache kann bei der Lösung der folgenden Aufgaben Pascal, Modula, C, C++ oder Java verwendet werden.

Aufgabe 1

An einer Rundstrecke befinden sich 20 Zapfsäulen. Vorausgesetzt ist, dass die i-te Zapfsäule eine Benzinmenge a_i enthält und dass eine Benzinmenge b_i nötig ist, um die nächste Zapfsäule zu erreichen. Es soll ein Programm erstellt werden, das die Nummern aller Zapfsäulen ausgibt, von denen aus ein Auto mit anfangs leerem (aber genügend großem) Tank die Rundstrecke abfahren kann, bis der Ausgangspunkt wieder erreicht ist. Das Programm soll zunächst die Werte a_i und b_i in geeignete Felder einlesen.

Aufgabe 2

In einer Wetterstatistik werden für 30 europäische Großstädte die täglichen Durchschnittstemperaturen des Jahres 2000 erfasst.

a) Deklarieren Sie geeignete Datentypen oder Klassen (zunächst ohne Methoden) zur Darstellung der Wetterstatistik, der Großstädte und der Messungen, so dass

- die Wetterstatistik über eine Liste aller Großstädte verfügt,
- jede Großstadt einen Namen hat und über eine Liste ihrer Messungen im Jahr 2000 verfügt,
- jede Messung einen Wert (die Durchschnittstemperatur) und einen Tag und einen Monat (zur Bestimmung des Messdatums) hat!

b) Geben Sie eine Funktion (bzw. Prozedur) oder eine Methode an, die für eine Großstadt die Durchschnittstemperatur im Jahr 2000 berechnet! Falls die Implementierung in C++ oder Java angegeben wird, soll die Methode zur Klasse „Großstadt" gehören, ansonsten soll die Prozedur einen formalen Parameter vom Typ „Großstadt" verwenden.

c) Geben Sie eine Funktion (bzw. Prozedur) oder eine Methode an, die den Namen der (bzw. einer) Großstadt mit maximaler Durchschnittstemperatur im Jahr 2000 berechnet! Falls die Implementierung in C++ oder Java angegeben wird, soll die Methode zur Klasse „Wetterstatistik" gehören, ansonsten soll die Prozedur einen formalen Parameter vom Typ „Wetterstatistik" verwenden.

Fortsetzung nächste Seite!

Aufgabe 3

Im Folgenden soll die Datenstruktur der endlichen Teilmengen ganzer Zahlen implementiert werden.

a) Deklarieren Sie einen Datentyp „SetInt" oder eine Klasse „SetInt" (zunächst ohne Methoden), so dass endliche Mengen ganzer Zahlen durch einfach verkettetete Listen (Zeigerstrukturen) dargestellt werden!

b) Es sollen die Operationen „isempty", „contains", „add" und „remove" implementiert werden. „isempty" stellt fest, ob eine Menge s leer ist; „contains" stellt fest, ob ein Element x zu einer Menge s gehört, „add" fügt ein Element x zu einer Menge s hinzu (falls x noch nicht zu s gehört) und „remove" entfernt ein Element x aus einer Menge s.

   Falls Sie in Teil a) einen Datentyp „SetInt" deklariert haben, sollen die Operationen durch geeignete Funktionen bzw. Prozeduren mit Parametern s und (ggf.) x implementiert werden.

   Falls Sie in Teil a) eine Klasse „SetInt" deklariert haben, sollen die Operationen durch geeignete Methoden dieser Klasse (ggf. mit Parameter x) implementiert werden.

Aufgabe 4

Gegeben ist die folgende rekursive Funktionsdefinition:

```
function f (x: integer) : integer;
  if x = 1 then 0
  else if x = 2 then 1
    else f(x-2) - f(x-1)
    endif
  endif
```

a) Welche Werte liefert die Funktion für die Aufrufe f(3), f(4), f(5) und f(6)? Wie viele rekursive Aufrufe erfolgen jeweils?

b) Bestimmen Sie die Menge aller Integer-Zahlen x, für die der Aufruf f(x) terminiert!

c) Geben Sie eine rekursive Funktionsdefinition mit Kopf

   function anzahl (x: integer) : integer;

   an, die für jede Integer-Zahl x, für die der Aufruf f(x) terminiert, die Anzahl der rekursiven Aufrufe von f(x) berechnet!

Thema Nr. 2

1. Programmiermethodik

Wir betrachten die Implementierung einer endlichen Menge. Sie sei softwaremäßig als ein Modul gekapselt, der nach außen die folgenden Funktionen bereitstellt:

`empty_set:`		→	*set*	kreiert eine neue (leere) Menge
`add:`	*set* x *item*	→	*set*	fügt ein Element in die Menge ein
`remove:`	*set* x *item*	→	*set*	entfernt ein Element, sofern es in der Menge enthalten ist
`is_empty:`	*set*	→	*bool*	prüft, ob die Menge leer ist
`is_in`	*item* x *set*	→	*bool*	prüft, ob das Element in der Menge ist

a) Die Semantik der Funktion `is_empty` kann folgendermaßen definiert werden:

```
is_empty(empty_set)   = true
is_empty(add(s,e))    = false
```

Definieren Sie in der gleichen Weise die Semantik von `is_in`!

b) Wenn Sie die Semantik von `remove` definieren wollen, müssen Sie an die Möglichkeit denken, dass `add` ein Element mehrfach in die Menge aufnimmt. Wie lässt sich das Zusammenspiel von `remove` und `add` definieren? (Es gibt zwei grundlegend verschiedene Möglichkeiten.)

c) Ergänzen Sie das Beispiel durch Syntax und Semantik einer Funktion `card`, die die Kardinalität der Menge bestimmt! (Beachten Sie Ihre Überlegungen zu b)!)

d) Eine Funktion $f : M_1 \rightarrow M_2$ kann als Menge implementiert werden, indem wir *item* $= M_1 \times M_2$ setzen und folgende Funktion ergänzen: `val`: *set* x $M_1 \rightarrow M_2$, die den Funktionswert $f(x)$ heraussucht. Definieren Sie die Semantik von `val`, und passen Sie die Semantik von `add` an die neue Aufgabenstellung an!

Fortsetzung nächste Seite!

2. Systementwurf

Ein aktuelles Thema in der Diskussion um die Strukturierung von Softwaresystemen ist die Objektorientierung.

a) Erläutern Sie an dem folgenden (in einer Phantasiesprache geschriebenen) Beispiel die Begriffe Klasse, Objekt und Methode:

```
class konto is
   eroeffne_konto  (kunde);
   einzahlen  (betrag);
   auszahlen  (betrag);
   kontostand () returns betrag;
end konto;
```

(`betrag` und `kunde` seien geeignet definierte Klassen.) Erläutern Sie den Unterschied zwischen Ins-tanzen- und Klassenvariablen, und geben Sie zu der Klasse `konto` je ein Beispiel an!

b) Ein zentrales Konzept des objektorientierten Software-Entwurfs ist die Vererbung. Welche Möglichkeiten zur Konstruktion neuer Klassen bietet dieses Konzept?

c) Einige objektorientierte Programmiersprachen (z. B. C++) erlauben die Mehrfachvererbung, während andere (z. B. Java) sich auf die Einfachvererbung beschränken. Erläutern Sie den Unterschied und diskutieren Sie Vor- und Nachteile beider Möglichkeiten! Welchen Ausweg bietet Java für den Fall, dass die Problemstruktur in natürlicher Weise zu einer Mehrfachvererbung führen würde?

d) Erläutern Sie, was man in der objektorientierten Programmierung unter virtuellen Funktionen und „dynamischem Binden" versteht! Welche Vorteile bietet dieses Konzept bei der Systementwicklung?

3. Algorithmen und Datenstrukturen

Streuspeicherverfahren (Hash-Verfahren) erlauben die Speicherung einer Menge von Daten, die über Schlüssel aus einer Schlüsselmenge K identifiziert werden können, wobei nur ein kleiner Teil der potentiell möglichen Schlüssel wirklich auftritt. Man reduziert daher die Größe der Schlüsselmenge mit Hilfe einer Funktion $h : K \rightarrow A$ auf einen kleineren Adressbereich. Dabei kann es vorkommen, dass zwei oder mehr real auftretende Schlüssel auf die gleiche Adresse abgebildet werden (Kollision).

a) Bei der „offenen" Adressierung zur Kollisionsauflösung definiert jeder Schlüssel k eine Folge von Speicherplätzen $h(k,i)$. Beschreiben Sie verbal und durch ein Struktogramm das Einfügen eines neuen Objektes mit dem Schlüssel k in die Menge!

Fortsetzung nächste Seite!

b) Beschreiben Sie verbal und durch ein Struktogramm das Suchen nach einem Element mit gegebenem Schlüssel k! Wie erkennt Ihr Algorithmus, dass kein Element mit dem Schlüssel k in der Menge gespeichert ist?

c) Bei der häufig verwendeten, linearen Kollisionsauflösung

$$h(k,i) = [h_0(k) + ci] \bmod m,$$

wobei m die Größe des Adressbereichs A und c eine Konstante mit $ggT(m, c) = 1$ ist, kann es zur „Clusterbildung" kommen. Was versteht man darunter, wie lässt sich der Effekt formelmäßig charakterisieren, und welche Auswirkung hat er?

d) Die Anwendung der Streuspeicherung mit offener Adressierung ist nicht unbedingt zu empfehlen, wenn es erlaubt sein soll, dass Daten aus der Menge gestrichen werden können. Beschreiben Sie das auftretende Problem und eine denkbare Lösung!

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Frühjahr 2005

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____ Kennwort: _____ Arbeitsplatz-Nr.: _____	**Frühjahr 2005**	**46114**

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 7

Bitte wenden!

Thema Nr. 1

Aufgabe 1: Verwaltung von Schülerinnen und Schülern (80 Minuten)

Es soll ein Programm zur Verwaltung von Schülerinnen und Schülern entwickelt werden. Dazu soll die Klasse Schüler erstellt werden, die folgende Informationen speichert:

- Name des Schülers
- Vorname des Schülers
- Geschlecht des Schülers (m, w)
- Liste mit erzielten Noten dieses Schülers
- Laufende Nummer des Schülers

Die Klasse soll einen Konstruktor besitzen, der ein neues Objekt anlegt. Dem Konstruktor sollen der Vor- und Nachname des Schülers sowie dessen Geschlecht übergeben werden. Die laufende Nummer soll vom Programm selbst vergeben werden. Dabei muss darauf geachtet werden, dass kein Schüler eine bereits vergebene Nummer erhält.

Außerdem soll die Klasse über eine Methode zum Hinzufügen einer Note sowie zur Ausgabe des Objekts verfügen. Letztere soll zunächst die Durchschnittsnote des Schülers berechnen und die relevanten Informationen in einer Zeile an die Standardausgabe übergeben. Das könnte so aussehen:

Beispiel 1:

22 Hopper, Grace (w) 1,3

Beispiel 2:

23 Zuse, Konrad (m) 1,3

a. Spezifizieren Sie diese Klasse, indem Sie die Signaturen des Konstruktors und der Methode in einer Programmiersprache Ihrer Wahl angeben. Überlegen Sie sich außerdem, welche Werte für die jeweiligen Parameter zulässig sind, bzw. bei welchen Parametern eine Ausnahme generiert werden soll. (20 Minuten)

b. Implementieren Sie diese Klasse. Achten Sie darauf, dass die Implementierung zu der von Ihnen erstellten Spezifikation aus Teilaufgabe a. passt. Insbesondere sollen ungültige Parameter erkannt und mit einer Ausnahme behandelt werden. (20 Minuten)

Des Weiteren soll eine Klasse Schülerliste implementiert werden. Diese soll über einen Konstruktur verfügen, der eine leere Liste generiert. Außerdem sollen zwei Methoden implementiert werden, mit denen der Liste ein Schüler hinzugefügt bzw. ein Schüler aus der Liste entfernt werden kann. Beim Entfernen soll der entsprechende Schüler durch seine laufende Nummer spezifiziert werden. Eine weitere Methode soll die Schülerliste auf der Standardausgabe ausgeben. Dazu soll die Liste zunächst bzgl. Vor- und Nachname sortiert werden.

Fortsetzung nächste Seite!

c. Spezifizieren Sie die Klasse Schülerliste (analog a)! (10 Minuten)

d. Implementieren Sie die Klasse entsprechend Ihrer Spezifikation. Greifen Sie bei der Implementierung der Ausgabe auf einen Sortieralgorithmus Ihrer Wahl zurück, den Sie als bereits implementierte Methode verwenden können. (30 Minuten)

Aufgabe 2: Maximum eines Feldes (50 Minuten)

Gegeben sei ein Feld von Ganzzahlen der Länge n. Gesucht ist die Position des größten Elements in diesem Feld. Für diese Aufgabe sollen zwei Algorithmen entwickelt und anschließend miteinander verglichen werden.

a. Entwickeln Sie zunächst die Funktion maxIterativ, welche die Aufgabe mit einem iterativen Programmieransatz löst. (10 Minuten)

b. Entwickeln Sie die Funktion maxRekursiv, welche die Aufgabe mittels einer rekursiven Funktion löst, die das Feld in jedem Rekursionsschritt in zwei Hälften teilt, jeweils das Maximum der beiden Hälften errechnet und anschließend das Maximum der beiden Ergebnisse bildet. Überlegen Sie sich zunächst ein geeignetes Rekursionsende. (20 Minuten)

c. Entwickeln Sie ein Hauptprogramm, das
- solange Werte von der Standardeingabe liest, bis das Eingabeende erreicht wird,
- diese in einem Feld ablegt,
- das Feld mit Hilfe der obigen Funktion sortiert und
- das sortierte Feld über die Standardausgabe ausgibt.

Hinweis: Sie dürfen auf die Teilergebnisse a) und b) auch dann zurückgreifen, wenn Sie diese Aufgaben noch nicht vollständig gelöst haben. (10 Minuten)

d. Vergleichen Sie Ihre Lösungen aus Aufgabe a) und b) bezüglich
- Speicherkomplexität und
- Zeitkomplexität

Wie sinnvoll ist in diesem Zusammenhang die Verwendung des Landau-Symbols O? (10 Minuten)

Fortsetzung nächste Seite!

Aufgabe 3: Verkette Listen (50 Minuten)

Im Folgenden soll die flexible Datenstruktur List in einer objektorientierten Sprache Ihrer Wahl implementiert werden. Die Klasse soll Elemente vom abstrakten Datentyp DATA speichern können und dabei so flexibel sein, dass List-Objekte sowohl FIFO-Listen (First in First Out bzw. Warteschlangen) als auch LIFO-Listen (Last In First Out bzw. Keller) sein können.
Implementieren Sie eine solche Klasse mit den folgenden Konstruktoren und Methoden:

(Lesen und Planen: 20 Minuten)

Konstruktor:	Erzeugt eine leere Liste. Als Argument wird übergeben, ob es sich um eine Warteschlange oder einen Keller handeln soll. (5 Minuten)
Method add:	Fügt der Liste ein Objekt hinzu. (15 Minuten)
Method remove:	Entfernt ein Objekt und liefert dieses als Rückgabewert. Je nach Art der Liste (FIFO bzw. LIFO) wird das „älteste" bzw. „jüngste" Objekt in der Liste zurückgegeben. (10 Minuten)

Achten Sie bei der Implementierung darauf, dass die Zeitkomplexität des Konstruktors und der beiden Methoden in allen Fällen unabhängig von der Länge der Liste ist. Die Komplexität soll also in allen Fällen zur Klasse O (1) gehören.

Thema Nr. 2

Aufgabe 1: Baumdarstellung arithmetischer Ausdrücke

Arithmetische Ausdrücke in Infixnotation mit den binären Operatoren $+, -, *, /$ und dem unären Operator $-$ und Variablen oder Konstanten als Operanden können induktiv wie folgt definiert werden:
Basis: Einzelne Variablen und Konstanten sind arithmetische Ausdrücke.
Induktion: Wenn E_1 und E_2 arithmetische Ausdrücke sind, so sind auch

$$(E_1 + E_2),\ (E_1 - E_2),\ (E_1 * E_2),\ (E_1 / E_2) \text{ und } (-E_1)$$

arithmetische Ausdrücke.

Entsprechend dieser Definition kann ein arithmetischer Ausdruck durch einen Ausdrucksbaum dargestellt werden, dessen Blätter einzelne Variablen oder Konstanten sind und dessen innere Knoten die jeweiligen Teilausdrücke repräsentieren.

a) Geben Sie den Ausdrucksbaum zum arithmetischen Ausdruck $(y * (-(x + 9)))$ an!
b) Definieren Sie in einer Programmiersprache Ihrer Wahl eine Datenstruktur zur Darstellung arithmetischer Ausdrücke, die zu jedem Ausdruck den Operator und die Teilausdrücke, auf die der Operator angewendet wird, angibt.
c) Spezifizieren Sie in einer Programmiersprache Ihrer Wahl eine rekursive Funktion inorder(), die aus einem gegebenen Ausdrucksbaum, der als Parameter übergeben wird, den zugehörigen arithmetischen Ausdruck in Infixnotation ausgibt.

Achten Sie in allen Teilaufgaben auf eine korrekte Klammerung der Teilausdrücke!

Aufgabe 2: Warteschlangen als abstrakter Datentyp

Eine Warteschlange ist eine lineare Datenstruktur zur Ablage von Elementen des gleichen Typs mit der Eigenschaft, dass nur am Ende der Datenstruktur Elemente eingefügt und nur am Anfang der Datenstruktur Elemente entnommen werden können (FIFO-Prinzip). Dementsprechend stehen die folgenden beiden elementaren Operationen zur Verfügung:

enqueue (e): Einfügen eines Elementes e am Ende der Warteschlange.
e=dequeue(): Entnahme eines Elementes e am Anfang der Warteschlange (falls vorhanden).

Gehen Sie dabei davon aus, dass jedes Warteschlangenelement einen einzelnen Integerwert enthält.

a) Realisierung von Warteschlangen mit einfach verketteten Listen in einer imperativen Programmiersprache Ihrer Wahl:

   α) Definieren Sie eine Knoten-Datenstruktur zur Verwendung in der Listenrealisierung.
   β) Geben Sie eine Funktion `make_element()` an, die einen Knoten der Liste mit zugehörigem Integerwert, der als Parameter übergeben wird, anlegt.

Fortsetzung nächste Seite!

