Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Herbst 2016

66115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 6

Bitte wenden!

Thema Nr. 1

(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Teilaufgabe I

Gegeben ist der deterministische endliche Automat $(Q,\{0,1\}\delta,q_0,F)$, wobei $Q=\{A,B,C,D,E\}, q_0=A, F=\{E\}$ und

δ	0	1
A	B	C
B	E	C
C	D	C
D	E	A
E	E	E

- Minimieren Sie den Automaten mit dem bekannten Minimierungsalgorithmus. Dokumentieren Sie die Schritte geeignet.
- Geben Sie einen regulären Ausdruck für die erkannte Sprache an.
- Geben Sie die Äquivalenzklassen der Myhill-Nerode-Äquivalenz der Sprache durch reguläre Ausdrücke an.
- Geben Sie ein Beispiel einer regulären Sprache an, für die kein deterministischer endlicher Automat mit höchstens zwei Endzuständen existiert.
- Geben Sie ein Beispiel einer regulären Sprache an, für die kein deterministischer endlicher Automat mit weniger als fünf Zuständen existiert.

Teilaufgabe II

Ordnen Sie die folgenden Sprachen über $\Sigma=\{a,b\}$ bestmöglich in die Chomsky-Hierarchie ein und geben Sie jeweils eine kurze Begründung (1-2 Sätze).

- $L_1 = \{a^n b^n \mid n \geq 1\}$
- $L_2 = \{a^n b^n \mid$ die Turingmaschine mit Gödelnummer n hält auf leerer Eingabe$\}$
- $L_3 = \Sigma^* \setminus L_1$
- $L_4 = \Sigma^* \setminus L_2$
- $L_5 = \{a^n b^m \mid n+m$ ist Vielfaches von drei$\}$
- $L_6 = \{a^n b^n \mid n$ Quadratzahl$\}$

Teilaufgabe III

Sie sind an der Entwicklung eines Konfigurationssystems für Desktopumgebungen beteiligt. Eine gültige Desktopumgebung besteht aus k Komponenten, z.B. Mailer, Editor, Browser, Tabellenkalkulation, etc. Für jede Komponente $i = 1,\ldots k$ gibt es eine Menge A_i von unterschiedlichen Versionen, z. B. kmail, Outlook, thunderbird für den Mailer etc.

Leider sind manche Versionen nicht miteinander kompatibel, so harmoniert Outlook nicht mit dem Betriebssystem Linux. Die bekannten Inkompatibilitäten sind in einer Menge P von Paaren zusammengefasst.

Fortsetzung nächste Seite!

Gefragt ist, ob es möglich ist, für jede verlangte Komponente eine Version so auszuwählen, dass keine zwei ausgewählten Versionen inkompatibel sind. Formal sind also endliche Mengen $A_1,\ldots, A_k$ und eine endliche Menge P von Paaren gegeben.
Gefragt ist, ob eine Auswahl von Elementen $y_1 \in A_1,\ldots, y_n \in A_k$ existiert, sodass $(y_i, y_j) \notin P$ für alle i, j.
Wir nennen dieses Problem KPAKET.

- Begründen Sie, dass KPAKET in NP liegt. Beschreiben Sie eine Reduktion von KNFSAT (Erfüllbarkeit von konjunktiven Normalformen) auf KPAKET. Sie müssen dazu eine Instanz von KNFSAT in eine äquivalente Instanz von KPAKET übersetzen. Idee: die Komponenten entsprechen den Klauseln, die Literale einer Klausel den Versionen.
- Geben Sie konkret die Übersetzung der KNFSAT Instanz $C_1 = \{\neg A, \neg B, \neg D\}$, $C_2 = \{\neg E\}$, $C_3 = \{\neg C, A\}$, $C_4 = \{C\}$, $C_5 = \{B\}$, $C_6 = \{\neg G, D\}$, $C_7 = \{G\}$ an.
- Was können Sie aufgrund dieser Reduktion über die Komplexitätsklassen, in denen KPAKET liegt, aussagen?

Teilaufgabe IV
Es sei $A[0..n-1]$ ein Array von paarweise verschiedenen ganzen Zahlen.
Wir interessieren uns für die Zahl der Inversionen von A; das sind Paare von Indices (i, j), sodass $i < j$ aber $A[i] > A[j]$. Die Inversionen im Array [2, 3, 8, 6, 1] sind (0, 4), da $A[0] > A[4]$ und weiter (1, 4), (2, 3), (2, 4), (3, 4). Es gibt also 5 Inversionen.

1. Wie viel Inversionen hat das Array [3, 7, 1, 4, 5, 9, 2]?
2. Welches Array mit den Einträgen $\{1, \ldots, n\}$ hat die meisten Inversionen, welches hat die wenigsten?
3. Entwerfen Sie eine Prozedur
   ```
   int merge(int[] a, int i, int h, int j);
   ```
   welche das Teilarray a[$i..j$] sortiert und die Zahl der in ihm enthaltenen Inversionen zurückliefert, wobei die folgenden Vorbedingungen angenommen werden:
   * $0 \le i \le h < j < n$, wobei n die Länge von a ist ($n = a.length$).
   * $a[i..h]$ und $a[h+1..j]$ sind aufsteigend sortiert.
   * Die Einträge von $a[i..j]$ sind paarweise verschieden.

