Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Frühjahr
2018

66115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)** // Vertieft

Einzelprüfung: **Theoretische Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 8

Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

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

δ	0	1
A	$\{A, B\}$	$\{A, C\}$
B	$\{B, C\}$	$\{B\}$
C	$\{C\}$	$\{C\}$

a) Führen Sie für diesen NEA die Potenzmengenkonstruktion durch; geben Sie alle acht entstehenden Zustände mit ihren Transitionen an, nicht nur die erreichbaren.

b) Es sei L eine beliebige reguläre Sprache über dem Alphabet $\Sigma = \{a, b, c\}$. Die Sprache L' über dem Alphabet $\Sigma' = \{a, b\}$ umfasst alle Wörter, die aus Wörtern in L durch Streichen aller cs entstehen. Ist zum Beispiel $L = \{acb, acbcc, abbaccc\}$, so ist $L' = \{ab, abba\}$.
Zeigen Sie, dass L' regulär ist.

c) Die Sprache $L_1 = \{a^n b^n \mid n \geq 1\}$ ist bekanntlich kontextfrei aber nicht regulär. Obwohl die kontextfreien Sprachen nicht unter Komplement abgeschlossen sind, ist $\overline{L_1}$ kontextfrei. Die Sprache $L_2 = \{a^n b^n c^n \mid n \geq 1\}$ ist bekanntlich nicht kontextfrei.
Geben Sie eine kontextfreie Grammatik für $\overline{L_1}$, das Komplement von L_1, an.

d) Die Sprache $L_2 = \{a^n b^n c^n \mid n \geq 1\}$ kann bekanntlich als Schnitt zweier kontextfreier Sprachen dargestellt werden: $L_2 = L_1 c^+ \cap a^+ L_1'$ wobei $L_1' = \{b^n c^n \mid n \geq 1\}$. Zeigen Sie, dass die Komplemente von $L_1 c^+$ und $a^+ L_1'$ kontextfrei sind.

e) Leiten Sie daraus einen Beweis her, dass die kontextfreien Sprachen nicht unter Komplement abgeschlossen sind.

Aufgabe 2:

Beim Problem CLIQUE ist ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl k gegeben. Gefragt ist, ob es in G eine Teilmenge von k Knoten gibt, die paarweise miteinander verbunden sind ("k-Clique"). Dieses Problem ist bekanntlich NP-vollständig.

a) Das Problem 3CLIQUE ist der Spezialfall $k = 3$, gegeben ist also nur ein Graph und gefragt ist, ob er drei paarweise miteinander verbundene Knoten enthält.
Zeigen Sie, dass 3CLIQUE in P ist.

b) Das Problem CLIQUE' ist die Einschränkung von CLIQUE auf den Spezialfall von CLIQUE, bei dem $k \leq |V|/2$ ist. Zeigen Sie, dass CLIQUE' auch NP-vollständig ist.

Fortsetzung nächste Seite!

Aufgabe 3:

In einem ungerichteten Graphen ist eine Kante eine Brückenkante, falls deren Entfernung den Graphen in zwei unzusammenhängende Teilgraphen zerschneidet.

- Entwerfen Sie einen Algorithmus, der alle Brückenkanten findet. Schreiben Sie den Algorithmus in Pseudo-Code auf. Hinweis: beim Test, ob eine Kante eine Brückenkante ist, versuchen Sie durch geeignete Maßnahmen zu vermeiden, dass Knoten mehrfach besucht werden. Sie können dabei geeignete Datenstrukturen für Knoten und Kanten voraussetzen.
- Analysieren Sie die Komplexität Ihres Algorithmus in Abhängigkeit der Anzahl Kanten und Knoten.

