Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Herbst 2011

66115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoret. Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): **2**

Anzahl der Druckseiten dieser Vorlage: **6**

Bitte wenden!

Thema Nr. 1

Aufgabe 1: Automatentheorie

1.1 Reguläre Sprachen und Endliche Automaten

Gegeben ist die Grammatik $G = (\{S, A, B, C\}, \{a,b\}, \Phi, S)$ mit

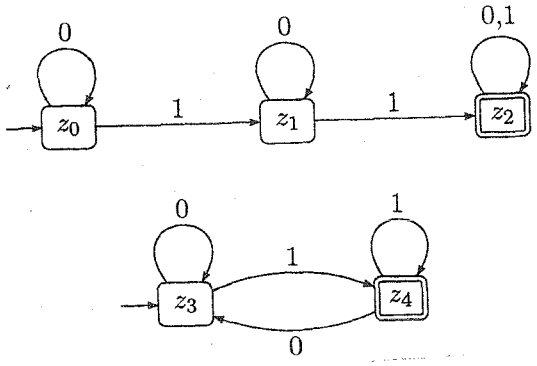
$\Phi = \{ \; S \rightarrow aA \mid bB \mid a \mid b \mid \epsilon,$
$A \rightarrow aC \mid bB \mid b,$
$B \rightarrow aA \mid bC \mid a,$
$C \rightarrow aC \mid bC\}.$

a) Konstruieren Sie einen DFA M mit $L(M) = L(G)$.

b) Welche Sprache erzeugt die Grammatik G?

1.2 Minimierung

Gegeben seien die beiden folgenden deterministischen endlichen Automaten M1 und M2:



a) Zeigen Sie, dass M_1 und M_2 minimal sind.

b) Bilden Sie einen DFA $\hat{M}$, der die Sprache $L(M_1) \cap L(M_2)$ akzeptiert.

c) Welche Sprache akzeptiert der Automat $\hat{M}$? Geben Sie einen regulären Ausdruck γ an, der diese Sprache beschreibt, so dass γ möglichst wenig Zeichen aus {0; 1} enthält.

Aufgabe 2: Formale Sprachen

2.1 Bestimmung der Grammatiktypen

Bestimmen Sie für die folgenden drei Sprachen jeweils den restriktivsten Typ einer Grammatik, die diese Sprache erzeugt, und geben Sie eine entsprechende Grammatik an (n, k $\in \mathbb{N}$).

$L_1 = \{a^n \, c \, a^n \mid n \geq 0\}$

$L_2 = \{a^n \, b^k \, c \, a^n \mid n \geq 0, k \geq 1\}$

$L_3 = \{a \, b^n \, a^k \mid n \geq 1, k \geq 1\}$

2.2 Pumping Lemma

Wählen Sie eine nicht-reguläre Sprache aus Teilaufgabe (2.1) aus und beweisen Sie mit Hilfe des *Pumping Lemma*, dass diese nicht regulär ist.

Fortsetzung nächste Seite!

2.3 Eindeutige Grammatik

Es sei folgende Grammatik $G = (V_N, V_T, \Phi, S)$ mit $V_N = \{S, T\}, V_T = \{a,b\}$ und $\Phi = \{S \rightarrow aT, S \rightarrow aTbT, T \rightarrow aT, T \rightarrow aTbT, T \rightarrow \in\}$ gegeben.

a) Zeigen Sie, dass die Grammatik mehrdeutig ist.

b) Welche Sprache wird von dieser Grammatik erzeugt?

c) Geben Sie eine eindeutige, $\in$-freie Grammatik an, die dieselbe Sprache erzeugt.

Aufgabe 3: Berechenbarkeit

3.1 Rekursive Aufzählbarkeit

Sei U eine beliebige entscheidbare Menge und $W \subset U$. Zeigen Sie, dass W genau dann entscheidbar ist, wenn W und $U \backslash W$ rekursiv aufzählbar sind.

3.2 Entscheidbarkeit

Beweisen Sie: Wenn eine nicht-leere Menge $M \subset \mathbb{N}$ eine monoton wachsende Aufzählungsfunktion α_M besitzt, ist M entscheidbar.

Aufgabe 4: Komplexität

4.1 O-Notation

Formulieren Sie möglichst einfache Algorithmen mit den folgenden asymptotischen Laufzeitkomplexitäten:

a) $O(n^2)$

b) $O(\log n)$

c) $O(n \log n)$

Versuchen Sie, die Komplexität durch geeignete Schachtelung von Schleifen zu erreichen. Erläutern Sie Ihre Lösungen. Eine Verwendung der Ausdrücke der asymptotischen Laufzeitkomplexität als Grenzen für Laufvariablen ist nicht zulässig.

