

**Sammlung aller Staatsexamensaufgaben der  
Prüfungsnummer**

**66115**

**Theoretische Informatik / Algorithmen  
(vertieft)**

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2006**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	<b>Frühjahr</b>	
Kennwort: _____	<b>2006</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

**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: 7

**Thema Nr. 1**

1. Spracheindeutigkeit

Gegeben sei folgende Grammatik:

$$G : (\{S\}, \{+, x, y, z\}, \{S \rightarrow S+S, S \rightarrow z, S \rightarrow x, S \rightarrow y\}, S)$$

1.1. Welche Sprache  $L$  wird von  $G$  erzeugt (ohne Beweis)?

1.2. Beweisen oder widerlegen Sie: Die Grammatik  $G$  ist eindeutig.

1.3. Ist die Sprache  $L$  eindeutig? Begründen Sie Ihre Antwort.

2. Chomsky-Hierarchie

Gegeben sei die folgende Grammatik:

$$G := (\{S, M\}, \{x, \#\}, \{S \rightarrow SMx, S \rightarrow \#, xM \rightarrow Mx, \#M \rightarrow xx\#\}, S)$$

- 2.1. Welchen Typ (Namen und Nummer in der Chomsky-Hierarchie) hat die Grammatik  $G$ ?
- 2.2. Welche Sprache  $L$  wird von  $G$  erzeugt (ohne Beweis)?
- 2.3. Zu welchen Sprachtypen gehört die Sprache  $L$  und zu welchen gehört sie nicht?  
Zitieren Sie, soweit möglich, die Chomsky-Hierarchie. Geben Sie an der entscheidenden Stelle eine erzeugende Grammatik an und benutzen Sie das entsprechende Pumping-Lemma.

### 3. Turing-Maschinen

Eine deterministische Turingmaschine ist ein Tupel  $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ . Dabei ist  $Q$  die endliche Zustandsmenge,  $\Sigma$  das endliche Eingabealphabet,  $\Gamma$  das endliche Bandalphabet,  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$  die Übergangsfunktion,  $q_0 \in Q$  der Startzustand,  $B$  das Blank-symbol und  $F \subseteq Q$  eine Menge von Endzuständen. Eine solche, deterministische Turingmaschine liest also in jedem Schritt das aktuelle Zeichen unter dem Schreib-/Lesekopf, entscheidet abhängig vom aktuellen Zustand und dem gelesenen Zeichen welches neue Zeichen geschrieben, in welchen Folgezustand übergegangen und ob dabei der Schreib-/Lesekopf nach rechts ( $R$ ), nach links ( $L$ ) oder gar nicht ( $N$ ) bewegt werden soll.

Sei nun  $\Sigma = (0, 1)$  und  $\Gamma = (0, 1, B)$ . Konstruieren Sie eine Turingmaschine, welche eine Zahl ungleich Null in Binärdarstellung um Eins dekrementiert und die Zahl Null ggf. unberührt lässt. Beachten Sie dabei die folgenden Vorgaben:

Auf dem Band steht die gegebene Zahl in Binärdarstellung mit dem niederwertigsten Bit ganz rechts. Führende Nullen sind zugelassen. Beispiel für die Dezimalzahl 13:  $B\underline{B}00001101BB$ . Der Schreib-/Lesekopf steht zu Beginn der Verarbeitung auf dem ersten  $B$  links von der Eingabe und soll auch am Ende wieder dort stehen.

Schreiben Sie die Übergangsfunktion  $\delta$  in Tabellenform nieder, pro Zustand eine Spalte, pro Zeile ein Bandsymbol und dann in jeder Zelle den Folgezustand, das zu schreibende Zeichen sowie die Kopfbewegung.

### 4. Endliche Automaten

Gegeben sei folgender nichtdeterministischer, endlicher Automat  $A_a$ :

$$A_a = (Q, \Sigma, \delta, q_0 \{q_4\})$$

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

$$\Sigma = \{a, b\}$$

$$\delta \subseteq (Q \times \Sigma) \times Q, \delta = \{(q_0, a, q_1), (q_0, a, q_2), (q_0, b, q_3), (q_2, b, q_4),$$

$$(q_3, a, q_4), (q_4, a, q_5), (q_5, a, q_4), (q_1, a, q_0)\}$$

- 4.1. Stellen Sie den Automaten  $A_a$  graphisch dar.
- 4.2. Beschreiben Sie die vom Automaten  $A_a$  akzeptierte Sprache durch einen regulären Ausdruck (ohne Beweise).
- 4.3. Konstruieren Sie aus  $A_a$  einen äquivalenten, deterministischen, endlichen Automaten  $A_b$  und stellen Sie ihn graphisch dar.
- 4.4. Ist der in 4.3. konstruierte Automat minimal? Begründen Sie Ihre Antwort.  
Ist der Automat noch nicht minimal, so geben Sie den Minimalautomaten in graphischer Form an.

## 5. Komplexitätstheorie

Sei  $RUCKSACK := \left\{ (A, g, w, G, W) \mid A \text{ endliche Menge}, g : A \rightarrow \mathbb{N}, w : A \rightarrow \mathbb{N}, G \in \mathbb{N}, W \in \mathbb{N} \right\}$   
und

$$RUCKSACK^+ := \left\{ (A, g, w, G, W) \in RUCKSACK \mid \exists B \subseteq A : \sum_{a \in B} g(a) \leq G \wedge \sum_{a \in B} w(a) \geq W \right\}.$$

Das *Rucksackproblem* besteht darin, für ein gegebenes Tupel  $x = (A, g, w, G, W) \in RUCKSACK$  zu entscheiden, ob  $x \in RUCKSACK^+$  gilt.

Sei  $TEILE := \left\{ A \mid A \subset \mathbb{N} \text{ endlich} \right\}$  und  $TEILE^+ := \left\{ A \in TEILE \mid \exists B \subseteq A : \sum_{b \in B} b = \sum_{b \in A \setminus B} b \right\}$ .

Das *Teileproblem* besteht darin, für eine gegebene Menge  $A \in TEILE$  zu entscheiden, ob  $A \in TEILE^+$  gilt. Das *Teileproblem* ist NP-vollständig.

- 5.1. Beschreiben Sie einen nichtdeterministischen Algorithmus zur Lösung des Rucksackproblems in polynomieller Zeit abhängig von der Länge der Eingabe.
- 5.2. Zeigen Sie, dass das *Teileproblem* polynomiell auf das Rucksackproblem reduziert werden kann  $TEILE^+ \leq_{pol} RUCKSACK^+$ .
- 5.3. Schließen Sie daraus formal die NP-Vollständigkeit des Rucksackproblems.

## 6. Verifikation

Gegeben ist folgender Algorithmus welcher verifiziert werden soll:

```
// {x ≥ 0} = P
y = 0
z = 0
while (z ≤ x - 1)
{
    y = y + z + z + 1
    z = z + 1
}
// {...} = Q
```

- 6.1. Geben Sie die Bedingungen an, welche nach dem Ablauf der while-Schleife für das Prädikat  $Q$  gelten und geben Sie die Schleifeninvariante an. Begründen Sie Ihre Antwort.
- 6.2. Führen Sie für diesen Algorithmus eine Verifikation durch, wobei die einzelnen Beweisschritte mit den Regeln der axiomatischen Semantik ausführlich zu beschreiben und zu begründen sind.

## 7. Rekursion und Iteration

- 7.1. Es gibt zwei Möglichkeiten die Fakultät  $F : \mathbb{N}^+ \rightarrow \mathbb{N}^+$ , mit  $F(n) = \prod_{i=1}^n i$  zu berechnen.

Zum einen kann sie iterativ und zum anderen rekursiv berechnet werden. Geben Sie für jede der Möglichkeiten jeweils eine Methode in einer höheren Programmiersprache an.

- 7.2. In der Linearen Algebra ist die Determinante eine Funktion, die jeder quadratischen Matrix eine Zahl zuordnet. Zum Beispiel hat die  $2 \times 2$ -Matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \text{ die Determinante } \det(A) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

Allgemein wird die Determinante einer  $n \times n$  Matrix berechnet, indem aus einer gewählten Zeile  $i$  jedem Element  $a_{i,1}, \dots, a_{i,n}$  eine Untermatrix  $A_{ij}$  gebildet wird. Die Untermatrix  $A_{ij}$  entsteht aus  $A$  durch Streichen der  $i$ -ten Zeile und  $j$ -ten Spalte. Für die Determinante von  $A$  gilt:  $\det(A) = \sum_{j=1}^n (-1)^{i+j} \cdot a_{ij} \cdot \det(A_{ij})$ .

Schreiben Sie eine rekursive Funktion in einer höheren Programmiersprache, welche die Determinante der Matrix  $A$  bestimmt. Die Funktionsdeklaration könnte in C wie folgt aussehen: `float det (float[ ][ ] Matrix, int dimension)`

## 8. Listen, Bäume, Komplexität

- 8.1. Nennen Sie den Aufwand von sortierten Listen, balancierten Suchbäumen und sortierten Arrays in Bezug zueinander bei der Speicherung von  $n$  Elementen für die Operationen Element finden und Element einfügen bzw. löschen. Geben Sie ebenfalls den zusätzlichen Speicherbedarf für die Verwaltung der Datenstrukturen an.

Kriterium	Listen	Bäume	Arrays
Element finden			
Element einfügen bzw. löschen			
zusätzlicher Speicherbedarf			

- 8.2. Gesucht ist ein nicht notwendigerweise balancierter binärer Suchbaum, bei welchem in jedem Knoten eine Zahl  $n \in \mathbb{N}$  gespeichert ist. Für jeden Knoten gilt, dass alle Knoten, welche an seinem linken (rechten) Ast hängen, kleinere (größere) Elemente als  $n$  gespeichert haben. Ferner gibt es in dem Suchbaum keine doppelten Elemente  $n$ . Geben Sie eine Methode `insert ( n )` zum Einfügen und eine Methode `search ( n )` zum Suchen eines Elementes an. Fügen Sie mit der Methode `insert ( n )` die Elemente 5, 14, 2, 8, 14, 7 in einen leeren Baum ein und geben Sie den Baum nach jedem Einfügen eines Elementes an.
- 8.3. Im Weinkeller eines grausamen Königs befinden sich  $n$  wertvolle Weinflaschen. Seine Wächter haben einen Hexer gefangen genommen, der genau eine Flasche vergiftet hat. Unglücklicherweise wissen sie nicht welche. Das Gift ist jedoch so stark, dass man sogar dann sterben würde, wenn man den Wein aller Flaschen vermischt und davon kostet. Allerdings wirkt das Gift so langsam, dass man erst einen Monat später daran erkrankt. Mit welcher Methode könnte der König innerhalb eines Monats feststellen, welche Flasche vergiftet ist und dabei höchstens  $O(\log n)$  Vorkosten einsetzen?

**Thema Nr. 2**Teilaufgabe I

Wir fixieren das Alphabet  $\Sigma = \{0,1\}$  und definieren  $L \subseteq \Sigma^*$  durch  $L = \{w \mid \text{in } w \text{ kommt genau einmal das Teilwort } 0010 \text{ vor}\}$

1. Zeigen Sie, dass  $L$  regulär ist!
2. Geben Sie die Äquivalenzklassen der Myhill-Nerode Äquivalenz von  $L$  durch Repräsentanten an. (Diese Äquivalenz ist definiert durch  $x \sim_L y \Leftrightarrow \forall u. xu \in L \Leftrightarrow yu \in L$ )
3. Zeichnen Sie den Minimalautomaten für  $L$ .

Teilaufgabe II

Sei  $\Sigma_n$  das Alphabet  $\{0,1\}^n$ . Ein Buchstabe in  $\Sigma_n$  ist also ein  $n$ -Tupel von 0en und 1en; das Alphabet  $\Sigma_n$  hat  $2^n$  Buchstaben. Ist  $w \in \Sigma_n^*$  so bezeichne  $w_1$  das Wort über  $\Sigma$ , das aus den ersten Komponenten der Buchstaben in  $w$  besteht, formal also  $w_1 = f(w)$  für den durch  $f((x_1, \dots, x_n)) = x_1$  definierten Homomorphismus  $f$ .

Analog definiert man  $w_2, w_3$ , etc. Also gilt für  $w = (0,1,1)(1,1,0)(0,0,1)(1,1,1)$ , dass  $w_1 = 0101, w_2 = 1101, w_3 = 1011$ . Für  $w \in \Sigma_1^*$  bezeichne  $\text{num}(w) \in \mathbb{N}$  die Bedeutung von  $w$  aufgefasst als von rechts nach links gelesene Binärdarstellung. Also  $\text{num}(10110000) = 13$ . Die "Inverse" (bis auf Nullen am Ende) von  $\text{num}$  bezeichnen wir mit  $\text{bin}$ , also  $\text{bin}(19) = 11001$ .

1. Zeigen Sie, dass  $L = \{w : \Sigma_3^* \mid \text{num}(w_3) = \text{num}(w_1) + \text{num}(w_2)\}$  regulär ist.

2. Für  $u \in \Sigma^*$  und  $v \in \Sigma_n^*$  sei  $(u, v) \in \Sigma_{n+1}^*$  das durch

$$\begin{aligned}|(u, v)| &= \max(|u|, |v|) \\ \text{num}((u, v)_1) &= \text{num}(u) \\ \text{num}((u, v)_{i+1}) &= \text{num}(v_i)\end{aligned}$$

eindeutig definierte Wort.

Sei  $L \subseteq \Sigma_{n+1}^*$  regulär. Es sei

$$L' = \{w \in \Sigma_n^* \mid \text{es existiert } n \in \mathbb{N}, \text{ sodass } (\text{bin}(n), w) \in L\}$$

Zeigen Sie, dass  $L'$  regulär ist.

3. Programmieren Sie die vier Funktionen  $\text{bin}$ ,  $\text{num}$ ,  $w \mapsto w_i$  und  $u, v \mapsto (u, v)$  in einer funktionalen Programmiersprache Ihrer Wahl oder Pseudocode. Zur Beachtung: das möglicherweise erforderliche Auffüllen von  $v$  mit Nullen bei der Berechnung von  $(u, v)$  beinhaltet eine gewisse Schwierigkeit, da  $v \in \Sigma_n^*$  für beliebiges  $n$ . Es bietet sich an, die Tupel als Listen zu implementieren. Programmieren Sie auch eine boole'schwertige Funktion, die die Zugehörigkeit zur in 1) definierten Sprache prüft.

**Teilaufgabe III**

Es soll ein Mühlespiel programmiert werden.

Die beiden Spieler sollen abwechselnd mit der Maus Züge wählen; das Programm soll überprüfen, ob ein Zug erlaubt ist und das Ergebnis des Zuges auf dem Bildschirm anzeigen.

1. Skizzieren Sie einen objektorientierten Entwurf für diese Aufgabenstellung in Form eines Klassendiagramm (in UML o.ä.). Ihr Diagramm sollte nicht weniger als fünf Klassen haben und neben den Beziehungen der Klassen untereinander auch deren wichtigste Methoden und Attribute beinhalten.
2. Das Programm muss mehrere Zustände haben, aus denen hervorgeht, welcher Spieler am Zug ist, ob die Maus gedrückt wurde, ob das Spiel zu Ende ist, etc. Modellieren Sie diese Programmzustände als Ablaufdiagramm.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2006**

Kennzahl: \_\_\_\_\_

**Herbst**

Kennwort: \_\_\_\_\_

**2006**

Arbeitsplatz-Nr.: \_\_\_\_\_

**66115**

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**  
**- Prüfungsaufgaben -**

Fach: **Informatik (vertieft studiert)**Einzelprüfung: **Theoretische Informatik, Algorithmen**Anzahl der gestellten Themen (Aufgaben): **2**Anzahl der Druckseiten dieser Vorlage: **7**

**Thema Nr. 1**

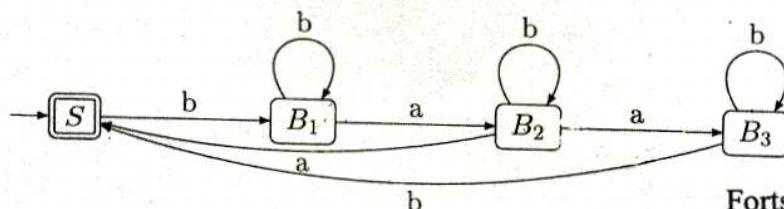
**Automatentheorie**

Anmerkungen zur Notation: Sei  $T(A)$  die von einem endlichen Automaten  $A = (Z, \Sigma, \delta, S, E)$  erzeugte Sprache. Dann wird  $\hat{\delta} : P(Z) \times \Sigma^* \rightarrow P(Z)$  wie folgt induktiv definiert:

$$\hat{\delta}(Z', \varepsilon) = Z' \text{ für alle } Z' \subseteq Z$$

$$\hat{\delta}(Z', ax) = \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x)$$

Sei nun der nichtdeterministische endliche Automat (NFA)  $N = (\{S, B_1, B_2, B_3\}, \{a, b\}, \gamma, \{S\}, \{S\})$  mit folgender Überführungsfunktion  $\gamma$  gegeben:



Fortsetzung nächste Seite!

- a) Bestimmen Sie die Menge  $\hat{\delta}(\{S\}, baabb)$ . Gilt  $baabb \in T(N)$ ?
- b) Konstruieren Sie einen deterministischen endlichen Automaten (DFA)  $M$ , der die Bedingung  $T(N) = T(M)$  erfüllt!
- c) Geben Sie einen regulären Ausdruck für  $T(N)$  an!
- d) Bilden Sie zu  $N$  eine reguläre Grammatik mit  $L(G) = T(N)$ !

## Formale Sprachen

Gegeben sei eine Grammatik für bedingte Anweisungen, deren Produktionsmenge unter anderem Folgendes enthält:

$$\begin{aligned}\text{Anweisung} &::= \text{if-Anweisung} \mid \text{Andere-Anweisung} \\ \text{if-Anweisung} &::= \underline{\text{if}} \text{ Bedingung } \underline{\text{then}} \text{ Anweisung} \mid \\ &\quad \underline{\text{if}} \text{ Bedingung } \underline{\text{then}} \text{ Anweisung } \underline{\text{else}} \text{ Anweisung}\end{aligned}$$

Terminale sind dabei unterstrichen. Die Nicht-Terminale *Andere-Anweisung* und *Bedingung* können mit dem gegebenen Ausschnitt der Grammatik nicht weiter abgeleitet werden. Deuten Sie daher in den Syntaxbäumen die entsprechenden Teilbäume mit einem Dreieck an!