   Ihre Prozedur soll in linearer Zeit, also $O(j - i)$ laufen.
   Orientieren Sie sich bei Ihrer Lösung an der Mischoperation des bekannten Mergesort-Verfahrens.
4. Entwerfen Sie nun ein Divide-and-Conquer-Verfahren zur Bestimmung der Zahl der Inversionen, indem Sie angelehnt an das Mergesort-Verfahren einen Algorithmus ZI beschreiben, der ein gegebenes Array in sortierter Form liefert und gleichzeitig dessen Inversionsanzahl berechnet.
   Im Beispiel wäre also
   $$ZI([2, 3, 8, 6, 1]) = ([1, 2, 3, 6, 8], 5)$$
   Die Laufzeit Ihres Algorithmus auf einem Array der Größe n soll $O(n \log(n))$ sein.
   Sie dürfen die Hilfsprozedur `merge` aus dem vorherigen Aufgabenteil verwenden, auch, wenn Sie diese nicht gelöst haben.
5. Begründen Sie, dass Ihr Algorithmus die Laufzeit $O(n \log(n))$ hat.
6. Geben Sie die Lösungen folgender asymptotischer Rekurrenzen (in O-Notation) an:
   (a) $T(n) = 2 * T(n/2) + O(\log n)$
   (b) $T(n) = 2 * T(n/2) + O(n^2)$
   (c) $T(n) = 3 * T(n/2) + O(n)$

Thema Nr. 2
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

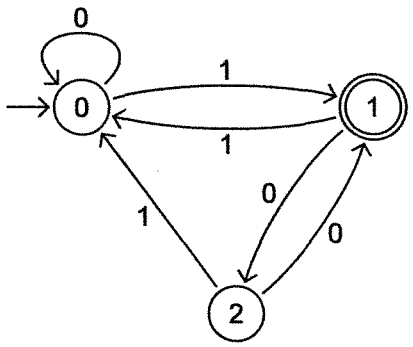
Aufgabe 1:

Geben Sie eine kontextfreie Grammatik für folgende Sprache über dem Alphabet $\{a, b\}$ an.

$$L_0 = \{a^m b^n \mid m, n \in \mathbb{N} \text{ und } m \neq n\}$$

Aufgabe 2:

a) Sei L_1 die von dem folgenden deterministischen, endlichen Automaten akzeptierte Sprache. Geben Sie einen nichtdeterministischen, endlichen Automaten an, der die Sprache L_1^* akzeptiert.



b) Konstruieren Sie einen deterministischen, endlichen Automaten für folgende Sprache.

$L_2 = \{w \in \{a, b\}^* \mid$ die Anzahl der Buchstaben a in w ist ungerade und w endet nicht auf $a\}$

c) Sei $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$ die Menge der positiven natürlichen Zahlen und sei $g : \mathbb{N}^+ \to \mathbb{N}^+$ definiert durch

$$g(x) = \text{größte Zweierpotenz, die Teiler von } x \text{ ist.}$$

Geben Sie das Programm einer Turing-Maschine an, die g berechnet, wobei Ein- und Ausgabe binär dargestellt werden. Es genügt, wenn Ihr Programm für Binärzahlen ohne führende Nullen korrekt arbeitet.

Beispiel: Ihr Programm muss bei Eingabe der Binärdarstellung von 28 die Binärdarstellung von 4 ausgeben, denn 4 ist die größte Zweierpotenz, die Teiler von 28 ist.

Aufgabe 3:

Sei $M_0, M_1, \ldots$ eine Gödelisierung aller Registermaschinen (RAMs). Beantworten Sie folgende Fragen zur Aufzählbarkeit und Entscheidbarkeit. Beweisen Sie Ihre Antworten.

a) Ist folgende Menge entscheidbar?
$A = \{x \in \mathbb{N} \mid x \geq 100$ oder M_x hält bei Eingabe $x\}$

b) Ist folgende Menge entscheidbar?
$B = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid M_x$ hält bei Eingabe x genau dann, wenn M_y bei Eingabe y hält$\}$

Fortsetzung nächste Seite!

c) Ist folgende Menge aufzählbar?
$C = \{x \in \mathbb{N} \mid M_x \text{ hält bei Eingabe 0 mit dem Ergebnis 1}\}$

Aufgabe 4:

a) G bezeichne die Menge der geraden natürlichen Zahlen und $K_0 \subseteq \mathbb{N}$ das spezielle Halteproblem. Definieren Sie eine in polynomieller Zeit berechenbare Funktion r, die G auf K_0 reduziert. Beweisen Sie, dass r in polynomieller Zeit berechenbar ist und die Reduktion von G auf K_0 leistet.

b) Beweisen Sie, dass die Fakultätsfunktion $f(x) = x!$ nicht in polynomieller Zeit berechenbar ist. (Zahlen werden hier wie üblich binär kodiert.)

