Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Herbst 2015

46115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 9

Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

Die Sprache L über dem Alphabet $\Sigma = \{a, b\}$ enthält alle Wörter, die das Wort baab oder das Wort abba oder beide enthalten. Z.B. ist $baababb \in L$, aber $babababa \notin L$.

1. Geben Sie einen regulären Ausdruck für L an.
2. Konstruieren Sie einen nichtdeterministischen endlichen Automaten mit höchstens 8 Zuständen für L.
3. Konstruieren Sie einen deterministischen Automaten für L mit der Potenzmengenkonstruktion. Sie dürfen sich wie üblich auf die erreichbaren Zustände beschränken.
4. Konstruieren Sie den Minimalautomaten für L. Sie können entweder das übliche Verfahren anwenden und Ihre Schritte geeignet dokumentieren, oder aber den Minimalautomaten direkt angeben und geeignet begründen, dass es sich um den richtigen Automaten handelt.

Aufgabe 2:

Beim *Postschen Korrespondenzproblem* (PCP) ist bekanntlich eine Liste von Paaren $(u_1, v_1), \ldots, (u_n, v_n)$ mit $u_i, v_i \in \Sigma^*$ für ein Alphabet Σ gegeben. Zum Beispiel $(u_1, v_1) = (b, bab)$ und $(u_2, v_2) = (ba, aa)$ und $(u_3, v_3) = (abb, bb)$, wobei hier also $n = 3$ und $\Sigma = \{a, b\}$ sind. Gefragt ist, ob es eine Indexfolge $i_1, i_2, \ldots, i_N$ mit $i_k \leq n$ gibt, so dass gilt $u_{i_1} u_{i_2} \ldots u_{i_N} = v_{i_1} v_{i_2} \ldots v_{i_N}$. Im Beispiel wäre $1, 3, 2, 3$ solch eine Lösung, also genauer $N = 4$ und $i_1 = 1, i_2 = 3, i_3 = 2, i_4 = 3$, denn es ist hier $u_1 u_3 u_2 u_3 = b\ abb\ ba\ abb = bab\ bb\ aa\ bb = v_1 v_3 v_2 v_3$.

Es ist bekannt, dass das PCP ein unentscheidbares Problem ist.

1. Erläutern Sie, was es genau bedeutet, dass das PCP unentscheidbar ist.
2. Begründen Sie, dass das PCP rekursiv aufzählbar ist.
3. Seien jetzt zwei Alphabete Δ und Σ sowie Funktionen $f, g : \Delta \to \Sigma^*$ durch ihre Wertetabellen gegeben, z.B. $\Delta = \{c, d, e\}$, $\Sigma = \{a, b\}$ und $f(c) = b, f(d) = ba, f(e) = abb$. Man setzt diese Funktionen in offensichtlicher Weise ("homomorph") auf Wörter über Δ fort, z.B. $f(ccde) = bbbaabb$.

   Zeigen Sie durch Reduktion vom PCP, dass es unentscheidbar ist, festzustellen, ob ein Wort $w \in \Delta^+$ mit $f(w) = g(w)$ existiert.

Fortsetzung nächste Seite!

Aufgabe 3:

Eine Folge von Zahlen $a_1, \ldots, a_n$ heiße *unimodal*, wenn sie bis zu einem bestimmten Punkt echt ansteigt und dann echt fällt. Zum Beispiel ist die Folge $1, 3, 5, 6, 5, 2, 1$ unimodal, die Folgen $1, 3, 5, 4, 7, 2, 1$ und $1, 2, 3, 3, 4, 3, 2, 1$ aber nicht.