γ) Geben Sie Funktionen zur Realisierung der beiden elementaren Funktionen `enqueue()` und `dequeue()` an. Geben Sie beim Entnahmeversuch aus einer leeren Warteschlange eine geeignete Fehlermeldung aus. Verwenden Sie zur Realisierung von `enqueue()` die Funktion `make_element()`.

b) Realisierung einer Warteschlange mit Hilfe eines Feldes fester Länge in einer imperativen Programmiersprache Ihrer Wahl.
Verwenden Sie für die Realisierung zwei globale Feldindices:
- der Index `head` gibt das erste belegte Element der Warteschlange an;
- der Index `tail` gibt die Feldposition an, an der das nächste einzufügende Element abgelegt wird.

Verwenden Sie für eine effiziente Ausnutzung des Feldes eine zirkuläre Implementierung (wraparound), die beim Erreichen des Feldendes zum Einfügen neuer Elemente Feldpositionen am Anfang des Feldes verwendet, wenn dort durch Entfernen von Warteschlangenelementen freie Feldpositionen entstanden sind.

α) Spezifizieren Sie die verwendeten Datenstrukturen und geben Sie eine Funktion `queue_init()` zur Initialisierung der Warteschlange an.

β) Geben Sie Funktionen zur Realisierung von `enqueue()` und `dequeue()` an und beachten Sie dabei jeweils die Möglichkeit, dass durch die Aktualisierung von `head` und `tail` das Feldende erreicht werden kann.
Geben Sie geeignete Fehlermeldungen aus, wenn `enqueue` auf eine volle Liste angewendet wird und wenn `dequeue` auf eine leere Liste angewendet wird.

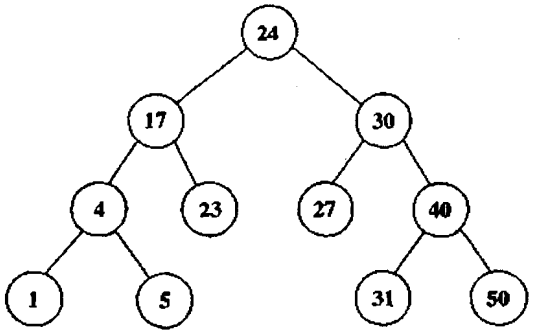
c) Diskutieren Sie Vor- und Nachteile der Realisierung von Warteschlangen mit einfach verketteten Listen und Feldern fester Länge!

Fortsetzung nächste Seite!

Aufgabe 3 (binäre Suchbäume)

Ein (unbalancierter) binärer Suchbaum ist eine dynamische Datenstruktur zur knotenorientierten Ablage von Daten mit Suchschlüsseln, welche zum Sortieren der Einträge verwendet werden. Als Suchschlüssel sollen Integerwerte verwendet werden. Jeder Knoten x enthält neben den Daten einen Suchschlüssel und einen Verweis auf den linken und den rechten Unterbaum des Knotens. Die Knoten sollen dabei so angeordnet sein, dass alle Knoten im linken Unterbaum von x Suchschlüssel besitzen, die kleiner als der Suchschlüssel von x sind, und dass alle Knoten im rechten Unterbaum von x Suchschlüssel besitzen, die größer als der Suchschlüssel von x sind. Jeder Suchschlüssel soll nur einmal in dem Suchbaum auftreten. Verwenden Sie für die folgende Realisierung eine imperative Programmiersprache Ihrer Wahl.

a) Definieren Sie eine Datenstruktur zur Realisierung eines Knotens des Suchbaumes mit Daten, Suchschlüssel und Verweisen auf den linken bzw. rechten Unterbaum.
b) Geben Sie eine rekursive Funktion `tree_lookup()` an, die für einen als Parameter übergebenen Suchschlüssel den zugehörigen Eintrag in einem Suchbaum zurückliefert.
c) Geben Sie eine rekursive Funktion `tree_insert()` an, die Daten mit einem gegebenen Suchschlüssel in einen Suchbaum als neuen Eintrag einfügt.
d) Löschen Sie aus folgendem binären Suchbaum, in dem nur die Suchschlüssel angegeben sind, nacheinander die Elemente mit Suchschlüssel 5,4,24. Geben Sie die nach jedem Löschvorgang resultierenden Suchbäume an.



e) Geben Sie eine rekursive Funktion `delete_min()` an, die aus einem Unterbaum, dessen Wurzel als Parameter übergeben wird, den Eintrag mit dem kleinsten Suchschlüssel löscht!
f) Geben Sie eine rekursive Funktion `delete()` an, die einen über den Suchschlüssel spezifizierten Eintrag aus einem Suchbaum löscht. Verwenden Sie dazu die Funktion `delete_min()` aus Aufgabenteil e)!

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Herbst 2005

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Herbst**	**46114**
Kennwort: _____	**2005**	
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 5

Bitte wenden!

Thema Nr. 1

Aufgabe 1: Algorithmen und Datenstrukturen

Sei G ein gerichteter Graph. Die *Transposition* von G hat dieselben Knoten wie G, aber alle Kanten sind entgegengesetzt ausgerichtet.
Bei der Darstellung von Graphen durch *Adjazenzlisten* wird zu jedem Knoten die Liste der benachbarten Knoten angegeben.

a) Zeichnen Sie einen gerichteten Graphen mit mindestens 5 Knoten und 10 Kanten und geben Sie dessen Repräsentation durch Adjazenzlisten an!

b) Entwerfen Sie ein Verfahren, welches die Transposition eines durch Adjazenzlisten gegebenen Graphen in ebendieser Form bestimmt! Beschreiben Sie das Verfahren detailliert, z.B. durch Angabe von Pseudocode!

c) Geben Sie die Komplexität Ihres Verfahrens unter Verwendung der O-Notation als Funktion der Knotenzahl m und der Kantenzahl n an!

d) Beschreiben Sie nun ein Verfahren, welches zu zwei Knoten s, t in einem durch Adjazenzlisten gegebenen Graphen feststellt, ob t von s aus erreichbar ist, und, falls ja, die Länge des kürzesten verbindenden Pfades angibt! Geben Sie auch hier die Zeitkomplexität in der unter c) beschriebenen Form an!

Aufgabe 2: Objektorientierter Entwurf

Im Betriebssystem *Phindows* gibt es vier Arten von Dateien, die jeweils die angegebenen Attribute haben:

- Text: Name, Größe, Anzahl der Zeilen, Verzeichnis. Das Suffix einer Text-Datei ist .txt
- HTML: Name, Größe, Anzahl der Zeilen, Titel, Verzeichnis. Das Suffix einer HTML-Datei ist .htm oder .html
- GIF: Name, Größe, Breite, Höhe, Verzeichnis. GIF-Dateien haben das Suffix .gif
- Verzeichnis: Name, Verzeichnis, enthaltene Dateien. Verzeichnisse haben entweder kein Suffix oder eines, welches von den bisher genannten verschieden ist.

Das Attribut "Verzeichnis" gibt jeweils das Verzeichnis an, in dem sich eine Datei befindet.

Bei den folgenden Teilaufgaben können Sie statt des objektorientierten Pseudocodes auch Java verwenden.
Aus dem Pseudocode muss insbesondere die Schnittstelle Ihrer Methoden (also Klassenzugehörigkeit, Anzahl und Typ der Parameter, Typ des Rückgabewertes) eindeutig hervorgehen. Außerdem muss das intendierte Verhalten aller von Ihnen geschriebenen Methoden dokumentiert sein!

Fortsetzung nächste Seite!

a) Modellieren Sie diese Situation in UML.

b) Schreiben Sie objektorientierten Pseudocode für eine Methode, die zu gegebenem Namen eine Datei der entsprechenden Art anhand des Suffixes bildet.

c) Zugriffspfade sind in Phindows durch „|“ getrennte Listen von Dateinamen, etwa
„`|home|user|test.txt`“
Schreiben Sie objektorientierten Pseudocode für eine Methode, die zu einer gegebenen Datei den vollständigen Zugriffspfad als Zeichenkette ausgibt. In Phindows beginnen alle Pfade bei einem Wurzelverzeichnis (im Beispiel home), das sich selbst als Verzeichnis hat.

d) Schreiben Sie objektorientierten Pseudocode für eine Methode, die in einem gegebenen Verzeichnis und all seinen Unterverzeichnissen nach einer Datei eines gegebenen Namens sucht!

Thema Nr. 2

Aufgabe 1

Gegeben seien die Schlüsselwerte (3, 5,11, 8, 4, 1, 12, 7, 2, 6, 10, 9)

a) Generieren Sie den binären Suchbaum, der durch sukzessives Einsetzen dieser Werte entsteht. Gehen Sie dabei von einem leeren Baum aus!
b) Geben Sie zu den jeweiligen Knoten die zugehörige Balance ein! Ist der Baum ein *AVL*-Baum ?
c) Generieren Sie wie in a) erneut einen Suchbaum durch sukzessives Einsetzen der gegebenen Werte und stellen Sie nach jedem Schritt die AVL-Eigenschaft wieder her, so dass beim nächsten Einsetzen ein AVL-Baum zugrunde liegt und auch der Baum am Ende ein AVL-Baum ist.
d) Stellen Sie jetzt für obige Schlüsselwertmenge einen B-Baum der Ordnung 3 auf! Gehen Sie schrittweise vor, so dass die Vorgehensweise nachvollziehbar ist!

Aufgabe 2

Die Backus-Naur-Form kann dazu verwendet werden, die Syntax der Kommandozeilenparameter eines Programms zu beschreiben. Als Beispiel wird das Bildkonvertierungstool `convert` herangezogen. Mit diesem können die verschiedensten Operationen auf Bilder, welche durch ihre Dateinamen `file` gegeben sind, ausgeführt werden.

Beschreibung (Auszug):

```
Version: ImageMagick 6.0.7 11/03(04 Y16 http://www.imagemagick.org
Copyright: Copyright (C) 1999-2004  ImageMagick Studio LLC

Usage: convert {<options>} file {{<options>} file} {<options>} file
   <options> ::= -antialias | -contrast | -dither | -monochrome |
   -normalize

Where options  include:
-antialias          remove pixel -aliasing
-contrast           enhance or reduce the image contrast
-dither             apply Floyd/Steinberg error diffusion to image
-monochrome         transform image to black and white
-normalize          transform image to span the full range of colors
```

a) Sind folgende Programmaufrufe für die Bilder `bild1.jpg`, `bild2.jpg` und `bild3.jpg` gültig?
Begründen Sie die Antwort!

i) `convert bild1.jpg -antialias bild2.jpg`
ii) `convert bild1.jpg`
iii) `convert bild1.jpg bild2.jpg -normalize -monochrome bild3.jpg -dither`
iv) `convert bild1.jpg bild2.jpg -normalize -monochrome bild3.jpg`

b) Wandeln Sie die Backus-Naur-Form in ein Syntax-Diagramm um!

Fortsetzung nächste Seite!

Aufgabe 3:

Gegeben sind folgende Ersetzungsregeln:

<Summe> ::= <Produkt> | <Produkt> + <Produkt> | <Produkt> - <Produkt>
<Produkt> ::= <Faktor> | <Faktor> * <Faktor> | <Faktor> / <Faktor>
<Faktor> ::= <Zahl> | (<Summe>) | <Summe>
<Zahl> ::=0 | (1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9) { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

a) Erstellen Sie ein Syntaxdiagramm zu obigen Ersetzungsregeln!

b) Erweitern Sie die Regeln, um das arithmetische Potenzieren darstellen zu können! Verwenden Sie dabei das '^' Zeichen für die Potenz $a\hat{}b \triangleq a^b$. Geben Sie auch die Änderungen am Syntaxdiagramm an!

Aufgabe 4

Eine Rechnung besteht aus mehreren Posten, von denen jeder eine Artikelnummer, Anzahl, Einzelpreis und Gesamtpreis enthält. Eine Rechnung ist an eine Person gerichtet und kann mit unterschiedlichen Zahlungsmethoden bezahlt werden. Zahlungsmethoden sind:

- Barzahlung, ohne weitere Attribute
- ec-Karte mit BLZ und PIN-Nummer
- Kreditkarte mit Nummer und Verfallsdatum

a) Modellieren Sie den Sachverhalt mit einem ersten UML Klassendiagramm. Geben Sie alle relevanten Beziehungen, Attribute und Methoden an! Begründen Sie Ihre Entwurfsentscheidung! Verwenden Sie hier noch keine Vererbung!

b) Geben Sie ein Objektdiagramm, welches eine Situation beschreibt, in der alle Zahlungsmethoden verwendet werden!

c) Wenden Sie nun Generalisierung (Vererbung) an, um den Entwurf zu verbessern!

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Frühjahr 2006

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Frühjahr**	**46114**
Kennwort: _____	**2006**	
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 10

Bitte wenden!

Thema Nr. 1

Vorbemerkung:

Es wird empfohlen, zur Lösung der Programmieraufgaben eine der Programmiersprachen Pascal oder Java zu verwenden. Dazu ähnliche Programmiersprachen wie C oder C++ sind jedoch auch zugelassen.

Teilaufgabe A (Dateiverwaltung)

Adressen und Telefonnummern sollen in nach Namen sortierten Binärbäumen abgelegt werden. Beim Auftreten mehrerer Einträge unter gleichem Familiennamen werden zur Unterscheidung zunächst die Vornamen und schließlich, falls diese ebenfalls übereinstimmen, die Geburtsdaten verwendet. Mit diesen drei Merkmalen wird dann aber Eindeutigkeit vorausgesetzt, so dass das Tripel (Familienname, Vorname, Geburtsdatum) als Zugriffsschlüssel (*Schlüssel*) verwendet werden kann.
Familienname und Vorname werden als *String* (bei Programmierung in Java) bzw. als *string* (bei Programmierung in Pascal), das Geburtsdatum wird als *ganze Zahl* im Format JJJJMMTT eingegeben.
Als Schnittstelle für die Programmieraufgaben steht Ihnen die Arithmetik der ganzen Zahlen (einschließlich der Vergleichsoperationen) und genau eine Vergleichsoperation *kl(String a, String b)* (in Java) bzw. *kl(string a, string b)* (in Pascal) zwischen zwei Zeichenreihen zur Verfügung, die jeweils den Wahrheitswert true genau dann liefert, wenn *a* lexikographisch vor *b* kommt. Sind *a* und *b* gleich oder kommt *b* vor *a*, ist das Resultat der Wahrheitswert *false*.

1. Geben Sie eine für diese Teilaufgabe A geeignete Datenstruktur an! Adresse und Telefonnummer sollen dabei als *String* (Java) bzw. *string* (Pascal) eingegeben werden. Da das Tripel (Familienname, Vorname, Geburtsdatum) als Zugriffsschlüssel für die Einträge im sortierten Binärbaum dient, soll dabei für diese *Schlüssel* eine eigene Klasse (Java) bzw. ein eigener Verbund (Pascal) eingeführt werden.

2. Programmieren Sie unter dem Namen *eintrag* eine Instanzmethode (Java) bzw. eine Prozedur (Pascal), die folgender Spezifikation genügt:

   2.1 Eingabeparameter sind (zum *Schlüssel* zusammengefasst) Familienname, Vorname und Geburtsdatum einer Person sowie die zu dieser Person gehörende Adresse und Telefonnummer.
   Hinweis: Bei Programmierung in Pascal muss auch der entsprechende Binärbaum als Parameter übergeben werden.

   2.2 Die Methode bzw. die Prozedur trägt die Eingabedaten (nach 2.1) in den entsprechend 1. aufgebauten sortierten Binärbaum ein, falls es zu dem angegebenen *Schlüssel* noch keinen Eintrag gibt. Danach soll der Binärbaum wieder sortiert sein.
   Hinweis: Es ist empfehlenswert, für diese Programmieraufgabe die Schnittstelle zunächst um eine Methode bzw. eine Funktionsprozedur *kl1 (Schlüssel a, Schlüssel b)* zu erweitern.

Fortsetzung nächste Seite!

3. Programmieren Sie unter dem Namen *ausgabe* eine Instanzmethode (Java) bzw. eine Prozedur (Pascal), die folgender Spezifikation genügt:

   3.1 Eingabeparameter sind (zum *Schlüssel* zusammengefasst) Familienname, Vorname und Geburtsdatum einer Person.
   Beachten Sie den Hinweis bei 2.1!

   3.2 Die Methode bzw. die Prozedur druckt die zu den Daten nach 3.1 gehörende Adresse und Telefonnummer auf dem Bildschirm (Standard-Output) aus, falls zu der Eingabe ein Eintrag im Binärbaum enthalten ist. Andernfalls druckt sie die Meldung „`kein Eintrag`" aus.

4. Es soll eine Methode *loesche* entwickelt werden, die einen Eintrag in einem entsprechend 1. aufgebauten sortierten Binärbaum löscht.

   4.1 Programmieren Sie zunächst zu einem nichtleeren nach 1. aufgebauten sortierten Binärbaum *sb* eine Instanzmethode (Java) bzw. eine Funktionsprozedur (Pascal) unter dem Namen *max*, die folgender Spezifikation genügt:

      4.1.1 Bei Programmierung in Java hat diese Methode keinen Parameter; bei Programmierung in Pascal hat die Funktionsprozedur nur *sb* als Parameter.

      4.1.2 Die Methode bzw. die Funktionsprozedur liefert als Resultat den Teilbaum von *sb*, der in seiner Wurzel den maximalen Schlüssel von *sb* enthält.

   4.2 Formulieren Sie mit Hilfe der nunmehr gegebenen Schnittstelle (also einschließlich *max* nach 4.1), wie eine Methode bzw. eine Prozedur zu programmieren ist, die einen Eintrag zu einem gegebenen Schlüssel in einem sortierten Binärbaum löscht, so dass der Binärbaum danach wieder sortiert ist. Die Formulierung soll so konzis (d.h. knapp und unmissverständlich) sein, dass eine erfahrene Programmiererin (ein erfahrener Programmierer) das Programm ohne weiteres schreiben kann.

Fortsetzung nächste Seite!