- a) Zeigen Sie, dass die Grammatik mehrdeutig ist, indem Sie Syntaxbäume angeben!
- b) Wählen Sie logische Ausdrücke und Anweisungen für die Syntaxbäume aus der vorherigen Teilaufgabe so, dass die Ausführung der zugeordneten Programme zu verschiedenen Ergebnissen führt!
- c) Wie kann diese Mehrdeutigkeit aufgelöst werden?
- d) Geben Sie Produktionsregeln an, die Ihre Lösung aus Aufgabenteil (c) realisieren, so dass die Grammatik eindeutig wird!

## Berechenbarkeit

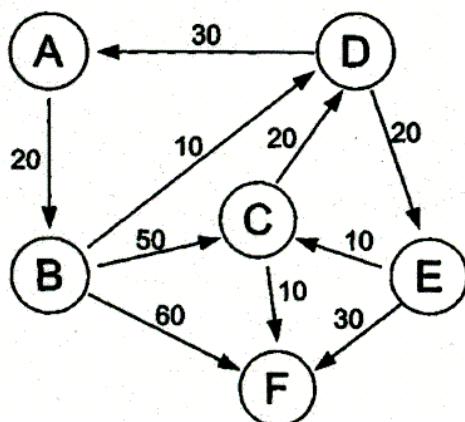
Untersuchen Sie, ob die folgenden Mengen und Sprachen entscheidbar bzw. semi-entscheidbar sind! Begründen Sie jeweils Ihre Antwort!

- a)  $f^{-1}(A) = \{n \in \mathbb{N} \mid f(n) \in A\}$ , wobei  $A \subseteq \mathbb{N}$  eine entscheidbare Menge und  $f : \mathbb{N} \rightarrow \mathbb{N}$  eine totale und berechenbare Funktion ist.
- b)  $L_1 \setminus L_2$ , wobei  $L_1 \subseteq \mathbb{N}$  eine semi-entscheidbare und  $L_2 \subseteq \mathbb{N}$  eine entscheidbare Sprache ist.

## Algorithmen und Datenstrukturen

- a) Gegeben sei die folgende Folge ganzer Zahlen: 6, 13, 4, 8, 11, 9, 10.  
 ➤ Fügen Sie obige Zahlen der Reihe nach in einen anfangs leeren AVL-Baum ein und stellen Sie den Baum nach jedem Einfügeschritt dar!  
 ➤ Löschen Sie das Wurzelement des entstandenen AVL-Baums und stellen Sie die AVL-Eigenschaft wieder her!

- b) Gegeben sei der folgende gerichtete und gewichtete Graph:



- Bestimmen Sie mit Hilfe des *Algorithmus von Dijkstra* die kürzesten Wege vom Knoten A zu allen anderen Knoten! Geben Sie dabei nach jedem Verarbeitungsschritt den Zustand der Hilfsdatenstruktur an!
  - Skizzieren Sie einen Algorithmus für den Tiefendurchlauf von gerichteten Graphen, wobei jede Kante nur einmal verwendet werden darf!
- c) Ein wesentlicher Nachteil der Standardimplementierung des **QUICKSORT** Algorithmus ist dessen rekursiver Aufruf.
- Implementieren Sie den Algorithmus **QUICKSORT** ohne den rekursiven Prozederaufruf!

## Systementwurf

- a) Erklären Sie den Begriff Vererbung und benennen Sie die damit verbundenen Vorteile!
- b) Erstellen Sie zu der folgenden Beschreibung eines Systems zur Buchung von Flügen ein Klassendiagramm, das neben Attributen und Assoziationen mit Kardinalitäten auch Methoden zur Tarifberechnung enthält! Setzen Sie dabei das Konzept der Vererbung sinnvoll ein!
  - Die Fluggesellschaft bietet verschiedene Flugrouten an, die durch den jeweiligen Startflughafen und Zielflughafen charakterisiert werden.
  - Jeder Flug besitzt eine Flugnummer, eine Abflugzeit, eine geplante Ankunftszeit und ist genau einer Flugroute zugeordnet. Flugrouten sollen auch gespeichert werden, falls noch keine zugehörigen Flüge existieren.
  - Flugbuchungen beziehen sich auf einzelne Plätze im Flugzeug. Sowohl in der Economy Class als auch in der Business Class gibt es Nichtraucher- und Raucherplätze. Zu jeder Buchung wird das Datum vermerkt.
  - Zu jedem Passagier müssen die Adressinformationen erfasst werden.
  - Die Berechnung des Tarifs soll vom System unterstützt werden. Jeder Flug besitzt einen Grundpreis. Für Plätze der Business Class wird ein Aufschlag verrechnet. Auf diesen ermittelten Zwischenpreis sind zwei Arten von Rabatten möglich:
    - Jugendliche Privatkunden unter 25 Jahren erhalten einen Nachlass auf den Flugpreis.
    - Geschäftsreisende erhalten Vergünstigungen in Abhängigkeit ihrer gesammelten Flugmeilen.

- c) Erstellen Sie ein exemplarisches Objektdiagramm! Es soll mindestens einen Flug enthalten, in dem sowohl ein privater Kunde als auch ein Geschäftskunde einen Platz gebucht haben! Wählen Sie geeignete Attributwerte!
- d) Beschreiben Sie den Vorgang „Tarifberechnung“ wahlweise als Sequenzdiagramm oder Kommunikationsdiagramm!

## Thema Nr. 2

### Aufgabe 1

Gegeben sei ein Variablenalphabet  $V = \{A, B\}$  und ein Terminalalphabet  $T = \{a, b\}$ . Es sei  $G_1 = (V, T, P_1, A)$  die kontextfreie Grammatik mit den Produktionen

$$P_1 = \{A \rightarrow aAb | bB, B \rightarrow bB | aB | \lambda\}$$

und  $G_2 = (V, T, P_2, A)$  die kontextfreie Grammatik mit den Produktionen

$$P_2 = \{A \rightarrow aAb | Ba, B \rightarrow bB | aB | \lambda\},$$

wobei  $\lambda$  für das leere Wort steht

- a) Welches sind die von  $G_1$  bzw.  $G_2$  generierten Sprachen  $L(G_1)$  bzw.  $L(G_2)$ ? Geben Sie Beschreibungen von  $L(G_1)$  bzw.  $L(G_2)$ , die nicht auf die Grammatiken  $G_1$  bzw.  $G_2$  Bezug nehmen! Beweisen Sie Ihre Behauptungen!
- b) Zeigen Sie, dass die kontextfreien Sprachen  $L(G_1)$  und  $L(G_2)$  nicht regulär sind!
- c) Welches sind die Sprachen  $L(G_1) \cup L(G_2)$  und  $L(G_1) \cap L(G_2)$ ?
- Welche dieser Sprachen ist regulär?

### Aufgabe 2

Es sei  $\Sigma = \{0, 1\}$ . Das Alphabet  $\Sigma_2$  bestehe aus allen Paaren von Elementen aus  $\Sigma$ , geschrieben als Spaltenvektoren der Länge 2 über  $\Sigma$ , also

$$\Sigma_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}.$$

Ein Wort  $w = w_1 w_2 \dots w_n \in \Sigma_2^n$  mit  $w_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}, 1 \leq i \leq n$ , kann aufgefasst werden als ein Paar  $(x, y) \in \Sigma^n \times \Sigma^n$  mit  $x = x_1 x_2 \dots x_n$ ,  $y = y_1 y_2 \dots y_n$ , d.h.

$$w = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \dots \begin{bmatrix} x_n \\ y_n \end{bmatrix}.$$

- a) Jedes Wort  $a = a_1 a_2 \dots a_{n-1} a_n \in \Sigma^n$  stellt die natürliche Zahl

$$\text{bin}(a) = a_1 \cdot 2^{n-1} + a_2 \cdot 2^{n-2} + \dots + a_{n-1} \cdot 2 + a_n$$

dar (binäre Zahldarstellung).

Die Sprache des Größenvergleichs ist

$$\text{LEQ} := \left\{ w = \begin{bmatrix} x \\ y \end{bmatrix} \in \Sigma_2^*; \text{bin}(x) \leq \text{bin}(y) \right\}.$$

Es gilt also beispielsweise

$$\begin{bmatrix} 0101 \\ 0110 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \in \text{LEQ}, \quad \begin{bmatrix} 0110 \\ 0101 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \notin \text{LEQ}.$$

Zeigen Sie, dass die Sprache LEQ regulär ist!

Hinweis:

Sie können - falls Ihnen das hilfreich erscheint - hier die Tatsache verwenden, dass eine Sprache  $L$  genau dann regulär ist, wenn die gespiegelte Sprache  $L^R = \{w^R; w \in L\}$  regulär ist. Dabei ist  $(w_1 w_2 \dots w_{n-1} w_n)^R = w_n w_{n-1} \dots w_2 w_1$ .

- b) Zeigen Sie, dass die Sprache

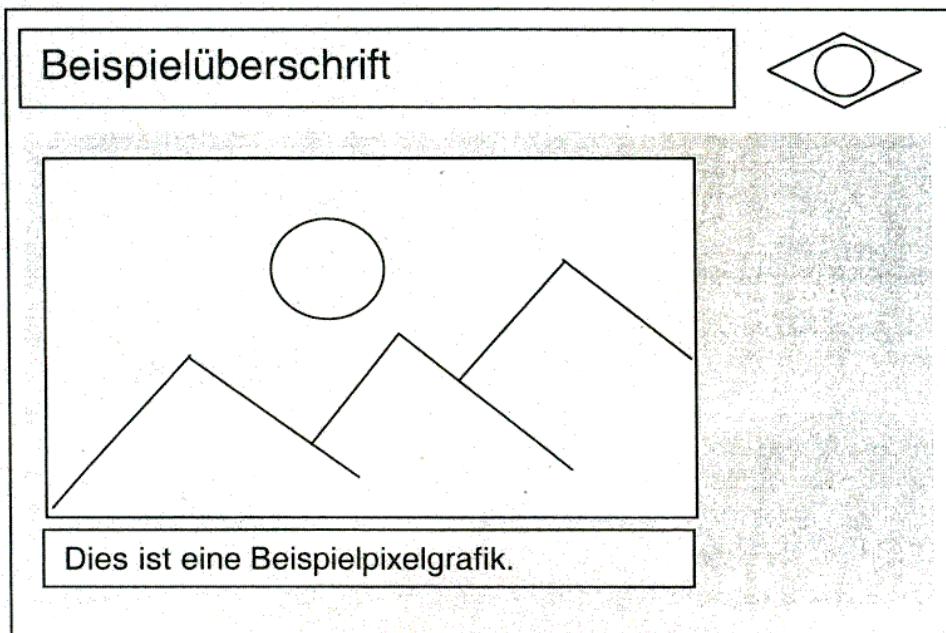
$$\left\{ w = \begin{bmatrix} x \\ y \end{bmatrix} \in \Sigma_2^*; y = x^R \right\}$$

nicht regulär ist!

### Aufgabe 3

Im Folgenden sollen Zusammenhänge innerhalb einer Präsentationssoftware als Klassendiagramm dargestellt werden. Folgende Zusammenhänge sind zu berücksichtigen: Eine Präsentation besteht aus mehreren Folien (mindestens aus einer), die sich in einer bestimmten Reihenfolge befinden. Eine Folie kann Textboxen sowie einfache Vektor- und Pixelgrafiken enthalten. Textboxen enthalten Text und haben eine Position sowie eine Breite und eine Höhe. Sie können eine Füllfarbe haben und umrandet sein, in diesem Fall kann eine Randstärke und eine Randfarbe angegeben werden. Für den enthaltenen Text kann die Schriftart und die Schriftgröße erfasst werden. Vektorgrafiken können aus Linien, Rechtecken und Kreisen zusammengesetzt werden. Für diese können jeweils die Position, die Linienstärke und Linienfarbe sowie ggf. eine Füllfarbe erfasst werden. Pixelgrafiken liegen als externe Dateien vor und werden an einer bestimmten Position mit einer gegebenen Höhe und einer Breite eingebunden. Die Elemente auf einer Folie werden in einer bestimmten Reihenfolge gezeichnet, um bestimmte Elemente im Vordergrund andere Elemente im Hintergrund verdecken zu lassen. Jeder Folie ist ein Folienmaster zugeordnet, auf dem immer wiederkehrende Elemente (z.B. ein Logo) zur Erstellung eines einheitlichen Layouts für mehrere Folien erfasst werden können. Dabei kann ein Folienmaster prinzipiell dieselben Elemente enthalten, wie eine normale Folie. Eine Präsentation kann ggf. mehrere Folienmaster verwenden. Alle Elemente auf einer Folie (Textboxen, Rechtecke etc.) können zu Gruppen zusammengefasst werden. Gruppen können ihrerseits weitere Gruppen sowie die genannten Folienelemente enthalten.

- a) Erstellen Sie für die beschriebenen Zusammenhänge ein UML-Klassendiagramm! Um eine Präsentation vorführen zu können, müssen verschiedene Elemente der Folie über eine Operation darstellen zur Darstellung verfügen. Ergänzen Sie diese! Jede Klasse soll mindestens über ein Attribut verfügen. Spezifizieren Sie alle Attribute, Operationen und Beziehungen durch Angabe von Datentypen, Beziehungstypen und -namen sowie Kardinalitäten!
- b) Gegeben sei folgende Folie:



Die Folie enthält eine Überschrift. Im Hauptteil befindet sich eine Pixelgrafik mit einer Bildunterschrift. Grafik und Bildunterschrift bilden eine Gruppe. Der Folienmaster enthält ein hellgraues Rechteck als Hintergrund für den Hauptteil (ohne Überschrift) und ein Logo (Pixelgrafik) in der oberen rechten Ecke.

Geben Sie zu dieser Folie und dem zugehörigen Folienmaster ein UML-Objektdiagramm an!

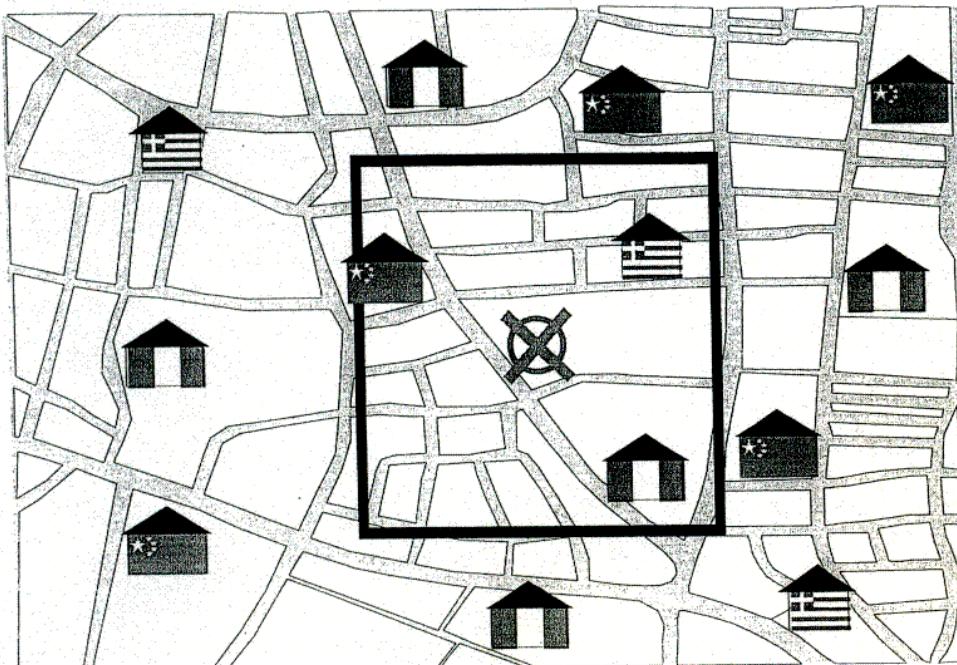
Hinweis:

Gehen Sie von den üblichen Folienmaßen aus: Höhe=21cm, Breite=29,7cm aus. Alle Angaben für Breiten, Höhen und Positionen von Folienelementen dürfen Sie schätzen, keinesfalls müssen diese maßstabsgetreu sein! Das Koordinatensystem habe in der oberen linken Ecke seinen Nullpunkt, Punkte im Inneren des Folienbereichs haben positive x- und y-Koordinaten.

- c) Die Folie aus Aufgabe b soll präsentiert (dargestellt) werden. Zeichnen Sie für diese Situation ein UML-Sequenzdiagramm!

#### Aufgabe 4

Ein Navigationssystem soll folgenden Service anbieten. Ausgehend von einem aktuellen Standort eines Anfragenden sollen alle Restaurants einer bestimmten Küchenrichtung (z. B. italienisch, chinesisch) ausgegeben werden, die sich innerhalb eines quadratischen Bereichs einer anzugebenden Größe um diesen befinden (s. Abbildung):



Standort



Chinese



Italiener



Griechen

Gehen Sie vereinfachend davon aus, dass sowohl der Standort, als auch alle Restaurants jeweils mit ihren  $(x, y)$ -Koordinaten vorliegen, wobei  $0 \leq x \leq x_{\max}$  und  $0 \leq y \leq y_{\max}$  gelten soll. Verwenden Sie zur Formulierung von Algorithmen bzw. Datentypen eine gängige höhere Programmiersprache oder einen entsprechenden Pseudocode! Erläutern Sie Ihre Lösung ausgiebig durch Kommentare!

- Geben Sie einen geeigneten Datentyp zur Verwaltung der Restaurants an! Zusätzlich zur Lage  $((x, y)$ -Koordinaten) soll der Name, die Adresse, die Telefonnummer und die Küchenrichtung angegeben werden!
- Geben Sie einen Algorithmus an, der als Eingabe einen Standort in  $(x, y)$ -Koordinaten, eine Bereichsgröße und eine bevorzugte Küchenrichtung erhält und der als Ergebnis eine Datenstruktur liefert, die alle Restaurants dieser Richtung innerhalb eines achsenparallelen, quadratischen Bereichs um den Standort enthält!

Lösungshinweis:

Eine mögliche Strategie besteht darin, zunächst nur die Restaurants mit passender  $x$ -Koordinate zu identifizieren und aus diesen diejenigen mit passender  $y$ -Koordinate auszuwählen.

- Geben Sie die Laufzeit Ihres Verfahrens in  $O(n)$ -Notation an und begründen Sie Ihr Ergebnis!