1. Entwerfen Sie einen Algorithmus, der zu (als Array) gegebener *unimodaler* Folge $a_1, \ldots, a_n$ in Zeit $O(\log n)$ das Maximum $\max a_i$ berechnet. Ist die Folge nicht unimodal, so kann Ihr Algorithmus ein beliebiges Ergebnis liefern. Größenvergleiche, arithmetische Operationen und Arrayzugriffe können wie üblich in konstanter Zeit ($O(1)$) getätigt werden. Hinweise: binäre Suche, divide-and-conquer.
2. Begründen Sie, dass Ihr Algorithmus tatsächlich in Zeit $O(\log n)$ läuft.
3. Schreiben Sie Ihren Algorithmus in Pseudocode oder in einer Programmiersprache Ihrer Wahl, z.B. Java, auf. Sie dürfen voraussetzen, dass die Eingabe in Form eines Arrays der Größe n vorliegt.
4. Beschreiben Sie in Worten ein Verfahren, welches in Zeit $O(n)$ feststellt, ob eine vorgelegte Folge unimodal ist oder nicht.
5. Begründen Sie, dass es kein solches Verfahren (Test auf Unimodalität) geben kann, welches in Zeit $O(\log n)$ läuft.

Thema Nr. 2
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

"Streuspeicherung"

Fügen Sie die folgenden Werte in der gegebenen Reihenfolge in eine Streutabelle der Größe 8 (mit den Indizes 0 7) und der Streufunktion h(x) = x mod 8 ein. Verwenden Sie die jeweils angegebene Hash-Variante bzw. Kollisionsauflösung:

15, 3, 9, 23, 1, 8, 17, 4

a) Offenes Hashing: Zur Kollisionsauflösung wird Verkettung verwendet.
Beispiel: Für die beiden Werte 8 und 16 würde die Lösung wie folgt aussehen:

Bucket	*Inhalt*
0	8 - 16
1	
2	

b) Geschlossenes Hashing: Zur Kollisionsauflösung wird lineares Sondieren (nur hochzählend) mit Schrittweite +5 verwendet. Treten beim Einfügen Kollisionen auf, dann notieren Sie die Anzahl der Versuche zum Ablegen des Wertes im Subskript (z.B. das Einfügen des Wertes 8 gelingt im 5. Versuch: „$8_{(5)}$").
Beispiel: Für die beiden Werte 8 und 16 würde die Lösung wie folgt aussehen:

Bucket	*Inhalt*
0	8
1	
2	
3	
4	
5	$16_{(2)}$

c) Welches Problem tritt auf, wenn zur Kollisionsauflösung lineares Sondieren mit Schrittweite 4 verwendet wird? Warum ist 5 eine bessere Wahl?
d) Geben Sie für Ihre Lösungen zu a) und b) jeweils den Lastfaktor an.
e) Gegeben sei eine Streutabelle, in der die gleichen Schlüssel mehrfach enthalten sein können. Welches der beiden Verfahren aus den Teilaufgaben a) und b) ist in der Worst-Case-Laufzeit performanter beim Einfügen eines neuen Eintrags? Welches Verfahren ist performanter beim Abrufen des Wertes zu einem Schlüssel? Begründen Sie kurz Ihre Antworten.

Fortsetzung nächste Seite!

Aufgabe 2:

"Abstrakte Datentypen (adt)"

Der Abstrakte Datentyp **(adt)** **`AA`** (*AssociativeArray*) entspricht einer Reihung vom Typ **`T`** mit beliebiger Länge und hat folgende Eigenschaften: Der Konstruktor **`create`** instanziiert eine neue leere Reihung. Mit den Methoden **`get`** und **`set`** kann man die Elemente nach ganzzahligen (nicht notwendigerweise positiven) Indizes auslesen bzw. mit einem Wert ≠ **`null`** aktualisieren (Aufrufe von **`set`** mit **`null`** sollen vom **adt** „ignoriert" werden). **`delete`** löscht (falls vorhanden) die Belegung eines Indexes. Lesende Zugriffe auf unbelegte/gelöschte Indizes liefern **`null`**. Die Anzahl der tatsächlich (also mit Wert ≠ **`null`**) belegten Indizes kann mit **`size`** abgefragt werden.

