Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Frühjahr 2021

46115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik**

Einzelprüfung: **Algorithmen und Datenstrukturen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 12

Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Teilaufgabe I: Theoretische Informatik

Aufgabe 1 (Kaugummi-Automat) [20 PUNKTE]

Ein moderner Kaugummi-Automat erhalte 20- und 50-Cent-Münzen. Eine Packung Kaugummi kostet 120 Cent. Die interne Steuerung des Kaugummi-Automaten verwendet einen deterministischen endlichen Automaten, der die Eingabe als Folge von 20- und 50-Cent-Münzen (d. h. als Wort über dem Alphabet $\{20, 50\}$) erhält, und genau die Folgen akzeptiert, die in der Summe 120 Cent ergeben.

(a) Geben Sie **zwei Worte** aus $\{20, 50\}^*$ an, die der Automat akzeptiert.

(b) Zeichnen Sie einen **deterministischen endlichen Automaten** als **Zustandsgraph**, der für die interne Steuerung des Kaugummi-Automaten verwendet werden kann (d. h. der Automat akzeptiert genau alle Folgen von 20- und 50-Cent-Münzen, deren Summe 120 ergibt).

c) Geben Sie einen **regulären Ausdruck** an, der die akzeptierte Sprache des Automaten erzeugt.

Aufgabe 2 (Kontextfreie Sprachen) [30 PUNKTE]

a) Verwenden Sie den Algorithmus von Cocke, Younger und Kasami (CYK-Algorithmus), um für die folgende kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit Variablen $V = \{S, A, B, C, D\}$, Terminalzeichen $\Sigma = \{a, b\}$, Produktionen

$$P = \{S \to SB \mid AC \mid a,$$

$$A \to a,$$

$$B \to b,$$

$$C \to DD \mid AB,$$

$$D \to AB \mid DC \mid CD\}$$

und Startsymbol S zu prüfen, ob das Wort *aababbabb* in der durch G erzeugten Sprache liegt. Erläutern Sie dabei Ihr Vorgehen und den Ablauf des CYK-Algorithmus.

Fortsetzung nächste Seite!

b) Mit a^i, wobei $i \in \mathbb{N}_0 = \{0, 1, 2, \ldots\}$, wird das Wort bezeichnet, das aus der i-fachen Wiederholung des Zeichens a besteht (d. h. $a^0 = \varepsilon$ und $a^i = aa^{i-1}$, wobei ε das leere Wort ist).

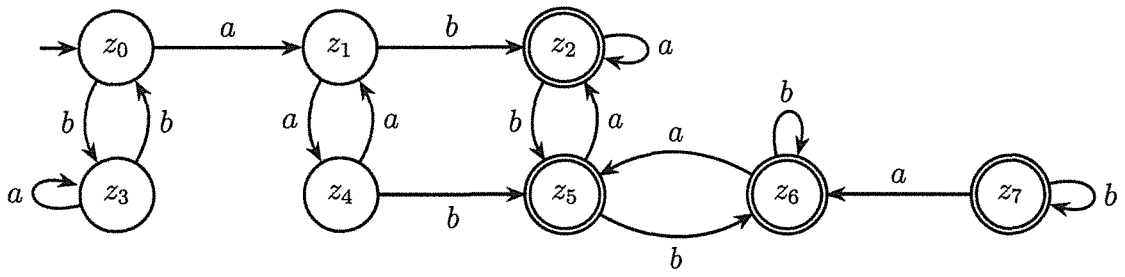
Sei die Sprache L definiert über dem Alphabet $\{a, b\}$ als

$$L = \{a^{2i}b^{4i} \mid i \in \mathbb{N}_0, i \geq 1\}.$$

Zeigen Sie, dass die Sprache L nicht vom Typ 3 der Chomsky-Hierarchie (d. h. nicht regulär) ist.