## Aufgabe 5

- Geben Sie einen rekursiven Algorithmus vom Typ „Teile und Herrsche“ zum Zeichnen einer Approximation des Geradenabschnitts an, welcher zwei gegebene Punkte  $(x_1, y_1)$  und  $(x_2, y_2)$  verbindet, indem Pixel mit ganzzahligen Koordinaten gezeichnet werden! Dabei soll der erste zu zeichnende Punkt etwa in der Mitte zwischen den beiden gegebenen Punkten liegen. Wann kann die Rekursion abgebrochen werden? Gehen Sie davon aus, dass zum Zeichnen eines Pixels eine Operation `zeichne` mit geeigneten Parametern zur Verfügung steht. Verwenden Sie (auch im Teil c.) zur Formulierung eine gängige höhere Programmiersprache oder einen entsprechenden Pseudocode. Erläutern Sie Ihre Lösung ausgiebig durch Kommentare!
- Warum sollte man dieses Problem nicht mit einem Algorithmus vom Typ „Teile und Herrsche“ lösen?
- Geben Sie eine bessere Lösung für die unter a.) beschriebene Aufgabenstellung an!

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2007**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	<b>Frühjahr</b>	
Kennwort: _____	<b>2007</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

---

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**

**— Prüfungsaufgaben —**

---

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

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

Anzahl der Druckseiten dieser Vorlage: **5**

---

Bitte wenden!

Thema Nr. 1**Aufgabe 1:**

Gegeben sei das Alphabet  $\Sigma = \{a, b\}$  und die Sprache  $L_1 = \{waba | w \in \Sigma^*\}$

- Zeigen Sie, dass  $L_1$  regulär ist.
- Geben Sie einen vollständigen deterministischen endlichen Automaten an, der die Sprache  $L_1$  akzeptiert.
- Konstruieren Sie einen minimalen deterministischen Automaten, der  $L_1$  akzeptiert und weisen Sie dessen Minimalität nach.

Gegeben sei weiter die Sprache  $L_2 = \{ww^Raba | w \in \Sigma^*\}$ .

- Stellen Sie eine geeignete Grammatik G auf, die die Sprache  $L_2$  erzeugt, und geben Sie eine Ableitung der Wörter aba und aabbaaaba in G an.
- Ist  $L_2$  regulär bzw. kontextfrei? Begründen Sie Ihre Antworten.

**Aufgabe 2:**

- Die loop-Anweisung kann in WHILE-Programmen durch while-Anweisungen ersetzt werden. Beweisen Sie diese Aussage für die folgende loop-Anweisung:  
**loop a do P enddo** (P sei ein beliebiges WHILE-Programm.)
- Terminieren GOTO-Programme immer? Begründen Sie Ihre Antwort.
- Geben Sie eine Turingmaschine  $M = (Z, \{|, 0\}, \Gamma, \delta, z_0)$  mit einem Band an, die überprüft, ob die Anzahl der 0-Symbole im Eingabewort gerade ist.

**Aufgabe 3:**

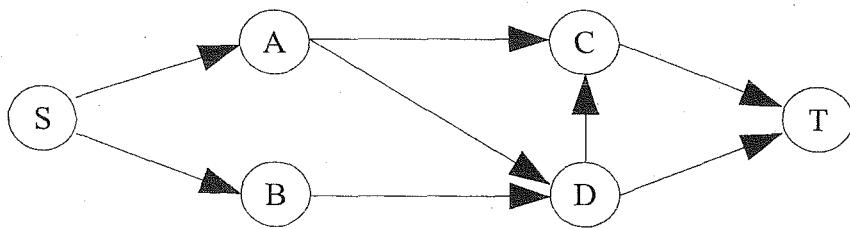
Die Multiplikationsfunktion für zwei ganze Zahlen  $n, m$  kann durch fortgesetzte Addition realisiert werden:

$$n \cdot m = \text{mult}(n, m) = \begin{cases} m + m \cdot (n - 1) : & \text{falls } n > 0 \\ 0 : & \text{sonst} \end{cases}$$

Geben Sie eine **iterative** und eine **rekursive** Implementierung der Multiplikationsfunktion in einer imperativen oder objektorientierten Programmiersprache an. Verwenden Sie dabei nicht den Multiplikationsoperator dieser Programmiersprache.

**Aufgabe 4:**

Repräsentieren Sie den folgenden Graphen mit Hilfe einer Adjazenzmatrix und einer Adjazenzliste.

**Aufgabe 5:**

Fügen Sie nacheinander die Schlüssel 7, 20, 30, 2, 14, 18, 9, 12, 4, 6, 16 in einen anfangs leeren binären Suchbaum ein. Zeichnen Sie den kompletten Baum.

**Aufgabe 6:**

Gegeben seien die Zahlen 2, 3, 8, 5, 6, 4, 1, 7. Sortieren Sie diese in aufsteigender Reihenfolge mit dem Sortierverfahren *Mergesort*. Geben Sie alle Zwischenschritte in einer Tabelle an, in der Sie die durch Aufteilung entstehenden Teilfolgen durch senkrechte Striche trennen.

**Aufgabe 7:**

Implementieren Sie die angegebenen Methoden einer Klasse Queue für Warteschlangen. Eine Warteschlange soll eine unbeschränkte Anzahl von Elementen aufnehmen können. Elemente sollen am Ende der Warteschlange angefügt und am Anfang aus ihr entfernt werden.

Sie können davon ausgehen, dass eine Klasse QueueElement mit der folgenden Schnittstelle bereits implementiert ist.

```

class QueueElement {
    QueueElement(Object contents);
    Object getContents();
    QueueElement getNext();
    void setNext(QueueElement next);
}
  
```

Von der Klasse Queue ist folgendes gegeben:

```

class Queue {
    QueueElement first;
    QueueElement last;
}
  
```

- a) Schreiben Sie eine Methode void append (Object contents), die ein neues Objekt in die Warteschlange einfügt.
- b) Schreiben Sie eine Methode Object remove (), die ein Element aus der Warteschlange entfernt und dessen Inhalt zurückliefert. Berücksichtigen Sie, dass die Warteschlange leer sein könnte.
- c) Schreiben Sie eine Methode boolean isEmpty (), die überprüft, ob die Warteschlange leer ist.

**Aufgabe 8:**

Gegeben sei folgende Funktion zur Berechnung der Quadratwurzel einer positiven, reellen Zahl  $a \geq 1$ : Es lässt sich zeigen, dass bei Terminierung der while-Schleife  $x = y$  gilt. Diese Tatsache

```
double heron(double a) {
    double x, y;
    x = a; y = 1;
    while (x>y) {
        x = (x+y)/2;
        y = a/x;
    }
    return x;
}
```

können Sie für Ihre Lösung ausnutzen.

- a) Bestimmen Sie eine Vorbedingung, eine Nachbedingung und die Invariante der while-Schleife.
- b) Begründen Sie, warum aus der Invariante und der Nachbedingung folgt, dass der Algorithmus tatsächlich die Quadratwurzel von  $a$  berechnet.

Thema Nr. 2**Aufgabe 1:**

- a) Beschreiben Sie in Pseudocode oder einer Programmiersprache Ihrer Wahl einen *Greedy*-Algorithmus, der einen Betrag von  $n$  Cents mit möglichst wenigen Cent-Münzen herausgibt. Bei  $n = 29$  wäre die erwartete Antwort etwa  $1 \times 20ct, 1 \times 5ct, 2 \times 2ct$ .
- b) Beweisen Sie die Korrektheit Ihres Verfahrens, also dass tatsächlich die Anzahl der Münzen minimiert wird.
- c) Nehmen wir an, Bayern führe eine Sondermünze im Wert von  $7ct$  ein. Dann liefert der naheliegende Greedy-Algorithmus nicht immer die minimale Zahl von Münzen. Geben Sie für dieses Phänomen ein konkretes Beispiel an und führen Sie aus, warum Ihr Beweis aus Aufgabenteil a) in dieser Situation nicht funktioniert.

**Aufgabe 2:**

Sei  $\Sigma$  endliches Alphabet. Sei  $L \subseteq \Sigma^*$  eine reguläre Sprache über  $\Sigma$ . Die Sprache  $L^{rev}$  besteht aus allen Wörtern der Form  $a_1 \dots a_n$ , wobei  $a_n \dots a_1 \in L$  ist, also die Menge aller Wörter die, wenn von rechts nach links gelesen, in  $L$  sind.

Die Sprache  $L^{suf}$  bezeichnet alle Wörter  $w$ , deren sämtliche Suffixe in  $L$  sind; für die also gilt: wenn immer  $w = uv$ , so folgt  $v \in L$ .

Zeigen Sie, dass auch  $L^{rev}$  und  $L^{suf}$  regulär sind.

**Aufgabe 3:**

Sei  $\Sigma = \{0, 1, 2, \dots, t\}$  und  $L \subseteq \Sigma^*$  beliebige kontextfreie Sprache. Sei  $L^{ord}$  die Menge der absteigend geordneten Wörter aus  $L$ , das sind diejenigen Wörter  $a_1 \dots a_n \in L$  für die gilt  $a_1 \geq a_2 \geq \dots \geq a_n$  an. Zeigen Sie, dass  $L^{ord}$  kontextfrei ist.

**Aufgabe 4:**

Das Partyveranstaltungsproblem ist das folgende. Gegeben ist eine Menge  $B$  von Bekannten und eine Menge  $H \subseteq B \times B$  von (Paaren von) Bekannten, die sich nicht leiden können. Es ist festzustellen, ob eine Auswahl  $U \subseteq B$  von  $k$  Bekannten ( $|U| = k$ ) existiert (die Gästeliste), sodass in dieser keine zwei Bekannten enthalten sind, die sich nicht leiden können. Man zeige, dass das Partyveranstaltungsproblem NP-vollständig ist.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2007**

---

Prüfungsteilnehmer

Prüfungstermin

Einzelprüfungsnummer

---

Kennzahl: \_\_\_\_\_

**Herbst  
2007**

**66115**

Kennwort: \_\_\_\_\_

Arbeitsplatz-Nr.: \_\_\_\_\_

---

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen  
— Prüfungsaufgaben —**

---

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

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

Anzahl der Druckseiten dieser Vorlage: **5**

---

Bitte wenden!

Thema Nr. 1Hinweis:

Die einzelnen Teilaufgaben bauen oftmals aufeinander auf, sind aber im Prinzip in beliebiger Reihenfolge lösbar. Sie dürfen hierbei die Angaben und Ergebnisse früherer Teilaufgaben uneingeschränkt zur Lösung späterer Teilaufgaben verwenden! Außerdem dürfen Sie Tatsachen aus dem Informatik-Duden ohne weitere Begründung als bekannt voraussetzen.

**Aufgabe 1:**

Ordnen Sie die folgenden formalen Sprachen bestmöglich in die Chomsky-Hierarchie ein und geben Sie eine *ausreichende Begründung* an:

- $L_1 = \{a^n b^n a^n \mid n \geq 1\}$
- $L_2$  sei die Menge aller terminierenden Java-Programme.
- $L_3$  sei die Menge aller *vollständig und korrekt geklammerten* arithmetischen Ausdrücke in den Variablen  $a$  und  $b$  mit den Operationen  $+$  und  $\times$ .  
Zur Illustration:  $((a + (b + a)) \times a) \in L_3$ ,  $((a + b + a)) \times b \notin L_3$  (nicht korrekt geklammert),  $a \times (b + b)) \notin L_3$  (nicht vollständig geklammert).
- $L_4 = \{w \in \{a, b\}^* \mid w \text{ enthält mindestens } 4 \text{ Vorkommen von } a\}$ .
- $L_5 = \{a^n b^n \$ w \mid n \geq 1 \text{ und } w \in \{a, b\}^*\}$

**Aufgabe 2:**

Geben Sie zu dem nichtdeterministischen endlichen Automaten in der Abbildung einen äquivalenten deterministischen Automaten an.

Ist Ihr Automat minimal?

Falls nein, so geben Sie mindestens ein Paar von Zuständen an, die zu einem einzigen Zustand zusammengefasst werden können.

Falls ja, so geben Sie für mindestens drei Zustandspaare Ihrer Wahl jeweils eine Begründung dafür, dass diese nicht zusammengefasst werden können.

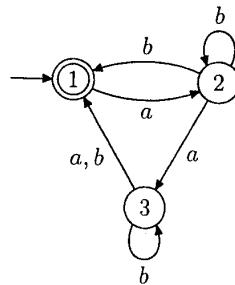


Abbildung: ein nichtdeterministischer endlicher Automat mit 3 Zuständen

**Aufgabe 3:**

Für eine (deterministische) Turingmaschine  $T = (I, \Sigma, Q, \delta, q_0, F, b)$  und ein Wort  $w \in \Sigma^*$  ist die partielle Funktion  $\text{TIME}_T(w)$  definiert als die Anzahl von Arbeitsschritten, die  $T$  bei Eingabe  $w$  ausführt. Falls  $T$  bei Eingabe  $w$  nicht hält, ist  $\text{TIME}_T(w) = \perp$ , also undefiniert. Im Folgenden sei  $\Sigma = \{0, 1\}$  fest; das leere Wort wird wie üblich mit  $\epsilon$  bezeichnet.

Die *Busy-Beaver Funktion*  $\text{BB}(n)$  ist definiert als

$$\text{BB}(n) := \max \{ \text{TIME}_T(\epsilon) \mid \begin{array}{l} \text{Turingmaschine } T \text{ hat höchstens } n \text{ Zustände und hält auf leerer} \\ \text{Eingabe } \} \}$$

- a) Das Halteproblem bei leerer Eingabe ist die Menge  $H_0 = \{T \mid \text{TIME}_T(\epsilon) \neq \perp\}$ . Bekanntlich ist  $H_0$  unentscheidbar. Geben Sie eine Reduktion des Graphen von  $\text{BB}$ , also der Menge  $G = \{(n, b) \mid b = \text{BB}(n)\}$  auf  $H_0$  an.
- b) Zeigen Sie durch Widerspruch:  $\text{BB}(n)$  wächst schneller als jede berechenbare Funktion, d. h. für jede berechenbare Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  gilt:  $\text{BB}(n) \notin O(f(n))$ .

**Aufgabe 4:**

Wrestling ist ein Showkampf, bei dem es zwei Arten von Teilnehmern gibt: Gute und böse Wrestler. Wer gut und wer böse ist, wird von den Organisatoren vorab festgelegt, die Wrestler haben sich dann entsprechend zu kleiden und zu benehmen.

Zwischen manchen Wrestlern bestehen persönliche Rivalitäten und um die Kämpfe zusätzlich anzuheizen, ist man bestrebt, die Einteilung in Gute und Böse so vorzunehmen, dass es keine Rivalitäten zwischen zwei Guten oder zwischen zwei Bösen gibt, sondern nur zwischen „Gut“ und „Böse“.

Helfen Sie dem Management, indem Sie einen effizienten Algorithmus entwerfen, der entscheidet, ob solch eine Einteilung existiert und sie ggf. berechnet. Gegeben ist hierbei eine Menge von  $W$  Wrestlern repräsentiert durch  $\{1, \dots, W\}$  und einer Liste von  $R$  Paaren einander rivalisierender Wrestler. „Effizient“ bedeutet hier, dass die Laufzeit  $O(W + R)$  sein muss. Beschreiben Sie Ihre Lösung in Pseudocode oder einer Programmiersprache Ihrer Wahl.

*Beispiel:*

Bei drei Wrestlern  $\{1, 2, 3\}$  und Rivalitäten zwischen 1,2 sowie 1,3 könnte man 1 als „gut“ und 2,3 jeweils als „böse“ einteilen. Besteht zusätzlich noch eine Rivalität zwischen 2 und 3, so existiert keine Lösung.

*Hinweis:*

Bauen Sie Ihren Algorithmus auf einem geeigneten Verfahren zum Durchlaufen von (ungerichteten) Graphen auf.

Thema Nr. 2**Aufgabe 1:**

Gegeben sei der nichtdeterministische endliche Automat  $M$  mit dem Alphabet  $\Sigma = \{a, b\}$ , der Zustandsmenge  $\{z_0, z_1, z_2, z_3\}$ , Anfangszustand  $z_0$ , Endzustand  $\{z_3\}$  und der Überführungsfunktion  $\delta$  mit:

$$\begin{aligned}\delta(z_0, a) &= \{z_1, z_2\}, \\ \delta(z_1, b) &= \{z_0, z_1\}, \\ \delta(z_2, a) &= \{z_2, z_3\}, \\ \delta(z_0, b) &= \delta(z_1, a) = \delta(z_2, b) = \delta(z_3, a) = \delta(z_3, b) = \emptyset\end{aligned}$$

$L(M)$  sei die von  $M$  akzeptierte Sprache.

a) Gelten folgende Aussagen?

- i) Es gibt Zeichenreihen in  $L(M)$ , die genauso viele  $a$ 's enthalten wie  $b$ 's.
- ii) Jede Zeichenreihe in  $L(M)$ , die mindestens vier  $b$ 's enthält, enthält auch mindestens vier  $a$ 's.

Begründen Sie Ihre Antworten.

- b) Geben Sie eine reguläre (Typ-3-)Grammatik an, die  $L(M)$  erzeugt.
- c) Beschreiben Sie  $L(M)$  durch einen regulären Ausdruck.
- d) Konstruieren Sie aus  $M$  mit der Potenzmengen-Konstruktion (und entsprechender Begründung) einen deterministischen endlichen Automaten, der  $L(M)$  akzeptiert.

**Aufgabe 2:**

Für beliebiges  $m \in \mathbb{N}$  sei  $L_m$  die Sprache  $L_m = \{a^i b^m a^i b^m \in \{a, b\}^* | i \in \mathbb{N}\}$ .

- a) Beweisen Sie:  $L_3$  ist nicht regulär.
- b) Ist  $L_m$  für jedes  $m \in \mathbb{N}$  nicht regulär? Begründen Sie Ihre Antwort.
- c) Geben Sie die allgemeine Form einer kontextfreien (Typ-2-)Grammatik an, die  $L_m$  (für beliebiges  $m$ ) erzeugt.
- d) Ist jede der Sprachen  $L_m$  mit einer deterministischen Turing-Maschine mit einer Zeitkomplexität  $O(n^2)$  entscheidbar ( $n$  ist die Länge der jeweiligen Eingabe)? Begründen Sie Ihre Antwort.

**Aufgabe 3:**

Es seien  $\Sigma$  ein Alphabet,  $L_1$  und  $L_2$  zwei Sprachen über  $\Sigma$ .  $\epsilon$  bezeichne die leere Zeichenreihe. Gelten folgende Aussagen? Begründen Sie Ihre Antworten.

