Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Frühjahr
2021

66115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (vertieft)**

Einzelprüfung: **Algorithmen und Datenstrukturen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 17

Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Teilaufgabe I: Theoretische Informatik

Aufgabe 1 (Reguläre Sprachen und Endliche Automaten) [54 PUNKTE]

Im Folgenden bezeichnet $a^i = \underbrace{a \cdots a}_{i\text{-mal}}$ und ε steht für das leere Wort (d.h. insbesondere $a^0 = \varepsilon$).
Die Menge $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ ist die Menge aller nicht-negativer Ganzzahlen.
Die Sprachen $L_1, \ldots, L_{12}$ seien definiert als:

$L_1 = \{a^{2 \cdot n} \mid n \in \mathbb{N}_0\}$

$L_2 = \{a^{2 \cdot n} \mid n \in \mathbb{N}_0, n \geq 1\}$

$L_3 = \{a^{2^n} \mid n \in \mathbb{N}_0\}$

$L_4 = \{a^{2^n} \mid n \in \mathbb{N}_0, n \geq 1\}$

$L_5 = \{a^n \mid n \in \mathbb{N}_0\}$

$L_6 = \{a^n \mid n \in \mathbb{N}_0, n \geq 1\}$

$L_7 = \{a^n \mid n \in \mathbb{N}_0, n \geq 2\}$

$L_8 = \{a^{2 \cdot n+1} \mid n \in \mathbb{N}_0\}$

$L_9 = \{a^{2 \cdot n+1} \mid n \in \mathbb{N}_0, n \geq 1\}$

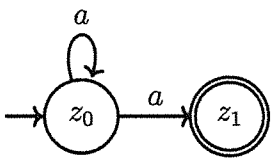
$L_{10} = \{a^n \mid n \in \mathbb{N}_0, n \text{ ist Primzahl}\}$

$L_{11} = \emptyset$

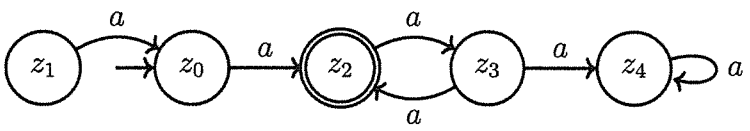
$L_{12} = \{\varepsilon\}$

a) Ordnen Sie jedem der folgenden nichtdeterministischen endlichen Automaten N_j, $j = 1, \ldots, 6$, (die alle über dem Alphabet $\Sigma = \{a\}$ arbeiten) **jeweils eine** der Sprachen $L_i \in \{L_1, \ldots, L_{12}\}$ zu, sodass L_i genau die von N_j **akzeptierte Sprache** ist.

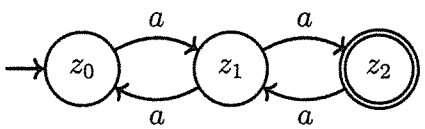
N_1 :



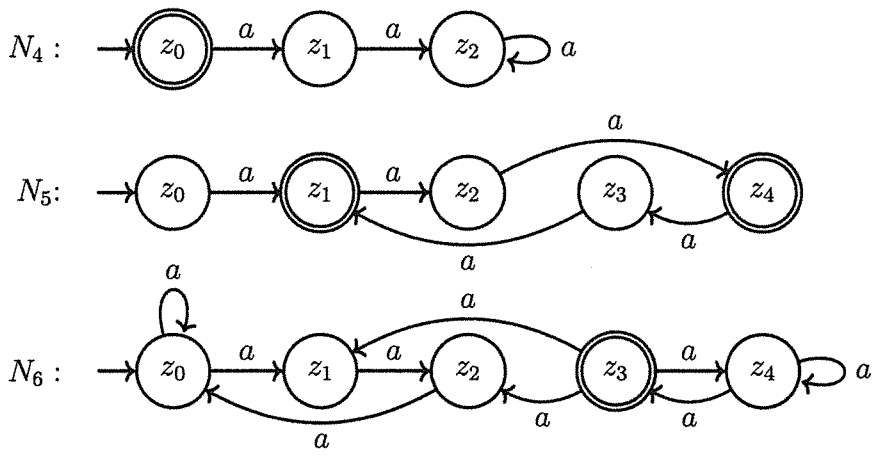
N_2 :