Aufgabe 3 (Minimierung von Endlichen Automaten) [30 PUNKTE]

Betrachten Sie den unten gezeigten deterministischen endlichen Automaten, der Worte über dem Alphabet $\Sigma = \{a, b\}$ verarbeitet. Bestimmen Sie den dazugehörigen Minimalautomaten, d. h. einen deterministischen endlichen Automaten, der die gleiche Sprache akzeptiert und eine minimale Anzahl an Zuständen benutzt. Erläutern Sie Ihre Berechnung, indem Sie z. B. eine Minimierungstabelle angeben.



Aufgabe 4 (Entscheidbarkeit) [10 PUNKTE]

Beweisen Sie, dass das folgende Entscheidungsproblem semi-entscheidbar ist.

Eingabe: Eine Turingmaschine M
Aufgabe: Entscheiden Sie, ob ein Eingabewort existiert, auf das M hält.

Fortsetzung nächste Seite!

Teilaufgabe II: Algorithmen

Aufgabe 1 (Sortieren) [26 PUNKTE]

a) Geben Sie für folgende Sortierverfahren jeweils zwei Felder A und B an, so dass das jeweilige Sortierverfahren angewendet auf A seine Best-Case-Laufzeit und angewendet auf B seine Worst-Case-Laufzeit erreicht. (Wir messen die Laufzeit durch die Anzahl der Vergleiche zwischen Elementen der Eingabe.) Dabei soll das Feld A die Zahlen $1, 2, \ldots, 7$ genau einmal enthalten; das Feld B ebenso. Sie bestimmen also nur die Reihenfolge der Zahlen.

Wenden Sie als Beleg für Ihre Aussagen das jeweilige Sortierverfahren auf die Felder A und B an und geben Sie nach jedem größeren Schritt des Algorithmus den Inhalt der Felder an.

Geben Sie außerdem für jedes Verfahren asymptotische Best- und Worst-Case-Laufzeit für ein Feld der Länge n an.

Für drei der Sortierverfahren ist der Pseudocode angegeben. Beachten Sie, dass die Feldindizes hier bei 1 beginnen. Die im Pseudocode verwendete Unterroutine Swap(A, i, j) vertauscht im Feld A die Elemente mit den Indizes i und j miteinander.

i) Insertionsort

ii) Bubblesort

iii) Quicksort

```
Insertionsort(int[] A)
for j = 2 to A.length do
    key = A[j]
    i = j − 1
    while i > 0 and A[i] > key do
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

```
Bubblesort(int[] A)
n := length(A)
repeat
    swapped = false
    for i = 1 to n − 1 do
        if A[i − 1] > A[i] then
            Swap(A, i − 1, i)
            swapped := true
until not swapped
```

```
Quicksort(int[] A, ℓ = 1, r = A.length)
if ℓ < r then
    m = Partition(A, ℓ, r)
    Quicksort(A, ℓ, m − 1)
    Quicksort(A, m + 1, r)
```