Teilaufgabe B (Textverarbeitung)

Gegeben sei das Alphabet $I = (a, b, c, \ldots, z)$. Als Schnittstelle für Zeichenreihen aus I^* sowie für deren Bearbeitung stehen Ihnen einfach verkettete Listen mit den folgenden Methoden (Java) bzw. Funktionsprozeduren (Pascal) zur Verfügung:

In Java:

```
class Zr{
char c; Zr next;
static int laenge (Zr x)                          // liefert die Länge von x
static boolean istanfang (Zr x,Zr y){ . . . }     // liefert genau dann true,
                                                  // wenn y ein Anfang von x ist
static Zr copy (Zr x){ . . . }                    // liefert eine Kopie von x
}
```

bzw.

in Pascal:

```
type zr =↑ zrel;
type zrel =record c :char; next:zr end;
function laenge(x : zr) :integer;                 (* liefert die Länge von x *)
function istanfang(x : zr,y : zr) :boolean;       (* liefert genau dann true, *)
                                                  (* wenn y ein Anfang von x ist *)
function copy(x : zr) : zr;                       (* liefert eine Kopie von x *)
```

1. Programmieren Sie mit Hilfe der angegebenen Schnittstelle unter dem Namen *ersetze* eine Methode (Java) bzw. eine Prozedur (Pascal), die folgender Spezifikation genügt:

   1.1 Eingabeparameter sind drei nichtleere Wörter x, y und z aus I^*.

   1.2 Die Methode ersetzt das erste (von links gelesen) in x vorkommende y durch z und nennt das so entstehende Wort wieder x. Mit diesem neuen Wort x wird die Methode mit den (unveränderten) Wörtern y und z solange wiederholt, bis y nicht mehr in x vorkommt.
   Hinweis:
   Es ist empfehlenswert, für diese Programmieraufgabe die Schnittstelle (siehe oben) zunächst um die Methode `int` $pos(Zr\,x, Zr\,y)$ {...} (Java) bzw. um die Funktionsprozedur `function` $pos(x : zr, y : zr)$ `:integer;` (Pascal) zu erweitern.
   Mit den Bezeichnungen

   $$x = X_1 X_2 \ldots X_i X_{i+1} \ldots X_{i+|y|} X_{i+|y|+1} \ldots X_{|x|};$$
   $$y = X_{i+1} \ldots X_{1+|y|}$$ (das erste Vorkommen von y in x, falls y in x überhaupt vorkommt)

   liefert *pos* den Index i, wenn y in x vorkommt, und sonst den Wert -1.

Fortsetzung nächste Seite!

2. x, y und z seien drei Wörter aus I^* wie bei 1.1

   2.1 Die Methode *ersetze* (aus 1.) terminiert sicher für alle x, wenn $|y| > |z|$ ist. Beweisen Sie diese Aussage!

   2.2 Geben Sie eine hinreichende Bedingung dafür an, dass die Methode *ersetze* (aus 1.) für alle x terminiert, wenn $|y| \leq |z|$ ist und wenigstens ein Zeichen aus I sowohl in y als auch in z vorkommt, und beweisen Sie Ihre Aussage.

Thema Nr. 2

Aufgabe 1 (Objektorientierung)

a) Definieren Sie die Begriffe *Klasse, Objekt, Vererbung, Interface* und *abstrakte Klasse*!

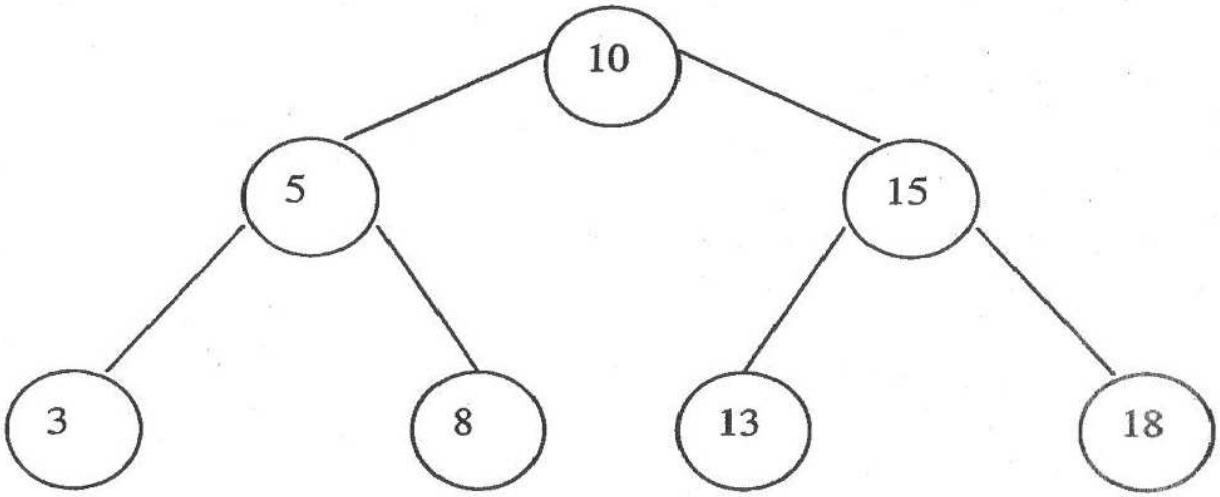
b) Schreiben Sie in Java (oder in einer anderen objektorientierten Programmiersprache) eine Klasse `Konto`, die das Guthaben eines Bankkontos verwaltet. Die Klasse soll Methoden zum `einzahlen` und `abheben` vom Konto haben. Das Konto soll man nicht überziehen können. Die Klasse soll außerdem eine Methode `getKontostand` haben, die den aktuellen Kontostand zurückliefert. Implementieren Sie den Konstruktor so, dass ihm das Startkapital des Kontos übergeben werden muss. Schreiben Sie ein Hauptprogramm, das die Klasse verwendet und zuerst ein Konto mit 100 € Startkapital eröffnet und dann 20 € abhebt.

c) Schreiben Sie eine von der Klasse `Konto` abgeleitete Klasse `Girokonto`. Im Gegensatz zu einem normalen Konto darf ein Girokonto um ein vorher festgelegtes Dispolimit überzogen werden. Schreiben Sie für die Klasse Girokonto Methoden zum Festlegen und Abfragen des Dispolimits! Der Einfachheit halber gehen Sie davon aus, dass für das Überziehen des Kontos keine Zinsen gezahlt werden müssen. Wenn nötig, überschreiben Sie Methoden der Klasse Konto.

Fortsetzung nächste Seite!

Aufgabe 2 (Binärer Suchbaum)

Gegeben sei der folgende binäre Suchbaum:



Löschen Sie den Schlüssel 13. Aus dem resultierenden Baum löschen Sie dann den Schlüssel 15. Im dritten Schritt löschen Sie dann den Schlüssel 5. Geben Sie dabei nach jeder Löschoperation den resultierenden Suchbaum an.

Aufgabe 3 (Code-Verständnis)

Unten ist der Java-Code einer Klasse `Stack` (ein Keller) abgedruckt, der alle Datenstrukturen und Methoden enthält, um einen Keller von natürlichen Zahlen zu verwalten. Die Klasse `Stack` besitzt eine Variable, die die Elemente des Kellers in einem Feld hält. Außerdem werden folgende Methoden nach außen zur Verfügung gestellt:

- `void init()`
  Initialisierung der Keller-Datenstruktur (mit leerem Keller zu Beginn)

- `boolean isEmpty()`
  Abfrage, ob der Keller gerade leer ist

- `void push(int i) throws StackException`
  soll die als Parameter angegebene Zahl auf den Keller legen; falls der Speicherbereich erschöpft ist, wird eine Ausnahmesituation ausgelöst.

- `int pop()`
  nimmt die oberste Zahl vom Keller und liefert sie zurück; ist der Keller leer, wird -1 zurückgegeben.

Fortsetzung nächste Seite!

Der folgende Code enthält allerdings einige syntaktische und semantische Fehler. Geben Sie jeweils die Zeilennummer des Fehlers und den korrigierten Code an.

```
1   void class Stack {
2      private int stackSize;
3      private int currentPos;
4      private int elements[];
5
6        Stack(int size) {
7           init();
8           if (size > 0) {
9              stackSize = size;
10             elements = new int[size];
11          }
12       }
13
14       // Initialisierung des Stacks
15       public void init() {
16          currentPos = 1;
17       }
18
19       // Ist der Stack leer?
20       public boolean isEmpty() {
21          return (currentPos = 0);
22       }
23
24       // Ablegen eines Wertes auf dem Stack
25       private void push(int i) {
26          if (currentPos == stackSize)
27             System.out.println ("Der Keller ist voll.");
28          else elements[currentPos++] = i;
29       }
30
31       // Holen des obersten Wertes vom Stack
32       public int pop() {
33          if (currentPos == 0)
34             return -1;
35
36          return (elements[--currentPos])
37       }
38    }
```

Fortsetzung nächste Seite!

Aufgabe 4 (Rekursion und Iteration)

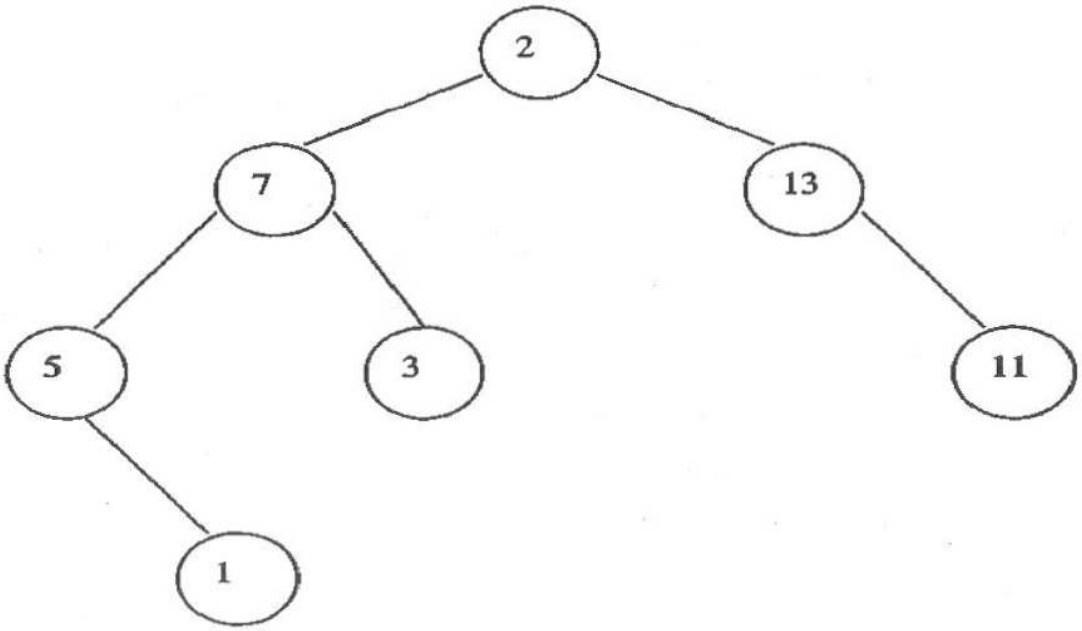
Die Potenzierungsfunktion für zwei natürliche Zahlen n, m kann durch fortgesetzte Multiplikation realisiert werden:

$$m^n = pot(n,m) = \begin{cases} m \cdot m^{n-1} & \text{falls } n > 0 \\ 1 & \text{sonst} \end{cases}$$

Geben Sie eine **iterative** und eine **rekursive** Implementierung der Potenzfunktion an! Benutzen Sie dabei keine Funktionen aus einer Bibliothek!

Aufgabe 5 (Rekursion und Iteration)

Gegeben sei der folgende Baum:



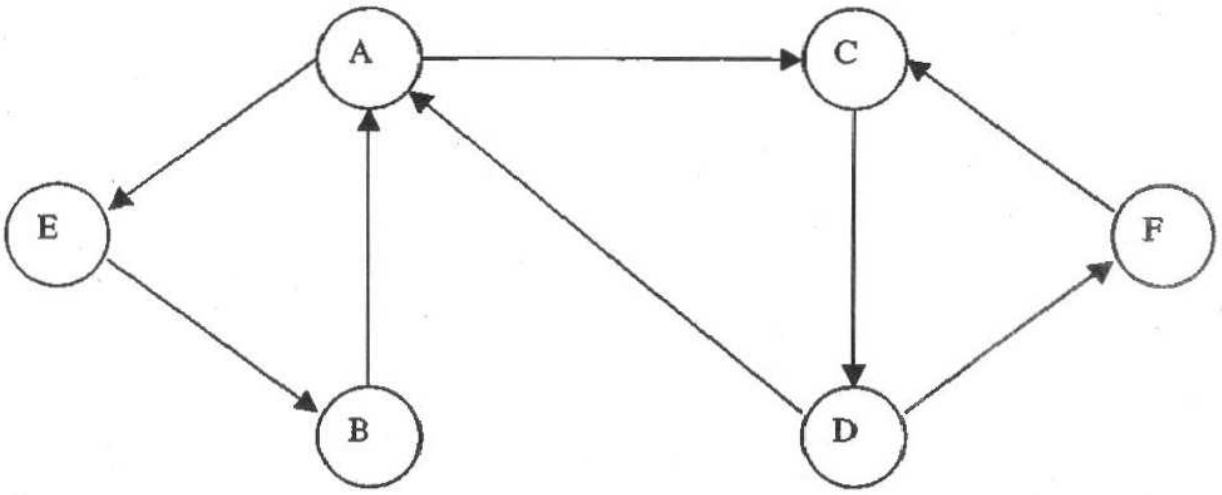
a) Durchlaufen Sie diesen Baum einmal in Tiefen- und einmal in Breitensuche und geben Sie jeweils an, in welcher Reihenfolge die im Baum enthaltenen Zahlen dabei ausgegeben werden!

b) Handelt es sich bei dem oben abgebildeten Baum um einen binären Suchbaum bzgl. der natürlichen Zahlen und der *kleiner*- Relation? Begründen Sie Ihre Antwort!

Fortsetzung nächste Seite!

Aufgabe 6 (Graphrepräsentation)

Repräsentieren Sie den folgenden Graphen sowohl mit einer Adjazenzmatrix als auch mit einer Adjazenzliste.



Aufgabe 7 (UML-Klassendiagramm)

Erzeugen Sie ein Analysediagramm (Klassendiagramm) für ein Informatiksystem einer Kart-Bahn, das die Vermietung der Kart-Wagen verwaltet. Folgende Anforderungen werden daran gestellt:

- Nur registrierte Kunden der Bahn können Kart-Wagen ausleihen. Jeder Kunde kann entweder einen oder sechs Karts ausleihen (zur Erklärung: bei sechs Karts möchte der Kunde für eine Gruppe mieten).
- Zu jedem Kunden werden Name, Adresse, Datum der ersten Miete und Kundennummer gespeichert.
- Zu jedem Kart ist das TÜV-Siegel (bescheinigt technisch korrekten Zustand des Karts), der Mietpreis, eine Inventarnummer und eine Typenbezeichnung festgehalten.
- Ein Kunde wählt zwischen der Miete von Karts für Erwachsene und solcher für Kinder.
- Zu einem Kart für Erwachsene wird die Leistung angegeben. Karts für Kinder haben alle dieselbe reduzierte Leistung.

Erstellen Sie ein UML-Klassendiagramm für die Analyse, das ausschließlich mit den Klassen Kunde, Kart, KartErwachsene und KartKinder arbeitet und das die gestellten Anforderungen erfüllt. Kardinalitäten/Multiplizitäten geben Sie bitte auch im Fall einer Kardinalität/ Multiplizität von 1 explizit an.

Fortsetzung nächste Seite!

Aufgabe 8 (Algorithmenverständnis)

Analysieren Sie nachfolgenden Pseudocode eines bekannten Algorithmus.

```
1    op1( k, x ):
2         vater := null;
3         sohn  := k.root;
4         while not (sohn = null) do
5              vater := sohn;
6              if x.key < sohn.key
7                   then sohn := sohn.left
8                   else sohn := sohn.right
9              fi
10        od
11        if vater = null
12             then k.root := x
13             else
14                  if x.key < vater.key
15                       then vater.left := x
16                       else vater.right := x
17                  fi
18        fi
```

a) Erläutern Sie kurz, was in jeder Zeile geschieht!

b) Um welchen Algorithmus handelt es sich hier?

c) Gehen Sie davon aus, dass zu Beginn k.root=null ist. Wählen Sie fünf Werte für x aus und wenden Sie den Algorithmus nacheinander auf diese Werte an! Geben Sie hierzu nach jedem der fünf Durchläufe das Ergebnis an!

Aufgabe 9 (Textformatierung)

Schreiben Sie ein Programm in einer Programmiersprache Ihrer Wahl, das ein zweidimensionales Integer-Feld (Array) der Größe 5×5 auf dem Bildschirm ausgibt. Jede Zeile des Feldes soll in einer eigenen Zeile der Ausgabe erscheinen. Die Zahlen innerhalb der Zeile des Feldes sollen am Bildschirm mit einem Komma voneinander getrennt sein. Nach der letzten Zahl einer Zeile soll kein Komma stehen.

Beispiel: Das Java-Feld

```
int [][] a =   { {1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15},
                 {16,17,18,19,20}, {21,22,23,24,25}};
```

soll wie folgt formatiert ausgegeben werden:

```
1,2,3,4,5
6,7,8,9,10
11,12,13,14,15
16,17,18,19,20
21,22,23,24,25
```

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Herbst 2006

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____ Kennwort: _____ Arbeitsplatz-Nr.: _____	**Herbst 2006**	**46114**

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Algorithmen/Datenstrukturen/Programmiermethodik**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 5

Thema Nr. 1

Hinweis für die Aufgaben 1, 2 und 4:
Verwenden Sie zur Formulierung von Algorithmen bzw. Datentypen eine gängige höhere Programmiersprache oder einen entsprechenden Pseudocode. Erläutern Sie Ihre Lösung ausgiebig durch Kommentare!

Aufgabe 1

a) Realisieren Sie den Datentyp der Brüche als 3-Tupel bestehend aus dem Betrag des ganzzahligen Zählers, dem Betrag des ganzzahligen Nenners und einer Komponente, die die Zahl 1 für ein positives und die Zahl -1 für ein negatives Vorzeichen des Bruches enthält.

b) Geben Sie unter Verwendung einer gegebenen Operation für den größten gemeinsamen Teiler (ggT) eine Operation an, die für die so dargestellten Brüche die Addition zweier Brüche realisiert. Als Ergebnis dieser Operation soll wiederum ein Bruch (soweit wie möglich gekürzt) in der o. g. Darstellung zurückgeliefert werden.

Fortsetzung nächste Seite!

Aufgabe 2

Eine Prioritätswarteschlange (priority queue) ist eine Datenstruktur, in die Datensätze mittels einer Operation `insert` eingefügt werden können und aus der mittels einer Operation `remove` der Datensatz mit dem jeweils größten Schlüssel wieder ausgelesen werden kann. Vereinfachend kann davon ausgegangen werden, dass die Prioritätswarteschlange jeden Schlüssel nur genau einmal enthält.

a) Geben Sie Implementierungen der genannten Operationen an, die folgende Eigenschaften erfüllen: `Insert` soll in konstanter Zeit erfolgen, `remove` in linearer Zeit (jeweils abhängig von der Zahl der Elemente in der Schlange).

b) Geben Sie Implementierungen der genannten Operationen an, die folgende Eigenschaften erfüllen: `Insert` soll in linearer Zeit erfolgen, `remove` in konstanter Zeit (jeweils abhängig von der Zahl der Elemente in der Schlange).

c) Geben Sie eine Operation `changeMax` an, mittels der die Priorität des Elements mit der höchsten Priorität auf einen anderen Wert gesetzt werden kann.

d) Mittels der unter a) und b) realisierten Operationen lassen sich leicht zwei bekannte Sortierverfahren nachbilden. Erläutern Sie, um welche Verfahren es sich hierbei handelt.