- a) Ist  $L_1$  kontext-sensitiv (Typ-1), so ist die Sprache

$$L = \{w \in \Sigma^* | w \notin L_1 \text{ und } w \neq \epsilon\}$$

entscheidbar.

- b) Wird  $L_1$  von einem linear beschränkten Automaten  $M$  mit Zustandsmenge  $Z$  und Endzustandsmenge  $E$  akzeptiert, so akzeptiert der linear beschränkte Automat  $M'$ , der aus  $M$  entsteht, wenn man  $E$  durch  $Z \setminus E$  ersetzt, die Sprache  $\sum^* \setminus L_1$ .
- c) Sind  $L_1$  und  $L_2$  entscheidbar, so ist auch die Sprache

$$L_1 \circ L_2 = \{w_1 w_2 \in \sum^* \mid w_1 \in L_1, w_2 \in L_2\}$$

entscheidbar.

- d) Ist  $L_1$  entscheidbar und  $L_2$  semi-entscheidbar, so ist die Funktion  $f : \sum^* \rightarrow \sum^*$  mit

$$f(w) = \begin{cases} \epsilon & \text{falls } w \notin L_1 \cap L_2 \\ \text{undefiniert} & \text{sonst} \end{cases}$$

berechenbar.

- e) Sind sowohl  $L_1$  als auch  $L_2$  mit einer deterministischen Turing-Maschine mit polynomieller Zeitkomplexität entscheidbar, so gilt dies auch für  $L_1 \setminus L_2$ .

#### Aufgabe 4:

- a) Beschreiben Sie kurz allgemein die Wirkungsweise des Sortier-Algorithmus Heapsort.
- b) Welche Zeitkomplexität hat Heapsort? Begründen Sie Ihre Antwort.
- c) Beschreiben Sie konkret die einzelnen Schritte, die durchgeführt werden, wenn die Zahlenfolge

$$13, 8, 25, 3, 9, 20, 5, 21$$

mit Heapsort aufsteigend sortiert wird. Geben Sie dabei die jeweiligen Heapstrukturen sowohl als Baum als auch in ihrer üblichen Darstellung als Feld an.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2008**

---

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	Frühjahr	
Kennwort: _____	2008	66115
Arbeitsplatz-Nr.: _____		

---

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**  
— Prüfungsaufgaben —

---

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 6

---

Bitte wenden!

Thema Nr. 1**Aufgabe 1:****Reguläre Sprachen**

Sei  $\Sigma = \{a, b\}$  ein Alphabet. Wir betrachten die Sprache

$$L = \left\{ w_1 \cdots w_n \in \Sigma^* \mid n \geq 2 \wedge \exists 1 \leq i < n : w_n = w_i \wedge \forall 1 \leq i \leq n : w_i \in \Sigma \right\}$$

als die Sprache, die alle Wörter enthält, deren letzter Buchstabe vorher bereits schon einmal im Wort vorkam.

- Geben Sie für das Alphabet  $\Sigma = \{a, b\}$  einen nichtdeterministischen endlichen Automaten  $N_1$  an, der  $L$  erkennt.
- Übersetzen Sie den Automaten  $N_1$  in einen deterministischen endlichen Automaten, indem Sie die Potenzmengenkonstruktion durchführen.
- Geben Sie eine Chomsky-Typ-3-Grammatik  $G$  an, die  $L$  erzeugt.
- Zeigen Sie, dass  $G$  das Wort  $aaabab$  erzeugt.

**Aufgabe 2:****Kontextfreie Sprachen**

Zeigen Sie, dass die Sprache  $L = \{a^n b^m \mid n > m\}$  genau eine Chomsky-Typ-2-Sprache ist!

**Aufgabe 3:****Turing-Maschinen**

Sei  $\Sigma = \{a, b, c\}$  ein endliches Alphabet. Wir betrachten die Sprache  $L = \{w \in \Sigma^* \mid \forall s \in \Sigma \setminus \{a\} : |w|_s < |w|_a\}$ , also die Menge der Wörter, in denen  $a$  echt häufiger als die anderen Buchstaben vorkommt.

- Geben Sie eine deterministische Turingmaschine an, die  $L$  für das Alphabet  $\Sigma$  entscheidet. Die Maschine soll dabei wie folgt verfahren:
  - Das Arbeitsalphabet  $\Gamma$  soll zusätzliche Zeichen  $\square$  und  $\diamond$  enthalten.
  - Das Wort steht initial als  $\square w \square$  auf dem Band der Turingmaschine.
  - Die Turingmaschine sucht nach jeweils einem  $a$ ,  $b$  und  $c$  und überschreibt das jeweils erste gefundene Zeichen mit  $\diamond$ .
  - Wird dabei kein  $a$  gefunden, ist das Wort zu verwerfen. Wird ein  $a$  gefunden, aber weder  $b$  noch  $c$ , wird das Wort akzeptiert. Beachten Sie, dass es dabei möglich ist, ein  $a$  und  $c$ , aber kein  $b$ , oder ein  $a$  und  $b$ , aber kein  $c$  zu finden.
  - Nach einem solchen Durchgang fährt die Turingmaschine nach links und beginnt von vorne.

Beschreiben Sie dabei die Bedeutung der Zustände  $q \in Q$  Ihrer Turingmaschine informell.

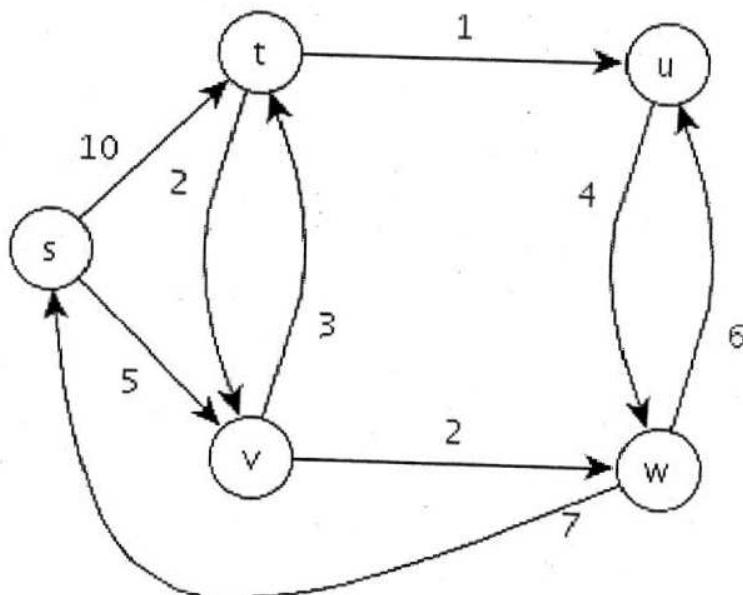
- b) Geben Sie den Ablauf Ihrer Turingmaschine über dem Wort *aba* an.
- c) Geben Sie Laufzeit- und Speicherplatzkomplexität Ihrer Turingmaschine  $M$  in  $O$ -Notation an.  
Kann die Laufzeitkomplexität durch die Verwendung einer Mehrband-Turingmaschine verbessert werden? Begründen Sie Ihre Antworten.

Thema Nr. 2

Hinweis: Die einzelnen Teilaufgaben bauen oftmals aufeinander auf, sind aber im Prinzip in beliebiger Reihenfolge lösbar. Sie dürfen hierbei die Angaben und Ergebnisse früherer Teilaufgaben uneingeschränkt zur Lösung späterer Teilaufgaben verwenden! Außerdem dürfen Sie Tatsachen aus dem Informatik-Duden ohne weitere Begründung als bekannt voraussetzen.

**Aufgabe 1:****Effiziente Algorithmen**

- a) Führen Sie den Dijkstra-Algorithmus zur Bestimmung aller kürzesten Pfade vom Startknoten  $s$  am folgenden Graphen aus:



Dokumentieren Sie Ihre Schritte geeignet.

- b) Der Dijkstra-Algorithmus benutzt einen Heap  $R$ , in dem diejenigen Knoten verwaltet werden, deren Entfernung zu  $s$  noch nicht endgültig feststeht. Sei  $v_1, v_2, \dots, v_n$  eine Anordnung der Knoten in der Reihenfolge, in der sie aus  $R$  mittels  $\text{deletemin}(R, v)$  herausgenommen werden. Weisen Sie nach, dass in dieser Anordnung die Knoten aufsteigend nach der Entfernung  $d[v]$  von  $s$  sortiert sind.

Hinweis:

Zeigen Sie zuerst, dass für alle benachbarten Paare  $(v_i, v_{i+1})$  von Knoten gilt:  $v_i \leq v_{i+1}$ .

**Aufgabe 2:****Reguläre Sprachen**

- a) Konstruieren Sie einen deterministischen endlichen Automaten, der die Sprache akzeptiert, die durch den regulären Ausdruck  $(a + b)^* \cdot a \cdot (a + b) \cdot (a + b) \cdot (a + b)$  beschrieben ist.

Hinweis:

Konstruieren Sie zuerst einen nichtdeterministischen endlichen Automaten, und wandeln Sie diesen anschließend in einen deterministischen endlichen Automaten um!

- b) Geben Sie einen nichtdeterministischen endlichen Automaten (NEA) mit maximal 4 Zuständen an, der die folgende reguläre Sprache akzeptiert:

$$L = \{ xy \mid x, y \in \{a, b\}^* \quad x \text{ endet mit } b \text{ und die Anzahl der Zeichen } a \text{ in } y \text{ ist durch 3 teilbar} \}$$

Ein Wort  $w$  ist somit in  $L$ , falls das Zeichen  $b$  so in  $w$  vorkommt, dass die Zahl der Zeichen  $a$ , die nach diesem  $b$  stehen, durch 3 teilbar ist.

- c) Im Gegensatz zu deterministischen endlichen Automaten ist kein einfacher Minimierungsalgorithmus für nichtdeterministische endliche Automaten bekannt. Sie sollen hier dennoch nachweisen, dass jeder nichtdeterministische endliche Automat, der die in Teilaufgabe b) beschriebene Sprache  $L$  akzeptiert, mindestens 4 Zustände besitzt.

Betrachten Sie eine Zerlegung des Wortes  $w = baaa$  in die folgenden Paare:  $(x_1, y_1) = (\epsilon, baaa)$ ,  $(x_2, y_2) = (ba, aa)$ ,  $(x_3, y_3) = (baa, a)$ ,  $(x_4, y_4) = (baaa, \epsilon)$

Sei  $A$  ein (unbekannter) endlicher Automat, der  $L$  akzeptiert. Da  $w$  vom Automaten  $A$  akzeptiert wird, gibt es für jedes Paar  $(x_i, y_i)$  einen Zustand  $q_i$ , so dass einer der Berechnungspfade den Automaten vom Startzustand aus auf Eingabe  $x_i$  in den Zustand  $q_i$  führt, und von  $q_i$  ausgehend auf der Eingabe  $y_i$  in einen der—möglicherweise mehreren—Endzustände. Führen Sie nun die Annahme zum Widerspruch, dass  $A$  drei Zustände *und zugleich* keine Wörter akzeptiert, die nicht in  $L$  sind.

**Aufgabe 3:****Kontextfreie Sprachen**

Gegeben sei die Sprache  $L = \{ ww \mid w \in \Sigma^* \}$  über dem Alphabet  $\Sigma = \{a, b\}$

- a) Beweisen Sie mit Hilfe des Pumping Lemma für kontextfreie Sprachen, dass die Sprache nicht kontextfrei ist.

Hinweis:

Gehen Sie hierzu von einem Wort der Form  $ww = a^n b^n a^n b^n$  aus, bei geeigneter Wahl von  $n$ .

Hinweis:

Das Pumping Lemma für kontextfreie Sprachen lautet wie folgt: Ist  $L$  kontextfrei, so existiert  $n \geq 0$  sodass für alle  $z \in L$  eine Zerlegung der Form  $z = uvwxy$  existiert, derart, dass  $|vx| \geq 1$  und  $|vwx| \leq n$  und  $uv^iwx^i y \in L$  für alle  $i \geq 0$ .

b) Begründen Sie, dass für jedes Wort  $z \in \Sigma^*$ , das nicht in  $L$  ist, einer der folgenden drei Fälle zutrifft:

- $z$  hat ungerade Länge, oder
- $z$  kann in der Form  $z = uavbw$  geschrieben werden mit  $|u| = m, |w| = n, |v| = m + n$  für geeignete  $m, n \geq 0$ , oder
- $z$  kann in der Form  $z = ubvaw$  geschrieben werden mit  $|u| = m, |w| = n, |v| = m + n$  für geeignete  $m, n \geq 0$ .

Zum Beispiel ist  $aabababbab \notin L$  und es passt der zweite Fall mit  $u = a, v = baba, w = bab$ , also  $m = 1, n = 3$ .

c) Begründen Sie nun, dass das Komplement  $\Sigma^* \setminus L$  kontextfrei ist.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2008**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
--------------------	----------------	----------------------

Kennzahl: \_\_\_\_\_

Kennwort: \_\_\_\_\_

Arbeitsplatz-Nr.: \_\_\_\_\_

**Herbst  
2008**

**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: **8**

---

**Bitte wenden!**

## Thema Nr. 1

### **Aufgabe 1: Datentypen**

Betrachten Sie das Verhalten von Kellern (stack), Warteschlangen (queue) und Prioritätswarteschlagen (priorityqueue oder heap), in die Zahlen eingefügt und wieder gelöscht werden.

Für die Prioritäten gilt: x vor y genau dann wenn  $x > y$ .

in(x) bedeutet, dass die Zahl x eingefügt wird (in der Java API add(E o) oder push(E o))  
out() gibt eine Zahl aus und löscht das ausgegebene Element (in der Java API pop() oder poll() ).

Am Anfang sind alle Datentypen leer.

In welcher Reihenfolge werden die Zahlen bei der nachfolgenden Sequenz

in(10), in(9), in(20), in(5), in(2), in(1), out(), out(), in(15), out(), out(), in(12), in(7), out(), out(), out(), in(3), out(), out(), out()

- a) bei einem Keller,
- b) bei einer Warteschlange,
- c) bei einer Prioritätswarteschlage (mit  $x > y$ ) ausgegeben?
- d) Beschreiben Sie in einer Programmiersprache ihrer Wahl oder in Pseudocode, wie man einen Keller in einem Array implementieren kann, so dass die Operationen "in" und "out" effizient (asymptotisch optimal) realisiert werden.  
Ihre Beschreibung muss die Deklarationen des Arrays und der relevanten Variablen und der Methoden/Funktionen enthalten.

Es reicht nicht, z. B. java.lang.stack zu importieren.

Achten Sie dabei auf mögliche Fehler bei der Ausführung der Operationen „in“ und „out“. Im Fehlerfall kann Ihr Programm mit "exit" beendet werden.

**Fortsetzung nächste Seite!**

**Aufgabe 2: O-Notation**

- a) Ordnen Sie die nachfolgenden Funktionen (über den positiven natürlichen Zahlen  $n \geq 1$ ) aufsteigend in der O-Notation. (Ein Beweis ist nicht gefordert.)

$$f_1(n) = 10n$$

$$f_2(n) = 0,1n^2$$

$$f_3(n) = n \log n$$

$$f_4(n) = 0,001n!$$

$$f_5(n) = 0,001n^3$$

$$f_6(n) = 100 \log_{10} n \quad (\text{Logarithmus zur Basis } 10)$$

- b) Zeigen Sie für zwei nach ihrer Ordnung benachbarte Funktionen  $g$  und  $h$ , dass  $g \in O(h)$  gilt.

**Aufgabe 3: Sortierverfahren**

Eine Menge von reellen Zahlen (type double) sei in einer Liste (oder wahlweise in einem Array) A gespeichert.

Betrachten Sie Quicksort in einer Implementierung (Version), bei der jeweils das erste Element der Liste (oder des betrachteten Abschnitts des Arrays) als Pivotelement gewählt wird.

- a) Beschreiben Sie informal die Arbeitsweise von Quicksort.  
Geben sie die benötigten Parameter und die Vorgehensweise an.
- b) Wie arbeitet das von Ihnen unter a) beschriebene Quicksort auf der folgenden Liste (Array) A?  
Geben Sie die Reihenfolge der Elemente jeweils nach den Durchläufen/Aufrufen bei Quicksort an.  
 $A = (3,0; 3,14; 0,0; 2,9; 7,7; 2,3; 11,0; 9,1; 5,2; 0,5)$
- c) Ist Quicksort in der obigen Version das asymptotisch beste Sortierverfahren?  
Begründen Sie Ihre Antwort!

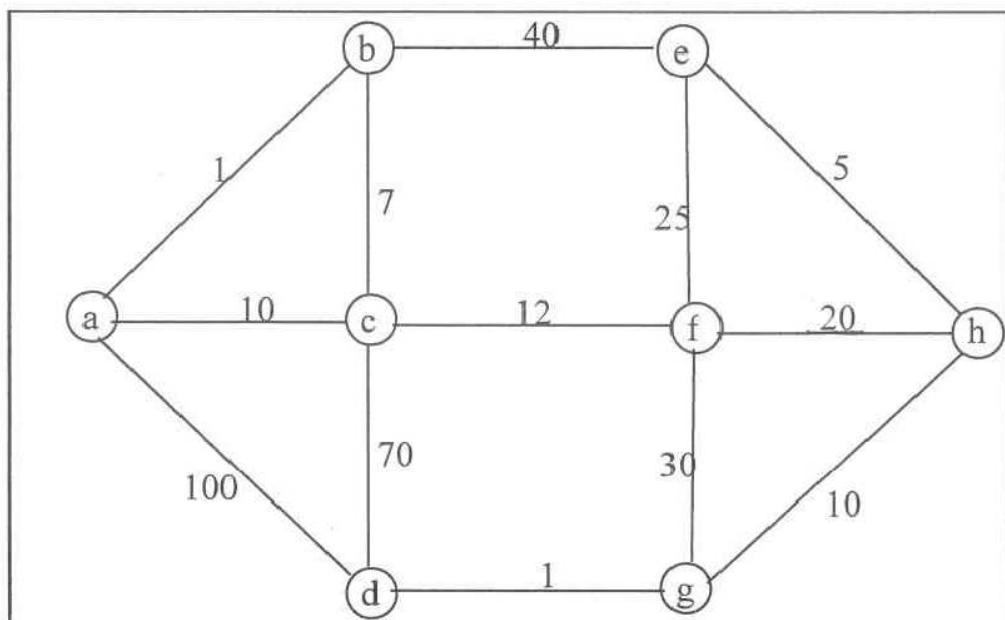
**Aufgabe 4: kürzeste Wege**

Gegeben sei der nachfolgend gezeichnete ungerichtete Graph  $G = (V, E)$  mit den Knoten a, ..., h.