N_3 :

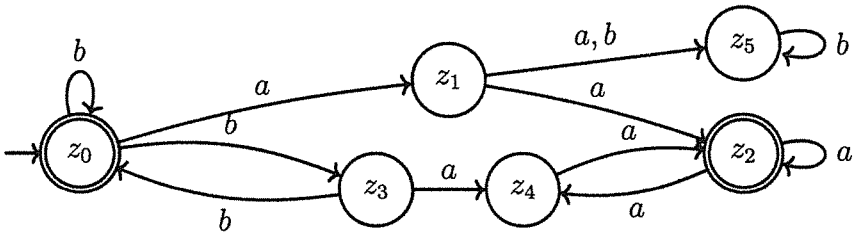


Fortsetzung nächste Seite!



b) Zeigen Sie für **eine** der Sprachen $L_1, \ldots, L_{12}$, dass diese **nicht regulär** ist.

c) Konstruieren Sie für den folgenden nichtdeterministischen endlichen Automaten (der Worte über dem Alphabet $\Sigma = \{a, b\}$ verarbeitet) einen äquivalenten deterministischen endlichen Automaten mithilfe der Potenzmengenkonstruktion. Zeichnen Sie dabei nur die vom Startzustand erreichbaren Zustände. Erläutern Sie Ihr Vorgehen.



Aufgabe 2 (CYK-Algorithmus) [15 PUNKTE]

Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik mit Variablen $V = \{S, A, B, C, D\}$, Terminalzeichen $\Sigma = \{a, b, c\}$, Produktionen

$$P = \{\ S \to AD \mid CC \mid c,$$

$$A \to a,$$

$$B \to b,$$

$$C \to CC \mid c,$$

$$D \to SB \mid CB\}$$

und Startsymbol S. Führen Sie den Algorithmus von Cocke, Younger und Kasami (CYK-Algorithmus) für G und das Wort *aaaccbbb* aus. Liegt *aaaccbbb* in der durch G erzeugten Sprache? Erläutern Sie Ihr Vorgehen und den Ablauf des CYK-Algorithmus.

Fortsetzung nächste Seite!

Aufgabe 3 (Disjunktives Postsches Korrespondenzproblem) [29 PUNKTE]

Das Postsche Korrespondenzproblem und das disjunktive Postsche Korrespondenzproblem sind definiert als:

Postsches Korrespondenzproblem (PCP)

Eine Instanz des Postschen Korrespondenzproblems (PCP) besteht aus einer endlichen Folge K von Wortpaaren $K = (x_1, y_1), \ldots, (x_k, y_k)$ mit $x_i, y_i \in \Sigma^+$, wobei Σ ein Alphabet mit $|\Sigma| > 1$ ist. Das Entscheidungsproblem ist die Frage, ob es eine Folge von Indizes $i_1, \ldots, i_m$ mit $i_j \in \{1, \ldots, k\}$ und $m > 0$ gibt, sodass $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$ gilt.

Disjunktives Postsches Korrespondenzproblem (DPCP)

Eine Instanz des Disjunktiven Postschen Korrespondenzproblems (DPCP) besteht aus einer endlichen Folge K von Worttripeln $K = (x_1, y_1, z_1), \ldots, (x_k, y_k, z_k)$ mit $x_i, y_i, z_i \in \Sigma^+$, wobei Σ ein Alphabet mit $|\Sigma| > 1$ ist. Das Entscheidungsproblem ist die Frage, ob es eine Folge von Indizes $i_1, \ldots, i_m$ mit $i_j \in \{1, \ldots, k\}$ und $m > 0$ gibt, sodass **mindestens eine** der Gleichungen $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$ oder $x_{i_1} \ldots x_{i_m} = z_{i_1} \ldots z_{i_m}$ oder $y_{i_1} \ldots y_{i_m} = z_{i_1} \ldots z_{i_m}$ gilt.

a) Sei $\Sigma = \{a, b\}$ und $K = (ab, b, a), (a, b, b), (b, b, ab)$ eine DPCP-Instanz.

   i) Geben Sie eine Folge $i_1, \ldots, i_4$ von Indizes mit $i_j \in \{1, 2, 3\}$ an, die **keine** Lösung für K ist. Begründen Sie, warum die Folge keine Lösung ist.

   ii) Geben Sie eine Folge von Indizes an, die **eine Lösung** für K ist. Begründen Sie, warum die Folge eine Lösung ist.

b) Sie dürfen als bekannt annehmen, dass PCP unentscheidbar ist. Zeigen Sie die Unentscheidbarkeit von DPCP, indem Sie PCP auf DPCP reduzieren.