Aufgabe 3

Für den Gastronomiebereich soll ein Verwaltungssystem objektorientiert entworfen werden. Folgende Zusammenhänge sollen umgesetzt werden: In einem Gastronomiebetrieb arbeiten mehrere Kellner. Zu jedem Kellner werden dessen Personalnummer, Name und Vorname erfasst. Jeder Kellner ist während seiner Arbeitszeit für mehrere Tische zuständig. Jeder Tisch hat eine eindeutige Nummer und eine kurze Beschreibung seines Standortes (z.B. „Saal1, hinten rechts"). Bestellungen von Gästen an einem Tisch werden zu einem bestimmten Zeitpunkt aufgenommen und untergliedern sich in mehrere Bestellpositionen. Alle bestellbaren Speisen und Getränke haben eine eindeutige Nummer, eine Bezeichnung und einen Preis. Wenn Gäste an einem Tisch bezahlen möchten, wird eine Rechnung erstellt. Diese enthält Informationen über den Namen und die Adresse des Betriebs, das Datum und die Uhrzeit, den Tisch, die einzelnen Rechnungspositionen und den im Rechnungsbetrag enthaltenen MwSt-Anteil.

a) Erstellen Sie für die beschriebenen Zusammenhänge ein Klassendiagramm. Jede Klasse soll mindestens zwei Attribute und zwei Methoden enthalten und die im Text beschriebenen Zusammenhänge berücksichtigen. Spezifizieren Sie alle Attribute, Methoden und Beziehungen durch Angabe von Datentypen, Beziehungsnamen und -typen und ggf. Kardinalitäten.

b) Erläutern Sie textuell und präzise, wie folgende Aufgaben mittels Ihres Klassenmodells bearbeitet werden können. Geben Sie dazu auch Pseudocode an. Ergänzen Sie Ihr Modell, falls erforderlich.

   1. Erstellung einer Rechnung
   2. Berechnung des Tagesumsatzes eines Kellners

Fortsetzung nächste Seite!

Aufgabe 4

Stellen Sie sich eine stark vereinfachte Straßenkarte vor! Die Straßen seien mit der Entfernung zwischen ihrem jeweiligen Start- und Zielpunkt beschriftet. Vereinfachend können Sie davon ausgehen, dass Straßen immer in Orten beginnen und enden.

a) Geben Sie einen geeigneten Datentyp zur Realisierung der Straßenkarte an.

b) Geben Sie einen Algorithmus an, mit dem Sie unter Verwendung des Datentyps aus Teilaufgabe a) die kürzeste Verbindung zwischen zwei Orten bestimmen können.

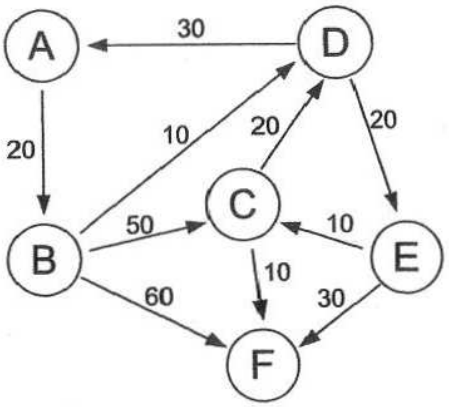
c) Gehen Sie im Folgenden vereinfachend davon aus, dass die Entfernung zwischen zwei über eine Straße miteinander verbundenen Orten überall den gleichen Wert a betrage. Beweisen Sie für diesen Fall mittels vollständiger Induktion, dass Ihr Algorithmus immer den kürzesten Weg findet.

Thema Nr. 2

Aufgaben zu Algorithmen und Datenstrukturen

Teilaufgabe 1 (Graphen)

Gegeben sei der folgende gerichtete Graph mit einer Kantenbewertung (Netzwerk):



Bestimmen Sie mit Hilfe des *Algorithmus von Dijkstra* die kürzesten Wege vom Knoten *A* zu allen anderen Knoten. Geben Sie dabei nach jedem Verarbeitungsschritt den Zustand des Arbeitsfeldes an.

Fortsetzung nächste Seite!

Teilaufgabe 2 (Binäre Bäume)

1. Zeichnen Sie einen binären Suchbaum mit den natürlichen Zahlen von 1 bis 15 und der Höhe *drei*! Ist das Ergebnis eindeutig? Begründen Sie Ihre Behauptung.

2. Geben Sie jeweils die Reihenfolge der Knoten an, wenn obiger Baum in (1) *Inorder*, (2) *Preorder*, (3) *Postorder* und (4) *Levelorder* durchlaufen wird.

3. Zeichnen Sie für jeden der obigen vier Fälle den resultierenden binären Suchbaum, wenn er ausgehend vom leeren Baum durch sukzessive Einfügeoperationen in der jeweiligen Reihenfolge ohne Ausbalancieren aufgebaut wird.

Teilaufgabe 3 (AVL-Bäume)

Gegeben sei die folgende Zahlenfolge:

120, 50, 70, 100, 90, 30, 80

Fügen Sie die Ziffern nacheinander in einen AVL-Baum ein. Notieren Sie jeweils sämtliche dabei notwendigen Zwischenschritte und das Endergebnis.

Teilaufgabe 4 (Bäume)

Gegeben sei ein binärer Baum T mit Wurzel v. Die Teilbäume von v seien balanciert und es sei $|b(v)|$ (Balance von v). Beschreiben Sie die Situation, bei der nach einer Einfach- oder Doppelrotation von v die Gesamthöhe von T nicht reduziert wird (Grafik).

Teilaufgabe 5

Das Newton-Verfahren ist ein numerisches Verfahren zur näherungsweisen Bestimmung von Nullstellen einer Funktion $f : \mathbb{R} \to \mathbb{R}$.
Zum Startwert x_0 wird der Punkt $(x_0, f(x_0))$ berechnet und die Tangente dieses Punktes bestimmt. Der Schnittpunkt der Tangente mit der x-Achse $(x_1, 0)$ liefert einen neuen Punkt: $(x_1, f(x_1))$. Zu diesem wird wieder die Tangente bestimmt und erneut der Schnittpunkt mit der x-Achse berechnet, etc.
Allgemein lautet die Iterationsschrift wie folgt:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Man bricht das Verfahren ab, wenn die Nullstelle hinreichend genau bestimmt ist oder wenn eine obere Grenze für die Anzahl der Iterationsschritte n erreicht wurde.
Ferner wird abgebrochen, falls für ein n $f'(x_n) = 0$ und zugleich $f(x_n) \neq 0$ gilt.
In den beiden letzten Fällen ist die Nullstelle mit dem Verfahren in dieser Form nicht bestimmbar.

Beispiel: $f(x) = x - \cos x$

Fortsetzung nächste Seite!

Iterationsvorschrift: $x_{n+1} = x_n - \frac{x_n - \cos x_n}{1 + \sin x_n}$

Iteration: $x_0 = 0,\ x_1 = 1,\ x_2 = 0{,}75,\ x_3 = 0{,}74,\ x_4 = 0{,}74$

Es soll nun ein Programm zur Berechnung einer Nullstelle einer beliebigen Funktion mit Hilfe des Newton-Verfahrens erstellt werden. Verwenden Sie geeignete Abbruchkriterien unter Berücksichtigung von Rundungsfehlern und Schnittstellen (Unterprogrammaufrufe) zur Berechnung von *f(x)* und *f '(x)*.

a) Erstellen Sie ein Programmablaufdiagramm nach DIN 66001 zum obigen Programm.

b) Erstellen Sie ein Blockdiagramm für das Programm und schreiben Sie das Programm in Pascal oder einer anderen Programmiersprache.

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Frühjahr 2007

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Frühjahr**	**46114**
Kennwort: _____	**2007**	
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Algorithmen, Datenstrukturen und Programmiermethodik**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 9

Bitte wenden!

Thema Nr. 1

Sämtliche Teilaufgaben sind zu bearbeiten!

Aufgabe 1:

Das Java-Programm `Converter` hat die Aufgabe, Dezimalzahlen als Zahlen zur Basis 16 darzustellen. Dabei wird die Dezimalzahl fortgesetzt ganzzahlig durch 16 dividiert, der jeweilige (eventuelle) Rest ist der Wert einer Ziffer der gesuchten Darstellung.

a) Finden Sie die Fehler in der Klasse `Hex` und korrigieren Sie diese, indem Sie jeweils die fehlerhafte(n) Zeilennummer(n) und den zugehörigen korrigierten Quelltext angeben und jeweils kurz erläutern, worin der Fehler besteht.

b) Überarbeiten Sie das Programm dahingehend, dass die Ausgabe der Ziffern in der richtigen Reihenfolge geschieht.

Quelltext `Converter.java`

```
1   class Hex {
2       char[] ziffern = {'0','1','2','3','4','5','6','7',
3                 '8','9','A','B','C','D','E','F'};
4       int h = 16;
5       void out(int z) {
6           int rest;
7           while ( z > 1 ) {
8               rest = z / h;
9               z    = z % h;
10              System.out.println(z);
11          }
12      }
13  }
14
15  class Converter {
16
17      public static void main(String[] args) throws IOException {
18
19          int zahl;
20          BufferedReader stdin =
21            new BufferedReader(new InputStreamReader(System.in));
22          String inData;
23
24          System.out.println("Gib mir eine Zahl:");
25          inData = stdin.readLine();
26          zahl   = Integer.parseInt(inData);
27
28          Hex hex = new Hex();
29          hex.out(zahl);
30      }
31  }
```

Fortsetzung nächste Seite!

Aufgabe 2:

Betrachten Sie das unten angegebene Java-Programm.

a) Welche Aufgaben haben die Methoden `foo_a()`, `foo_b()` und `foo_c()` bei der Lösung des Gesamtproblems?

b) Geben Sie das Abbruchkriterium für die Rekursion in `foo_c()` an.

c) Wie lautet der Inhalt des Arrays `data` nach dem ersten Aufruf der Methode `foo_b()`?

d) Welches Verfahren wird hier umgesetzt? Erklären Sie den Algorithmus anhand des Quellcodes.

Quelltext `Aufgabe.java`

```
1   class Meldung {
2       static void print_r(int[] arr){
3           int i;
4           for (i = 0; i < arr.length; i++){
5               System.out.print(arr[i] + ", ");
6           }
7           System.out.println();
8           }
9   }
10
11  class Algorithm {
12
13      void foo_a (int[] arr, int n, int m) {
14          int h;
15          h = arr[m];
16          arr[m] = arr[n];
17          arr[n] = h;
18      }
19
20      boolean foo_b (int[] data){
21          int i;
22          boolean doit = false;
23          for (i = 0; i < data.length - 1; i++){
24              if (data[i+1] < data[i]){
25                  doit = true;
26                  foo_a(data,i,i+1);
27              }
28          }
29          Meldung.print_r(data);
30          return doit;
31      }
32
33      void foo_c (int[] data){
34          if (foo_b(data)) {
35              foo_c(data);
36          }
37      }
38  }
39
40  class Aufgabe {
41
42      public static void main(String[] args) {
43          int[] data = {23,38,14,-3,0,14,9,103,0,-56};
44          Algorithm algorithm = new Algorithm();
45          algorithm.foo_c(data);
46      }
47  }
```

Fortsetzung nächste Seite!

Aufgabe 3:

Eine Bank verfügt über zwei Arten von Konten: Sparkonten, die keinen negativen Stand haben dürfen, und Girokonten, die überzogen werden dürfen. Der Aufruf zum Auszahlen von einem Konto soll für Spar- und Girokonten einheitlich realisiert werden. Alle Konten werden in einer gemeinsamen Liste verwaltet, die z. B. als Reihung (Array) realisiert ist. Die Liste verfügt über eine Methode, die zu einer gegebenen Kontonummer das zugehörige Konto findet.

a) Geben Sie ein Klassendiagramm mit allen für die Aufgabenbeschreibung relevanten Attributen und Methoden sowie die Kardinalitäten an.

b) Implementieren Sie die Methode, die zu einer gegebenen Kontonummer das zugehörige Konto findet (als binäre Suche).

c) Geben Sie ein Hauptprogramm an, das zur Kontonummer 1234567 das zugehörige Konto findet und bei diesem eine Auszahlung von 100 EUR vornimmt. Dabei soll keine explizite Falluntersuchung zwischen Spar- und Girokonten erforderlich sein.

Aufgabe 4:

Erläutern Sie, wie Sie ein Klassendiagramm des objektorientierten Entwurfs (z.B. UML-Klassendiagramm) in Code einer objektorientierten Programmiersprache Ihrer Wahl übersetzen. Berücksichtigen Sie hierbei folgende Elemente von Klassendiagrammen: Klassen, Attribute, Methoden, Vererbungsbeziehungen, Aggregationen, Assoziationen. Unterscheiden Sie bei Assoziationen die Fälle 1:1-, 1:n- und n:m-Beziehung.

Aufgabe 5:

Gegeben seien zwei Beispiele für einfache arithmetische Ausdrücke, in denen Variablen $a, b, c, \ldots$ sowie Operatoren $+, -, *, /$ vorkommen und die vollständig geklammert sind.

1. $((a + b) * c)$
2. $((x * y)/(x - y))$

a) Geben Sie ein Verfahren an, das mit Hilfe eines Stapels (Stack) beliebige, wohlgeformte arithmetische Ausdrücke auswertet. Verwenden Sie eine gängige höhere Programmiersprache Ihrer Wahl oder einen Pseudocode. Gehen Sie davon aus, dass die Stapeloperationen Push und Pop zur Verfügung stehen.

b) Für das 1. Beispiel soll das Verfahren folgende Belegungen des Stapels erzeugen. (Eine Stapelbelegung steht in einer Zeile, das oberste Stapelelement steht rechts.)

a	
$a+$	
$a + b$	
Z_1	($a + b$ wird zu Z_1 ausgewertet)
Z_1*	
$Z_1 * c$	
Z_2	($Z_1 * c$ wird zu Z_2 ausgewertet)

Welche Folge von Belegungen des Stapels wird für das 2. Beispiel erzeugt?

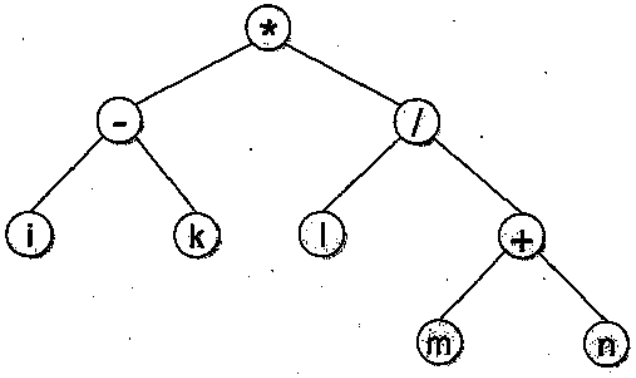
Fortsetzung nächste Seite!

Aufgabe 6:

Ein arithmetischer Ausdruck lautet in Infixnotation $((a + b) * c)$ und in Postfixnotation $ab + c*$. Bei der Postfixnotation kann auf Klammern verzichtet werden.

a) Geben Sie den Ausdruck $((x * y)/(x - y))$ in Postfixnotation an.

b) Ein arithmetischer Ausdruck kann in einem Binärbaum gespeichert werden. Traversieren Sie



den Binärbaum in

- Inorder,
- Postorder.

Geben Sie jeweils den arithmetischen Ausdruck an, der ausgegeben wird. Begründen Sie, ob und nach welcher Strategie Klammern gesetzt werden.

c) Geben Sie ein Verfahren an, das einen arithmetischen Ausdruck, der als Binärbaum vorliegt, in

- Infixnotation,
- Postfixnotation.

ausgibt. Verwenden Sie eine gängige höhere Programmiersprache oder einen entsprechenden Pseudocode.

Hinweis:

Sie können wahlweise davon ausgehen, dass der Binärbaum als Reihung (Array) oder mit Zeigern realisiert ist und die entsprechenden Operationen zur Verfügung stehen. Achten Sie darauf, dass Klammern gesetzt werden, falls erforderlich.

Thema Nr. 2

Sämtliche Teilaufgaben sind zu bearbeiten!

Aufgabe 1:

In der folgenden Aufgabe soll ein einfaches Programm zur Verwaltung eines Fußballvereins entwickelt werden.

Entwickeln Sie zunächst die Klasse Spieler. In Objekten dieser Klasse soll die folgende Information gespeichert sein:

*Name
*Vorname
*Geburtsdatum
*Anzahl gespielter Spiele
*Anzahl geschossener Tore
*Spielernummer

Die Klasse soll einen Konstruktor besitzen, der ein neues Objekt anlegt. Dem Konstruktor soll der Vor- und Nachnahme des Spielers und dessen Geburtsdatum übergeben werden. Die Spielernummer soll vom Programm selbst vergeben werden. Dabei muss darauf geachtet werden, dass kein Spieler eine bereits vergebene Nummer erhält.

Außerdem soll die Klasse eine Methode besitzen, die die Zahl der Spiele um eins erhöht. Dieser wird die Anzahl der Tore übergeben, die der Spieler in diesem Spiel erzielt hat. Zuletzt soll die Klasse noch über eine Methode verfügen, die alle relevanten Informationen über den Spieler in einer Zeile ausgibt. Dazu gehören: Name, Vorname, Geburtsdatum, Anzahl Spiele, Anzahl Tore und die durchschnittliche Anzahl von Toren pro Spiel.

A) Spezifizieren Sie die Klasse Spieler, indem Sie die Signaturen des Konstruktors und der Methoden in einer objektorientierten Programmiersprache Ihrer Wahl angeben. Überlegen Sie sich außerdem, welche Werte für die jeweiligen Parameter zulässig sind, bzw. bei welchen Parametern eine Ausnahme generiert werden soll. (15 Minuten)

B) Implementieren Sie diese Klasse. Achten Sie darauf, dass die Implementierung zu der von Ihnen erstellten Spezifikation aus Teilaufgabe A passt. Insbesondere sollen ungültige Parameter erkannt und mit einer Ausnahme behandelt werden. (20 Minuten)

Des Weiteren soll eine Klasse Mannschaft implementiert werden. Eine Mannschaft besteht aus bis zu elf Stammspielern und einer beliebigen Anzahl von Ersatzspielern. Sie soll über einen Konstruktor verfügen, der eine leere Mannschaft erzeugt. Mit Hilfe einer Methode sollen der Mannschaft Spieler hinzugefügt werden. Hat die Mannschaft weniger als 11 Stammspieler, wird der neu hinzugefügte Spieler zum Stammspieler, andernfalls zum Ersatzspieler. Eine zweite Methode dient dazu, einen Ersatzspieler mit einem Stammspieler zu tauschen: Der Stammspieler wird zum Ersatzspieler und umgekehrt.