Fortsetzung nächste Seite!

4.2 Algorithmen

Beachten Sie folgenden Algorithmus:

```
int vectorSum(int[] vector) {
    int n = vector.length;
    int sum = 0;

    for (int i = 0; i < n; i++) {
        if (i%3 == 0)
            sum += vector[i];
        else
            if (i % 3 == 1)
                sum -= vector[i];
    }

    return sum;
}
```

Bestimmen Sie die Anzahl der Rechenschritte in Abhängigkeit von *n* und die Komplexitätsklasse des Algorithmus (O-Notation). Die Operationen <, %, ==, ++, + = und - = benötigen jeweils einen Rechenschritt, ebenso der Arrayzugriff.

4.3 Algorithmen

Beachten Sie folgenden Algorithmus:

```
int vectorSum(int[] vector) {
    int n = vector.length;
    int sum = 0;

    for (int i = 0; i < n; i += 3)
         sum += vector[i];
    for (int i = 1; i < n; i += 3)
         sum -= vector[i];

    return sum;
}
```

a) Bestimmen Sie die Anzahl der Rechenschritte in Abhängigkeit von *n* und die Komplexitätsklasse des Algorithmus (O-Notation).

b) Vergleichen Sie die Anzahl der Rechenschritte und die Komplexitätsklasse mit dem Algorithmus aus Aufgabe 4.2. Welcher Algorithmus ist besser und was ist der Grund dafür?

Aufgabe 5: Effiziente Algorithmen und Datenstrukturen

a) Skizzieren Sie kurz die Funktionsweise des *Heap-Sort*-Algorithmus. Stellen Sie dabei insbesondere dar, welche beiden Phasen bei diesem Algorithmus unterschieden werden.

b) Gegeben sei die folgende Zahlenfolge:

6, 3, 7, 4, 5, 9, 8, 2

Sortieren Sie diese Zahlenfolge durch Anwendung des *Heap-Sort-Algorithmus*. Beschreiben Sie dabei die einzelnen Schritte des Algorithmus und stellen Sie die Entwicklung des Heaps und des durch den Heap dargestellten Arrays graphisch dar.

Thema Nr. 2

Aufgabe 1:

a) Beweisen Sie die Entscheidbarkeit der Menge
$A = \{n \in \mathbb{N} \mid$ es existieren Primzahlen p und q mit $n = p + q\}$.

b) Beweisen Sie die Aufzählbarkeit der Menge
$B = \{n \in \mathbb{N} \mid$ es existieren Primzahlen p und q mit $n = p - q\}$.

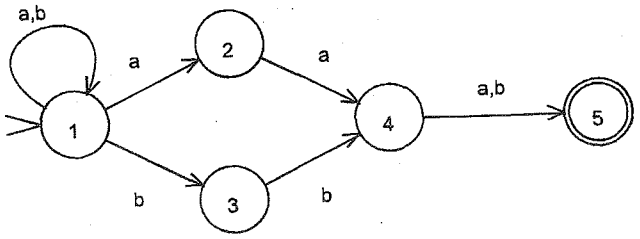
Aufgabe 2:

Konstruieren Sie einen deterministischen, endlichen Automaten, der folgende Sprache über dem Alphabet $\{a, b\}$ akzeptiert.

$C = \{w \in \{a,b\}^* \mid \exists u, v \in \{a, b\}^*$ mit $w = ubv$ und $|v|$ ist durch 3 teilbar$\}$

Aufgabe 3:

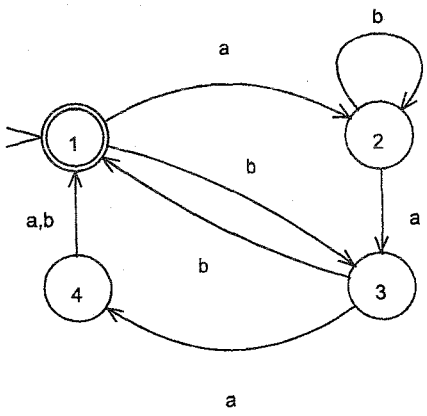
Sei D die von dem folgenden endlichen Automaten akzeptierte Sprache.



Konstruieren Sie einen endlichen Automaten, der die Sprache $\overline{D} = \{w \in \{a, b\}^* \mid w \notin D\}$ akzeptiert.

Aufgabe 4:

Berechnen Sie einen regulären Ausdruck, der die von dem folgenden endlichen Automaten akzeptierte Sprache beschreibt!