a) Ergänzen Sie die fehlenden Axiome, um das vorangehend beschriebene Verhalten formal darzustellen:

adt *AA*
sorts *AA, T, int*
ops

create:		→	*AA*
get:	*AA* × *int*	→	*T*
set:	*AA* × *int* × *T*	→	*AA*
size:	*AA*	→	*int*
delete:	*AA* × *int*	→	*AA*

axs

size(create) = 0

get(create, i) = null

delete(create, i) = create

end *AA*

b) Ergänzen Sie die fehlenden Methoden so, dass die folgende Klasse genau das Verhalten des obigen **adt** wiedergibt. Sie dürfen dabei *keine* anderen Datenstrukturen verwenden, also insbesondere auch *keine* **`HashMap`** aus der Java-API.

```
public class AA<T> {
	private int key;
	private T value;
	private AA<T> tail;

	private AA(int key, T value, AA<T> tail) {
		this.key = key;
		this.value = value;
		this.tail = tail;
	}

	public static <T> AA<T> create() {
		return new AA<T>(0, null, null);
	}

	// TODO: Code hier ergaenzen
```

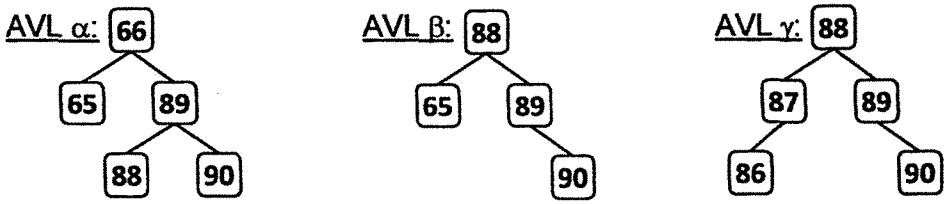
Fortsetzung nächste Seite!

Aufgabe 3:

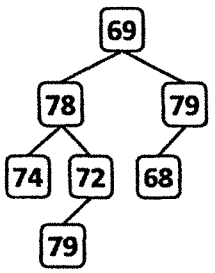
"Bäume"

a) Fügen Sie die Zahlen **83, 85, 67, 72, 69** in der gegebenen Reihenfolge
   i) in einen *linksvollständigen Binärbaum* bzw.
   ii) in einen *binären Suchbaum*
   ein. Stellen Sie jeweils nur das Endergebnis dar.

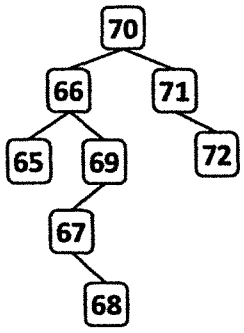
b) Fügen Sie **77** in jeden der folgenden *AVL-Bäume* ein und führen Sie anschließend *bei Bedarf* die erforderliche(n) Rotation(en) aus. Zeichnen Sie den Baum zunächst unmittelbar nach dem Einfügen und anschließend im Endzustand nach den Rotationen.



c) Traversieren Sie den folgenden Baum in der jeweils angegebenen Art (Abstieg zuerst im linken Teilbaum) und geben Sie dabei die Knoten in der entsprechenden Reihenfolge aus:
   i) in-order
   ii) post-order



d) Löschen Sie den Knoten **70** aus dem folgenden *binären Suchbaum* und ersetzen Sie ihn dabei durch einen geeigneten Wert aus dem linken Teilbaum. Zeichnen Sie nur den Endzustand:



Fortsetzung nächste Seite!

Aufgabe 4:

"Formale Verifikation - Induktionsbeweis"

Gegeben sei die folgende Methode `function`:

```
double function(int n) {
      if (n == 1)
            return 0.5 * n;
      else
            return 1.0 / (n * (n + 1)) + function(n - 1);
}
```

Beweisen Sie folgenden Zusammenhang mittels vollständiger Induktion:

$$\forall n \geq 1\text{: } \texttt{function(n)} = f(n) \text{ mit } f(n) \coloneqq 1 - \frac{1}{n+1}$$

Hinweis: Eventuelle Rechenungenauigkeiten, wie z. B. in Java, bei der Behandlung von Fließkommazahlen (z. B. double) sollen beim Beweis nicht berücksichtigt werden – Sie dürfen also annehmen, Fließkommazahlen würden mathematische Genauigkeit aufweisen.