Aufgabe 4 (NP-Vollständigkeit) [22 PUNKTE]

Eine 3-CNF ist eine Menge von Klauseln, wobei jede Klausel eine Menge aus **0, 1, 2 oder 3** Literalen ist und kein Literal sowohl positiv als auch negativ in derselben Klausel vorkommt. Ein positives Literal ist eine aussagenlogische Variable x_i. Ein negatives Literal ist eine negierte aussagenlogische Variable $\neg x_i$.

Z. B. ist $F = \{\{x_1, x_2, \neg x_3\}, \{x_1, x_2\}, \{\neg x_1\}\}$ eine 3-CNF.

Eine 3-CNF, in der genau die Variablen $\{x_1, \ldots, x_n\}$ vorkommen, ist **erfüllbar**, wenn es eine Belegung $B : \{x_1, \ldots, x_n\} \to \{0, 1\}$ gibt, sodass in jeder Klausel für mindestens ein Literal L gilt: $B(L) = 1$. Dabei gilt

$$B(\neg x_i) = \begin{cases} 1, & \text{wenn } B(x_i) = 0 \\ 0, & \text{wenn } B(x_i) = 1 \end{cases}$$

Fortsetzung nächste Seite!

Die 3-CNF F ist erfüllbar, was z.B. die Belegung B mit $B(x_1) = 0, B(x_2) = 1$ und $B(x_3) = 1$ bezeugt.

Die Entscheidungsprobleme 3-CNF-SAT und GENAU-3-CNF-SAT sind in gegeben/gefragt-Notation wie folgt definiert:

3-CNF-SAT

gegeben: Eine 3-CNF F

gefragt: Gibt es eine Belegung B, die F erfüllt?

GENAU-3-CNF-SAT

gegeben: Eine 3-CNF F, wobei jede Klausel aus **genau 3** (paarweise verschiedenen) Literalen besteht.

gefragt: Gibt es eine Belegung B, die F erfüllt?

a) Zeigen Sie, dass 3-CNF-SAT in Polynomialzeit auf GENAU-3-CNF-SAT reduziert werden kann. (Dies wird oft als 3-CNF-SAT $\leq_p$ GENAU-3-CNF-SAT notiert.)

b) Zeigen Sie, dass GENAU-3-CNF-SAT NP-vollständig ist. Dabei dürfen Sie als bekannt annehmen, dass 3-CNF-SAT NP-vollständig ist. Die Resultate aus Aufgabenteil a) dürfen verwendet werden.

c) Welche Schlussfolgerungen würden sich jeweils für das berühmte offene $P \stackrel{?}{=} NP$-Problem ergeben, wenn Sie die folgenden Aussagen zeigen könnten?

   i) GENAU-3-CNF-SAT kann in der Zeit $O((k+m) \cdot \log_2(k+m))$ für jede 3-CNF mit k Klauseln und m paarweise verschiedenen Variablen gelöst werden.

   ii) Jede deterministische Turingmaschine, die GENAU-3-CNF-SAT löst, benötigt im Worst-Case $\Omega(2^n)$ Schritte (als untere Schranke), wobei n die Anzahl der Klauseln in der Eingabe ist.

   iii) Es gibt eine deterministische Turingmaschine, die 3-CNF-SAT im Worst-Case in $O(2^n)$ Schritten (als obere Schranke) löst, wobei n die Anzahl der Klauseln in der Eingabe ist.

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.