Die Zahlen an den ungerichteten Kanten geben deren Länge an.

Gesucht ist ein (der) kürzester Weg von a nach h.

- a) Beschreiben Sie wie der Dijkstra-Algorithmus auf G arbeitet.  
Geben Sie dazu an, in welcher Reihenfolge die Knoten bearbeitet werden und welche Werte dort gespeichert werden.
- b) Wie lang ist der kürzeste Weg von a nach h?



**Fortsetzung nächste Seite!**

**Aufgabe 5: formale Sprachen**

Sei  $L = \{w \in \{a, b, c, d\}^* \mid w = a^p b^q c^r d^s \mid p, q, r, s > 1\}$ .

In welcher Klasse der Chomsky-Hierarchie (regulär, kontextfrei, kontextsensitiv, rekursiv aufzählbar) liegt  $L$ ?

Gesucht ist die kleinste Klasse.

Begründen Sie Ihre Antwort!

**Aufgabe 6: reguläre Mengen**

Zeigen Sie durch Angabe der Konstruktionsvorschrift für endliche Automaten, dass die Differenz von zwei regulären Sprachen  $L_1 \setminus L_2$  regulär ist.

**Aufgabe 7: formale Sprachen**

Ist die Sprache  $L = \{a^p b^q c^p \mid p, q \geq 0\}$

regulär

oder ist sie nicht regulär, aber kontextfrei

oder ist sie nicht kontextfrei?

Begründen Sie Ihre Aussagen!

Es darf als bekannt vorausgesetzt werden, dass es für die regulären bzw. die kontextfreien Sprachen passende Pumping-Lemmas gibt. Man muss das passende Pumping-Lemma ggf. anwenden.

**Aufgabe 8: Turingmaschinen**

- a) Konstruieren Sie eine deterministische Turingmaschine  $M$  mit  $L(M) = L$ .  
Beschreiben Sie zusätzlich, wie  $M$  arbeitet (Stil:  $M$  liest  $a$ 's und speichert ....)

$$L = \{a^p b^q c^p d^q \mid p, q \geq 1\}.$$

- b) Welche Zeit- und welche Speicherkomplexität hat Ihre Turingmaschine  $M$ ?  
Es reicht die Angabe einer geeigneten Funktion  $f(n)$  mit  $n = 2p+2q$  für die Länge der Eingabe.  
Die O-Notation darf verwendet werden.  
Erläutern Sie die Schranken für  $M$ .

## Thema Nr. 2

### Aufgabe 1: AVL-Bäume

- a) Fügen Sie nacheinander die Zahlen 9 8 7 2 4 6 1 5 3 in einen anfangs leeren AVL-Baum ein.  
 Repräsentieren (zeichnen) sie alle AVL-Bäume jeweils vor einer notwendigen Rotation und beschreiben Sie die Rotationen (z.B. LL-Rotation für die Zahlen ....)
- b) Konstruieren Sie einen AVL-Baum mit 12 Knoten mit maximaler Tiefe (Höhe) und begründen Sie Ihre Lösung.

### Aufgabe 2: abstrakte Datentypen

Gegeben sei ein abstrakter Datentyp ADT mit den folgenden Operationen (über den ganzen Zahlen)

empty()	Kreiert die leere Struktur
isempty()	Testet auf Leerheit
insert(x)	Einfügen des Elements x
deletemin()	Löschen des Elements mit dem kleinsten Wert
deletemax()	Löschen des Elements mit dem größten Wert

Entwerfen/beschreiben Sie eine Datenstruktur für eine effiziente Realisierung des ADT mit höchstens  $O(\log n)$  Kosten je Operation.

### Aufgabe 3: Wegealgorithmen

Ein gerichteter Graph  $G$  sei durch die folgende Entfernungsmatrix gegeben.

	s	a	b	c	d	e	f	t
s	25		80		45			
a		13	25	70				
b					6		70	
c					13	6		
d			10					40
e				7	16		31	
f				7	1			30
t								

- (i) Konstruieren Sie eine Zeichnung für  $G$ . (Tipp: Zeichnen Sie s ganz links und t rechts.)  
 (ii) Berechnen Sie die kürzesten Wege ab s mit dem Dijkstra-Algorithmus.  
 Protokollieren Sie, in welcher Reihenfolge der Dijkstra-Algorithmus den Knoten ihre kürzeste Entfernung zuweist.  
 (iii) Markieren (oder notieren) Sie den kürzesten Weg von s nach t und berechnen Sie seine Länge.

Fortsetzung nächste Seite!

**Aufgabe 4: Automaten und formale Sprachen**

Gesucht ist die Menge  $L$  aller Dezimalzahlen über  $\Sigma = \{0,1,2,3\}$  (mit führenden Nullen), die durch 2 oder (logisches oder) durch 3 teilbar sind.

Beschreiben Sie  $L$  durch einen Automaten oder durch eine Grammatik.

In welcher Klasse der Chomsky-Hierarchie liegt  $L$ ? Geben Sie die kleinstmögliche Klasse der Chomsky-Hierarchie an.

**Aufgabe 5: Klammersprachen**

Sei  $L$  die Menge der Wörter  $w \in \{( , ), a, b\}^*$  die an beliebigen Stellen die Zeichen „a“ oder „b“ haben können und die bezüglich „(“ und „)“ korrekt geklammert sind.

- (i) Ist  $L$  regulär?
- (ii) Ist  $L$  kontextfrei?
- (iii) Ist  $L$  entscheidbar?

Begründen Sie ihre Entscheidung.

**Aufgabe 6: entscheidbare oder rekursive Sprachen**

Zeigen und begründen Sie:

Die symmetrische Differenz  $X \oplus Y = \{w \mid w \in X \text{ und } w \notin Y \text{ oder } w \notin X \text{ und } w \in Y\}$  zweier entscheidbarer Sprachen ist entscheidbar.

Tipp: Es wird keine formale Konstruktion verlangt; es reicht eine informale "high-level" Argumentation. (Die rekursiven Sprachen sind definiert durch .... Daher haben sie die folgende Eigenschaften.... Somit kann man für  $X \oplus Y$  konstruieren ...)"

**Aufgabe 7: NP-Vollständigkeit**

Das  $k$ -Färbbarkeitsproblem auf Graphen ist das Problem, ob man die Knoten eines ungerichteten Graphen  $G = (V, E)$  so mit höchstens  $k$  Farben färben kann, dass benachbarte Knoten verschiedene Farben haben.

Es ist bekannt (und kann vorausgesetzt werden), dass das 3-Färbbarkeitsproblem auf Graphen NP-vollständig ist.

Ist auch das 4-Färbbarkeitsproblem NP-vollständig?

Begründen Sie Ihre Antwort!

Tipp: Ergänzen Sie einen neuen Knoten und viele Kanten.

**Fortsetzung nächste Seite!**

**Aufgabe 8: Klassifikation**

Gegeben seien die folgenden Klassen von Sprachen:

- A: die kontextfreien Sprachen, CFL
- B: NP
- C: die rekursiv aufzählbaren (oder positiv semi-entscheidbaren) Sprachen, RE
- D: die regulären Mengen, REG
- E: die rekursiven oder entscheidbaren Sprachen, REK

Ordnen Sie diese Klassen bezüglich der bekannten und bewiesenen Inklusionen und begründen Sie Ihre Antwort in der Form: Die Klasse X wird durch Maschinen des Typs x definiert und diese sind ein Spezialfall von Maschinen des Typs y für Y.

Wo gelten echte Inklusionen und warum?

Tipp: Komplexitätsklassen, z. B. mit exponentiellem Aufwand (Zeit- oder Speicherplatz) sind echte Teilmengen der rekursiven Sprachen REK.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2009**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	<b>Frühjahr</b>	
Kennwort: _____	<b>2009</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**

**— Prüfungsaufgaben —**

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 9

Bitte wenden!

Thema Nr. 1  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1** Ordnen Sie die folgenden formalen Sprachen bestmöglich in die Chomsky-Hierarchie ein, und geben Sie eine ausreichende Begründung dafür, dass die jeweilige Sprache in der von Ihnen genannten Hierarchiestufe liegt und in keiner niedrigeren eingeordnet werden kann!

1.  $L_1 = \{ a^n b^n a^n \mid n \geq 1 \}$
2.  $L_2 = \{ a^n b^m a^n \mid m \geq 1, n \geq 1 \}$
3.  $L_3 = \{ a^n b^m \mid m \geq 1, n \geq 1 \}$
4.  $L_4 = \text{Die Menge aller Dezimaldarstellungen von Vielfachen von } 3$
5.  $L_5 = \text{Die Menge aller terminierenden Java-Programme}$
6.  $L_6 = \{ w \in \{a, b\}^* \mid w \text{ enthält mindestens 4 Vorkommen von } a \}.$

**Aufgabe 2** Wir betrachten den deterministischen Automaten  $(\Sigma, Z, z_0, \delta, F)$ , wobei  $\Sigma = \{0, 1\}$  und  $Z = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$  und  $z_0 = \varepsilon$  und  $F = \{10\}$  und die Zustandsübergangsfunktion  $\delta$  schematisch durch die folgenden Gleichungen gegeben ist:

$$\begin{aligned}\delta(\varepsilon, x) &= x \\ \delta(x, y) &= xy \\ \delta(xy, z) &= yz\end{aligned}$$

wobei  $x, y, z \in \{0, 1\}$ . Es gilt also zum Beispiel  $\delta(0, 1) = 01$  und  $\delta(01, 1) = 11$ . Lassen Sie sich nicht dadurch verwirren, dass die Zustände hier Wörter sind.

1. Zeichen Sie den Automaten als Diagramm!
2. Geben Sie einen möglichst kurzen regulären Ausdruck für die erkannte Sprache  $L$  an!
3. Wenden Sie den Minimierungsalgorithmus auf den Automaten an und berechnen Sie so den Minimalautomaten!
4. Geben Sie die Äquivalenzklassen der Myhill-Nerode Äquivalenzrelation als reguläre Ausdrücke an! Hinweis: Die Myhill-Nerode Äquivalenz ist erklärt durch  $u \sim v \iff (\forall w : uw \in L \Leftrightarrow vw \in L)$ .

Tipp: Die Vereinigung aller Klassen muss  $\{0, 1\}^*$  ergeben.

**Aufgabe 3** Welche der folgenden Aussagen trifft zu, welche nicht? Begründen Sie jeweils Ihre Antwort durch Beweis oder Gegenbeispiel! Natürlich dürfen Sie sich auf allgemein bekannte Lehrsätze und den Informatikduden beziehen.

1. Ist eine Sprache  $L$  rekursiv aufzählbar, so ist sie auch entscheidbar.
2. Ist eine Sprache auf das Halteproblem reduzierbar, so ist sie rekursiv aufzählbar.
3. Die rekursiv aufzählbaren Sprachen sind unter Vereinigung und Durchschnitt abgeschlossen.
4. Die rekursiv aufzählbaren Sprachen sind unter Komplement abgeschlossen.
5. Es ist unentscheidbar, ob eine Turingmaschine  $M$  bei leerer Eingabe im Verlaufe der Berechnung eine Konfiguration erreicht, die grösser als 17 ist. Tipp: Es gibt nur eine feste Zahl von Konfigurationen der Grösse 17 oder kleiner.

**Aufgabe 4** Hier finden Sie Pseudocode<sup>1</sup> für den Dijkstra-Algorithmus in der Notation des Informatikdudens. Insbesondere ist  $R$  eine Prioritätsschlange mit  $D[]$  als Schlüssel. Der Algorithmus berechnet die kürzesten Entfernungen vom Startknoten  $s$  in einem gerichteten Graphen  $G = (V, E)$  mit nichtnegativen Kantengewichten  $d(\cdot, \cdot)$ . Der Aufruf  $\text{deletemin}(R, v)$  weist den kleinsten Eintrag von  $R$  der Variablen  $v$  zu und entfernt ihn aus  $R$ .

```

setze  $D[v] = \infty$  für alle Knoten  $v$ ;  

 $D[s] := 0$ ;  $R := V$   

while  $R \neq \emptyset$  do  $\text{deletemin}(R, v)$ ;  

    for all Knoten  $w$  mit  $(v, w) \in E$  do  

         $h := D[v] + d(v, w)$ ;  

        if  $h < D[w]$   

            then  $D[w] := h$ ;  

             $\text{decreasekey}(R, w)$ ;
```

- Arbeiten Sie den Algorithmus für folgende Eingabe ab und dokumentieren Sie die Schritte geeignet!

$$\begin{aligned}
V &= \{s, u, v, x, y\} \\
E &= \{(s, u), (s, x), (u, x), (x, u), (u, v), (x, y), (x, v), (v, y), (y, v), (y, s)\} \\
d(s, u) &= 10, d(s, x) = 5, d(u, x) = 2, d(x, u) = 3, d(u, v) = 1, \\
d(x, y) &= 2, d(x, v) = 9, d(v, y) = 4, d(y, v) = 6, d(y, s) = 7
\end{aligned}$$

- Der syldawische Informatiker W. E. ARTSKJID schlug 2007 vor, beim Abarbeiten eines Knotens  $v$  jeweils die Menge  $M = \{v_1, \dots, v_k\}$  derjenigen Knoten in  $R$  zu bilden, deren  $D$ -Wert während der Abarbeitung von  $v$  verringert wurde. Aus dieser Menge  $M$  wird nun der Knoten mit dem kleinsten  $D$ -Wert gesucht und als nächstes abgearbeitet. Sollte die Menge  $M$  leer sein, so wird wie üblich verfahren, es wird also der Knoten aus  $R$  mit kleinstem  $D$ -Wert als nächstes abgearbeitet.

Geben Sie Pseudocode für den so modifizierten Algorithmus an!

Geben Sie ein konkretes Beispiel dafür, dass der modifizierte Algorithmus bisweilen unkorrekte Ergebnisse liefert! Sie müssen dazu den entsprechenden Eingabegraphen mit Kantengewichten skizzieren und den Lauf des modifizierten Algorithmus auf dieser Eingabe geeignet dokumentieren.

---

<sup>1</sup> Die Version der Informatikdudens enthält eine hier störende Ungenauigkeit, welche es im Prinzip erlaubt, bereits abgearbeitete Knoten wieder in die Prioritätsschlange aufzunehmen.

Thema Nr. 2  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Formale Sprachen und Automatentheorie

- a) Wir betrachten die Sprache  $L$  über der Zeichenmenge  $T = \{a, b\}$ , die das Wort  $abab$  als Teilwort enthalten.
- Geben Sie einen regulären Ausdruck an, der  $L$  beschreibt.
  - Geben Sie eine reguläre, rechtslineare Grammatik an, die  $L$  erzeugt.
  - Geben Sie eine Ableitung des Wortes  $aababbb$  an.
- b) Gegeben sei der reguläre Ausdruck:

$$[0|1[01^*0]^*1]^*.$$

Wandeln Sie diesen regulären Ausdruck in einen nichtdeterministischen endlichen Automaten um.

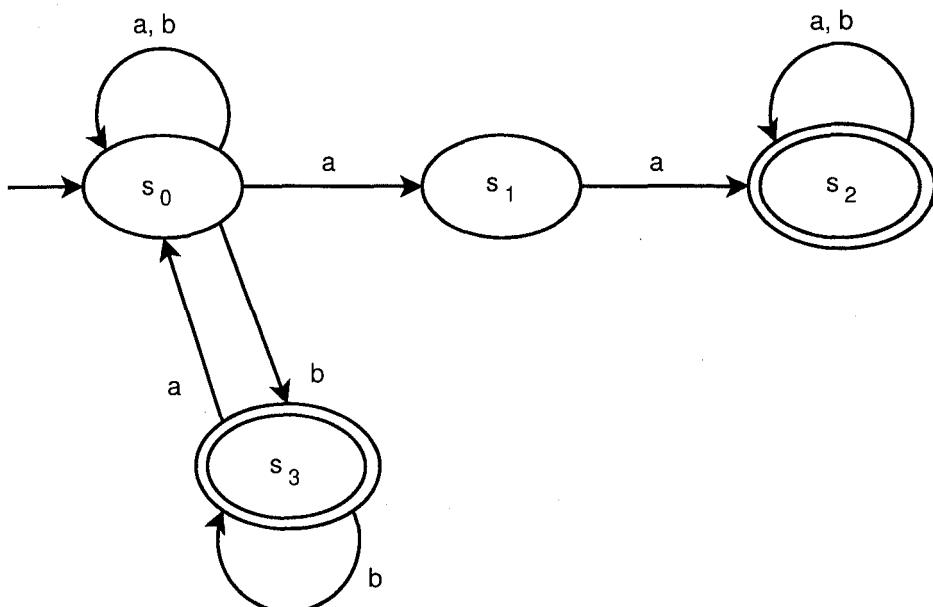
- c) Gegeben sei der endliche, nichtdeterministische Automat

$$A = (S, T, s_0, \delta) \text{ mit}$$

$$S = \{s_0, s_1, s_2, s_3\},$$

$$T = \{a, b\}, Z = \{s_2, s_3\}$$

mit der Übergangsfunktion  $\delta$  gemäß untenstehendem Diagramm.



- Konstruieren Sie einen äquivalenten, deterministischen, endlichen Automaten.
- Minimalisieren Sie den endlichen Automaten.

Fortsetzung nächste Seite!

d) Eine Turing-Maschine D, die natürliche Zahlen in Strichdarstellung halbiert (ganzzahlige Division durch 2), mit Eingabealphabet  $T = \{ | \}$  soll in mehreren Schritten entwickelt werden.

- i) Definieren Sie eine Turing-Maschine L mit Eingabealphabet T, die den Kopf zum nächsten leeren Bandfeld nach links (mindestens um ein Feld) bewegt und die Bandbeschriftung nicht ändert. L setzt voraus, dass der Kopf auf einem leeren Bandfeld steht (Beispiel für Startkonfiguration:  $\# ||| \#$ ).

Stellen Sie die Übergangsrelation von L in Form einer Tabelle und in Form eines erweiterten, endlichen Automaten dar.

**Hinweis:** Ein erweiterter, endlicher Automat ist ein endlicher Automat, dessen Zustandsübergänge neben der gelesenen Eingabe mit der geschriebenen Ausgabe und der Lese-/Schreibkopfbewegung annotiert sind.