Zuletzt soll eine Methode entwickelt werden, die alle Informationen über die Mannschaft in einer Tabelle auflistet.

Fortsetzung nächste Seite!

Dazu werden zunächst alle Stammspieler aufgelistet, anschließend alle Ersatzspieler. Außerdem wird zuletzt die durchschnittliche Anzahl von Toren pro Spieler und Spiel ausgegeben.

C) Spezifizieren Sie die Klasse Mannschaft (analog zu A). (15 Minuten)

D) Implementieren Sie die Klasse entsprechend ihrer Spezifikation. Erweitern Sie evtl. auch die Klasse Spieler, falls dies notwendig sein sollte. (20 Minuten)

Aufgabe 2:

Arithmetisches Mittel eines Feldes

Gegeben sei ein Feld von Fließkommazahlen der Länge n. Gesucht ist das arithmetische Mittel M dieser Zahlen, definiert durch:
M = Summe der Elemente / n

Für diese Aufgabe sollen zwei Algorithmen entwickelt und anschließend miteinander verglichen werden.

A) Entwickeln Sie zunächst die Funktion MIterativ, welche die Aufgabe mit einem iterativen Programmieransatz löst. (10 Minuten)

B) Entwickeln Sie die Funktion MRekursiv, welche die Aufgabe mittels einer rekursiven Funktion löst, die das Feld in jedem Rekursionsschritt in zwei Hälften teilt, jeweils das Mittel der beiden Hälften berechnet und anschließend das Mittel der beiden Ergebnisse bildet. Überlegen Sie sich zunächst ein geeignetes Rekursionsende. Der Einfachheit halber dürfen Sie annehmen, dass die Länge des Feldes eine Zweierpotenz ist. Falls dies nicht der Fall ist, soll eine Fehlermeldung ausgegeben werden. (20 Minuten)

C) Entwickeln Sie ein Hauptprogramm, das

* solange Werte von der Standardeingabe liest, bis das Dateiende erreicht wird,
* diese in einem Feld ablegt,
* das arithmetische Mittel mit Hilfe der obigen Funktionen ermittelt,
* dieses auf der Standardausgabe ausgibt.

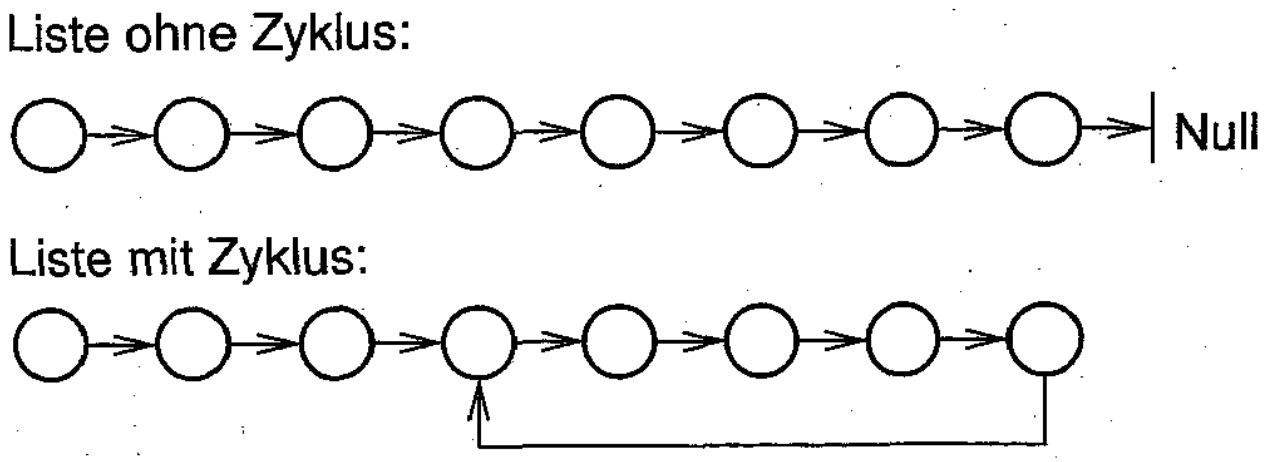
Hinweis:
Sie dürfen auf die Teilergebnisse A und B auch dann zurückgreifen, wenn Sie diese Aufgaben noch nicht vollständig gelöst haben. (20 Minuten)

D) Vergleichen Sie die Lösungen aus Aufgabe A) und B).
Welche Vor- und Nachteile gibt es?
Für welche Lösung würden Sie sich entscheiden? Warum? (10 Minuten)

Fortsetzung nächste Seite!

Aufgabe 3:
Zyklen in verketteten Listen

Gegeben sei eine verkettete Liste, die einen Zyklus enthalten kann. Der Hase-Igel-Algorithmus überprüft, ob dies der Fall ist und soll im Folgenden implementiert werden.



Beim Hasen bzw. Igel handelt es sich jeweils um eine Referenz auf ein Listenelement. Beim Durchlauf durch die Liste wird der Igel um jeweils eine Position, der Hase jedoch um zwei Positionen vorgerückt.

Die Liste bietet also folgende Methoden an:

boolean kannIgel();//	prüft, ob der Igel um eine Position vorrücken kann oder ob das Ende der Liste bereits erreicht ist.
boolean kannHase();//	prüft, ob der Hase um zwei Positionen vorrücken kann oder ob das Ende der Liste bereits erreicht ist.
ListItem bewegeIgel();//	rückt den Igel um eine Position vor und liefert das entsprechende Listenelement. Beim ersten Aufruf wird das erste Listenelement geliefert.
ListItem bewegeHase();//	rückt den Hasen um zwei Positionen vor und liefert das entsprechende Listenelement. Beim ersten Aufruf wird das zweite Listenelement geliefert.

A) Implementieren Sie die oben angegebenen Methoden der Klasse List. Gehen Sie dabei davon aus, dass die Klasse über zwei Referenzen auf Objekte vom Typ ListItem verfügt, die den Hasen und den Igel implementieren. Zu Beginn sind beide Referenzen mit null initialisiert. Außerdem gibt es noch die Referenz start, die auf das erste Element der Liste zeigt. Die Klasse ListItem können Sie als gegeben betrachten. Sie verfügt über folgende Methode:

ListItem next();// liefert das nächste Listenelement oder null, falls das Ende der Liste erreicht ist.

Andere Klassen brauchen/dürfen nicht verwendet werden. (20 Minuten)

Fortsetzung nächste Seite!

B) Beim Hase-Igel-Algorithmus wird der Igel auf das erste Listenelement gesetzt, der Hase auf das zweite. Hase und Igel rücken dann jeweils zwei bzw. ein Feld vor. Der Algorithmus terminiert, falls:

- einer der Referenzen das Listenende erreicht. In diesem Fall besitzt die Liste keinen Zyklus.
- der Igel vom Hasen eingeholt wird, d.h., beide Referenzen zeigen auf dasselbe Listenelement. In diesem Fall besitzt die Liste einen Zyklus.

Implementieren Sie den Hase-Igel-Algorithmus unter Verwendung der Ergebnisse aus A! (20 Minuten)

C) Wie ist die Zeitkomplexität des Hase-Igel-Algorithmus?
Wie ist die Speicherkomplexität? (10 Minuten)

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Herbst 2007

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	**Herbst 2007**	**46114**
Arbeitsplatz-Nr.: _____		

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: **6**

Bitte wenden!

Thema Nr. 1

Aufgabe 1

Eine Liste ist ein abstrakter Datentyp, der die Ablage einer Folge von Elementen gleichen Typs erlaubt und der die folgenden Operationen unterstützt:

- Einfügen eines Elements am Anfang oder an einer beliebigen Position der Liste
- Löschen eines Elements am Anfang oder an einer beliebigen Position der Liste
- Suchen eines Elements in der Liste

Wir betrachten im Folgenden Listen, in denen Integerwerte in aufsteigender Reihenfolge ihres Wertes sortiert abgelegt sind.
Zur Realisierung des abstrakten Datentyps betrachten wir doppelt verkettete Listen, die für jeden Integerwert ein separates Datenobjekt verwenden, welches mit dem Vorgänger- und dem Nachfolgerobjekt durch jeweils einen Verweis verbunden ist. Realisieren Sie die folgenden Aufgaben in C, C++ oder Java ohne Verwendung vordefinierter Klassen oder Bibliotheken zur Realisierung von Listen!

a) Deklarieren Sie einen Datentyp ELEM, der als Datenstruktur für eine doppelt verkettete Liste zur Ablage von Integerwerten verwendet werden kann!

b) Geben Sie eine Funktion `make_element(int value)` an, die ein Listenelement für eine doppelt verkettete Liste erzeugt, den als Parameter mitgelieferten Wert ablegt und einen Verweis auf das Listenelement als Rückgabewert zurückliefert!

c) Geben Sie eine Funktion `insert_element()` an, die einen als Parameter mitgelieferten Integerwert in eine doppelt verkettete, sortierte Liste so einfügt, dass die resultierende Liste wieder sortiert ist! Beachten Sie dabei auch die Spezialfälle, dass das einzufügende Element an den Anfang oder das Ende der Liste gehört! Ist der einzufügende Integerwert bereits in der Liste enthalten, soll er nicht erneut eingefügt werden. Eine Referenz auf die Liste, in die das Element eingefügt werden soll, soll ebenfalls als Parameter übergeben werden. Die Funktion `insert_element()` soll eine Referenz auf die neue Liste als Rückgabewert zurückliefern.

d) Geben Sie eine Funktion `delete_element()` an, die einen als Parameter mitgelieferten Integerwert aus einer doppelt verketteten, sortierten Liste löscht, falls der Integerwert in der Liste vorhanden ist!
Beachten Sie auch die Spezialfälle, dass das Element am Anfang oder am Ende der Liste gelöscht wird! Eine Referenz auf die Liste, aus der das Element gelöscht werden soll, soll ebenfalls als Parameter übergeben werden! Die Funktion `delete_element()` soll eine Referenz auf die neue Liste als Rückgabewert zurückliefern.

e) Geben Sie eine Funktion `compare_lists()` an, die zwei doppelt verkettete, sortierte Listen miteinander vergleicht! Die Funktion soll `true(1)` zurückliefern, falls die beiden Listen die gleichen Elemente in der gleichen Reihenfolge enthalten. Andernfalls soll `false(0)` zurückgeliefert werden. Referenzen auf die beiden Listen sollen als Parameter mitgeliefert werden. Beachten Sie auch den Spezialfall, dass eine der beiden Listen leer ist!

Fortsetzung nächste Seite!

f) Geben Sie eine Funktion `merge_lists()` an, die zwei doppelt verkettete, sortierte Listen so mischt, dass die resultierende Liste ebenfalls sortiert ist und jedes in einer der beiden Eingabelisten enthaltene Element genau einmal enthält! Referenzen auf die beiden Eingabelisten sollen als Parameter mitgeliefert werden! Die neu entstehende Liste soll als Rückgabewert zurückgeliefert werden.

Aufgabe 2

Sortieren durch Mischen (Mergesort) ist ein Sortierverfahren, das nach dem divide-and-conquer-Prinzip arbeitet. Wir betrachten im Folgenden die Anwendung dieses Verfahrens zum Sortieren von Integerzahlen. Die Sortierung soll in aufsteigender Reihenfolge der Werte erfolgen. Wir nehmen dabei an, dass die zu sortierenden Zahlen in einer doppelt verketteten Liste abgelegt sind (vgl. Aufgabe 1).

a) Beschreiben Sie die Arbeitsweise des divide-and-conquer-Prinzips im allgemeinen Fall! Geben Sie dabei die Bedeutung der Schritte divide (Aufteilen), conquer (Lösung der Unterprobleme) und combine (Zusammensetzen) an!

b) Beschreiben Sie die Arbeitsweise des Algorithmus Mergesort. Geben Sie dabei an, worin die Schritte divide, conquer und combine im konkreten Fall bestehen!

c) Geben Sie eine Funktion `split()` in C, C++ oder Java an, die eine vorliegende (unsortierte) doppelt verkettete Liste von Integerzahlen so aufspaltet, dass dies dem divide-Schritt des Mergesort-Algorithmus entspricht!

d) Geben Sie eine Funktion `mergesort()` in C, C++ oder Java an, die in Form einer (unsortierten) doppelt verketteten Liste vorliegende Zahlen nach dem Prinzip von Mergesort sortiert! Verwenden Sie für das Aufspalten der Liste die Funktion `split()` aus Aufgabenteil c. Verwenden Sie für das Zusammenführen zweier sortierter Listen die Funktion `merge_lists()` aus Aufgabenteil f von Aufgabe 1. Formulieren Sie die Funktion `mergesort()` nach Vorgabe des divide-and-conquer-Prinzips als rekursive Funktion!

e) Identifizieren Sie in der von Ihnen im Aufgabenteil d realisierten Funktion die Schritte divide, conquer und combine. Schätzen Sie anhand der von Ihnen realisierten Funktion die asymptotische Laufzeit dieser Schritte ab, indem Sie die O-Notation verwenden!
Leiten Sie aus der Laufzeit der Einzelschritte die Laufzeit des Gesamtverfahrens ab! Begründen Sie Ihre Antwort!

Aufgabe 3

Ein Keller ist ein abstrakter Datentyp zur Ablage von Elementen des gleichen Typs, wobei ein Zugriff nur an einem Ende des Kellers möglich ist. Auf einem Keller sind folgende Operationen vorgesehen:

- `push(e)` : Einfügen eines Elements `e` am freien Ende des Kellers
- `e=pop()` : Entfernen eines Elements vom freien Ende des Kellers

Fortsetzung nächste Seite!

Wir betrachten im Folgenden einen Keller zur Ablage von Integerwerten. Für die Realisierung des Datentyps verwenden wir eine doppelt verkettete Liste. Verwenden Sie zur Realisierung der folgenden Aufgabenpunkte C, C++ oder Java ohne Verwendung vordefinierter Klassen oder Bibliotheken!

a) Geben Sie eine Funktion `push()` zur Ablage eines als Parameter mitgelieferten Elements am freien Ende des Kellers an!

b) Geben Sie eine Funktion `pop()` zur Entnahme des obersten Kellerelements an; dieses soll als Rückgabewert zurückgeliefert werden!

c) Geben Sie eine Funktion `count()` an, die die Anzahl der in dem Keller aktuell abgelegten Werte zählt!

Thema Nr. 2

Allgemeines zu den Aufgaben

Entwerfen und programmieren Sie ein Telefonverzeichnis, d. h. eine Liste mit einzelnen Einträgen, die Name, Vorname, Telefonvorwahl und Telefonnummer enthalten! Verwenden Sie als Programmier-Modell OOP („Objekt orientierte Programmierung") (z. B. Java, C++). In jedem Fall sollte Ihr Programm modular aufgebaut sein!

Entwerfen Sie zunächst in Aufgabe 1 eine Spezifikation für diese Aufgabe und begründen Sie dabei die einzelnen Entscheidungen! Formulieren Sie dann in Aufgabe 2 Ihren Entwurf in einer Ihnen geläufigen Programmiersprache, wobei Sie sich nicht exakt an die Syntax der verwendeten Programmiersprache halten müssen! Sie dürfen dabei, soweit es notwendig ist, elementare Hilfsfunktionen (etwa für die Ein-/Ausgabe) als vorhanden annehmen.

Aufgabe 1 (Entwurf)

Das Telefonverzeichnis soll in jedem Fall die beiden folgenden Klassen (Module) enthalten: Eine Klasse **„Telefoneintrag"**, die einen einzelnen Telefoneintrag beschreibt und eine zweite Klasse **„Liste"**, die eine Listenverwaltung realisiert, aber so allgemein, dass sie auch andere Objekte verwalten könnte (z. B. Bücher). Entwerfen Sie zusätzlich ein Hauptprogramm, mit dem Sie ein Telefonverzeichnis bearbeiten könnten! (Nur skizzieren!)

1.1 Klasse „Telefonbucheintrag"

1.1.1 Die Klasse **TelefonEintrag**. soll folgende Eigenschaften (Einträge) haben:

- Familienname
- Vorname
- Vorwahl
- Telefonnummer

Welche Datentypen verwenden Sie für die einzelnen Eigenschaften? Begründen Sie Ihre Entscheidung!

Fortsetzung nächste Seite!

1.1.2 Skizzieren Sie folgende Methoden (Funktionen/Prozeduren) für die Klasse Telefoneintrag: (Ein neues Objekt muss mindestens Name und Vorname enthalten!)

- setTelefonNummer // soll Vorwahl und Telefonnummer in das Objekt eintragen.
- getTelefonNummer // liefert Vorwahl und Telefonnummer des Objekts.
- vergleicheMit // soll zwei Objekte nach Name und Vornamen miteinander.
  // lexikographisch vergleichen und größer, kleiner, gleich liefern.

(Beim Vergleich soll ein Eintrag als größer als ein anderer gelten, wenn er in einem üblichen Telefonbuch hinter diesem stehen würde.)

1.2 Listen

1.2.1 Beschreiben Sie zunächst verschiedene Datenstrukturen, mit denen man Listen implementieren kann! Diskutieren Sie Vor- und Nachteile der verschiedenen Möglichkeiten!

1.3 Klasse Liste

1.3.1 Skizzieren Sie eine **Klasse „Liste“**, die folgende Methoden zur Verfügung stellt:

- eintragen // trägt ein Objekt in die Liste ein.
  // Die Liste darf keine zwei „gleichen“ Einträge enthalten.
- loeschen // löscht ein Objekt aus der Liste.
- suchen // sucht ein Objekt in der Liste. Verwenden Sie dabei zum
  // Vergleichen die in der Klasse der verwalteten Objekte
  // angebotene Methode **„vergleicheMit“**.
- eintrag_aendern // damit kann eine Telefonnummer geändert werden.
  // Diese Methode muss so allgemein sein, dass sie auch
  // für Listeneinträge einer anderen Klasse funktioniert!