Die im Pseudocode verwendete Unterroutine **Swap (A,i,j)** vertauscht im Feld A die jeweiligen Elemente mit den Indizes i und j miteinander.

i) **Insertionsort**

ii) Standardversion von **Quicksort** (Pseudocode s.u., Feldindizes beginnen bei 1), bei der das *letzte* Element eines Teilfeldes als Pivot-Element gewählt wird.

iii) **QuicksortVar**: Variante von **Quicksort**, bei der immer das *mittlere* Element eines Teilfeldes als Pivot-Element gewählt wird (Pseudocode s.u., nur eine Zeile neu).

Bei einem Aufruf von **PartitionVar** auf ein Teilfeld $A[\ell..r]$ wird also erst mithilfe der Unterroutine **Swap** $A[\lfloor(\ell + r - 1)/2\rfloor]$ mit $A[r]$ vertauscht.

```
Quicksort(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
```

```
QuicksortVar(A, ℓ = 1, r =
  A.length)
if ℓ < r then
    m = PartitionVar(A, ℓ, r)
    QuicksortVar(A, ℓ, m − 1)
    QuicksortVar(A, m + 1, r)
```

```
int PartitionVar(int[] A, int ℓ, int r)
Swap(A, ⌊(ℓ + r − 1)/2⌋, r)  // neu!
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
```

Fortsetzung nächste Seite!

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

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) [16 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!

c) Sei Z die Menge der zusammenhängenden Graphen und $G \in Z$. Sei n die Anzahl der Knoten von G und m die Anzahl der Kanten von G. Entscheiden Sie mit Begründung, ob $\log m \in \Theta(\log n)$ gilt.

Aufgabe 5 (Projektplanung) [24 PUNKTE]

Sie sind Projektleiter und sollen einen Projektplan aufstellen. In der nächsten Zeit sind eine Menge A von Aufgaben zu erledigen, wobei Sie wissen, dass die Erledigung jeder Aufgabe a in A eine gewisse Anzahl $\ell(a)$ Stunden benötigt.

Außerdem hängen manche Aufgaben von anderen Aufgaben ab. Sie haben also zu jeder Aufgabe a eine (möglicherweise leere) Menge $N(a) \subset A$ von Aufgaben, die erst erledigt werden können, wenn a abgeschlossen ist. Zusätzlich gibt es für jede Aufgabe $a' \in N(a)$ eine Zeitdauer $w(a, a')$, die Sie nach der Erledigung von a warten müssen, bis Sie a' beginnen können.

a) Modellieren Sie das Problem graphentheoretisch.

Beantworten Sie folgende Fragestellungen aus graphentheoretischer Sicht.

b) Wovon hängt es ab, ob es einen Projektplan gibt, der von vorne bis hinten abgearbeitet werden kann?

c) Wie können Sie einen solchen Projektplan berechnen, d. h. wie können Sie eine Reihenfolge der Aufgaben ermitteln, bei der alle Abhängigkeiten berücksichtigt sind?

d) Angenommen, Sie haben eine sinnvolle Reihenfolge $a_1, a_2, \ldots, a_n$ der Aufgaben in A gefunden – geben Sie einen Algorithmus an, der berechnet, wie viele Stunden es dauert, Ihren Projektplan auszuführen.

Ihr Algorithmus soll für $i = 2, 3, \ldots, n$ die Startzeit s_i und die Endzeit e_i der Aufgabe a_i berechnen. Für die erste Aufgabe a_1 gelte $s_1 = 0$ und $e_1 = \ell(a_1)$.

Aufgabe 6 (Sondierfolgen für Hashing mit offener Adressierung) [22 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?

Fortsetzung nächste Seite!

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

Fügen Sie die Schlüssel 14, 9, 8, 3, 2 nacheinander 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) [30 PUNKTE]

a) Sei

$$L_1 = \{w \in \{a,b,c\}^* \mid w \text{ enthält genau zweimal den Buchstaben a und der vorletzte Buchstabe ist ein } c\}.$$

Geben Sie einen regulären Ausdruck für die Sprache L_1 an.

b) Konstruieren Sie einen deterministischen endlichen Automaten für die Sprache L_2:

$$L_2 = \{w \in \{a,b\}^* \mid w \text{ enthält } \textbf{genau einmal} \text{ das Teilwort } aab\} .$$

c) Sei $\mathbb{N} = \{1,2,3,\ldots\}$ die Menge der strikt positiven natürlichen Zahlen. Sei

$$L_3 = \{\#a^{i_1}\#a^{i_2}\#\cdots a^{i_{n-i}}\#a^{i_n}\# \mid n, i_1, \ldots, i_n \in \mathbb{N} \text{ und es existiert } j \in \mathbb{N} \text{ mit } i_j = n+1\}$$

eine Sprache über Alphabet $\{\#, a\}$.