Aufgabe 4:

Gegeben ist folgender Pseudo-Code einer Prozedur MAGICSORT, welcher als Argumente ein Array A und zwei Zahlen l, r übergeben bekommt:

```
MAGICSORT(A, l, r)
1  if l + 7 ≥ r
2      then
3          INSERTIONSORT(A, l, r)      //sortiere A von l bis r mit InsertionSort
4      else
5          m := ⌊(r − l + 1)/4⌋
6          for i := 1 to 2 do
7              MAGICSORT(A, l, r − 2m)
8              MAGICSORT(A, l + 2m, r)
9              MAGICDSORT(A, l + m, r − m)
```

Informell: Wenn der zu sortierende Bereich weniger als 9 Elemente besitzt, so werden diese mit Insertion-Sort sortiert. Andernfalls wird der Bereich in Viertel unterteilt; danach werden rekursiv zuerst die ersten beiden Viertel sortiert; dann die letzten beiden Viertel und schließlich die mittleren beiden Viertel. Diese drei Sortierungen werden noch einmal insgesamt wiederholt.

a) Begründen Sie, warum dieser Algorithmus ein korrekter Sortieralgorithmus ist, also den Teilbereich $A[l..r]$ (einschließlich der Grenzen) korrekt sortiert und den Restbereich unverändert lässt. Sie sollten per Induktion argumentieren und die Elemente in vier Typen einteilen: Typ-1 sind die Elemente, die bei korrekter Sortierung im ersten Viertel stehen sollen, etc. Überlegen Sie sich, dass die Typ-1 Elemente nach den ersten drei Aufrufen in der richtigen (also der ersten) Hälfte stehen.

b) Bestimmen Sie die Laufzeit der Methode MAGICSORT als Funktion der Größe n des zu sortierenden Bereiches in O-Notation.

Thema Nr. 2
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:
Geben Sie für folgende Menge $A \subseteq \mathbb{N}$ einen kommentierten Entscheidungsalgorithmus an, der nur Integer-Variablen, For-Schleifen (for i=a to b), Zuweisungen (=), Integer-Addition (+), If-Anweisungen zum Test auf Gleichheit von Integern (if a=b then ...) und Return-Anweisungen (return True/False) verwendet.

$$A = \{n^2 \mid n \in \mathbb{N}\}$$

Aufgabe 2:
Gegeben sei die Sprache $B = (\{a,b\}^*abb\{a,b\}^* + \varepsilon)$ über dem Alphabet $\{a,b\}$.

a) Geben Sie einen nichtdeterministischen endlichen Automaten A_1 an, der B akzeptiert.

b) Konstruieren Sie aus A_1 einen (deterministischen oder nichtdeterministischen) endlichen Automaten A_2, der $\overline{B} = \{a,b\}^* \setminus B$ akzeptiert.

c) Konstruieren Sie aus A_2 eine rechtslineare Grammatik für $\overline{B}$.

Aufgabe 3:
Gesucht ist eine reguläre Sprache $C \subseteq \{a,b\}^*$, deren minimaler deterministischer endlicher Automat (DEA) mindestens 4 Zustände mehr besitzt als der minimale nichtdeterministische endliche Automat (NEA). Gehen Sie wie folgt vor:

a) Definieren Sie $C \subseteq \{a,b\}^*$ und erklären Sie kurz, warum es bei dieser Sprache NEAs gibt, die deutlich kleiner als der minimale DEA sind.

b) Geben Sie den minimalen DEA M für C an.
(Zeichnung des DEA genügt; die Minimalität muss nicht begründet werden)

c) Geben Sie einen NEA N für C an, der mindestens 4 Zustände weniger als M besitzt.
(Zeichnung des NEA genügt)

Aufgabe 4:
Sei $M_0, M_1, \ldots$ eine Gödelisierung der Registermaschinen (RAMs). Beantworten Sie folgende Fragen und beweisen Sie Ihre Antworten.

a) Ist folgende Menge entscheidbar?
$D_1 = \{x \in \mathbb{N} \mid M_x \text{ hält bei Eingabe } 0\}$

b) Ist folgende Menge entscheidbar?
$D_2 = \{y \in \mathbb{N} \mid \text{es existiert ein } x \in \mathbb{N}, \text{sodass } M_x \text{ bei Eingabe } y \text{ hält}\}$

c) Ist folgende Menge aufzählbar?
$D_3 = \{x \in \mathbb{N} \mid \text{es existiert ein } y \in \mathbb{N}, \text{sodass } M_x \text{ bei Eingabe } y \text{ hält}\}$

Fortsetzung nächste Seite!