Aufgabe 2: (Programmierung)

2.1 Programmieren Sie die Klasse Telefoneintrag!

2.2 Programmieren Sie die Klasse Liste!

2.3 Skizzieren Sie ein Hauptprogramm, mit dem Sie ein Telefonverzeichnis anlegen und bearbeiten können, mit den folgenden Funktionen:
- Eintragen, - Eintrag löschen, - Telefonnummer suchen, - Telefonnummer ändern.
Dazu können Sie beliebige Ein-/Ausgabe-Methoden aufrufen, mit denen Sie Werte von der Tastatur einlesen und in einem Bildschirmfenster ausgeben können. Zusätzlich stehen Ihnen nach Belieben „Befehlsschalter“ zur Verfügung.

Fortsetzung nächste Seite!

Aufgabe 3: (Systementwurf)

Ein Programm zum Lesen und Bearbeiten eines Telefonbuches soll im Allgemeinen ja von mehreren Personen gleichzeitig benutzt werden können.

3.1 Beschreiben Sie die Probleme, die bezüglich der Datenintegrität auftreten können, wenn mehrere Personen berechtigt sind, das Telefonbuch zu ändern!

3.2 Skizzieren Sie eine Lösung für dieses Problem!

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Frühjahr 2008

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____ Kennwort: _____ Arbeitsplatz-Nr.: _____	**Frühjahr 2008**	**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: **10**

Bitte wenden!

Thema Nr. 1

Aufgabe 1: (Rekursion und Komplexität)

In einer funktionalen Programmiersprache Ihrer Wahl:

a) Definieren Sie rekursiv die Funktion *app*, die zwei Listen verbindet!
Beispiel: *app* ([1,2,3],[4,5,6]) = [1,2,3,4,5,6]

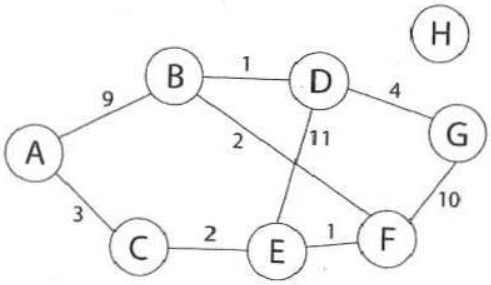
b) Welche Zeitkomplexität im schlimmsten Fall hat *app*? Geben Sie die Aufwandsklasse in der O-Notation in Abhängigkeit von der Größe der beiden Eingabeparameter an und begründen Sie Ihre Aussage!

c) Definieren Sie auch rekursiv die Funktion *rev*, die eine Liste revertiert!
Beispiel: *rev* [1,2,3,4] = [4,3,2,1]

d) Welche Zeitkomplexität im schlimmsten Fall hat *rev* in Abhängigkeit von dem Eingabeparameter (mit Begründung)?

Aufgabe 2: (Dijkstra-Algorithmus für kürzeste Wege)

Führen Sie auf folgendem Graphen den Algorithmus von Dijkstra zur Bestimmung der kürzesten Wege aus! Der Startknoten ist *A*. Geben Sie dabei nach jedem Schritt den Zustand des Heaps an! Geben Sie am Ende den kürzesten Pfad von *A* zu jedem Knoten sowie dessen Gewicht an!



Aufgabe 3: (Java-Arrays und Verifikation)

a) Implementieren Sie in Java eine Methode

```
int maximumPosition(int[] a),
```

die den kleinsten Index (Position) im Array `a` liefert, an der ein Element mit dem größten Wert gespeichert ist! Falls das Array leer ist, soll die Zahl `-1` zurückgegeben werden.
Beispiel: `maximumPosition`, angewendet auf das Array `{3, 9, 5, 9}`, soll den Wert `1` liefern.

b) Wie lautet die Schleifeninvariante und an welchen Stellen des Programms muss sie gelten?

Fortsetzung nächste Seite!

Aufgabe 4: (Abstiegsfunktion)

Hier ist eine rekursive Funktion (in Haskell kodiert), die angibt, ob die ihr übergebene ganze Zahl gerade oder ungerade ist:

```
even :: Int → Bool
even x = if x<0 then even (-x)
                else if x==0 then True
                             else if x==1 then False
                                          else even (x-2)
```

Geben Sie eine Abstiegsfunktion *h* an, die die Terminierung von `even` für alle Eingabewerte belegt:

a) Geben Sie die Signatur von *h* an!
b) Geben Sie die Funktionsvorschrift von *h* an!
c) Zeigen Sie, dass *h* die Eigenschaften einer Abstiegsfunktion erfüllt!

Aufgabe 5: (Parameterübergabe)

Gegeben ist folgendes imperative Programm in Java-Syntax:

```
int   i;
int[] a;

void p(int k) {
   System.out.print(k);              // Pos 1
   i = 1;
   k = k+1;
   System.out.print(a[0]);           // Pos 2
   System.out.print(k);              // Pos 3
}

void main() {
   a = new int[] {2, 9};
   i = 0;
   p(a[i]);
   System.out.print(a[0]);           // Pos 4
   System.out.print(a[1]);           // Pos 5
}
```

Übertragen Sie die folgende Tabelle und geben Sie darin die Zahlen an, die das Programm an den kommentierten Programmpositionen unter den verschiedenen Parameterübergabearten ausgibt!

call by	value	value-result	reference	name
Pos 1				
Pos 2				
Pos 3				
Pos 4				
Pos 5				

Fortsetzung nächste Seite!

Aufgabe 6: (Abstrakte Datentypen)

Betrachten wir einen axiomatisch definierten abstrakten Datentyp! Die *Signatur* vereinbart u. a. die Definitions- und Wertbereiche der Operationen, die für den Datentyp definiert sind. Die Bedeutungen (Semantik) der Operationen werden dann mit Gleichungen angegeben, die die Operationen zueinander in Beziehung setzen.

a) Geben Sie die Signaturen und die beschreibenden Gleichungen für die Operationen eines Kellers (Stacks) an, als da sind:
   - `push`: lege ein neues Element auf den Stack
   - `pop`: entferne ein Element vom Stack
   - `top`: gib den Wert des obersten Elements auf dem Stack an
   - `isempty`: gib an, ob der Stack leer ist oder nicht

b) Die Implementierung eines abstrakten Datentyps wird in der Regel vor dem Benutzer versteckt (Geheimnisprinzip). Geben Sie zwei Gründe dafür an.

Aufgabe 7: (Heap und Suchbaum)

a) Gegeben ist ein MinHeap mit der Belegung

1	3	2	4	5	6	8	7

Das Element 1 soll entfernt und danach die Heapeigenschaft wiederhergestellt werden. Geben Sie die Belegung des Heaps nach jeder einzelnen Vertauschung an!

b) Wir betrachten Eingabe und Löschen in einem binären Suchbaum.
   - Zeichnen Sie den Baum, der durch die folgende Eingabesequenz entsteht:
     ```
     insert(Nadine); insert(Roland); insert(Don);
     insert(Alfons); insert(Leo); insert(Petra);
     insert(Dieter); insert(Martin); insert(Fredo);
     insert(Gregor); insert(Berta); insert(Andrea);
     insert(Chris); insert(Gerd);
     ```
   - Zeichnen Sie nun den Baum, der nach Löschen des Eintrags `Don` entsteht!

Thema Nr. 2

Aufgabe 1

Betrachten Sie die beiden rekursiven Algorithmen:

```
1 long fakultaet(int n){
            return (n==1) ? 1 : n * fakultaet(n-1);
        }
```

```
2 void lege(int scheibe, char quelle, char ziel, char ablage) {
        if (scheibe > 0){
            lege(scheibe - 1, quelle, ablage, ziel);
            System.out.println("Ich lege eine Scheibe von "
                + quelle + " nach " + ziel);
            lege(scheibe - 1, ablage, ziel, quelle);
        }
    }
```

a) Welche Rekursionsform liegt jeweils vor?
b) Geben Sie den Algorithmus 1 in iterativer Form an!
c) Geben Sie für beide rekursiven Algorithmen eine Aufwandsabschätzung in O-Notation!

Aufgabe 2

Gegeben ist folgendes JAVA-Programm, das den Gesamtlohn der Mitarbeiter einer Gärtnerei ermittelt:

```
1   abstract class Mitarbeiter {
2       private String name;
3       *   private double stdlohn = 11.50;
4       float std;
5       *   public void Mitarbeiter(String name){
6           this.name = name;
7       }
8
9       abstract double getLohn();
10
11      public void setStdLohn(double arg) {
12          this.stdlohn = arg;
13      }
14
15      public void setStd(float arg) {
16          this.std = arg;
17      }
18  }
19  public class MitList {
20      private static Mitarbeiter[] alle = new Mitarbeiter[3];
```

Fortsetzung nächste Seite!

```
21
22      static void set(Mitarbeiter m, int n){
23          alle[n] = m;
24      }
25
26      static double getGesamtlohn(){
27          double l = 0;
28          for (int i = 0; i < alle.length; i++) {
29              l += alle[i].getLohn();
30          }
31          return l;
32      }
33  }
34  public class Gaertner extends Mitarbeiter {
35      float ueberstd;
36
37      public double getLohn(){
38          return (this.std + this.ueberstd) * this.stdlohn;
39      }
40
41      public void setUeberstd(float x){
42          this.ueberstd = x;
43      }
44  }
45  public class Verkaeufer extends Mitarbeiter {
46
47      public double getLohn(){
48          return this.std * this.stdlohn;
49      }
50  }
51  public class Gaertnerei {
52
53      public static void main(String[] args) {
54          *    Gaertner albert = new Mitarbeiter("Albert");
55          *    Gaertner.setUeberstd(2);
56          Verkaeufer betti = new Verkaeufer("Betti");
57          betti.setStdLohn(12.50);
58          albert.setStd(38.5);
59          betti.setStd(42);
60
61          MitList.set(albert,0);
62          MitList.set(betti,1);
63          MitList.set(new Verkaeufer("Charlie"),2);
64
65          System.out.println(MitList.getGesamtlohn());
66      }
67
68  }
```

Fortsetzung nächste Seite!

a) Das gegebene Java-Programm enthält vier fehlerhafte Codezeilen, die mit einem Stern (*) markiert sind. Geben Sie die Zeilen korrekt an.

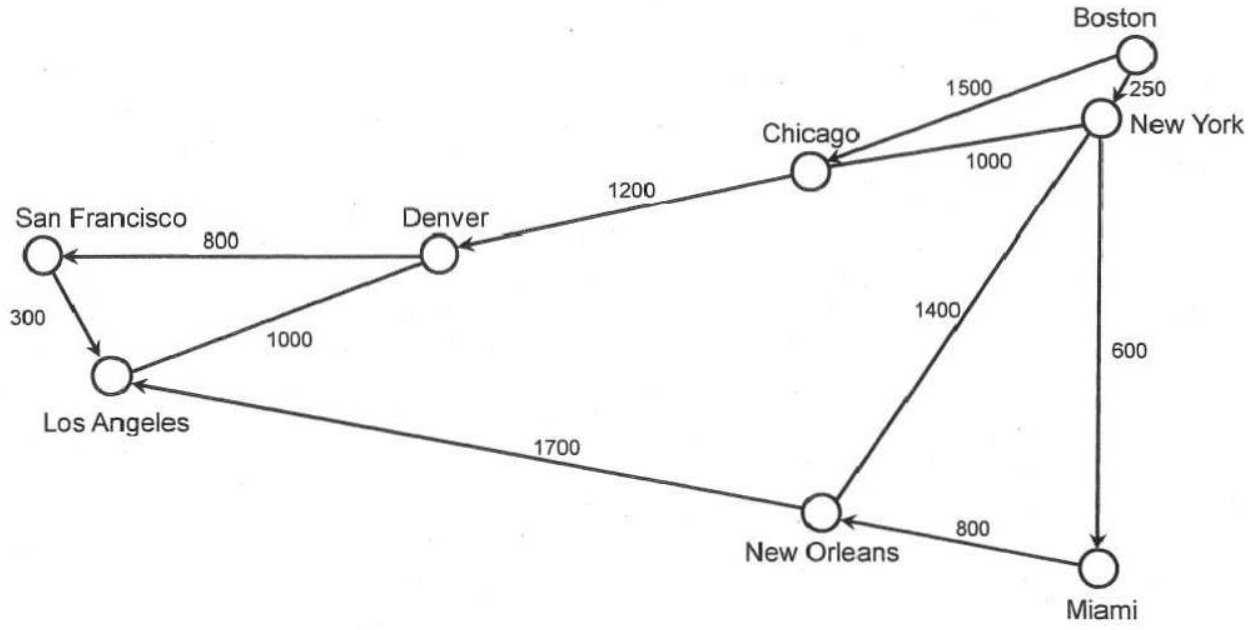
b) In den Klassen `Verkaeufer` und `Gaertner` fehlen jeweils die Konstruktoren. Geben Sie geeignete Konstruktoren an!

c) Erklären Sie das Konzept der polymorphen Operationen am Beispiel der Operation `getLohn()` in höchstens drei Sätzen!

d) An welchen Stellen des Programms wird Polymorphie von Variablen benutzt? Begründen Sie Ihre Antworten in höchstens drei Sätzen!

Aufgabe 3

Gegeben ist folgende Darstellung von Flugentfernungen verschiedener Orte:



a) Stellen Sie die Adjazenzmatrix für den obigen Graphen auf! Beachten Sie insbesondere, dass der Graph sowohl gerichtete als auch ungerichtete Teilabschnitte enthält!

b) Erläutern Sie in höchstens drei Sätzen, welche Bedeutung eine vollständig mit nicht-negativen Zahlen gefüllte Zeile in der Adjazenzmatrix besitzt!

c) Stellen Sie den Graphen auch als Adjazenzliste dar!

d) Stellen Sie tabellarisch Vor- und Nachteile der Speicherung des Graphen unter Verwendung dieser beiden Darstellungsvarianten gegenüber!

e) Berechnen Sie nun mit Hilfe des Algorithmus von Dijkstra die kürzesten Wege von Boston zu allen anderen angegebenen Städten! Geben Sie alle Zwischenschritte an!

Fortsetzung nächste Seite!

f) Es ergeben sich während des Ablaufs des Algorithmus verschiedene Wege mit gleichen Entfernungen für einen bestimmten Zielort! Entscheiden Sie sich in diesem Fall systematisch für eine der Varianten und begründen Sie Ihre Entscheidung!

g) Begründen und erläutern Sie in höchstens drei Sätzen die Aussage: *„Bei der Implementierung bestimmter Teile des Algorithmus von Dijkstra kann ein Heap (eine Halde) vorteilhaft eingesetzt werden.“*!

Aufgabe 4

Im Folgenden werden drei rekursive Definitionen eines Binärbaumes angegeben. Nur die Definition I stellt eine korrekte Beschreibung eines Binärbaumes dar. Begründen Sie, warum die Definitionen II und III keine korrekten Spezifikationen sind. Illustrieren Sie Ihre Begründung jeweils mit einem Beispiel.

I. Ein Binärbaum ist ein Blatt oder ein Knoten mit zwei Bäumen als linkem und rechtem Sohn.
II. Ein Binärbaum ist ein Knoten mit zwei Bäumen als linkem und rechtem Sohn.
III. Ein Binärbaum besteht aus einem Wurzelknoten und endlich vielen weiteren Knoten. Jeder Knoten ist entweder ein Blatt oder wieder Wurzelknoten eines Baumes.

Aufgabe 5

Ein Suchbaum sei definiert als Binärbaum mit folgender charakterisierender Eigenschaft:

> „In einem binären Suchbaum gilt für jeden Knoten: Alle Schlüssel, die kleiner sind als der Schlüssel des Knotens werden im linken Teilbaum gespeichert, alle Schlüssel, die größer sind, im rechten Teilbaum. Ein Schlüssel kommt höchstens einmal vor.“

Der abstrakte Datentyp (ADT) `TREE` enthalte mindestens die Signaturen:

```
create:                        -> TREE
bintree:  TREE x KEY x TREE    -> TREE
insert:   TREE x KEY           -> TREE
delete:   TREE x KEY           -> TREE
empty:    TREE                 -> Boolean
search:   TREE x KEY           -> Boolean
```

und mindestens die Axiome:

```
i,k: KEY, l,r,t: TREE
empty(create) = TRUE
insert(create,k) = bintree(create,k,create)
search(delete(t,k),k) = FALSE
```

a) Nennen Sie die wichtigsten Gründe für die Verwendung abstrakter Datentypen!

b) Ergänzen Sie das unvollständige Axiomensystem des ADT `TREE` für die Operation `insert`, so dass das Einfügen eines Schlüssels in einen binären Suchbaum gemäß der obigen Charakterisierung korrekt spezifiziert wird!

c) Fügen Sie die Schlüssel 5, 4, 2, 3, 8, 6, 5, 9, 1, 7 in einen anfangs leeren Suchbaum ein!

Fortsetzung nächste Seite!

Für die Operation `dunno` seien die Signatur und das folgende Axiom gegeben.

```
dunno: TREE -> KEY
k: KEY, l,r: TREE
dunno(bintree(l,k,r)) = { k          falls empty(r)
                        { dunno(r)   sonst
```

d) Beschreiben Sie kurz die Wirkungsweise und das Resultat der Operation `dunno`!

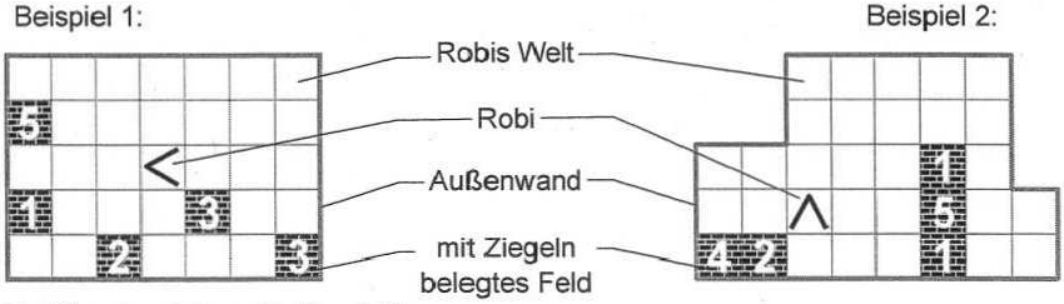
e) Schätzen Sie den Aufwand der Operation `dunno` im schlechtesten Fall (worst case) ab!

f) Wie kann der Suchbaum aufgebaut werden, um den worst case-Aufwand der Prozedur `dunno` und beliebiger Suchvorgänge zu verbessern? Skizzieren Sie eine gängige Vorgehensweise in wenigen Sätzen!