```
int Partition(int[] A, int ℓ, int 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
```

---

b) Geben Sie die asymptotische Best- und Worst-Case-Laufzeit von Mergesort an.

Fortsetzung nächste Seite!

Aufgabe 2 (Minimum und Maximum) [16 PUNKTE]

a) Argumentieren Sie, warum man das Maximum von n Zahlen nicht mit weniger als $n-1$ Vergleichen bestimmen kann.

b) Geben Sie einen Algorithmus im Pseudocode an, der das Maximum eines Feldes der Länge n mit genau $n-1$ Vergleichen bestimmt.

c) Wenn man das Minimum *und* das Maximum von n Zahlen bestimmen will, dann kann das natürlich mit $2n-2$ Vergleichen erfolgen. Zeigen Sie, dass man bei jedem beliebigen Feld mit deutlich weniger Vergleichen auskommt, wenn man die beiden Werte statt in zwei separaten Durchläufen in einem Durchlauf geschickt bestimmt.

Aufgabe 3 (Breitensuche) [16 PUNKTE]

Wir betrachten eine Variante der Breitensuche (BFS), bei der die Knoten markiert werden, wenn sie das erste Mal besucht werden. Außerdem wird die Suche einmal bei jedem unmarkierten Knoten gestartet, bis alle Knoten markiert sind. Wir betrachten gerichtete Graphen. Ein gerichteter Graph G ist *schwach zusammenhängend*, wenn der ungerichtete Graph (der sich daraus ergibt, dass man die Kantenrichtungen von G ignoriert) zusammenhängend ist.

a) Beschreiben Sie für ein allgemeines $n \in \mathbb{N}$ mit $n \geq 2$ den Aufbau eines schwach zusammenhängenden Graphen G_n mit n Knoten, bei dem die Breitensuche $\Theta(n)$ mal gestartet werden muss, bis alle Knoten markiert sind.

b) Welche asymptotische Laufzeit in Abhängigkeit von der Anzahl der Knoten (n) und von der Anzahl der Kanten (m) hat die Breitensuche über alle Neustarts zusammen? Beachten Sie, dass die Markierungen nicht gelöscht werden. Geben Sie die Laufzeit in Θ-Notation an. Begründen Sie Ihre Antwort.

Aufgabe 4 (Kürzeste-Wege-Bäume und minimale Spannbäume) [13 PUNKTE]

Die Algorithmen von Dijkstra und Jarník-Prim gehen ähnlich vor. Beide berechnen, ausgehend von einem Startknoten, einen Baum. Allerdings berechnet der Algorithmus von Dijkstra einen Kürzesten-Wege-Baum, während der Algorithmus von Jarník-Prim einen minimalen Spannbaum berechnet.

a) Geben Sie einen ungerichteten gewichteten Graphen G mit höchstens fünf Knoten und einen Startknoten s von G an, so dass Dijkstra(G, s) und Jarník-Prim(G, s) ausgehend von s verschiedene Bäume in G liefern. Geben Sie beide Bäume an.

b) Geben Sie eine unendlich große Menge von Graphen an, auf denen der Algorithmus von Jarník-Prim asymptotisch schneller ist als der Algorithmus von *Kruskal*, der ebenfalls minimale Spannbäume berechnet.

Hinweis: Für einen Graphen mit n Knoten und m Kanten benötigt Jarník-Prim $O(m + n \log n)$ Zeit, Kruskal $O(m \log m)$ Zeit.

Fortsetzung nächste Seite!

Aufgabe 5 (Sondierfolgen für Hashing mit offener Adressierung) [19 PUNKTE]

Eine *Sondierfolge* $s(k,i)$ liefert für einen Schlüssel k aus einem Universum U und Versuchsnummern $i = 0, 1, \ldots, m-1$ eine Folge von Indizes für eine Hashtabelle $T[0..m-1]$. Mithilfe einer Sondierfolge wird beim Hashing mit offener Adressierung z. B. beim Einfügen eines neuen Schlüssels k nach einem noch nicht benützten Tabelleneintrag gesucht. Seien h und h' zwei verschiedene Hashfunktionen, die U auf $\{0, 1, \ldots, m-1\}$ abbilden. Beantworten Sie die folgenden Fragen und geben Sie an, um welche Art von Sondieren es sich jeweils handelt.

a) Was ist problematisch an der Sondierfolge $s(k,i) = (h(k) + 2i) \bmod m$, wobei $m = 1023$ die Größe der Hashtabelle ist?

b) Was ist problematisch an der Sondierfolge $s(k,i) = (h(k) + i(i+1)) \bmod m$, wobei $m = 1024$ die Größe der Hashtabelle ist?

c) Was ist vorteilhaft an der Sondierfolge $s(k,i) = (h(k) + i \cdot h'(k)) \bmod m$, wobei m die Größe der Hashtabelle ist?