Aufgabe 5:

Reduzieren Sie das Problem NAE-SAT auf SAT, d.h. geben Sie eine totale, in Polynomialzeit berechenbare Funktion f mit der Eigenschaft $\forall x: \; x \in \text{NAE-SAT} \Leftrightarrow f(x) \in \text{SAT}$ an.

$$\text{NAE-SAT} = \{\varphi \mid \varphi \text{ ist eine aussagenlogische Formel in konjunktiver Normalform, für die eine Belegung der Variablen existiert, sodass in keiner Klausel alle Literale den gleichen Wahrheitswert haben}\}$$

Beachten Sie, dass die in der Definition von NAE-SAT genannte Belegung die Formel φ erfüllt. Z.B. gilt $(x \vee y) \wedge (\neg x \vee \neg y) \in$ NAE-SAT (belege z.B. x mit 1 und y mit 0) und $(x \vee y) \wedge (x \vee \neg y) \notin$ NAE-SAT (belegt man x und y mit dem gleichen Wahrheitswert, so haben alle Literale in der ersten Klausel den gleichen Wahrheitswert, andernfalls haben alle Literale in der zweiten Klausel den gleichen Wahrheitswert).
Eine aussagenlogische Formel in konjunktiver Normalform ist also genau dann in NAE-SAT, wenn es eine Belegung gibt, die

- jede Klausel der Formel erfüllt und
- zugleich in jeder Klausel ein Literal unerfüllt lässt.

Nutzen Sie diese Charakterisierung, um die Reduktion zu formulieren.

Aufgabe 6:

Der Hauptsatz der Laufzeitfunktionen ist bekanntlich folgendermaßen definiert:

Satz (Mastertheorem)

- Sei $T(n) = \begin{cases} d \in \Theta(1), & \text{falls } n \leq k \\ a \cdot T(\frac{n}{b}) + g(n), & \text{sonst} \end{cases}$ mit $k \in \mathbb{N}$, $a \geq 1$ und $b > 1$
- Dann gilt
  1. $g(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0 \Rightarrow T(n) \in \Theta(n^{\log_b a})$
  2. $g(n) \in \Theta(n^{\log_b a}) \Rightarrow T(n) \in \Theta(n^{\log_b a} \cdot \log n) = \Theta(g(n) \cdot \log n)$
  3. $g(n) \in \Omega(n^{\log_b a + \epsilon})$ für ein $\epsilon > 0$ und $a \cdot g(\frac{n}{b}) \leq c \cdot g(n)$ für fast alle n und ein c mit $0 < c < 1$ $\Rightarrow T(n) \in \Theta(g(n))$

a) Betrachten Sie die folgende Methode `m` in Java, die initial mit `m(r, 0, r.length)` für das Array `r` aufgerufen wird. Geben Sie dazu eine Rekursionsgleichung $T(n)$ an, welche die Anzahl an Rechenschritten von `m` in Abhängigkeit von der Länge `n = r.length` berechnet.

```
public static int m(int[] r, int lo, int hi) {
  if (lo < 0 || hi <= lo || lo >= r.length || hi > r.length) {
    throw new IllegalArgumentException();
  }

  if (hi - lo == 1) {
    return r[lo];
  } else if (hi - lo == 2) {
    return Math.max(r[lo], r[lo + 1]);    // O(1)
  } else {
    int s = (hi - lo) / 3;
    int x = m(r, lo, lo + s);
    int y = m(r, lo + s, lo + 2 * s);
    int z = m(r, lo + 2 * s, hi);
    return Math.max(Math.max(x, y), z);  // O(1)
  }
}
```

Fortsetzung nächste Seite!

b) Ordnen Sie die rekursive Funktion $T(n)$ aus (a) einem der drei Fälle des Mastertheorems zu und geben Sie die resultierende Zeitkomplexität an. Zeigen Sie dabei, dass die Voraussetzung des Falles erfüllt ist.