So ist z.B. $\#a\#aaa\# \in L_3$ (da das Teilwort $a^3 = aaa$ vorkommt) und $\#a\#a\#a\# \notin L_3$ (da das Teilwort $a^5 = aaaaa$ nicht vorkommt). Beweisen Sie, dass L_3 nicht regulär ist.

Aufgabe 2 (Kontextfreie Sprachen) [30 PUNKTE]

a) Zeigen Sie, dass die Sprache

$$L_4 = \{ww_1ww_2 \mid w, w_1, w_2 \in \{a,b,c\}^* \text{ und } 2|w| \geq |w_1| + |w_2|\}$$

nicht kontextfrei ist.

b) Betrachten Sie die Aussage

Seien $L_1, \ldots, L_n$ beliebige kontextfreie Sprachen.
Dann ist $\bigcap_{i=1}^{n} L_i$ immer eine entscheidbare Sprache.

Entscheiden Sie, ob diese Aussage wahr ist oder nicht und begründen Sie Ihre Antwort.

c) Sei $\mathbb{N}_0 = \{0,1,2,\ldots\}$ die Menge der nicht negativen natürlichen Zahlen. Es ist bekannt, dass $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ keine kontextfreie Sprache ist. Ist die Komplementsprache $L_5 = \{a,b,c\}^* \setminus L$ kontextfrei? Begründen Sie Ihre Antwort.

Fortsetzung nächste Seite!

Aufgabe 3 (Entscheidbarkeit) [30 PUNKTE]

Wir betrachten eine Gödelisierung von Turingmaschinen und bezeichnen mit M_w die Turingmaschine, die gemäß der Kodierung des Binärworts w kodiert wird. Außerdem bezeichnen wir mit $M_w(x)$ die Ausgabe der Maschine M_w bei Eingabe x. Sie dürfen davon ausgehen, dass x immer ein Binärstring ist. Der bekannte Satz von Rice sagt:

> Sei S eine Menge berechenbarer Funktionen mit $\emptyset \neq S \neq \mathcal{R}$, wobei $\mathcal{R}$ die Menge aller berechenbaren Funktionen ist. Dann ist die Sprache $\{w \mid f_{M_w} \in S\}$ unentscheidbar.

Hier ist f_{M_w} die von M_w berechnete Funktion.

Zeigen Sie für jede der nachfolgenden Sprachen über dem Alphabet $\{0, 1\}$ entweder, dass sie entscheidbar ist, oder zeigen Sie mit Hilfe des Satzes von Rice, dass sie unentscheidbar ist. Geben Sie beim Beweis der Unentscheidbarkeit die Menge S der berechenbaren Funktionen an, auf die Sie den Satz von Rice anwenden. Wir bezeichnen die Länge der Eingabe x mit $|x|$.

a) $\{w \mid M_w$ akzeptiert die Binärkodierungen der Primzahlen (und lehnt alles andere ab)$\}$

b) $\{w \mid$ es gibt eine Eingabe x, so dass $M_w(x)$ das Symbol 1 enthält$\}$

c) $\{w \mid M_w(x)$ hält für jedes x mit $|x| \leq 1000$ nach höchstens 100 Schritten an$\}$

d) $\{w \mid M_w$ hat für jede Eingabe dieselbe Ausgabe$\}$

e) $\{w \mid$ die Menge der Eingaben, die von M_w akzeptiert werden, ist endlich$\}$

Aufgabe 4 (Komplexität) [30 PUNKTE]

Betrachten Sie die folgenden Probleme:

CLIQUE	
Gegeben:	Ein ungerichteter Graph $G = (V, E)$, eine Zahl $k \in \mathbb{N}$
Frage:	Gibt es eine Menge $S \subseteq V$ mit $\|S\| = k$, sodass für alle Knoten $u \neq v \in V$ gilt, dass $\{u, v\}$ eine Kante in E ist?