- ii) Definieren Sie analog zu i) eine Turing-Maschine R, die den Kopf zum nächsten leeren Bandfeld nach rechts bewegt. Geben Sie eine vollständige Berechnung von R zur Startkonfiguration  $\# ||| \#$  an.
- iii) Geben Sie nun unter Verwendung von L und R eine Turing-Maschine D (für die ganzzahlige Division durch 2 einer Strichzahl) in Form eines erweiterten, endlichen Automaten an. D setzt voraus, dass der Kopf auf dem ersten leeren Bandfeld rechts von der Eingabe steht. Beispielberechnung:  $(\# \# \# ||| \#) \rightarrow_* (\# | \# \# \# \# \#)$

## Aufgabe 2:

### Berechenbarkeit

- a) Gegeben sei die Funktion  $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  mit  $f(x) = x - 1$  wobei '-' definiert ist als Subtraktion über den natürlichen Zahlen mit der Besonderheit, dass  $a - b = 0$  ergibt, falls  $a \leq b$ .
- i) Zeigen Sie, dass die Funktion f Einband-Turing-berechenbar ist, indem Sie eine solche Turing-Maschine angeben. x sei dabei wieder in Strichdarstellung gegeben. Der Kopf beginnt rechts.
- ii) Nennen Sie zwei weitere Modelle für Berechenbarkeit.
- iii) Was besagt die Church These im Hinblick auf die obigen Modelle?
- b) i) Begründen Sie, dass die Menge der Primzahlen entscheidbar ist.
- ii) Zeigen Sie folgende Eigenschaften für entscheidbare Mengen:
- Jede endliche Menge, einschließlich der leeren Menge, ist entscheidbar.
  - Das Komplement einer entscheidbaren Menge ist entscheidbar.
- c) Sei  $A \subseteq \mathbb{N}_0$ .
- i) Geben Sie die Definition für rekursive Aufzählbarkeit dieser Menge an.
- ii) Zeigen Sie, dass eine Menge A genau dann entscheidbar ist, wenn A und  $\overline{A}$  rekursiv aufzählbar sind.

**Aufgabe 3:**

Komplexität

- a) Die folgenden Fragen beziehen sich auf nichtdeterministische Turing-Maschinen.
- Geben Sie eine Definition in üblicher Tupelschreibweise an.
  - Erarbeiten Sie die wesentlichen Unterschiede zu einer deterministischen Turing-Maschine.
  - Was bedeutet die Klasse  $NP$ ?
- b) Gegeben sei das SOS (Sum of Subset) - Problem

$$SOS := \left\{ (a_1, \dots, a_m, b) : m, a_1, \dots, a_m, b \in \mathbb{N}_0 \right. \\ \text{und es existieren } c_1, \dots, c_m \in \{0, 1\} \text{ mit} \\ \left. \sum_{i=1}^m c_i \cdot a_i = b \right\}$$

- Zeigen Sie, dass das SOS - Problem in polynomialer Zeit auf einer nichtdeterministischen Maschine entscheidbar ist (verwenden Sie dazu ein geeignetes übliches nichtdeterministisches Berechnungsmodell in dem Arrays als primitive Datenstruktur vorhanden sind).
- Gegeben sei das NP-vollständige Problem  $TS'$ , eine Variante des "Travelling Salesperson" Problems:

$$TS' := \left\{ (m, M, k) : m, k \in \mathbb{N}_0, M : \{1, 2, \dots, m\}^2 \rightarrow \mathbb{N}_0 \right. \\ \text{und es existiert eine Permutation } \pi \text{ mit} \\ \left. \sum_{i=1}^{m-1} M(\pi(i), \pi(i+1)) + M(\pi(m), \pi(1)) = k \right\}$$

Zeigen Sie damit, dass das obige SOS-Problem ebenfalls ein NP-vollständiges Problem ist.

**Aufgabe 4:**

Effiziente Algorithmen und Datenstrukturen

- a) Die Ausdrücke  $O(\cdot)$  und  $\Omega(\cdot)$  sind gebräuchliche Notation in der Informatik.
- Wie sind sie jeweils definiert?
  - Benennen Sie die wesentlichen Unterschiede.

- iii) Geben Sie in  $O(\cdot)$ -Notation die Berechnungskomplexität in Abhängigkeit von  $a$  und  $b$  für folgenden Algorithmus an:

```

function fastmult(a,b);
begin
  A[0] := 1;
  B[0] := a;
  j := 0;
  while (A[j] <= b) do begin
    A[j+1] := (A[j] + A[j]);
    B[j+1] := (B[j] + B[j]);
    j := (j +1)
  end;
  c := 0;
  while (j > 0) do begin
    j := (j-1);
    if (A[j] <= b) then begin
      b := (b - A[j]);
      c := (c + B[j])
    end
  end;
  fastmult := c;
end

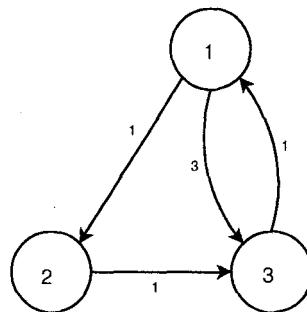
```

- iv) Zeigen oder widerlegen Sie:

$$2^{n+1} = O(2^n)?$$

$$2^{2n} = O(2^n)?$$

- b) Gegeben sei der Graph  $G$ :



Somit ist die Adjazenzmatrix von  $G$  die Matrix  $D_0$ :

$$D_0 = \begin{pmatrix} \infty & 1 & 3 \\ \infty & \infty & 1 \\ 1 & \infty & \infty \end{pmatrix}$$

Um den kürzesten Pfad in diesem gewichteten, gerichteten Graphen zu finden, setzen wir den Floyd-Warshall-Algorithmus ein:

```
n := rows[D0]  
for k ← 1 to n do  
    for i ← 1 to n do  
        for j ← 1 to n do  
            Dk[i][j] ← min( Dk-1[i][j], Dk-1[i][k] + Dk-1[k][j])  
return Dn
```

- i) Führen Sie den Floyd-Warshall Algorithmus auf der den Graphen  $G$  repräsentierenden Matrix  $D_0$  aus. Stellen Sie dabei die Zwischenergebnisse der Matrizen  $D_1$  und  $D_2$  sowie das Ergebnis  $D_3$  in Matrixform dar.
- ii) Auf welchen gewichteten, gerichteten Graphen berechnet der Floyd-Warshall-Algorithmus genau die transitive Hülle?

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2009**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	<b>Herbst 2009</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

---

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**  
**— Prüfungsaufgaben —**

---

Fach: **Informatik(vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

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

Anzahl der Druckseiten dieser Vorlage: **7**

---

Bitte wenden!

Thema Nr. 1  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Sei  $\Sigma = \{0, 1, \$\}$  und sei  $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ .

a) Sei

$$L_1 = \{0^n\$1^{3n} \mid n \in \mathbb{N}_0\} \cup \{0^{3n}\$1^n \mid n \in \mathbb{N}_0\}.$$

Beispiele:  $00\$111111 \in L_1$ ,  $\$ \in L_1$ ,  $000\$1 \in L_1$ .

i) Zeigen Sie, dass  $L_1$  kontextfrei ist, indem Sie eine kontextfreie Grammatik  $G$  angeben mit  $L(G) = L_1$ .

ii) Begründen Sie, warum Ihre Grammatik *genau* die Sprache  $L_1$  erzeugt.

b) Formulieren Sie das Pumping-Lemma für kontextfreie Sprachen:

„Sei  $L$  eine kontextfreie Sprache über dem Alphabet  $\Sigma$ . Dann gibt ...“

c) Zeigen Sie z. B. mittels Pumping-Lemma für kontextfreie Sprachen, dass die Sprache

$$L_2 = \{0^n\$1^{3n}\$0^n \mid n \in \mathbb{N}_0\}$$

nicht kontextfrei ist.

Beispiel:  $00\$111111\$00 \in L_2$

**Aufgabe 2:**

a) Definieren Sie die zum Halteproblem für Turing-Maschinen gehörende Menge  $H$ .

b) Gegeben sei das folgende Problem  $E$ :

„Entscheide, ob es für die deterministische Turing-Maschine mit der Gödelnummer  $n$  eine Eingabe gibt, für die die Turing-Maschine hält.“

Zeigen Sie, dass  $E$  nicht entscheidbar ist. Benutzen Sie, dass  $H$  aus (a) nicht entscheidbar ist.

c) Angenommen, es wurde gezeigt, dass  $P = NP$  ist. Zeigen Sie, dass dann *jede* Sprache  $L \in P$  sogar NP-vollständig ist.

**Aufgabe 3:**

- a) Geben Sie einen deterministischen endlichen Automaten an (Zeichnung), der die Sprache

$$L_3 = \{a^i b^j \mid i, j \in \mathbb{N}_0, i \text{ und } j \text{ sind durch 3 teilbar}\}$$

akzeptiert. ( $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ )

Beispiele:  $aaabbbbb \in L_3$ ,  $aaabbb \in L_3$ ,  $\varepsilon \in L_3$ .

- b) Beweisen oder widerlegen Sie die folgende Behauptung:

Ist die Sprache  $L$  regulär, dann ist auch jede Obermenge von  $L$  regulär.

- c) Zu einem Wort  $w = a_1 a_2 a_3 \dots a_n$  ist

$$\text{suffix}(w) = \{\varepsilon, a_n, a_{n-1}a_n, a_{n-2}a_{n-1}a_n, \dots, a_1 a_2 a_3 \dots a_n\}$$

die Menge aller Suffices von  $w$ .

Beweisen oder widerlegen Sie die folgende Behauptung:

Ist die Sprache  $L$  regulär, dann ist auch

$$\text{suffix}(L) = \bigcup_{w \in L} \text{suffix}(w)$$

regulär.

**Aufgabe 4:**

Erklären Sie die jeweils wesentliche(n) Idee(n) folgender Informatik-Konzepte in jeweils maximal drei Sätzen:

- a) Adjazenzliste
- b) Mergesort
- c) Lineare Suche
- d) Rekursive Datenstruktur

**Aufgabe 5:**

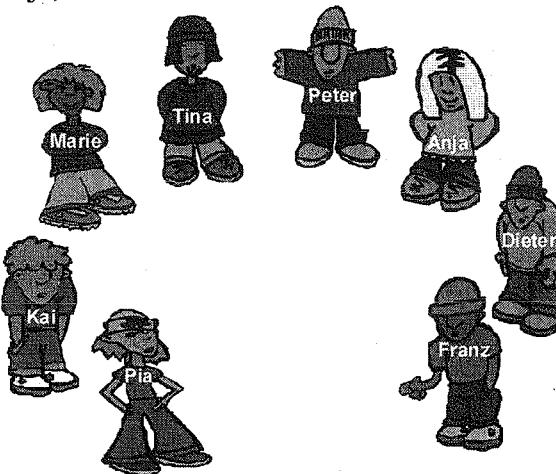
Gegeben seien  $n$  in einem Kreis stehende Kinder, die einen Abzählreim mit  $k$  Silben spielen. Dabei wird ausgehend von einem ersten Kind mit jeder weiteren Silbe des Abzählreims im Kreis herum (im Uhrzeigersinn) auf das jeweils nächste Kind gezeigt. Das Kind, auf das bei der letzten Silbe des Reims gezeigt wird, scheidet aus. Anschließend wird mit dessen Nachfolger fortgesetzt und der Abzählreim erneut gesprochen. Der Nachfolger eines Kindes sei dabei das Kind, das im Uhrzeigersinn hinter diesem steht unter der Annahme, dass alle in die Mitte des Kreises schauen. Das Ganze wird so lange durchgespielt, bis nur noch ein Kind übrig ist. Dieses hat gewonnen.

Gegeben sei der Abzählreim:

1, 2, 3, 4, 5, 6, 7  
*eine alte Frau kocht Rüben,  
 eine alte Frau kocht Speck -  
 und du bist weg!*

Im Kreis stehen dabei in dieser Reihenfolge folgende Kinder:

Pia, Kai, Marie, Tina, Peter, Anja, Dieter und Franz



- a) Wer gewinnt, wenn bei Pia begonnen wird?
- b) Geben Sie in einer gängigen Programmiersprache oder als Pseudocode einen gut kommentierten Algorithmus an, der ausgehend von einer Menge von  $n$  Kindern, einem Kind, das startet, und einem Abzählreim mit  $k$  Silben das gewinnende Kind ermittelt. Ihre Lösung soll als Datenstrukturen nur einfache Variablen und Arrays verwenden, aber keine Zeigerstrukturen.
- c) Welche Laufzeit- und welche Platzkomplexität hat Ihr Algorithmus?

### Aufgabe 6: Datenstruktur für Intervalle

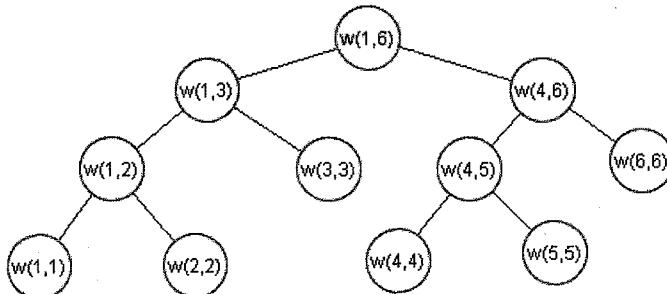
Sei  $M = \{1, 2, \dots, n\}$  eine Menge aufeinander folgender natürlicher Zahlen. Für jedes  $i \in M$  sei ein Wert  $w(i)$  definiert. Gesucht ist dann ein hinsichtlich Speicherbedarf und Laufzeit effizienter Algorithmus, der zu einem gegebenen Intervall  $[i, j]$  mit  $i, j \in M$  den Wert

$$w(i, j) = \begin{cases} w(i) + \dots + w(j) & , \text{falls } i < j \\ 0 & , \text{falls } i > j \\ w(i) & , \text{falls } i = j \end{cases}$$

berechnet.

Der Algorithmus soll aus zwei Phasen bestehen:

1. Aufbau einer geeigneten Datenstruktur (Preprocessing).
  2. Berechnung eines Wertes  $w(i, j)$ .
- a) Formulieren Sie gut kommentiert in einer gängigen Programmiersprache oder als Pseudocode einen Algorithmus, der dieses Problem mit Hilfe eines eindimensionalen Arrays löst, das die Werte  $w(i)$  nach  $i$  aufsteigend sortiert speichert. Geben Sie mit Begründung die Laufzeit, den der Algorithmus nach dem Preprocessing in Phase 2 benötigt, und den allgemeinen Speicherplatzbedarf an.
  - b) Formulieren Sie nun gut kommentiert in einer gängigen Programmiersprache oder als Pseudocode einen Algorithmus, der dieses Problem mit Hilfe eines zweidimensionalen Arrays löst. Geben Sie mit Begründung die Laufzeit, die der Algorithmus nach dem Preprocessing in Phase 2 benötigt, und den allgemeinen Speicherplatzbedarf an.
  - c) Eine effiziente Möglichkeit zur Lösung dieses Problems ist die Verwendung eines so genannten Segmentbaumes. Für das Beispiel  $M = \{1, 2, 3, 4, 5, 6\}$  sieht ein solcher wie folgt aus:



Formulieren Sie wiederum gut kommentiert in einer gängigen Programmiersprache oder Pseudocode einen Algorithmus, der das Problem mit Hilfe eines solchen Segmentbaumes löst. Geben Sie dann mit Begründung die Laufzeit, die der Algorithmus nach dem Preprocessing in Phase 2 benötigt, und den allgemeinen Speicherplatzbedarf an.

Thema Nr. 2  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Stufen Sie die folgenden Sprachen über dem Alphabet  $\Sigma = \{a, b, c\}$  in die Chomsky-Hierarchie ein. Begründen Sie jeweils Ihre Antwort.

- $L_1 = \{a^n b^m c^k \mid n, m, k \geq 1\}$
- $L_2 = \{a^n b^m c^n \mid m, n \geq 1\}$
- $L_3 = \{a^n b^n c^n \mid n \geq 1\}$
- $L_4 = \{(ab)^n a (ba)^n \mid n \geq 1\}$
- $L_5 = \{w \mid w \text{ codiert eine Turingmaschine, deren Ausführung bei leerer Eingabe hält}\}$
- $L_6 = \{w \mid w \text{ codiert eine Turingmaschine, deren Ausführung bei leerer Eingabe nicht hält}\}$

Bei den letzten zwei Aufgabenteilen ist eine sinnvolle Codierung von Turingmaschinen durch Zeichenketten über  $\Sigma$  zugrunde gelegt.

**Aufgabe 2:**

- Es seien  $L_1$  und  $L_2$  reguläre Sprachen über  $\Sigma = \{a, b, c\}$ . Begründen Sie, dass die folgende Sprache dann auch regulär ist:

$$L = \{w \mid \text{für alle Zerlegungen } w = uvx \text{ gilt: } u \in L_1 \Rightarrow x \in L_2\}$$

Hinweis: Zeigen Sie zunächst, dass das Komplement von  $L$  regulär ist.

- Es sei  $\Sigma = \{0, \dots, 9\}$  und  $L \subseteq \Sigma^*$  die Menge aller Dezimaldarstellungen von Zahlen, die durch 2009 teilbar sind. Begründen Sie unter Verwendung des aus der Grundschule bekannten schriftlichen Divisionsalgorithmus, dass  $L$  regulär ist.

**Aufgabe 3:**

Alle Sprachen dieser Aufgabe sind über dem Alphabet  $\Sigma = \{a, b, c\}$  definiert.

- Beweisen oder widerlegen Sie folgende Aussage: Sind  $L_1$  und  $L_2$  kontextfrei, so auch ihre Differenz  $L_1 \setminus L_2$ .
- Zeigen Sie, dass die Sprache aller Wörter  $w$ , die die folgenden zwei Bedingungen erfüllen, kontextfrei ist:
  - Nach einem  $b$  kommt kein  $c$  mehr, also  $w = ubv \Rightarrow v \in \{a, b\}^*$ ;
  - Es sind nie mehr  $c$ 's als  $a$ 's, also  $w = uv \Rightarrow |u|_a \geq |u|_c$

Hier bedeutet  $|w|_x$  die Zahl der Vorkommen des Symbols  $x$  im Wort  $w$ .

**Aufgabe 4:**

Das Problem der Schulstundenplanung ist wie folgt formuliert:

Gegeben: Eine Menge  $L$  von Lehrern, eine Menge  $K$  von Klassen, eine Menge  $Z$  von Zeiten und eine Menge  $U$  von „Unterrichten“. Außerdem Funktionen  $k : U \rightarrow K$  und  $l : U \rightarrow L$ , die jedem Unterricht  $u \in U$  eine Klasse  $k(u)$  und einen Lehrer  $l(u)$  zuordnen.