Aufgabe 7:

Sortieren mit Quicksort

a) Gegeben ist das folgende Array von Zahlen: $[23, 5, 4, 67, 30, 15, 25, 21]$.
Sortieren Sie das Array mittels Quicksort *in-situ aufsteigend* von links nach rechts. Geben Sie die (Teil-)Arrays nach jeder Swap-Operation (auch wenn Elemente mit sich selber getauscht werden) und am Anfang jedes Aufrufs der rekursiven Methode an. Verwenden Sie als Pivotelement jeweils das *rechteste* Element im Teilarray und markieren Sie dieses entsprechend. Teilarrays der Länge ≤ 2 dürfen im rekursiven Aufruf durch direkten Vergleich sortiert werden. Geben Sie am Ende das sortierte Array an.

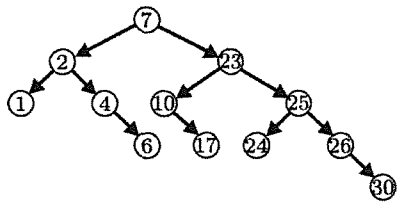
b) Welche Worst-Case-Laufzeit (O-Notation) hat Quicksort für n Elemente? Geben Sie ein Array mit fünf Elementen an, in welchem die Quicksort-Variante aus (a) diese Wort-Case-Laufzeit benötigt (ohne Begründung).

Aufgabe 8:

Bearbeiten Sie folgende Aufgaben zu AVL-Suchbäumen. Geben Sie jeweils bei jeder einzelnen Operation zum Einfügen, Löschen, sowie jeder elementaren Operation zum Wiederherstellen der AVL-Baumeigenschaften den entstehenden Baum als Baumzeichnung an. Geben Sie zur Darstellung der elementaren Operation auch vorübergehend ungültige AVL-Bäume an und stellen Sie Doppelrotationen in zwei Schritten dar. Dabei sollen die durchgeführten Operationen klar gekennzeichnet sein und die Baumknoten immer mit aktuellen Balancewerten versehen sein.

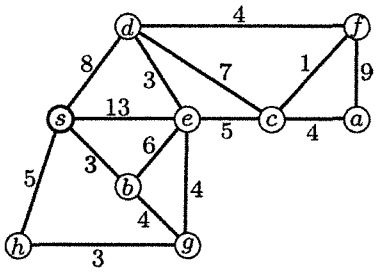
a) Fügen Sie (manuell) nacheinander die Zahlen 5, 14, 28, 10, 3, 12, 13 in einen anfangs leeren AVL-Baum ein.

b) Gegeben sei folgender AVL-Baum. Löschen Sie nacheinander die Knoten 1 und 23. Bei Wahlmöglichkeiten nehmen Sie jeweils den kleineren Wert anstatt eines größeren.



Fortsetzung nächste Seite!