Aufgabe 6

Zu Vermittlung der Grundlagen der Algorithmik im Unterricht kommen oft schulspezifische Lern- und Programmierumgebungen (wie z. B. Karol, der Roboter oder Kara, der Marienkäfer) zum Einsatz. Im Rahmen dieser Aufgabe sollen Sie eine solche Umgebung in Teilen entwerfen und implementieren. Die hier zu entwerfende Umgebung erhält den Namen *Robi, der Roboter*.

Robi ist ein virtuelles Robotersystem. Der Roboter *Robi* (dargestellt durch einen Pfeil „>") befindet sich in einer, durch ausschließlich parallel zu der x- und y-Achse verlaufende Außenwände begrenzten, zusammenhängenden Welt. Die Pfeilspitze zeigt die momentane Blickrichtung an.



Die Ziffern bezeichnen die Ziegelhöhen.

Auf einem Feld der Welt können bis zu einer maximalen Höhe von *hmax* Ziegel abgelegt werden, die *Robi* in unbegrenzter Anzahl mit sich herumtragen kann. Robi kann einen Höhenunterschied von einem Ziegel überbrücken, d. h. er kann z. B. von einem leeren Feld ein mit einem Ziegel belegtes Feld betreten.

Fortsetzung nächste Seite!

Robi beherrscht folgende Funktionen:

- *geheVor* (*Robi* geht in seiner Welt in Blickrichtung einen Schritt vor, sofern dies möglich ist)
- *dreheLinks* (*Robi* dreht sich in seiner Welt auf aktueller Position um 90 Grad nach links)
- *nimmAuf* (*Robi* nimmt in seiner Welt einen Ziegel von dem Feld, das sich in Blickrichtung vor ihm befindet, sofern dies möglich ist)
- *legeAb* (*Robi* legt in seiner Welt einen Ziegel auf das Feld, das sich in Blickrichtung vor ihm befindet, sofern dies möglich ist)

und verfügt über folgende Sensoren:

- *vorneWand* (liefert true, wenn sich vor dem nächsten Feld in seiner Welt in Blickrichtung eine Außenwand befindet)
- *ziegelVorhanden* (liefert true, wenn sich auf dem nächsten Feld in seiner Welt in Blickrichtung mindestens ein Ziegel befindet)

a) Entwerfen Sie eine geeignete Datenstruktur zur Repräsentation von *Robis Welt*. Erläutern Sie diese kurz!

b) Geben Sie ein Klassendiagramm an, das den Zusammenhang zwischen *Roboter* und *Welt* mit allen problemrelevanten Attributen, Methoden und Kardinalitäten enthält! Erläutern Sie dieses kurz!

c) Geben Sie eine kommentierte Implementierung Ihres Modells an: Implementieren Sie die Klasse *Roboter* sowie alle dazu erforderlichen Bestandteile der Klasse *Welt*! Die Bildschirmdarstellung brauchen Sie nicht zu berücksichtigen. Verwenden Sie eine gängige (objektorientierte) Programmiersprache oder einen geeigneten Pseudocode.

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Herbst 2008

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	**Herbst 2008**	**46114**
Arbeitsplatz-Nr.: _____		

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: **8**

Thema Nr. 1

Aufgabe 1

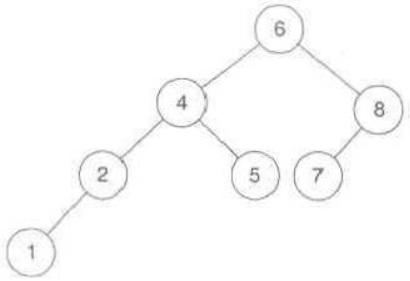
Gegeben sei folgende Zahlenfolge:

5, 2, 7, 6, 4, 1, 3

a) Fügen Sie obige Werte in der gegebenen Reihenfolge in einen anfänglich leeren binären Suchbaum ein und zeichnen Sie den resultierenden Baum!

b) Wie viele Vergleiche sind im besten und im schlechtesten Fall nötig, um festzustellen, ob eine gesuchte Zahl x in einem Suchbaum mit $n > 0$ Einträgen vorhanden ist?

c) Traversieren Sie folgenden Suchbaum in Breitensuche und schreiben Sie die Zahlen der jeweiligen besuchten Knoten der Reihe nach auf!



d) Welche drei Ausgabemuster für binäre Bäume kennen Sie, die sich aus einer Tiefensuche ableiten lassen? Welches produziert aus einem binären Suchbaum eine sortierte Liste?

Aufgabe 2:

Gegeben sei folgender Graph:

V: {a, b, c, d, e}
E: a → a,b
b → b,d,e
c → c,d
d → a,e

a) Stellen Sie den Graphen grafisch dar!
b) Berechnen Sie mit dem Algorithmus von Dijkstra schrittweise die Länge der kürzesten Pfade ab dem Knoten a! Nehmen Sie dazu an, dass alle Kantengewichte 1 sind. Erstellen Sie eine Tabelle gemäß folgendem Muster:

ausgewählt	a	b	c	d	e
-	0	∞	∞	∞	∞
A					

...

Ergebnis:					

Hinweis: Nur mit Angabe der jeweiligen Zwischenschritte gibt es Punkte. Es reicht also nicht, nur das Endergebnis hinzuschreiben.

Fortsetzung nächste Seite!

c) Welchen Aufwand hat der Algorithmus von Dijkstra bei Graphen mit |V| Knoten und |E| Kanten,

- wenn die Kantengewichte alle 1 sind?
  Mit welcher Datenstruktur und welchem Vorgehen lässt sich der Aufwand in diesem Fall reduzieren (mit kurzer Begründung)?

- wenn die Kantengewichte beliebig sind und als Datenstruktur eine Halde verwendet wird (mit kurzer Begründung)?

Aufgabe 3

Gegeben sei eine Reihung `a[i]` (`i = 0,1,...,n-1`), gefüllt mit unterschiedlichen, aufsteigend sortierten `int`-Zahlen. Für eine solche Reihung soll festgestellt werden, ob sie mindestens eine *Identität* hat. Identitäten sind die Stellen `i`, für die `a[i] == i` gilt.

a) Geben Sie alle Identitäten folgender zwei Reihungen an:

-7	**-3**	**-1**	**0**	**1**	**4**	**6**	**8**
-1	**1**	**2**	**5**	**7**	**8**	**10**	**12**

b) Sei `a[i]= 11`. Was ist die kleinste Zahl, die in `a[i+5]` stehen kann?

c) Seien `i < a[i]` und `a[i] = 11`. Geben Sie die Menge aller Indizes an, für die Sie dann sicher die Existenz von Identitäten ausschließen können.

d) Implementieren Sie eine rekursive Methode mit der Signatur

```
hatIdRek (int[] feld, int von, int bis),
```

die überprüft, ob die Reihung `feld` im Abschnitt [`von. . . bis`] eine Identität enthält! Der Zeitaufwand Ihrer Methode darf im ungünstigsten Fall O(log *n*) nicht überschreiten.

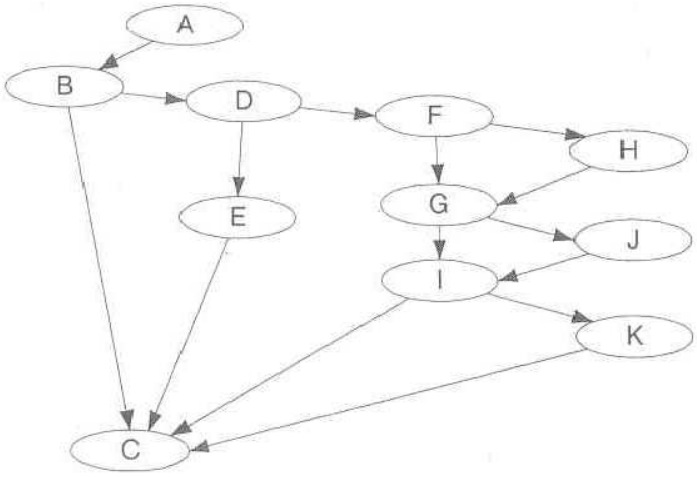
e) Begründen Sie kurz, wieso Ihr Verfahren terminiert!

Aufgabe 4

Gegeben ist die folgende Methode:

```
    static boolean checkISBN(char c, int pos) {
01      boolean result = false;
02      if (1 <= pos && pos <= 13) {
03          if (c == '-' && (pos == 2 || pos == 12)) {
04              result = true;
            } else {
05              if (c == '-' && (pos >= 5 && pos <= 10)) {
06                  result = true;
                }
07              if (c == 'X' && pos == 13) {
08                  result = true;
                }
09              if (c >= '0' && c <= '9') {
10                  result = true;
                }
            }
        }
11      return result;
    }
```

Gegeben ist ferner die dazugehörige grafische Darstellung des Kontrollflusses:



a) Ordnen Sie die Knoten des Graphen den Programmzeilen zu. Geben Sie dazu zu den nummerierten Programmzeilen die zugehörige Knotenbezeichnung an, also z. B. *01: A*

b) Eine Zeichenfolge ist eine gültige ISBN-Nummer genau dann, wenn folgende Bedingungen erfüllt sind:

`c` und `pos` sind Eingabeparameter

Fortsetzung nächste Seite!

- Gilt nicht `1 ≤ pos ≤ 13`, wird `false` zurückgegeben.
- Gilt `1 ≤ pos ≤ 13`, wird genau dann `true` zurückgegeben, wenn
  - `pos` den Wert `1, 3, 4` oder `11` hat und `c` eine Ziffer beinhaltet, oder
  - `pos` den Wert `2` oder `12` hat und `c` den Wert `'-'` beinhaltet, oder
  - `pos` den Wert `5, 6, 7, 8, 9` oder `10` hat und `c` eine Ziffer oder `'-'` beinhaltet oder
  - `pos` den Wert `13` hat und `c` eine Ziffer oder `'X'` beinhaltet.

Die oben gegebene Methode `checkISBN()` ist fehlerhaft bezüglich dieser Spezifikation. Weisen Sie die Anwesenheit des Fehlers durch einen geeigneten Testfall nach und geben Sie diesen an!

Hinweis: Es gibt einen solchen Testfall.

Thema Nr. 2

Aufgabe 1

Zeichenfolgen haben eine zentrale Bedeutung für Textverarbeitungssysteme. Wir betrachten im Folgenden Text-Zeichenfolgen, die aus Buchstaben, Ziffern und Sonderzeichen bestehen.

Eine grundlegende Operation auf Zeichenfolgen ist die Mustersuche: Gegeben sei eine Text-Zeichenfolge der Länge N und ein Muster der Länge M; gesucht ist ein Auftreten des Musters innerhalb der Text-Zeichenfolge. Ziel dieser Aufgabe ist die Implementierung eines Verfahrens zur Suche eines vorgegebenen Musters in einem eingelesenen Text.

Verwenden Sie für die Implementierung C, C++ oder Java.

(a) Das zu realisierende Programm soll schrittweise Zeilen von der Standardeingabe einlesen, bis keine Zeilen mehr eingegeben werden. Für jede eingelesene Zeile soll überprüft werden, ob die Zeile das vorgegebene Muster enthält. Wenn dies der Fall ist, soll die Zeile mit einer Treffermeldung ausgegeben werden. Für die Implementierung sollen zwei Funktionen verwendet und realisiert werden:

- Die Funktion

```
int getline (char s[], int lim)
```

  liest eine Zeile der maximalen Länge `lim-1` in das Zeichenfeld `s` ein und liefert die Länge der eingelesenen Zeile als Resultat zurück.

- Die Funktion

```
int strindex (char s[], char t[])
```

  sucht das Muster `t` in Zeile `s` und liefert die Position des ersten Auftretens von `t` in `s` als Resultat zurück. Wenn `t` in `s` nicht vorkommt, liefert die Funktion den Resultatwert -1 zurück.

Skizzieren Sie die Arbeitsweise des zu realisierenden Programms, indem Sie das Programm als Pseudocode angeben. Erläutern Sie die Struktur und Arbeitsweise des Programms.

Fortsetzung nächste Seite!

(b) Geben Sie eine Implementierung der Funktion `getline()` mit der oben beschriebenen Funktionalität an. Verwenden Sie zum Einlesen eine Systemfunktion, die einzelne Zeichen von der Standardeingabe liest und setzen Sie die eingelesenen Zeichen zur gesamten eingelesenen Zeile zusammen.

(c) Geben Sie eine Implementierung der Funktion `strindex()` mit der oben beschriebenen Funktionalität an. Verwenden Sie für die Realisierung zwei Indexzeiger, die parallel über `s` und `t` laufen und die jeweiligen Zeichen miteinander vergleichen, bis entweder das Ende von `t` erreicht ist (Erkennen eines Muster-Vorkommens) oder erkannt wird, dass kein Auftreten des Musters vorliegt. Testen Sie in `s` alle Möglichkeiten, an denen `t` auftreten kann, bis das erste Auftreten gefunden ist. Beschreiben Sie Ihr Vorgehen und die resultierende Implementierung.

(d) Geben Sie eine Implementierung des Hauptprogramms mit Verwendung von `getline()` und `strindex()` an. Das Hauptprogramm soll wiederholt Zeilen einlesen und das Auftreten eines vorgegebenen Musters in der Zeile suchen, bis keine Zeile mehr eingegeben wird. Wird das Auftreten eines Musters festgestellt, soll die Zeile ausgegeben werden.

(e) Welche Modifikation ist für `strindex()` nötig, um nicht nur das erste Auftreten des Musters, sondern alle Auftreten des Musters `t` in der Zeile `s` zu finden?

Realisieren Sie die notwendige Modifikation, indem Sie eine geänderte Implementierung von `strindex()` angeben. Rückgabewert soll nun die Anzahl der gefundenen Vorkommen des Musters t in s sein.

Beschreiben Sie die Funktionsweise der geänderten Funktion und begründen Sie, warum alle Auftreten gefunden werden.

Aufgabe 2

Eine Folge von Zahlen (integer-Werte) kann in Form einer einfach verketteten Liste, die für jede Zahl ein separates Datenobjekt verwendet, abgelegt werden. Dieses Datenobjekt enthält neben der Zahl einen Verweis auf das nachfolgende Datenobjekt, in dem die nächste Zahl der Folge abgelegt ist.
Realisieren Sie die folgenden Aufgaben in C, C++ oder Java ohne Verwendung vordefinierter Klassen oder Bibliotheken zur Realisierung von Listen.

(a) Deklarieren Sie einen Datentyp ELEM, der als Datenstruktur für eine einfach verkettete Liste zur Ablage von Zahlen verwendet werden kann.

(b) Geben Sie eine Funktion `make_element(int value)` an, die ein Listenelement für eine einfach verkettete Liste erzeugt, den als Parameter mitgelieferten Wert ablegt und einen Verweis bzw. Referenz auf das Listenelement als Rückgabewert zurückliefert.

(c) Geben Sie eine Funktion `int length()` an, die die Anzahl der in einer einfach verketteten Liste abgespeicherten Zahlen ermittelt und als Rückgabewert zurückliefert. Die Liste, deren Länge bestimmt werden soll, soll als Parameter der Funktion `length()` übergeben werden.

(d) Geben Sie eine Funktion `reverse()` an, die aus einer gegebenen einfach verketteten Liste eine neue einfach verkettete Liste erzeugt, die die gleichen Elemente wie die gegebene Liste in umgekehrter Reihenfolge enthält, d. h., wenn die gegebene Liste

Fortsetzung nächste Seite!

die Zahlen $a_1, \ldots, a_n$ in dieser Reihenfolge enthält, enthält die neue Liste die Zahlen $a_n, \ldots, a_1$ in dieser Reihenfolge. Die gegebene Liste soll als Parameter der Funktion `reverse()` zur Verfügung gestellt werden. Ein Verweis auf die neue Liste soll als Resultat der Funktion `reverse()` zurückgeliefert werden. Die neue Liste soll als neue Datenstruktur so aufgebaut werden, dass die gegebene Liste auch nach Verlassen der Funktion `reverse()` weiterhin verfügbar ist. Erläutern Sie die Funktionsweise Ihrer Implementierung.

(e) Geben Sie eine Funktion `split1()` an, die aus einer gegebenen einfach verketteten Liste mit Zahlen $a_1, a_2, \ldots, a_{n-1}, a_n$ zwei gleich lange einfach verkettete Listen erzeugt, so dass jedes Element der gegebenen Liste in genau einer der neuen Listen enthalten ist. Die erste neue Liste soll dabei die Elemente $a_1, \ldots, a_{\lfloor n/2 \rfloor}$, die zweite neue Liste die Elemente $a_{\lfloor n/2 \rfloor + 1}, \ldots, a_n$ enthalten, d. h. die gegebene Liste soll in der Mitte geteilt werden. Verwenden Sie die Funktion `length()` aus Aufgabenteil (c) zur Bestimmung der Länge der gegebenen Liste. Ein Verweis auf die gegebene Liste soll als Parameter der Funktion `split1()` zur Verfügung gestellt werden. Verweise auf die neuen Listen sollen über Parameter der Funktion `split1()` zurückgeliefert werden. Die neuen Listen sollen unter Verwendung der für die gegebene Liste verwendeten Datenelemente vom Datentyp ELEM aufgebaut werden, so dass keine Neu-Allokierung von Datenelementen erforderlich ist. Die gegebene Liste steht damit nach Beendigung der Funktion `split1()` nicht mehr zur Verfügung.

(f) Geben Sie eine Funktion `split2()` an, die ähnlich wie `split1()` eine gegebene Liste von Zahlen in zwei gleichgroße Listen aufspaltet. Im Gegensatz zu `split1()` soll durch Ausführung von `split2()` die erste Liste die Zahlen $a_1, a_3, a_5, \ldots$, d.h. die Zahlen mit ungeradem Index, enthalten; die zweite Liste soll die Zahlen $a_2, a_4, a_6, \ldots$, d.h. die Zahlen mit geradem Index, enthalten. Die Parameter sollen wie bei `split1()` verwendet werden. Wie bei `split1()` sollen für `split2()` die Datenelemente (vom Typ ELEM) der gegebenen Liste für den Aufbau der neuen Liste wiederverwendet werden, so dass keine Neu-Allokierung von Datenelementen erforderlich ist.

Aufgabe 3

Quicksort ist ein Sortierungsverfahren, das nach dem Divide-and-Conquer-Prinzip (Teile und Herrsche) arbeitet. Wir betrachten im Folgenden die Anwendung dieses Verfahrens zum Sortieren von Integerzahlen. Die Sortierung soll in aufsteigender Reihenfolge der Werte erfolgen. Wir nehmen dabei an, dass die zu sortierenden Zahlen in einem Feld fester Länge abgelegt sind.

(a) Beschreiben Sie die Arbeitsweise des Divide-and-Conquer-Prinzips im allgemeinen Fall. Geben Sie dabei die Bedeutung der Schritte divide, conquer und combine an.

