Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Herbst 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: **8**

Thema Nr. 1

Aufgabe 1

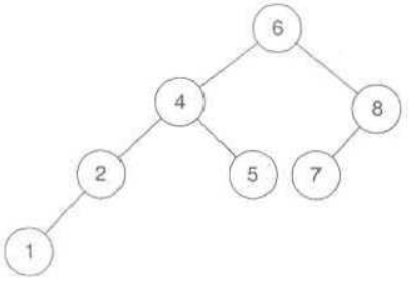
Gegeben sei folgende Zahlenfolge:

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.

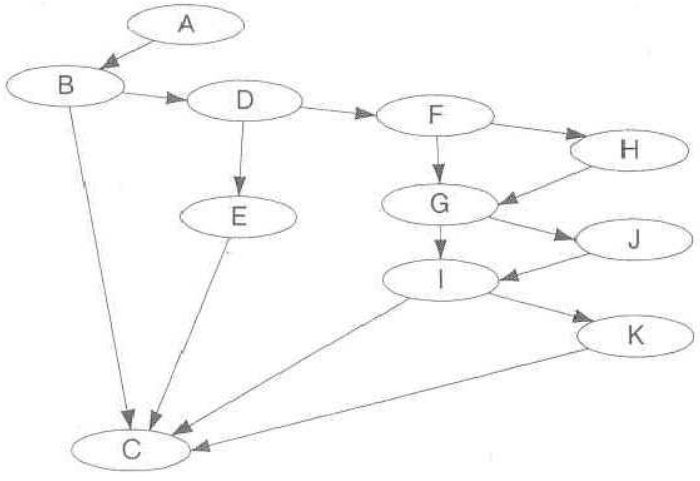
e) Begründen Sie kurz, wieso Ihr Verfahren terminiert!

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.