Fortsetzung nächste Seite!

Aufgabe 5:

Sei $\Sigma = \{a, b\}$, sei $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.

a) Sei

$$L_1 = \{w \in \Sigma^* \mid w \text{ enthält mehr } a\text{'s als } b\text{'s}\} \ .$$

Beispiele: $aba \in L_1$, $bbbaabaabaa \in L_1$, $a \in L_1$.

Zeigen Sie, dass L_1 kontextfrei ist, indem Sie die Arbeitsweise eines Kellerautomaten beschreiben, der L_1 akzeptiert.

b) Formulieren Sie das Pumping-Lemma für reguläre Sprachen:

„Sei L eine reguläre Sprache über dem Alphabet Σ. Dann gibt es ...“

c) Zeigen Sie mit Hilfe des Pumping-Lemmas für reguläre Sprachen, dass die Sprache L_1 nicht regulär ist.

Aufgabe 6:

a) Sei $m \in \mathbb{N}_0 = \{0, 1, 2, \ldots\}$. Definieren Sie formal die Menge H_m der Gödelnummern der Turing-Maschinen, die gestartet mit m halten.

b) Gegeben sei das folgende Problem E:

- Entscheide, ob es für die deterministische Turing-Maschine M mit der Gödelnummer $\langle M \rangle$ mindestens eine Eingabe $w \in \mathbb{N}_0$ gibt, so dass w eine *Quadratzahl* ist und die Maschine M gestartet mit w hält.

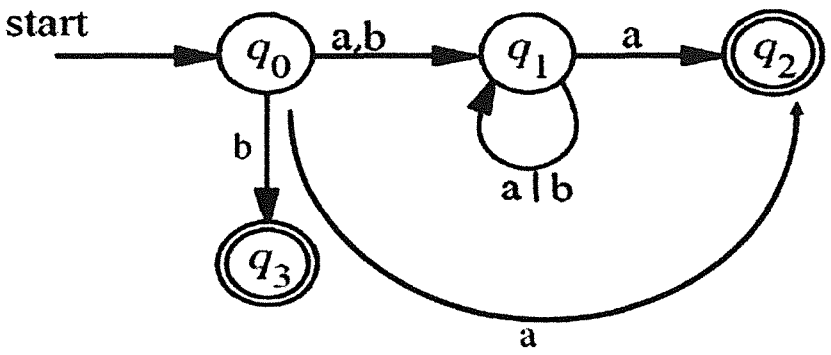
Zeigen Sie, dass E nicht entscheidbar ist. Benutzen Sie, dass H_m aus (a) für jedes $m \in \mathbb{N}_0$ nicht entscheidbar ist.

c) Zeigen Sie, dass das Problem E aus (b) partiell-entscheidbar (= rekursiv aufzählbar) ist.

Fortsetzung nächste Seite!

Aufgabe 7:

a) Gegeben sei der folgende nichtdeterministische endliche Automat N:



ε bezeichnet das leere Wort. Konstruieren Sie zu N mit der Potenzmengen-Konstruktion einen äquivalenten deterministischen endlichen Automaten A. Zeichnen Sie nur die vom Startzustand erreichbaren Zustände ein, diese aber *alle*. Die Zustandsnamen von A müssen erkennen lassen, wie sie zustande gekommen sind. Führen Sie *keine* „Vereinfachungen" durch!

Hinweise:

- $\{q_0\}$ ist **nicht** der Startzustand des deterministischen endlichen Automaten.
- In einem deterministischen endlichen Automaten darf es keine ε-Übergänge geben, und es muss an jedem Zustand für jedes Zeichen einen Übergang geben.

b) Geben Sie einen regulären Ausdruck $\alpha(N)$ für die Sprache, die der nichtdeterministische endliche Automat N aus (a) akzeptiert, an.

c) Beweisen oder widerlegen Sie die folgende Behauptung.

Beh.: Es gibt reguläre Sprachen L, die mindestens eine echte Teilmenge U enthalten, so dass U nicht regulär ist.