(b) Beschreiben Sie die Arbeitsweise des Algorithmus Quicksort. Geben Sie dabei an, worin die Schritte divide, conquer und combine im konkreten Fall bestehen.

(c) Geben Sie in C, C++ oder Java eine Implementierung des Algorithmus Quicksort an. Formulieren Sie die Implementierung als rekursive Funktion `quicksort()` und verwenden Sie das jeweils erste Element des (Teil-)Feldes für die Aufteilung.

Verwenden Sie für Ihre Implementierung von `quicksort()` drei Parameter: (1) das Feld, in dem die zu sortierenden Zahlen abgelegt sind; (2) den Index des am

Fortsetzung nächste Seite!

weitesten links gelegenen Elementes des zu sortierenden Teilfeldes; (3) den Index des am weitesten rechts gelegenen Elementes des zu sortierenden Teilfeldes.

Erläutern Sie die Arbeitsweise Ihrer Implementierung. Kennzeichnen Sie die Schritte divide, conquer und combine des zugrundeliegenden Divide-and-Conquer-Prinzips.

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Frühjahr 2009

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Frühjahr**	**46114**
Kennwort: _____	**2009**	
Arbeitsplatz-Nr.: _____		

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:

$$[\texttt{Hemd}, \texttt{Krawatte}]$$

$$[\texttt{Sakko}, \texttt{Mantel}]$$

$$[\texttt{Unterhemd}, \texttt{Hemd}]$$

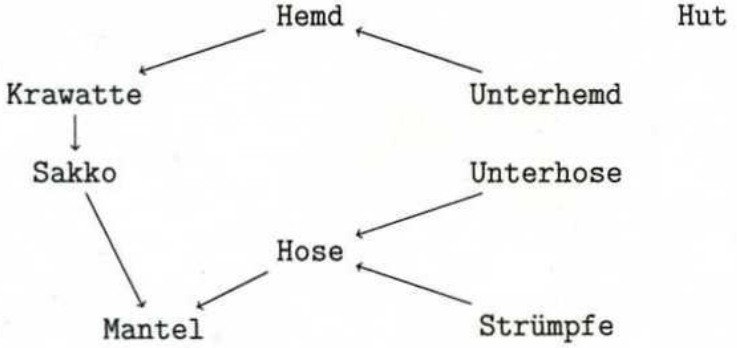
$$[\texttt{Unterhose}, \texttt{Hose}]$$

$$[\texttt{Strümpfe}, \texttt{Hose}]$$

$$[\texttt{Krawatte}, \texttt{Sakko}]$$

$$[\texttt{Hose}, \texttt{Mantel}]$$

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 $\circ$, ein bereits reservierter Sitz mittels $\bullet$ markiert. Für $R = 3, C = 5$ könnte sich bspw. also der folgende Sitzplan ergeben:

$$\begin{array}{ccccc} \circ & \circ & \circ & \bullet & \bullet \\ \circ & \bullet & \circ & \bullet & \bullet \\ \circ & \bullet & \bullet & \circ & \circ \end{array}$$

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 ($\circ$) enthält.

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

$$\begin{array}{ccccc} \circ & \circ & \circ & \bullet & \bullet \\ \circ & \bullet & \circ & \bullet & \bullet \\ \circ & \bullet & \bullet & \circ & \circ \end{array}$$

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.

46114

Algorithmen / Datenstrukturen / Programmiermethoden
(nicht vertieft)

Herbst 2009

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	**Herbst 2009**	**46114**
Kennwort: _____		
Arbeitsplatz-Nr.: _____		

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: **8**

Bitte wenden!

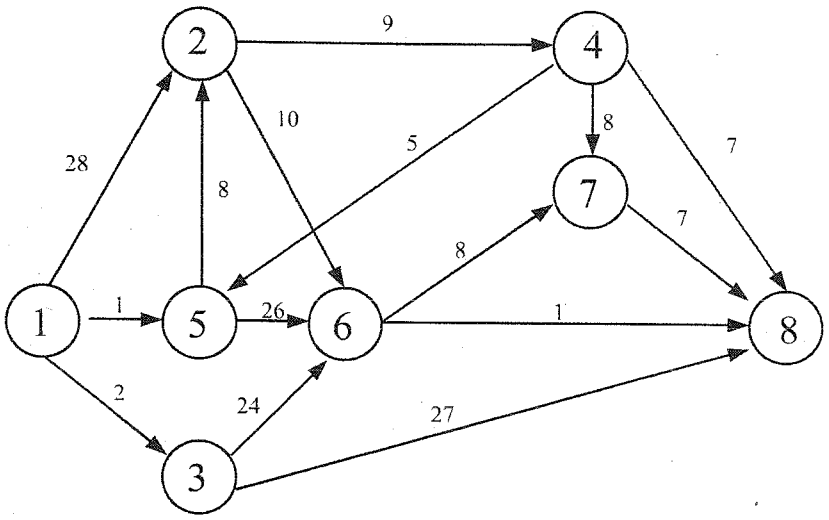
Thema Nr. 1

Teilaufgabe 1 (Listen)

Gegeben seien zwei einfach verkettete Listen $X = (x_1, x_2, ..., x_m)$ und $Y = (y_1, y_2, ..., y_n)$. Schreiben Sie eine Prozedur zum Mischen dieser Listen in der Form $Z = (x_1, y_1, x_2, y_2, ...)$, wobei eine neue Liste erzeugt werden soll. Achten Sie auf die Restglieder, falls $m \neq n$.

Teilaufgabe 2 (Kürzeste Wege)

Berechnen Sie für das folgende Netzwerk mit dem Algorithmus von Dijkstra ausgehend von der Ecke 1 die Längen der kürzesten Wege zu allen anderen Knoten. Stellen Sie die Kostenmatrix auf und geben Sie für jeden Schritt die Entfernungsangaben an.



Fortsetzung nächste Seite!

Teilaufgabe 3 (AVL-Bäume)

Gegeben seien die Schlüsselwerte 3, 5, 11, 8, 4, 1, 12, 7, 2, 6, 10, 9.

(a) Generieren Sie den binären Suchbaum, der durch sukzessives Einsetzen dieser Werte entsteht. Gehen Sie dabei von einem leeren Baum aus.

(b) Geben Sie zu den jeweiligen Knoten die zugehörige Balance ein. Ist der Baum ein AVL-Baum?

(c) Generieren Sie wie in a) erneut einen Suchbaum durch sukzessives Einsetzen der gegebenen Werte und stellen Sie nach jedem Schritt die AVL-Eigenschaft wieder her, so dass beim nächsten Einsetzen ein AVL-Baum zugrunde liegt und auch der Baum am Ende ein AVL-Baum ist.

(d) Stellen Sie jetzt für obige Schlüsselwertmenge einen B-Baum der Ordnung 3 auf. Gehen Sie schrittweise vor, so dass die Vorgehensweise nachvollziehbar ist.

Teilaufgabe 4 (Systementwurf und Programmiermethodik)

Das Newton-Verfahren ist ein nummerisches Verfahren zur näherungsweisen Bestimmung von Nullstellen einer reellen Funktion $f(x)$. Zum Startwert x_o wird der Punkt $(x_o, f(x_o))$ berechnet und die Tangente dieses Punktes bestimmt. Der Schnittpunkt der Tangente mit der x-Achse $(x_1,0)$ liefert einen neuen Punkt $(x_1, f(x_1))$. Zu diesem wird wieder die Tangente bestimmt und erneut der Schnittpunkt mit der x-Achse berechnet, etc. Allgemein lautet die Iterationsvorschrift wie folgt:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad .$$

Man bricht das Verfahren ab, wenn die Nullstelle hinreichend genau bestimmt ist oder wenn eine obere Grenze für die Anzahl der Iterationsschritte erreicht wurde. Ferner wird abgebrochen, falls $f'(x_n)$ gilt. Im letzten Fall ist die Nullstelle mit dem Verfahren in dieser Form nicht bestimmbar.

Es soll nun ein Programm zur Berechnung einer Nullstelle einer beliebigen Funktion mit Hilfe des Newton-Verfahrens erstellt werden. Verwenden Sie geeignete Abbruchkriterien unter Berücksichtigung von Rundungsfehlern und Schnittstellen (Unterprogrammaufrufe) zur Berechnung von $f(x)$ und $f'(x)$.

(a) Erstellen Sie ein Programmablaufdiagramm.

(b) Erstellen Sie ein Struktogramm und schreiben Sie das Programm in Pseudocode oder einer Programmiersprache Ihrer Wahl.

Fortsetzung nächste Seite!

Thema Nr. 2

Aufgabe 1

Aufgabe 1.1

Entwerfen Sie ein objektorientiertes Modell für gerichtete, gewichtete Graphen und beschreiben Sie Ihren Entwurf durch ein UML-Klassendiagramm. Verwenden Sie dabei die Klassen `Node`, `Edge` und `Graph` als Repräsentationen für Knoten, Kanten und den Graphen. Knoten sollen durch einen Namen identifiziert werden und Kantengewichte seien ganzzahlig (vom Typ `Integer`).
Im Diagramm sollen für alle Klassen die benötigten Attribute und Assoziationen sowie die dazugehörigen Akzessor- (Getter) und Mutator-Operationen (Setter) eingetragen werden. Zusätzlich soll die Klasse `Graph` Operationen für die folgenden Aufgaben enthalten:

- Hinzufügen von Knoten und Kanten
- Zugriff auf Knoten anhand ihres Namens
- Zugriff auf Kanten anhand ihres Start- und Endknotens
- Zugriff auf die Menge aller Knoten des Graphen
- Zugriff auf die Menge aller Nachfolger eines Knotens
- Zugriff auf die Menge aller ausgehenden Kanten eines Knotens

Aufgabe 1.2

Implementieren Sie die Klasse `Graph` aus dem in Aufgabe 1.1 erstellten Modell in einer objektorientierten Programmiersprache Ihrer Wahl. Die Klassen `Node` und `Edge` müssen nicht implementiert werden. Stattdessen können die in Aufgabe 1.1 definierten Operationen dieser Klassen als vorhanden angenommen und in der Implementierung von `Graph` verwendet werden. Ebenso können benötigte Konstruktoren mit geeigneten Parametern als gegeben betrachtet werden. Die Fehlerbehandlung kann bei der Implementierung vernachlässigt werden.

Für die Implementierung seien die folgenden Datenstruktur-Klassen gegeben:

`Hashtable`
Implementiert eine Hashtabelle, die Schlüssel und Werte mit beliebigen Typen speichert.

Operationen:

`put(key:Object, value:Object)`	Fügt das Objekt in `value` in die Hashtabelle ein und assoziiert es mit dem angegebenen Schlüssel. Sowohl `key` als auch `value` können von beliebigem Typ sein.
`get(key:Object):Object`	Gibt das Element zurück, das mit dem angegebenen Schlüssel assoziiert ist.
`values():LinkedList`	Gibt ein Objekt vom Typ `LinkedList` (siehe unten) zurück, das alle in der Hashtabelle gespeicherten Werte enthält.

Fortsetzung nächste Seite!

`LinkedList`
Implementiert eine verkettete Liste mit Elementen von beliebigem Typ.

Operationen:

`addLast(element:Object)`	Fügt das angegebene Element am Ende der Liste ein.
`addFirst(element:Object)`	Fügt das angegebene Element am Anfang der Liste ein.
`contains(element:Object): Boolean`	Gibt `true` zurück, wenn das angegebene Element in der Liste enthalten ist, sonst `false`.
`iterator():Iterator`	Gibt ein Objekt vom Typ `Iterator` (siehe unten) zurück, der eine Iteration alle Elemente der Liste ermöglicht. Nach Aufruf der Operation `iterator()` zeigt der aktuelle Wert des Iterators auf das erste Element der Liste.

`Iterator`
Ermöglicht eine Iteration über die Elemente einer Liste.

Operationen:

`hasNext():Boolean`	Gibt `true` zurück, wenn in der aktuellen Iteration noch weitere Elemente folgen, sonst `false`.
`next():Object`	Gibt das nächste Element aus der Iteration zurück.

Fortsetzung nächste Seite!

Aufgabe 1.3

Dijkstras Algorithmus zur Bestimmung der kürzesten Pfade ist ein bekannter Graphenalgorithmus, der von Edsger W. Dijkstra 1959 veröffentlicht wurde. Im Folgenden soll, aufbauend auf den Ergebnissen der Aufgaben 1.1 und 1.2, die Klasse `Dijkstra` implementiert werden, die diesen Algorithmus realisiert. Sie können dabei, wie in Aufgabe 1.2 erläutert, auf die Klassen `Hashtable`, `LinkedList` und `Iterator` zurückgreifen. Zusätzlich sollen folgende Klassen verwendet werden:

`NodeDistancePriorityQueue`

Implementiert eine Prioritätswarteschlange, die Elemente des Typs `Node` (aus Aufgabe 1.1) aufnehmen kann und diese gemäß einem Distanzwert sortiert. Der Distanzwert für einen Knoten wird dabei von einem Objekt abgerufen, das die Schnittstelle `NodeDistanceProvider` (siehe unten) implementiert und das bei der Konstruktion übergeben wird.

Operationen:

`add(node:Node)`	Fügt einen Knoten zur Prioritätswarteschlange hinzu.
`addAll(nodes:LinkedList)`	Fügt alle in der angegebenen Liste enthaltenen Knoten zur Prioritätswarteschlange hinzu.
`extractMinimum():Node`	Gibt den Knoten mit der geringsten Distanz zurück und löscht ihn aus der Liste.
`isEmpty():Boolean`	Gibt `true` zurück, falls die Prioritätswarteschlange kein Element enthält, sonst `false`.

`NodeDistanceProvider`

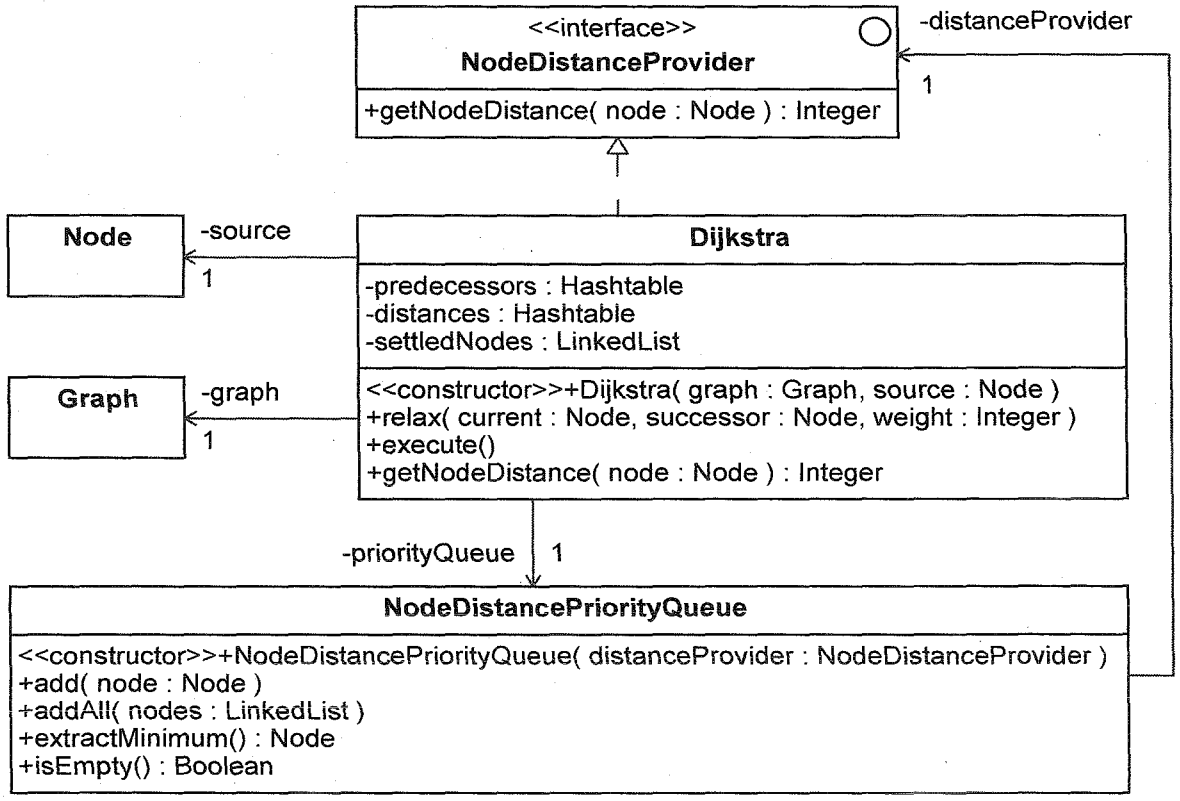
Schnittstelle, die eine Akzessor-Operation zum Zugriff auf die Distanz eines Knotens von einer implizit gegebenen Quelle liefert.

Operationen:

`getNodeDistance(node:Node): Integer`	Gibt die Distanz des angegebenen Knotens zurück.

Fortsetzung nächste Seite!

Das folgende Klassendiagramm zeigt die zu implementierende Klasse `Dijkstra` zusammen mit ihren Attributen, Assoziationen und Operationen.



a) Geben Sie zunächst eine informelle Beschreibung des Algorithmus von Dijkstra an (textuell oder in Pseudo-Code).

b) Geben Sie ein Quelltext-Gerüst für die Klasse `Dijkstra` an, das der Spezifikation im oben gegebenen Klassendiagramm entspricht. Ihr Code-Fragment soll dabei die Deklaration aller Instanzvariablen enthalten sowie leere Rümpfe für alle Operationen und den Konstruktor der Klasse. Die Implementierung der Operationen erfolgt in den folgenden Teilaufgaben.

c) Geben Sie eine Implementierung für die Operation `getNodeDistance` an sowie für den Konstruktor, der alle Instanzvariablen gemäß den Anforderungen des Dijkstra-Algorithmus initialisiert.

d) Geben Sie eine Implementierung für die Operation `relax` an, die in einem Schritt des Dijkstra-Algorithmus die Aktualisierung des aktuellen Distanz-Schätzwerts und des Vorgängers eines Knotens durchführt.

e) Geben Sie eine Implementierung für die Operation `execute` an, die den eigentlichen Ablauf des Algorithmus realisiert.

Fortsetzung nächste Seite!

Aufgabe 1.4

Welche Zeitkomplexität hat Ihre Lösung? Begründen Sie Ihre Antwort.

Sie können annehmen, dass die Prioritätswarteschlange die Knoten in einer verketteten Liste speichert und dass alle Zugriffe auf Hashtabellen in konstanter Zeit erfolgen.