Aufgabe 5:

In dieser Aufgabe sollen Sie eine Datenstruktur entwerfen, die den minimalen Abstand in einer dynamischen Menge M von ganzen Zahlen aufrechterhält. Zu Beginn sei M leer.

Die Laufzeiten sollen für alle Operationen logarithmisch in der Größe der aktuell gespeicherten Menge M sein.

Argumentieren Sie auch, warum Ihre Implementierung die Laufzeiten einhält.

Tipp: Erweitern Sie eine bekannte Datenstruktur.

a) Wir betrachten zunächst den einfacheren Fall einer halbdynamischen Menge, bei der Zahlen nicht wieder entfernt werden können. Entwerfen Sie hierfür eine Datenstruktur, die folgende Operationen bereitstellt:

- Insert(int k): Fügt die Zahl k in die Menge M, falls k nicht bereits enthalten ist. Sonst bleibt die Operation wirkungslos.
- MinDist(): Liefert den minimalen Abstand zwischen zwei Zahlen in M, falls M mindestens zwei Zahlen enthält, sonst wird -1 zurückgegeben.

b) Nun betrachten wir den allgemeinen Fall einer dynamischen Menge. Entwerfen Sie hierfür eine Datenstruktur, die zusätzlich zu den Operationen Insert und MinDist (definiert wie in Teilaufgabe a)), auch folgende Operation bereitstellt:

- Delete(int k): Löscht die Zahl k aus der Menge M, falls k in der Menge enthalten ist. Sonst bleibt die Operation wirkungslos.

Aufgabe 6:

In dieser Aufgabe betrachten wir nur *einfache* Pfade (also Pfade, die jeden Knoten höchstens einmal enthalten) und *Wurzelbäume* (also Bäume mit einem ausgezeichneten Knoten, der Wurzel). Die Kanten der Wurzelbäume sind nicht gerichtet; Pfade können also „aufsteigen“ und wieder „absteigen“.

a) Lässt sich in einem beliebigen ungerichteten Graphen der längste Pfad effizient bestimmen?

b) Betrachten wir nun einen Spezialfall. Geben Sie einen effizienten Algorithmus an, der für einen gegebenen Wurzelbaum den längsten Pfad bestimmt, der bei der Wurzel beginnt.

c) Geben Sie einen effizienten Algorithmus an, der in einem gegebenen Wurzelbaum den längsten Pfad insgesamt bestimmt. Verwenden Sie den Algorithmus aus Teilaufgabe b) als Unterroutine.

Fortsetzung nächste Seite!

d) Geben Sie einen rekursiven Algorithmus an, der in *Linearzeit* in einem gegebenen Wurzelbaum beides bestimmt: den längsten Pfad insgesamt und den längsten Pfad, der in der Wurzel beginnt.

Aufgabe 7:

Sortieren mit Quicksort

a) Gegeben ist die Ausgabe der Methode Partition (s. Pseudocode), rekonstruieren Sie die Eingabe.

Konkret sollen Sie das Array $A = \langle \_\_, \_\_, 1, \_\_, \_\_ \rangle$ so vervollständigen, dass der Aufruf Partition(A, 1, 5) die Zahl 3 zurückgibt und nach dem Aufruf gilt, dass $A = \langle 1, 2, 3, 4, 5 \rangle$ ist.

Geben Sie A nach jedem Durchgang der for-Schleife in Partition an.

b) Beweisen Sie die Korrektheit von Partition (z.B. mittels einer Schleifeninvarianten)!

c) Geben Sie für jede natürliche Zahl n eine Instanz I_n der Länge n an, so dass QuickSort(I_n) $\Omega(n^2)$ Zeit benötigt. Begründen Sie Ihre Behauptung.

d) Was müsste Partition (in Linearzeit) leisten, damit QuickSort Instanzen der Länge n in $O(n \log n)$ Zeit sortiert? Zeigen Sie, dass Partition mit der von Ihnen geforderten Eigenschaft zur gewünschten Laufzeit von QuickSort führt.

Algorithm: QuickSort(int[] A, $\ell = 1$, $r = A$.length)

```
if ℓ < r then
    m = Partition(A, ℓ, r)
    QuickSort(A, ℓ, m − 1)
    QuickSort(A, m + 1, r)
```

Algorithm: Partition(A, ℓ, r)

```
pivot = A[r]
i = ℓ
for j = ℓ to r − 1 do
    if A[j] ≤ pivot then
        Swap(A, i, j)
        i = i + 1
Swap(A, i, r)
return i
```

Algorithm: Swap(A, i, j)

```
temp = A[i]
A[i] = A[j]
A[j] = temp
```