d) Sei $h(k) = k \bmod 6$ und $h'(k) = k^2 \bmod 6$.

Fügen Sie die Schlüssel 14, 9, 8, 3, 2 in eine Hashtabelle der Größe 7 ein. Verwenden Sie die Sondierfolge $s(k,i) = (h(k) + i \cdot h'(k)) \bmod 7$ und offene Adressierung. Notieren Sie die Indizes der Tabellenfelder und vermerken Sie neben jedem Feld die erfolglosen Sondierungen.

Thema Nr. 2
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Teilaufgabe I: Theoretische Informatik

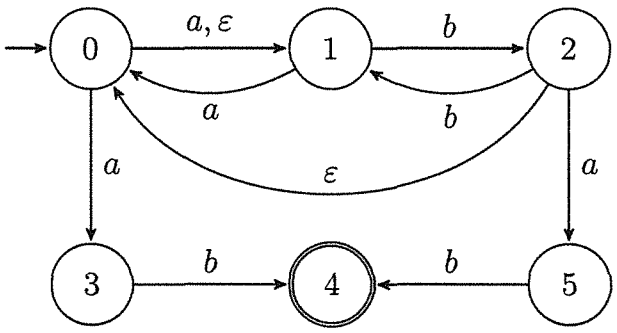
Aufgabe 1 (Reguläre Sprachen) [20 PUNKTE]

a) Betrachten Sie die formale Sprache $L \subseteq \{a, b, c\}^*$: aller Wörter, die entweder mit b beginnen oder mit a enden (aber nicht beides gleichzeitig) und das Teilwort cc enthalten.
Entwerfen Sie einen (vollständigen) deterministischen endlichen Automaten, der die Sprache L akzeptiert. (Hinweis: Es werden weniger als 10 Zustände benötigt.)

b) Ist die folgende Aussage richtig? Begründen Sie Ihre Antwort.
„Jede Teilsprache einer regulären Sprache ist regulär, d. h. für ein Alphabet Σ und formale Sprachen $L' \subseteq L \subseteq \Sigma^*$ ist L' regulär, falls L regulär ist."

Aufgabe 2 (Reguläre Sprachen) [20 PUNKTE]

Gegeben sei der folgende ε-nichtdeterministische endliche Automat A über dem Alphabet $\Sigma = \{a, b\}$:



a) Konstruieren Sie einen deterministischen endlichen Automaten für A. Wenden Sie dafür die Potenzmengenkonstruktion an.

b) Beschreiben Sie die von A akzeptierte Sprache $L(A)$ mit eigenen Worten und so einfach wie möglich.

Fortsetzung nächste Seite!

Aufgabe 3 (Reguläre und Kontextfreie Sprachen) [40 PUNKTE]

Sei $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ die Menge aller natürlichen Zahlen mit 0. Betrachten Sie die folgenden Sprachen.

a) $L_1 = \{a^{3n}b^{2n}a^n \mid n \in \mathbb{N}_0\}$

b) $L_2 = \{a^{3n}a^{2n}b^n \mid n \in \mathbb{N}_0\}$

c) $L_3 = \{(ab)^n a(ba)^n b(ab)^n \mid n \in \mathbb{N}_0\}$

Geben Sie jeweils an, ob L_1, L_2 und L_3 kontextfrei und ob L_1, L_2 und L_3 regulär sind. Beweisen Sie Ihre Behauptung und ordnen Sie jede Sprache in die kleinstmögliche Klasse (*regulär*, *kontextfrei*, *nicht kontextfrei*) ein. Für eine Einordnung in *kontextfrei* zeigen Sie also, dass die Sprache kontextfrei und nicht regulär ist.

Erfolgt ein Beweis durch Angabe eines Automaten, so ist eine klare Beschreibung der Funktionsweise des Automaten und der Bedeutung der Zustände erforderlich. Erfolgt der Beweis durch Angabe eines regulären Ausdruckes, so ist eine intuitive Beschreibung erforderlich. Wird der Beweis durch die Angabe einer Grammatik geführt, so ist die Bedeutung der Variablen zu erläutern.

Aufgabe 4 (Notationen) [10 PUNKTE]

Beweisen oder widerlegen Sie: Die Menge der geraden natürlichen Zahlen $G = \{2n \mid n \in \mathbb{N}\}$ ist auf das spezielle Halteproblem $K_0 = \{x \in \mathbb{N} \mid M_x \text{ hält auf Eingabe } x\}$ reduzierbar.

Teilaufgabe II: Algorithmen

Aufgabe 1 (O-Notation) [12 PUNKTE]

Sortieren Sie die unten angegebenen Funktionen der O-Klassen $O(a)$, $O(b)$, $O(c)$, $O(d)$ und $O(e)$ bezüglich ihrer Teilmengenbeziehungen. Nutzen Sie ausschließlich die echte Teilmenge $\subsetneq$ sowie die Gleichheit $=$ für die Beziehung zwischen den Mengen. Folgendes Beispiel illustriert diese Schreibweise für einige Funktionen f_1 bis f_5. (Diese haben nichts mit den unten angegebenen Funktionen zu tun.)

$$O(f_4) \subsetneq O(f_3) = O(f_5) \subsetneq O(f_1) = O(f_2)$$

Die angegebenen Beziehungen müssen weder bewiesen noch begründet werden.

- $a(n) = \sqrt{n^5} + 4n - 5$
- $b(n) = \log_2(\log_2(n))$
- $c(n) = 2^n$
- $d(n) = n^2 \log(n) + 2n$
- $e(n) = \frac{4^n}{\log_2 n}$

Fortsetzung nächste Seite!

Aufgabe 2 (Implementierung) [19 PUNKTE]

Gegeben sei die folgende Java-Implementierung eines Stacks.