ALMOSTCLIQUE	
Gegeben:	Ein ungerichteter Graph $G = (V, E)$, eine Zahl $k \in \mathbb{N}$
Frage:	Gibt es eine Menge $S \subseteq V$ mit $\|S\| = k$, sodass die Anzahl der Kanten zwischen Knoten in S genau $\frac{k(k-1)}{2} - 1$ ist?

Zeigen Sie, dass das Problem ALMOSTCLIQUE NP-vollständig ist. Nutzen Sie dafür die NP-Vollständigkeit von CLIQUE.

Hinweis: Die Anzahl der Kanten einer k-Clique sind $\frac{k(k-1)}{2}$.

Fortsetzung nächste Seite!

Teilaufgabe II: Algorithmen

Aufgabe 1 (O-Notation) [24 PUNKTE]

a) 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 a bis e zu tun.)

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

Die von Ihnen anzugebenden Teilmengenbeziehungen müssen weder bewiesen noch begründet werden.

- $a(n) = 3n^2 + \sqrt{4n}$
- $b(n) = 3^{\log_2(n)}$
- $c(n) = 3^{n-1}$
- $d(n) = n^2 - \log_2 n^{10}$
- $e(n) = 2^{n/2}$

b) Gegeben seien drei Funktionen $f, g, h\colon \mathbb{N} \to \mathbb{R}_0^+$. Wir definieren die Funktion $(f+g)\colon \mathbb{N}_0 \to \mathbb{R}_0^+, n \mapsto f(n) + g(n)$.

Beweisen Sie die folgenden Aussagen formal nach den Definitionen der O-Notation oder widerlegen Sie sie.

(i) $f \in \Theta(h) \wedge g \in O(h) \Rightarrow f + g \in \Theta(h)$.

(ii) $f \in \Theta(h) \wedge g \in \Omega(h) \Rightarrow f + g \in \Theta(h)$.

c) Bestimmen Sie eine asymptotische Lösung (in Θ-Schreibweise) für die folgende Rekursionsgleichung:

$$T(n) = 3 \cdot T(\frac{n}{5}) + \frac{n}{2}$$

Fortsetzung nächste Seite!

Aufgabe 2 (Implementierung) [19 PUNKTE]

Gegeben sei die folgende Java-Implementierung einer doppelt-verketteten Liste.

```
class DoubleLinkedList {
    private Item head;

    public DoubleLinkedList () {
        head = null;
    }

    public Item append(Object val) {
        if (head == null) {
            head = new Item(val, null, null);
            head.prev = head;
            head.next = head;
        } else {
            Item item = new Item(val, head.prev, head);
            head.prev.next = item;
            head.prev = item;
        }
        return head.prev;
    }

    public Item search(Object val) {
        // ...
    }

    public void delete(Object val) {
        // ...
    }

    class Item {
        private Object val;
        private Item prev;
        private Item next;

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

a) Skizzieren Sie den Zustand der Datenstruktur nach Aufruf der folgenden Befehlssequenz. Um Variablen mit Zeigern auf Objekte darzustellen, können Sie mit dem Variablennamen beschriftete Pfeile verwenden.

Fortsetzung nächste Seite!