Aufgabe 5:

Gegeben ist die Sprache $E = \{a^{2n}b^{3n} \mid n \geq 1\}$ über dem Alphabet $\{a, b\}$.

a) Beweisen Sie, dass E kontextfrei ist.
b) Beweisen Sie, dass E nicht regulär ist.

Aufgabe 6:

Beweisen Sie, dass folgende Menge in NP liegt.
$F = \{(a, b, c, d) \in \mathbb{N}^4 \mid \exists x, y \in \mathbb{N}$ mit $ax^3y + bx^2y^2 + cxy^3 = d\}$

Fortsetzung nächste Seite!

Aufgabe 7:

Geben Sie zwei Sortierverfahren an, deren asymptotische Worst-Case-Laufzeit sinkt, wenn die zu sortierenden Daten (über die ansonsten nichts bekannt ist) schon sortiert sind.

Aufgabe 8:

Sei $G = (V, E)$ ein ungerichteter Graph. Ein *Matching* (oder eine *Paarung*) in G ist eine Teilmenge M der Kantenmenge E, so dass keine zwei Kanten in M zum selben Knoten inzident sind. Sei nun $w: E \to \mathbb{R}_{>0}$ eine Abbildung, die den Kanten von G ein positives Gewicht zuordnet. Dann ist ein Matching M_{opt} ein *schwerstes* (oder *gewichtsmaximales*) Matching, falls $w(M_{\text{opt}}) := \sum_{e \in M_{\text{opt}}} w(e)$ maximal unter allen Matchings in G ist.

Professor Ydeerg schlägt vor, ein schwerstes Matching wie folgt zu berechnen. Man sortiere erst die Kanten nach Gewicht. Solange es noch Kanten gibt, nehme man eine schwerste Kante $\{u, v\}$ ins Matching und lösche alle (verbliebenen) Kanten, die zu u oder v inzident sind.

a) Zeigen Sie, dass Professor Ydeergs Algorithmus zwar immer ein Matching liefert – aber nicht immer ein schwerstes.

b) Nun behauptet Professor Ydeerg, dass sein Algorithmus immer ein Matching liefert, das wenigstens halb so schwer wie ein schwerstes Matching ist. Beweisen Sie seine Behauptung.

Betrachten Sie dazu einen Graphen G und ein schwerstes Matching M_{opt} in G. Was passiert, wenn Professor Ydeergs Algorithmus eine Kante $\{u, v\}$ auswählt und die zu u und v inzidenten Kanten löscht?

c) Welche Laufzeit hat der Algorithmus in Abhängigkeit von der Anzahl n der Knoten und der Anzahl m der Kanten von G?

d) Kann man diese Laufzeit verbessern, wenn man weiß, dass die Gewichtsfunktion w die Kantenmenge E in die Menge $\{1, 2, \ldots, m\}$ abbildet?

Aufgabe 9:

Sie verwalten eine Menge S von natürlichen Zahlen und müssen immer wieder für ein gegebenes Intervall $[a, b]$ (mit $a, b \in \mathbb{N}$) alle Zahlen in $S \cap [a, b]$ liefern.

a) Angenommen, die Menge S ändert sich nicht, wie würden Sie S vorverarbeiten, um Anfragen *ausgabesensitiv* bearbeiten zu können? Mit anderen Worten, Sie sollen durch die Vorverarbeitung erreichen, dass die Anfragezeit nicht nur von der Anzahl n der Elemente in S, sondern auch von der Größe k der Ausgabe abhängt.

Stellen Sie sicher, dass die Anfragezeit $T_{\text{query}}(n, k)$ *sublinear* von n abhängt. Die Abhängigkeit von k sollte linear sein. Wie lange dauert Ihre Vorverarbeitung?

b) Nehmen wir nun an, dass sich die Menge S im Lauf der Zeit ändert. Das heißt, Sie möchten eine dynamische Datenstruktur RangeSet anbieten, die neben obiger Anfrage auch das Einfügen und Löschen von Zahlen erlaubt.

Entwerfen Sie RangeSet, so dass die Laufzeiten aller drei Operationen sublinear von der aktuellen Größe n von S abhängen. Die Anfragezeit soll darüber hinaus wieder linear von k abhängen.

c) Eine andere Art von Anfrage soll die *Summe* der Zahlen in S liefern, die in einem gegebenen Intervall $[a, b]$ liegen.

Beschreiben Sie eine dynamische Datenstruktur, die diese Art von Anfragen sowie Einfügen und Löschen in $O(\log n)$ Zeit zulässt.