```
class Stack {
    private Item head;

    public Stack () {
        head = null;
    }

    public void push(int val) {
        if (head == null) {
            head = new Item(val, null);
        } else {
            head = new Item(val, head);
        }
    }

    public int pop() {
        // ...
    }

    public int size() {
        // ...
    }

    public int min() {
        // ...
    }

    class Item {
        private int val;
        private Item next;

        public Item(int val, Item next) {
          this.val = val;
          this.next = next;
        }
    }
}
```

Fortsetzung nächste Seite!

a) Implementieren Sie die Methode `pop` in einer objektorientierten Programmiersprache Ihrer Wahl, die das erste Item des Stacks entfernt und seinen Wert zurückgibt. Ist kein Wert im Stack enthalten, so soll dies mit einer `IndexOutOfBoundsException` oder Ähnlichem gemeldet werden.

Beschreiben Sie nun jeweils die notwendigen Änderungen an den bisherigen Implementierungen, die für die Realisierung der folgenden Methoden notwendig sind.

b) `size` gibt in Laufzeit $O(1)$ die Anzahl der enthaltenen `Items` zurück.

c) `min` gibt (zu jedem Zeitpunkt) in Laufzeit $O(1)$ den Wert des kleinsten Elements im Stack zurück.

Sie dürfen jeweils alle anderen angegebenen Methoden der Klasse verwenden, auch wenn Sie diese nicht implementiert haben. Sie können anstelle von objektorientiertem Quellcode auch eine informelle Beschreibung Ihrer Änderungen angeben.

Aufgabe 3 (Lineare und Binäre Suchverfahren) [34 PUNKTE]

Gegeben ist ein aufsteigend sortiertes Array A von n ganzen Zahlen und eine ganze Zahl x. Es wird der Algorithmus `BinarySearch` betrachtet, der A effizient nach dem Wert x absucht. Ergebnis ist der Index i mit $x = A[i]$ oder `NIL`, falls $x \notin A$.