```
DoubleLinkedList list = new DoubleLinkedList();
list.append("a");
list.append("b");
list.append("c");
```

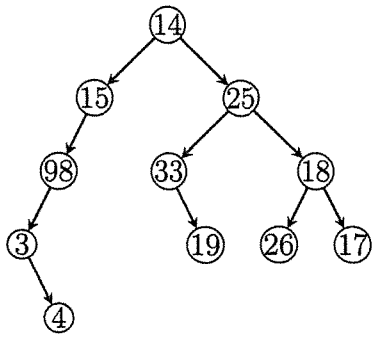
b) Implementieren Sie in der Klasse `DoubleLinkedList` die Methode `search`, die zu einem gegebenen Wert das `Item` der Liste mit dem entsprechenden Wert, oder `null` falls der Wert nicht in der Liste enthalten ist, zurückgibt.

c) Implementieren Sie in der Klasse `DoubleLinkedList` die Methode `delete`, die das erste Vorkommen eines Wertes aus der Liste entfernt. Ist der Wert nicht in der Liste enthalten, terminiert die Methode „stillschweigend", d. h. ohne Änderung der Liste und ohne Fehlermeldung. Sie dürfen die Methode `search` aus Teilaufgabe b) verwenden, auch wenn Sie sie nicht implementiert haben.

d) Beschreiben Sie die notwendigen Änderungen an der Datenstruktur und an den bisherigen Implementierungen, um eine Methode `size`, die die Anzahl der enthaltenen `Items` zurück gibt, mit Laufzeit $O(1)$ zu realisieren.

Aufgabe 3 (Binärbäume) [20 PUNKTE]

a) Betrachten Sie folgenden Binärbaum T.



Geben Sie die Schlüssel der Knoten in der Reihenfolge an, wie sie von einem Preorder-Durchlauf (= TreeWalk) von T ausgegeben werden.

b) Betrachten Sie folgende Sequenz als Ergebnis eines Preorder-Durchlaufs eines binären Suchbaumes T. Zeichnen Sie T und erklären Sie, wie Sie zu Ihrer Schlussfolgerung gelangen.

$$[8, 7, 4, 2, 1, 3, 5, 6, 10, 9, 11]$$

Hinweis: Welcher Schlüssel ist die Wurzel von T? Welche Knoten sind in seinem linken/rechten Teilbaum gespeichert? Welche Schlüssel sind die Wurzeln der jeweiligen Teilbäume?

c) Anstelle von sortierten Zahlen soll ein Baum nun verwendet werden, um relative Positionsangaben zu speichern. Jeder Baumknoten enthält eine Beschriftung und einen Wert (vgl. Abb. 1), der die ganzzahlige relative Verschiebung in horizontaler Richtung gegenüber seinem Elternknoten angibt. Die zu berechnenden Koordinaten für einen Knoten ergeben sich

Fortsetzung nächste Seite!

aus seiner Tiefe im Baum als y-Wert und aus der Summe aller Verschiebungen auf dem Pfad zur Wurzel als x-Wert. Das Ergebnis der Berechnung ist in Abb. 2 visualisiert. Geben Sie einen Algorithmus mit linearer Laufzeit in Pseudo-Code oder einer objektorientierten Programmiersprache Ihrer Wahl an. Der Algorithmus erhält den Zeiger auf die Wurzel eines Baumes als Eingabe und soll Tupel mit den berechneten Koordination aller Knoten des Baums in der Form (Beschriftung, x, y) zurück- oder ausgeben.

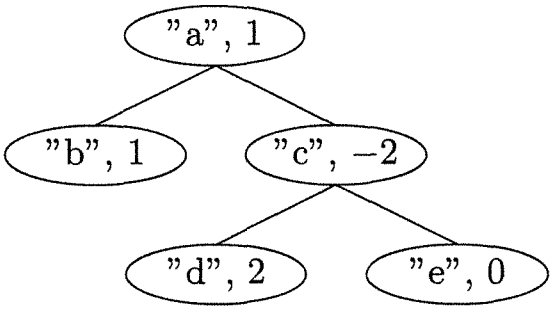


Abb. 1

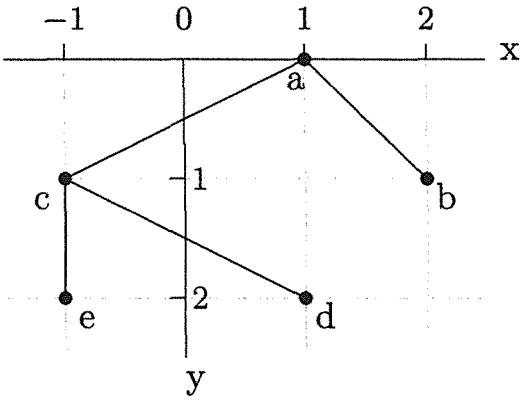


Abb.2.

Aufgabe 4 (TripleSort) [40 PUNKTE]

Im folgenden ist ein Algorithmus in Pseudocode gegeben, der die Zahlen in einem gegebenen Array sortieren soll.