Gesucht: Eine Funktion  $z : U \rightarrow Z$ , sodass für alle  $u, v \in U$  gilt:

$$u \neq v \wedge z(u) = z(v) \Rightarrow l(u) \neq l(v) \wedge k(u) \neq k(v)$$

d.h. gleichzeitig stattfindende Unterrichte gehören weder zum selben Lehrer, noch zur selben Klasse.

Zeigen Sie, dass das Problem der Schulstundenplanung NP-vollständig ist. Für die NP-Schwere können Sie z.B. auf Graphenfärbung reduzieren.

**Aufgabe 5:**

Es bezeichne  $F(h)$  die minimale Zahl von Knoten, die ein AVL-Baum der Höhe  $h$  enthalten kann. Die Höhe eines Baumes ist die Länge (Zahl der Kanten) des längsten Pfades von der Wurzel zu einem Blatt. Eine isolierte Wurzel hat Höhe 0. Eine Wurzel mit ein oder zwei Blättern als Kinder hat Höhe 1, etc.

- a) Geben Sie ein Beispiel eines AVL-Baums der Höhe 3 mit nur 7 Knoten und begründen Sie, dass kein AVL Baum der Höhe 3 mit weniger als 7 Knoten existiert, dass also  $F(3) = 7$  gilt.
- b) Begründen Sie ebenso, dass  $F(0) = 1, F(1) = 2, F(2) = 4, F(4) = 12$ .
- c) Leiten Sie aus diesen Beispielen eine Vorschrift ab, mit deren Hilfe man  $F(h+2)$  aus  $F(h)$  und  $F(h+1)$  berechnen kann.
- d) Begründen Sie die Allgemeingültigkeit dieser Vorschrift so genau wie möglich.

**Aufgabe 6:**

Die Wäscheleinenaufgabe besteht darin,  $n$  Wäschestücke der Breiten  $b_1, b_2, \dots, b_n$  auf Wäscheleinen der Breite  $b$  aufzuhängen. Idealerweise sollte die Zahl der benutzten Leinen möglichst klein werden. Formal ist eine Aufhängung der Wäsche auf  $l$  Leinen also eine Einteilung der Menge  $\{1, \dots, n\}$  in  $l$  Klassen  $L_1, \dots, L_l$ , sodass für alle  $j = 1 \dots l$  gilt  $\sum_{i \in L_j} b_i \leq b$ . Eine Lösung der Wäscheleinenaufgabe ist dann eine Zahl  $l$  und eine Aufhängung der Wäsche auf  $l$  Leinen. Eine Lösung ist umso besser, je kleiner  $l$  ist.

- a) Beschreiben Sie einen sinnvollen Greedy-Algorithmus für das Wäscheleinenproblem. (Also nicht einfach für jedes Wäschestück eine neue Leine)
- b) Geben Sie ein Beispiel einer Wäscheladung (Instanz des Wäscheleinenproblems), für die Ihr Algorithmus mehr als die minimal mögliche Zahl von Leinen verbraucht.
- c) Nennen Sie ein Beispiel einer Problemstellung, die mit einem Greedy-Algorithmus optimal gelöst werden kann.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2010**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	Frühjahr	
Kennwort: _____	2010	66115
Arbeitsplatz-Nr.: _____		

---

**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: 9

---

Bitte wenden!

Thema Nr. 1  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

- Konstruieren Sie einen deterministischen endlichen Automaten für die durch den regulären Ausdruck  $a(ba)^* + b + bb(ab)^*$  definierte Sprache über dem Alphabet  $\Sigma = \{a, b\}$ . Minimieren Sie den Automaten mit dem Minimierungsalgorithmus oder weisen Sie von Ihrem Automaten nach, dass er bereits minimal ist.
- Die Sprache  $L = \{a^n b^n \mid n \geq 1\}$  über dem Alphabet  $\Sigma = \{a, b\}$  ist bekanntlich kontextfrei. Ordnen Sie ihr Komplement  $\Sigma^* \setminus L$  bestmöglich in die Chomskyhierarchie ein (mit detaillierter Begründung). Sie können sich dazu überlegen, welche Möglichkeiten es für ein Wort  $w$  gibt, die Mitgliedschaft in  $L$  zu verfehlten.
- Wie man weiß, sind die kontextfreien Sprachen unter Schnitt mit regulären Sprachen abgeschlossen. Es ist auch wohlbekannt, dass die Sprache  $\{a^n b^n c^n \mid n \geq 0\}$  nicht kontextfrei ist. Benutzen Sie diese Tatsachen (ohne Beweis!), um nachzuweisen, dass die Sprache  $L = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$  nicht kontextfrei ist. Erinnerung:  $|w|_x$  bezeichnet für  $x \in \{a, b, c\}$  die Anzahl der Symbole  $x$  in  $w$ .

**Aufgabe 2:**

Die Ackermannfunktion genügt bekanntlich den Gleichungen  $a(0, y) = y + 1$  und  $a(x + 1, 0) = a(x, 1)$  und  $a(x + 1, y + 1) = a(x, a(x + 1, y))$  und wächst in beiden Argumenten streng monoton.

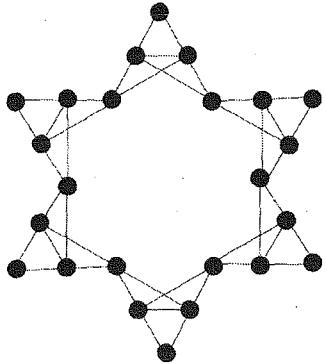
Eine Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  heiße *Ackermann-beschränkt*, wenn ein  $k \in \mathbb{N}$  existiert, sodass  $f(y) \leq a(k, y)$  für alle  $y \in \mathbb{N}$ . Zeigen Sie:

- $a(1, y) = y + 2$  und  $a(2, y) = 2y + 3$ .
- Sind  $f, g$  beide Ackermann-beschränkt, so auch  $f + g$ . Sie dürfen ohne Beweis verwenden: Für  $x > 2$  gilt:  $2a(x, y) \leq a(x + 1, y)$ . Hinweis:  $u + v \leq 2 \cdot \max(u, v)$ .
- Ist  $f$  Ackermann-beschränkt, so auch die Funktion  $g$ , die durch  $g(0) = 0$ ,  $g(y + 1) = f(g(y))$  definiert ist (also kurz  $g(y) = f^{(y)}(1)$ ).

**Aufgabe 3:**

Beim Graphenfärbungsproblem 3COL geht es darum, von einem gegebenen ungerichteten Graphen  $G = (V, E)$  zu entscheiden, ob er mit drei Farben gefärbt werden kann, also ob eine Funktion  $c : V \rightarrow \{0, 1, 2\}$  existiert, sodass aus  $\{u, v\} \in E$  folgt  $c(u) \neq c(v)$ . Bekanntlich ist dieses Problem NP-vollständig.

- a) Färben Sie den hier skizzierten Graphen mit drei Farben und begründen Sie dann, dass jede Färbung mit drei Farben den Zacken (die Knoten mit nur zwei Nachbarn) dieselbe Farbe zuweist.



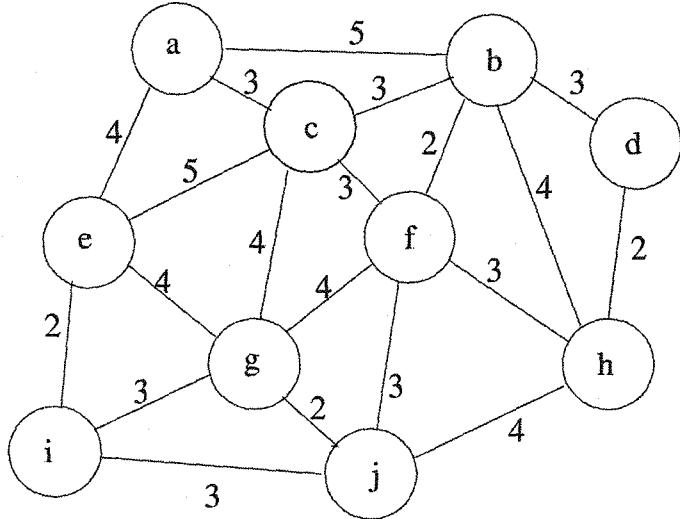
- b) Der obige Stern ist aus sechs Kopien eines "Eiffelturms"  zusammengesetzt. Begründen Sie, dass auch analog gebildete Sterne mit mehr als sechs Zacken auch mit drei Farben gefärbt werden können, aber wiederum nur so, dass alle Zacken gleichfarbig sind.
- c) Das Problem 3COL4 ist die Einschränkung des Problems 3COL auf Graphen, bei denen jeder Knoten höchstens vier Nachbarn hat.

Zeigen Sie durch Angabe einer geeigneten Reduktion von 3COL auf 3COL4, dass bereits 3COL4 NP-schwierig (NP-hart) ist (mit Begründung). Idee: Knoten mit vielen Nachbarn durch Sterne wie in Teilaufgabe b) ersetzen.

**Aufgabe 4:**

Professor Laksurk schlägt vor, minimale Spannbäume mit folgender Greedy-Strategie zu berechnen: Sortiere die Kanten nach absteigendem Gewicht, entferne solange Kanten in dieser Reihenfolge, wie der Graph (nach dem Entfernen) noch zusammenhängt.

- a) Berechnen Sie zunächst mit einem der Standardalgorithmen einen minimalen Spannbaum des hier dargestellten Graphen. Dokumentieren Sie die Arbeitsschritte geeignet.



- b) Berechnen Sie nunmehr einen minimalen Spannbaum mit Prof. Laksurks Verfahren. Die beiden Bäume sollten dasselbe Gewicht haben, denn die Strategie des Professors funktioniert tatsächlich!
- c) Versuchen Sie nun die Korrektheit von Prof. Laksurks Strategie zu beweisen. Sie können hierfür die folgende Invariante einsetzen: "zu jedem Zeitpunkt enthält der noch vorhandene Graph einen minimalen Spannbaum des ursprünglichen Graphen". Dieser Teil ist vergleichsweise schwierig. Auch Ideen oder Teillösungen werden gewertet.
- d) Mit welchem Algorithmus könnte man die Bedingung "der Graph hängt noch zusammen" überprüfen?
- e) Geben Sie eine möglichst gute obere Schranke für die Laufzeit des Verfahrens unter Verwendung Ihrer Lösung von Teilaufgabe d) an. Verwenden Sie dabei die  $O$ -Notation. Vergleichen Sie die so ermittelte Laufzeit mit der eines Standardalgorithmus.

Thema Nr. 2  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

### Aufgabe 1

Gegeben sei die Menge A:

A sei die Menge der natürlichen Zahlen n, die bei der Division durch 7 den Rest 3 haben.

$$A = \{n \in \mathbb{N} \mid n \bmod 7 = 3\}$$

- a) Geben Sie eine Turing-Maschine TM ( $TM = (Z, \Sigma, \Gamma, \delta, z_0)$ ) an, die die Menge A entscheidet. Die Überführungsfunktion können Sie als Tabelle oder Zustandsübergangsgraph darstellen.  
Ermitteln Sie die Zeitkomplexität Ihrer TM.
- b) Ist die Menge A semi-entscheidbar? Begründen Sie Ihre Entscheidung, indem Sie geeignete Änderungen an der Turingmaschine von a) vornehmen.
- c) Beweisen Sie den folgenden Satz: Eine Menge A ist genau dann entscheidbar, wenn sowohl A als auch das Komplement von A semi-entscheidbar sind.
- d) Beschreiben Sie das allgemeine Halteproblem. Begründen Sie informell, dass es nicht entscheidbar ist.

### Aufgabe 2

Sind die folgenden Behauptungen zu regulären Ausdrücken wahr? Beweisen Sie Ihre Aussage.

- a)  $(R + S)^* = R^* + S^*$
- b)  $(R + RS)^*R = R(SR + R)^*$

Hinweis:  $R^* = \bigcup_{i \geq 0} R^i$

### Aufgabe 3

Die Funktion f sei eine berechenbare Funktion auf den natürlichen Zahlen mit  $f(x) = 2x$ .

- a) Ist die Funktion f LOOP- oder WHILE-berechenbar?
- b) Geben Sie ein LOOP- oder WHILE-Programm an, das die Funktion f berechnet.

**Aufgabe 4**

Sei  $\Sigma = \{0, 1, \$\}$ , und sei  $w \in \Sigma^*$ .  $\#_0(w)$  gibt an, wie oft die 0 in  $w$  vorkommt. Zum Beispiel ist  $\#_0(0010) = 3$ ,  $\#_0(01101) = 2$ ,  $\#_0(1\$\$0\$) = 1$ .

- a) Sei

$$L_1 = \{\alpha \$ \beta \mid \alpha, \beta \in \{0, 1\}^*, \#_0(\alpha) = \#_0(\beta)\}.$$

Beispiele:  $11100\$1101011 \in L_1$ ,  $\$ \in L_1$ .

- (a1) Zeigen Sie, dass  $L_1$  kontextfrei ist, indem Sie eine kontextfreie Grammatik  $G$  angeben mit  $L(G) = L_1$ .

- (a2) Beschreiben Sie, warum Ihre Grammatik genau die Sprache  $L_1$  erzeugt.

- b) Formulieren Sie das Pumping-Lemma für kontextfreie Sprachen:

*„Sei  $L$  eine kontextfreie Sprache über dem Alphabet  $\Sigma$ . Dann gibt . . .“*

- c) Zeigen Sie mittels Pumping-Lemma für kontextfreie Sprachen, dass die Sprache

$$L_2 = \{\alpha \$ \beta \$ \gamma \mid \alpha, \beta, \gamma \in \{0, 1\}^*, \#_0(\alpha) = \#_0(\beta) = \#_0(\gamma)\}$$

nicht kontextfrei ist.

Beispiele:  $0100\$011010\$11000 \in L_2$ ,  $00\$00\$00 \in L_2$

**Aufgabe 5**

Sind folgende Aussagen *richtig* oder *falsch*?

- a) Die Suche nach einem Element in der Datenstruktur Keller/Stapel hat bei  $n$  Elementen den Aufwand  $O(n)$ .
- b) Um  $n$  Elemente zu sortieren, braucht das Sortieren durch Zerlegen (Quicksort) zusätzlichen Speicherplatz in der Größenordnung von  $(\log n)$ .
- c) Die Haldensortierung (Heapsort) hat einen niedrigeren Speicherplatzbedarf als das Sortieren durch Mischen (Mergesort).
- d) Ein Blattbaum kann auch dann die AVL-Eigenschaft erfüllen, wenn er nicht balanciert ist.
- e) Das Sortieren durch Mischen (Mergesort) hat für das Sortieren einer Reihung von  $n$  Zahlen im schlechtesten Fall einen Aufwand von  $O(n^2)$ .
- f) Das Sortieren durch Zerlegen (Quicksort) hat für das Sortieren einer Reihung von  $n$  Zahlen im schlechtesten Fall einen Aufwand von  $O(n^2)$ .

**Aufgabe 6**

Gegeben sei eine Reihung  $a[i]$  ( $i = 0, 1, \dots, n-1$ ) gefüllt mit paarweise verschiedenen ganzen Zahlen.

Ergänzen Sie folgendes Programmfragment so, dass die Elemente mittels Quicksort aufsteigend sortiert werden. Geben sie dazu den Code an, der in die Bereiche Ergänzung 1 und Ergänzung 2 einzusetzen ist.

```
// vertauscht zwei Positionen einer Reihung
static void swap (int a[], int i, int j){
    int t=a[i]; a[i]=a[j]; a[j]=t;
}
// sortiert die komplette Reihung
static void quickSort (int a[]){
    quickSort(a, 0, a.length - 1);
}
// sortiert den Bereich von start bis end (inklusive)
static void quickSort (int a[] , int start , int end) {
    // Ergänzung 1
    ...
    int pivot = a[start];
    int i = start ;
    int k = end;
    while (i < k) {
        while (i<k && a[i] < pivot) i++;
        while (i<k && a[k] > pivot) k--;
        if (i < k) swap (a, i, k);
    }
    // Ergänzung 2
    ...
}
```

**Aufgabe 7**

Gegeben ist folgender Ausschnitt der Implementierung einer Datenstruktur X:

```

public class E {
    E v1 = null;
    E v2 = null;
    int w;
}

public class X {
    E e;

    public boolean m1() {
        return e == null;
    }

    public int m2() {
        if (m1()) {
            return 0;
        } else {
            return e.w;
        }
    }

    public int m3() {
        int i = 0;
        E e1 = e;
        while (e1 != null) {
            e1 = e1.v2;
            i++;
        }
        return i;
    }

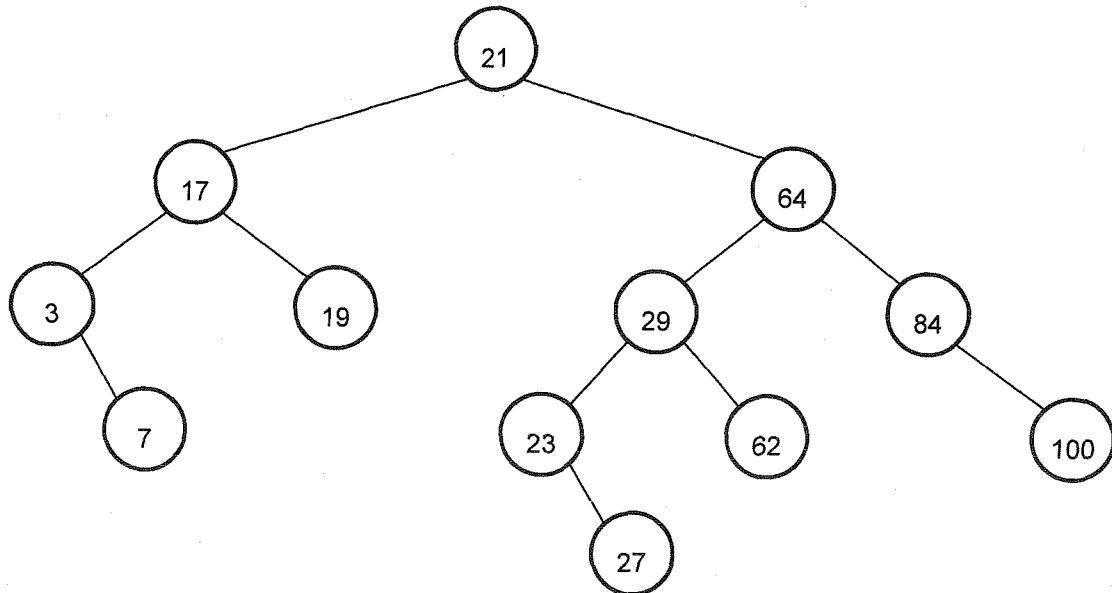
    public void m4(int j) {
        if (!m1()) {
            E e2 = new E();
            e2.v2 = e;
            e2.w = j;
            e = e2;
        } else {
            e = new E();
            e.w = j;
        }
    }
}

```

- Erläutern Sie, um welche Datenstruktur es sich hier handelt und welche Funktionalität die Methoden m1(), m2(), m3() und m4(int j) besitzen.
- Beschreiben Sie in Stichworten unter Verwendung von veranschaulichenden Skizzen, wie beim Löschen eines ersten Vorkommens eines Elements mit einem bestimmten Wert w1 aus dieser Datenstruktur prinzipiell vorgegangen werden muss.
- Implementieren Sie eine Methode, welche die in Teilaufgabe b) beschriebene Funktionalität besitzt. Verwenden Sie dabei eine gängige objektorientierte Programmiersprache oder einen entsprechenden Pseudocode und kommentieren Sie Ihre Lösung.