```
1  int BinarySearch(int[] A, int x)
2      ℓ = 1
3      r = A.length
4      while r ≥ ℓ do
5          m = ⌊(ℓ+r)/2⌋
6          if x < A[m] then
7              r = m − 1
8          else if x = A[m] then
9              return m
10         else
11             ℓ = m + 1
12     return NIL
```

a) Durchsuchen Sie das folgende Feld jeweils nach den in (i) bis (iii) angegebenen Werten mittels binärer Suche. Geben Sie für jede Iteration die Werte l, r, m und den betretenen `if`-Zweig an. Geben Sie zudem den Ergebnis-Index bzw. `NIL` an.

Index	1	2	3	4	5	6	7	8	9	10	11
Wert	1	5	6	7	10	13	14	21	31	33	36

(i) 10

(ii) 13

(iii) 22

Fortsetzung nächste Seite!

b) Betrachten Sie auf das Array aus Teilaufgabe a). Für welche Werte durchläuft der Algorithmus nie den letzten `else`-Teil in Zeile 11?
Hinweis: Unterscheiden Sie auch zwischen enthaltenen und nicht-enthaltenen Werten.

c) Wie ändert sich das Ergebnis der binären Suche, wenn im sortierten Eingabefeld zwei aufeinanderfolgende, unterschiedliche Werte vertauscht wurden? Betrachten Sie hierbei die betroffenen Werte, die anderen Feldelemente und nicht enthaltene Werte in Abhängigkeit vom Ort der Vertauschung.

d) Angenommen, das Eingabearray A für den Algorithmus für die binäre Suche enthält nur die Zahlen 0 und 1, aufsteigend sortiert. Zudem ist jede der beiden Zahlen mindestens ein Mal vorhanden. Ändern Sie den Algorithmus für die binäre Suche so ab, dass er den bzw. einen Index k zurückgibt, für den gilt: $A[k] = 1$ und $A[k-1] = 0$.

e) Betrachten Sie die folgende rekursive Variante von BinarySearch.

```
1 int RekBinarySearch(int[] A, int x, int ℓ, int r)
2 |  m = ⌊(ℓ+r)/2⌋
3 |  (rekursive Implementierung)
```

Der initiale Aufruf der rekursiven Variante lautet:

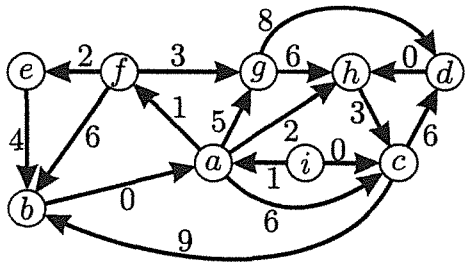
$$\texttt{RekBinarySearch}\ (A, x, 1, A.\text{length})$$

Vervollständigen Sie die Implementierung des Algorithmus in Zeile 3.

Fortsetzung nächste Seite!

Aufgabe 4 (Dijkstra-Algorithmus:) [25 PUNKTE]

a) Berechnen Sie im gegebenen gerichteten und gewichteten Graph $G = (V, E, w)$ mit Kantenlängen $w : E \to \mathbb{R}_0^+$ mittels des Dijkstra-Algorithmus die kürzesten (gerichteten) Pfade ausgehend vom Startknoten a.



Knoten, deren Entfernung von a bereits feststeht, seien als *schwarz* bezeichnet und Knoten, bei denen lediglich eine obere Schranke $\neq \infty$ für ihre Entfernung von a bekannt ist, seien als *grau* bezeichnet.

i) Geben Sie als Lösung eine Tabelle an. Fügen Sie jedes mal, wenn der Algorithmus einen Knoten schwarz färbt, eine Zeile zu der Tabelle hinzu. Die Tabelle soll dabei zwei Spalten beinhalten: die linke Spalte zur Angabe des aktuell schwarz gewordenen Knotens und die rechte Spalte mit der bereits aktualisierten Menge grauer Knoten. Jeder Tabelleneintrag soll anstelle des nackten Knotennamens v ein Tripel $(v, v.d, v.\pi)$ sein. Dabei steht $v.d$ für die aktuell bekannte kürzeste Distanz zwischen a und v. $v.\pi$ ist der direkte Vorgänger von v auf dem zugehörigen kürzesten Weg von a.

ii) Zeichnen Sie zudem den entstandenen Kürzeste-Pfade-Baum.

b) Warum berechnet der Dijkstra-Algorithmus auf einem gerichteten Eingabegraphen mit potentiell auch negativen Kantengewichten $w : E \to \mathbb{R}$ nicht immer einen korrekten Kürzesten-Wege-Baum von einem gewählten Startknoten aus? Geben Sie ein Beispiel an, für das der Algorithmus die falsche Antwort liefert.

c) Begründen Sie, warum das Problem nicht gelöst werden kann, indem der Betrag des niedrigsten (also des betragsmäßig größten negativen) Kantengewichts im Graphen zu allen Kanten addiert wird.