```
1  TripleSort(int[] A, int ℓ, int r)
2  |  n = r − ℓ + 1
3  |  if n = 2 then
4  |  |  min = min(A[ℓ], A[r])
5  |  |  max = max(A[ℓ], A[r])
6  |  |  A[ℓ] = min
7  |  |  A[r] = max
8  |  else if n > 2 then
9  |  |  d = ⌊n/3⌋
10 |  |  TripleSort(A, ℓ, r − d)
11 |  |  TripleSort(A, ℓ + d, r)
12 |  |  TripleSort(A, ℓ, r − d)
```

a) Führen Sie den Algorithmus auf dem absteigend sortierten Array $A = [6, 5, 4, 3, 2, 1]$ mit $\ell = 1$ und $r = 6$ aus. Geben Sie für jeden Aufruf die Rekursionstiefe, die Parameter und den Zustand des Arrays vor bzw. nach dem Aufruf an. Sie können hier z. B. eine Tabelle mit den Spalten Tiefe, ℓ, r, $A[1]$, $A[2]$, ..., $A[6]$ und zwei Zeilen je Aufruf (eine für davor und eine für danach) verwenden. Bereiche, die vom aktuellen Aufruf nicht betrachtet oder verändert werden, müssen Sie nicht erneut angeben. Rekursive Aufrufe von `tripleSort`, bei denen der

Fortsetzung nächste Seite!

zu sortierende Bereich drei Elemente oder weniger enthält, können Sie direkt lösen. Damit kann Ihre Antwort das folgende Format haben:

t	ℓ	r	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$	
1	1	6	6	5	4	3	2	1	
				...					
3	2	4	.	5	4	3	.	.	
3	2	4	.	3	4	5	.	.	(direkt sortiert)

b) Wie muss die Eingabe für den Algorithmus im Allgemeinen aussehen, sodass (im initialen Aufruf) der dritte rekursive Aufruf von `TripleSort` in Zeile 12 keine Änderung bewirkt?

c) Beweisen Sie die Korrektheit des Algorithmus. *Hinweis:* Wo befinden sich die $\frac{1}{3}n$ größten Werte nach dem ersten rekursiven Aufruf?

d) Bestimmen Sie die asymptotische Laufzeit des Algorithmus in Abhängigkeit von $n = r - \ell + 1$.

Aufgabe 5 (Hashing) [17 PUNKTE]

a) Nennen Sie zwei wünschenswerte Eigenschaften von Hashfunktionen.

b) Wie viele Elemente können bei Verkettung und wie viele Elemente können bei offener Adressierung in einer Hashtabelle mit m Zeilen gespeichert werden?

c) Angenommen, in einer Hashtabelle der Größe m sind alle Einträge (mit mindestens einem Wert) belegt und insgesamt n Werte abgespeichert.

Geben Sie in Abhängigkeit von m und n an, wie viele Elemente bei der Suche nach einem nicht enthaltenen Wert besucht werden müssen. Sie dürfen annehmen, dass jeder Wert mit gleicher Wahrscheinlichkeit und unabhängig von anderen Werten auf jeden der m Plätze abgebildet wird (einfaches gleichmäßiges Hashing).

d) Betrachten Sie die folgende Hashtabelle mit der Hashfunktion $h(x) = x \bmod 11$. Hierbei steht $\emptyset$ für eine Zelle, in der kein Wert hinterlegt ist.

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	$\emptyset$	$\emptyset$	3	$\emptyset$	16	28	18	$\emptyset$	$\emptyset$	32

Führen Sie nun die folgenden Operationen mit offener Adressierung mit linearem Sondieren aus und geben Sie den Zustand der Datenstruktur nach jedem Schritt an. Werden für eine Operation mehrere Zellen betrachtet, aber nicht modifiziert, so geben Sie deren Indizes in der betrachteten Reihenfolge an.

Fortsetzung nächste Seite!

i) Insert 7

ii) Insert 20

iii) Delete 18

iv) Search 7

v) Insert 5