Aufgabe 9:
Gegeben sei folgender Graph G.



a) Berechnen Sie mithilfe des Algorithmus von Dijkstra die kürzesten Wege vom Knoten s zu allen anderen Knoten im Graphen G. Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte den jeweils als nächstes fertigzustellenden Knoten v (wird sog. „schwarz") als Tripel (v, p, δ) mit v als Knotenname, p als aktueller Vorgängerknoten und δ als aktuelle Distanz von s zu v über p an. Führen Sie in der zweiten Spalten alle anderen bisher erreichten Knoten v ebenfalls als Tripel (v, p, δ) auf, wobei diese sog. „grauen Randknoten" in folgenden Durchgängen erneut betrachtet werden müssen. Zeichnen Sie anschließend den entstandenen Wegebaum, d.h. den Graphen G, in dem nur noch diejenigen Kanten vorkommen, die Teil der kürzesten Wege von s zu allen anderen Knoten sind.

b) Der Dijkstra-Algorithmus liefert bekanntlich auf Graphen mit negativen Kantengewichten unter Umständen ein falsches Ergebnis.

   i) Geben Sie einen Graphen mit negativen Kantengewichten an, sodass der Dijkstra-Algorithmus ausgehend von einem von Ihnen ausgezeichneten Startknoten ein korrektes Ergebnis liefert.

   ii) Geben Sie einen Graphen mit negativen Kantengewichten an, sodass der Dijkstra-Algorithmus ausgehend von einem von Ihnen ausgezeichneten Startknoten ein falsches Ergebnis liefert.

   Ein Beweis oder eine Begründung ist jeweils nicht erforderlich.

Aufgabe 10:
Algorithmus von Prim

a) Berechnen Sie mithilfe des Algorithmus von Prim ausgehend vom Knoten a einen minimalen Spannbaum des ungerichteten Graphen G, der durch folgende Adjazenzmatrix gegeben ist:

	a	b	c	d	e	f	g	h
a	-	1	4	6	-	-	-	5
b	1	-	3	-	4	-	7	-
c	4	3	-	1	-	-	-	-
d	6	-	1	-	9	6	2	0
e	-	4	-	9	-	5	5	-
f	-	-	-	6	5	-	-	-
g	-	7	-	2	5	-	-	8
h	5	-	-	0	-	-	8	1

Fortsetzung nächste Seite!

Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte denjenigen Knoten v, der vom Algorithmus als nächstes in den Ergebnisbaum aufgenommen wird (dieser sog. „schwarze" Knoten ist damit fertiggestellt), als Tripel (v, p, δ) mit v als Knotenname, p als aktueller Vorgängerknoten und δ als aktuelle Distanz von v zu p an. Führen Sie in der zweiten Spalte alle anderen vom aktuellen Spannbaum direkt erreichbaren Knoten v (sog. „graue Randknoten") ebenfalls als Tripel (v, p, δ) auf.
Zeichnen Sie anschließend den entstandenen Spannbaum und geben sein Gewicht an.

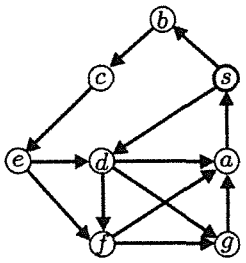
b) Welche Worst-Case-Laufzeitkomplexität hat der Algorithmus von Prim, wenn die grauen Knoten in einem Heap (= Halde) nach Distanz verwaltet werden? Sei dabei n die Anzahl an Knoten und m die Anzahl an Kanten des Graphen. Eine Begründung ist nicht erforderlich.

c) Zeigen Sie durch ein kleines Beispiel, dass ein minimaler Spannbaum eines ungerichteten Graphen nicht immer eindeutig ist.

d) Skizzieren Sie eine Methode, mit der ein maximaler Spannbaum mit einem beliebigen Algorithmus für minimale Spannbäume berechnet werden kann. In welcher Laufzeitkomplexität kann ein maximaler Spannbaum berechnet werden?

Aufgabe 11:

Gegeben sei der folgende gerichtete Graph G:



Traversieren Sie G ausgehend vom Knoten s mittels

a) Tiefensuche (DFS),

b) Breitensuche (BFS)

und geben Sie jeweils die erhaltene Nummerierung der Knoten an. Besuchen Sie die Nachbarn eines Knotens bei Wahlmöglichkeiten immer in alphabetisch aufsteigender Reihenfolge.