**Aufgabe 8**

- Erläutern Sie in Stichpunkten die Vor- und Nachteile der Datenstruktur AVL-Baum (jeweils mit Begründung).
- Geben Sie für den folgenden AVL-Baum für alle Knoten jeweils die Balancierung an.



- Erläutern Sie, ob es sich bei dem Baum aus Teilaufgabe b) immer noch um einen AVL-Baum handelt, wenn der Knoten mit dem Wert 62 entfernt wird. Skizzieren Sie gegebenenfalls erforderliche Maßnahmen, um den nach der Löschung des oben genannten Knotens entstandenen Baum wieder in einen AVL-Baum zu überführen.
- Geben Sie die Folge der Knoten an, wenn der in Aufgabe b) gegebene Baum in preorder-Reihenfolge traversiert wird.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2010**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl:		
Kennwort:		
Arbeitsplatz-Nr.:		

**Herbst  
2010**

**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**  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Hinweis:

Die einzelnen Teilaufgaben bauen oftmals aufeinander auf, sind aber im Prinzip in beliebiger Reihenfolge lösbar. Sie dürfen hierbei die Angaben und Ergebnisse früherer Teilaufgaben uneingeschränkt zur Lösung späterer Teilaufgaben verwenden! Außerdem dürfen Sie Tatsachen aus dem Informatik-Duden ohne weitere Begründung als bekannt voraussetzen.

**Aufgabe 1:**

Wir fixieren das Alphabet  $\Sigma = \{a, b\}$  und definieren  $L \subseteq \Sigma^*$  durch

$$L = \{w \mid \text{in } w \text{ kommt genau einmal das Teilwort } aaba \text{ vor}\}$$

Z.B. ist  $baababb \in L$ , aber  $baabaabab \notin L$ .

- a) Zeigen Sie, dass  $L$  regulär ist.
  - b) Konstruieren Sie den Minimalautomaten für  $L$ .
  - c) Geben Sie die Äquivalenzklassen der Myhill-Nerode Äquivalenz von  $L$  durch Repräsentanten an. (Diese Äquivalenz ist definiert durch  $x \sim_L y \iff \forall u. xu \in L \iff yu \in L$ .)
- Tipp: Die Vereinigung aller Klassen muss  $\{a, b\}^*$  ergeben und ihre Anzahl entspricht der Zustandszahl des Minimalautomaten.

**Aufgabe 2:**

Wir fixieren wieder das Alphabet  $\Sigma = \{a, b\}$  und betrachten die Sprache  $L = \{(ab)^n a (ba)^n \mid n \geq 1\}$ .

- a) Obwohl es auf den ersten Blick nicht so aussieht, ist diese Sprache regulär. Begründen Sie das.
- b) Professor Pump versucht fälschlicherweise mithilfe des Pumpinglemmas nachzuweisen, dass  $L$  nicht regulär ist. Er schreibt:

Nehmen wir an,  $L$  sei regulär. Dann gibt es eine Pumpingzahl  $n$ . Wir betrachten  $w = (ab)^n a (ba)^n$ . Sei jetzt  $w = xyz$  eine Zerlegung derart, dass  $|xy| \leq n$  und  $|y| \geq 1$ . Laut Pumpinglemma wäre nun aber  $xz$  auch in  $L$ . Das ist ein Widerspruch, da  $xy$  nach Annahme vollständig im  $(ab)^n$ -Block von  $w$  liegt und somit  $xz$  nicht in  $L$  sein kann.

Begründen Sie detailliert, an welcher Stelle Professor Pump irrt und geben Sie eine Pumpingzahl  $n$  für  $L$  an. Legen Sie konkret dar, wie jedes Wort  $w \in L$  mit  $|w| \geq n$  so zerlegt werden kann, wie es vom Pumpinglemma garantiert wird.

- c) Weisen Sie nun durch korrekte Anwendung des Pumpinglemmas nach, dass  $L' = \{(ab)^n b (ba)^n \mid n \geq 1\}$  wirklich nicht regulär ist.

**Aufgabe 3:**

Das Multiprozessor-Scheduling-Problem ist wie folgt formuliert:

- **GEGEBEN:** Eine Zahl  $n$  von Prozessen und deren Laufzeiten  $t_1, \dots, t_n$ ; eine Zahl  $m$  von Prozessoren; eine Deadline  $D$ .
- **GESUCHT:** Eine Aufteilung der  $n$  Prozesse auf die  $m$  Prozessoren, so dass die Summe der Laufzeiten der Prozesse, die einem Prozessor zugeordnet sind, jeweils unterhalb von  $D$  bleibt.
- Die Zahlen  $n, m, t_1, \dots, t_m, D$  sind natürliche Zahlen und in Binärdarstellung gegeben.

Weisen Sie nach, dass das Multiprozessor-Scheduling-Problem NP-vollständig ist, indem Sie ein geeignetes NP-vollständiges Problem auf das Multiprozessor-Scheduling-Problem reduzieren.

**Tipp:** Es bietet sich folgender Spezialfall einer vereinfachten Variante des *Rucksackproblems* an. Gegeben:  $n$  natürliche Zahlen  $a_1, \dots, a_n$ ; gesucht: eine Auswahl  $I \subseteq \{1, \dots, n\}$  so, dass  $\sum_{i \in I} a_i = b$  mit  $b = \frac{1}{2} \sum_{i=1}^n a_i$ .

**Aufgabe 4:**

Gegeben ist eine Menge  $V$  von "Arten" und für je zwei Arten  $v_1, v_2 \in V$  eine "genetische Distanz"  $d(v_1, v_2)$  die ein Maß dafür ist, wie sehr sich die Gene von  $v_1$  von denen von  $v_2$  unterscheiden. Je größer  $d(v_1, v_2)$ , umso unterschiedlicher sind  $v_1$  und  $v_2$  aus genetischer Sicht. Es gilt natürlich  $d(v, v) = 0$  für alle  $v \in V$  und  $d(v_1, v_2) = d(v_2, v_1) \geq 0$  für alle  $v_1, v_2 \in V$ .

Gesucht ist ein nicht notwendigerweise binärer Baum ("Stammbaum"), dessen Knoten die "Arten" sind und der die Eigenschaft hat, dass die Summe der "genetischen Distanzen" zwischen Eltern und Kindern im Baum möglichst klein ist.

- Welche bekannte graphentheoretische Problemstellung liegt dieser Aufgabe zugrunde?
- Nennen Sie einen Standardalgorithmus, mit dem diese Aufgabe gelöst werden kann.
- Skizzieren Sie den Ablauf des Algorithmus am folgenden Beispiel mit den Arten  $\{A, B, C, D, E, F, G\}$ . Die redundanten Einträge sind weggelassen.

d	A	B	C	D	E	F	G
A	7	19	5	16	18	16	
B		8	9	7	16	15	
C			16	5	18	17	
D				15	6	19	
E					8	9	
F						11	

**Thema Nr. 2**  
**(Aufgabengruppe)**

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1: Datentypen**

Betrachten Sie das Verhalten von Kellern (stack), Warteschlangen (queue) und Prioritätswarteschlangen (priorityqueue oder heap), in die Zahlen eingefügt und wieder gelöscht werden.

Für die Prioritäten gilt: x vor y genau dann wenn:  $x > y$ .

in(x) bedeutet, dass die Zahl x eingefügt wird (in der Java API add(E o) oder push(E o))  
out() gibt eine Zahl aus und löscht das ausgegebene Element (in der Java API pop() oder poll()).

Am Anfang sind alle Datentypen leer.

In welcher Reihenfolge werden die Zahlen bei der nachfolgenden Sequenz ausgegeben?

in(10), in(2), in(6), in(5), in(3), in(1), out(), out(), in(15), out(), out(), in(12), in(7), out(), out(), out(), in(4), out(), out(), out()

- a) bei einem Keller
- b) bei einer Warteschlange
- c) bei einer Prioritätswarteschlange (mit  $x > y$ ) ausgegeben.

**Aufgabe 2: O-Notation**

Zeigen Sie dass  $f(n) \in O(g(n))$  gilt für

$$f(n) = 4 \cdot n \cdot \log(n) + 20$$

$$g(n) = n^2$$

**Aufgabe 3: Hashing**

Gegeben sei ein Array der Größe 10, z.B. int [ ] hashfeld = new int [10].

Die Hashfunktion sei der Wert modulo 10,  $h(x) = x \% 10$ .

Kollisionen werden mit linearer Verschiebung um 1 (modulo 10) gelöst.

in(x) bedeutet, dass die Zahl x eingefügt wird,  
search(x), dass nach x gesucht wird mit den Antworten „ja“ bzw. „nein“ und  
out(x), dass x gelöscht wird, sofern x gespeichert ist.

Es wird folgende Sequenz von Operationen auf ein anfangs leeres Array ausgeführt:

in(19), in(29), in(39), in(10), out(29), out(39), search(29), in(11), in(17), out(10), in(2), in(22)

Geben Sie den Inhalt von hashfeld an

- nach search(29)
- nach out(10)
- und nach in(22).

**Fortsetzung nächste Seite!**

**Aufgabe 4: AVL Bäume**

- a) Fügen Sie nacheinander die Zahlen 10 15 22 6 7 20 1 in einen anfangs leeren AVL-Baum ein;  
 Repräsentieren (zeichnen) Sie alle AVL-Bäume jeweils vor einer notwendigen Rotation und beschreiben Sie die Rotationen (z.B. LL-Rotation für die Nummern ....)
- b) Die Zahl 15 soll anschließend gelöscht werden. Was muss man tun?

**Aufgabe 5: kürzeste Wege**

Ein gerichteter Graph G sei durch folgende Entfernungsmatrix gegeben:  
 Die 2 beim Eintrag (a,b) bedeutet, dass es eine Kante der Länge 2 von a nach b gibt.

	a	b	c	d	e	f	g	h
a	2	10	6					41
b		6		20				
c				12	9			
d			1				3	
e								6
f								14
g					14			
h								

Gesucht ist ein (der) kürzester Weg von a nach h.

- a) Beschreiben Sie, wie der Dijkstra-Algorithmus auf G arbeitet.  
 Geben Sie dazu an, in welcher Reihenfolge die Knoten bearbeitet werden, und welche Werte dort gespeichert werden.
- b) Wie lang ist der kürzeste Weg von a nach h?
- c) Der Dijkstra-Algorithmus soll nur auf Graphen mit Kanten mit nichtnegativer Länge angewendet werden.  
 Erläutern Sie, warum das so ist.

**Aufgabe 6: reguläre Mengen**

Gegeben sei die Menge aller Wörter  $w \in \{a,b\}^*$ , der Länge mindestens 2, bei denen das zweite und das zweitletzte Zeichen gleich sind.

Zeigen Sie, dass diese Sprache regulär ist.

Tipp: Beachten Sie auch die Wörter der Längen 2 und 3.

**Aufgabe 7: kontextfreie Sprachen**

Gegeben sei die Grammatik  $G = (\{S\}, \{a,b\}, S, P)$  mit den Produktionen

$$\begin{aligned} S &\rightarrow SS, \\ S &\rightarrow aSb, \\ S &\rightarrow bSa \text{ und} \\ S &\rightarrow \epsilon \end{aligned}$$

- Welche Eigenschaft haben die Wörter  $w \in L(G)$  bezüglich der Anzahl der Zeichen a und b?
- Geben Sie eine Ableitung (Ableitungsbaum) an für das Wort a a a b b b b a a.
- Geben Sie alle Wörter der Länge vier von  $L(G)$  an.
- Ist  $L(G)$  regulär? Begründen Sie Ihre Antwort.

Als Begründung müssen Sie entweder eine reguläre Beschreibung (Automat, Ausdruck) angeben oder den Widerspruch begründen. Dabei darf das Pumping Lemma als bekannt vorausgesetzt werden. Man muss das passende Pumping Lemma geeignet anwenden.

**Aufgabe 8: Turingmaschinen**

Konstruieren Sie eine Turingmaschine M mit  $L(M) = L$ , wobei  $p, q \geq 1$ .

Beschreiben Sie zusätzlich, wie M arbeitet (Stil: M liest das Zeichen a und speichert ....)

$$L = \{w \mid w \in \{a, b\}^*, w \text{ besteht aus } p \text{ Zeichen a und aus } q \text{ Zeichen b}\}.$$

**Aufgabe 9: NP und Graph Färbbarkeit**

Zeigen Sie durch Angabe eines nicht-deterministischen Verfahrens, dass 3-Färbbarkeit in NP liegt.

Instanz: Ein Graph  $G = (V, E)$

Problem: Gibt es eine 3-Färbung der Knoten, so dass benachbarte Knoten verschiedene Farben haben.

Hinweis: 3-Färbbarkeit ist sogar NP-schwer. Das muss nicht gezeigt werden.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2011**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl:		
Kennwort:		
Arbeitsplatz-Nr.:		

**Frühjahr  
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: **12**

**Bitte wenden!**

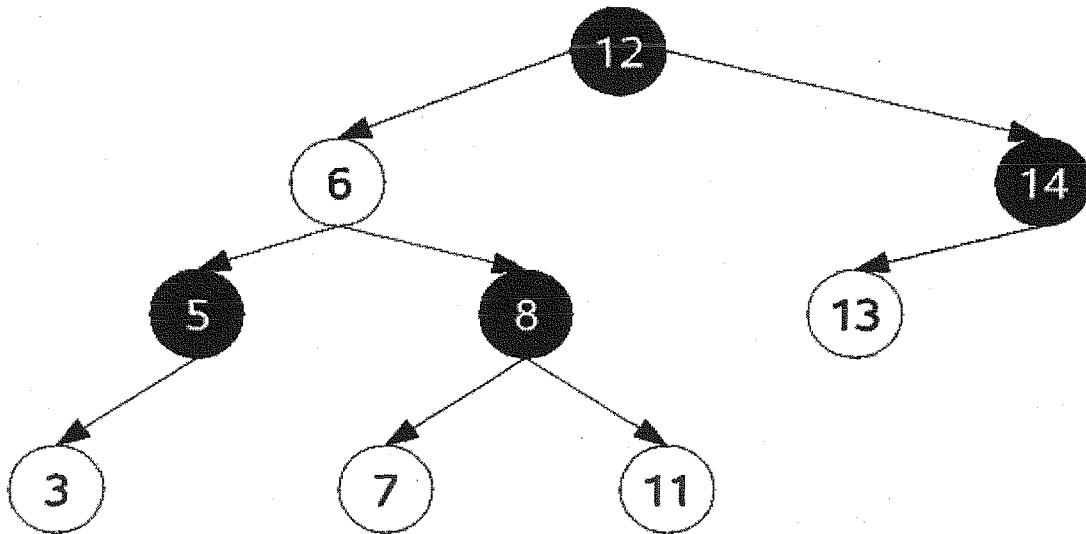
**Thema Nr. 1****Aufgabe 1:**

Bestimmen Sie mit Hilfe des Master-Theorems für die folgenden Rekursionsgleichungen möglichst scharfe asymptotische untere und obere Schranken, falls das Master-Theorem anwendbar ist!

Geben Sie andernfalls eine kurze Begründung, warum das Master-Theorem nicht anwendbar ist!

- a)  $T(n) = 16T(n/2) + 40n - 6$
- b)  $T(n) = 27T(n/3) + 3n^3 \log n$
- c)  $T(n) = 4T(n/2) + 3n^2 + \log n$
- d)  $T(n) = 4T(n/16) + 100 \log n + \sqrt{2n} + n^{-2}$

**Fortsetzung nächste Seite!**

**Aufgabe 2:**

- Zeichnen Sie die beiden Rot-Schwarz-Bäume (die roten Knoten sind weiß gekennzeichnet), die entstehen, wenn man nacheinander die Schlüssel 2 und 1 in den obigen Baum gemäß dem Einfügealgorithmus für Rot-Schwarz-Bäume einfügt.
- Zeichnen Sie die beiden Rot-Schwarz-Bäume, die entstehen, wenn man nacheinander die Schlüssel 7 und 6 aus dem in Teilaufgabe a) angegebenen Rot-Schwarz-Baum gemäß dem Löschalgorithmus für Rot-Schwarz-Bäume entfernt.

**Hinweis:** Falls Sie die Zwischenschritte geeignet dokumentieren, können auch für teilweise richtige Lösungen entsprechend Punkte vergeben werden.

**Aufgabe 3**

Ein Weinkellner benutzt das folgende System zur Lagerung seiner Weine. Die  $n$  Flaschen ( $n \geq 8$ ) werden in 3 Kategorien A, B und C eingeteilt:

**Kategorie A** soll möglichst wenige Flaschen enthalten, die zusammen mindestens 60% des Wertes ausmachen - falls dies möglich ist: Diese werden dann in einem speziellen Weinkühlschrank gelagert, dessen Kapazität  $n/\log n$  Flaschen fasst. Falls die  $n/\log n$  teuersten Flaschen also zusammen einen Wert von weniger als 60% haben, fallen eben die  $n/\log n$  teuersten Flaschen in Kategorie A.

**Kategorie C** enthält die 60% der Flaschen, d. h.  $\lfloor 0,6n \rfloor$  Stück, die den geringsten Wert haben. Diese sind zum alltäglichen Genuss bestimmt und werden aufrecht stehend im Vorratsschrank gelagert.

Die restlichen Flaschen bilden die **Kategorie B**, diese lagern im Keller im Weinregal.

Hinweis:  $\lfloor x \rfloor$  bezeichnet die größte ganze Zahl, die kleiner oder gleich  $x$  ist.

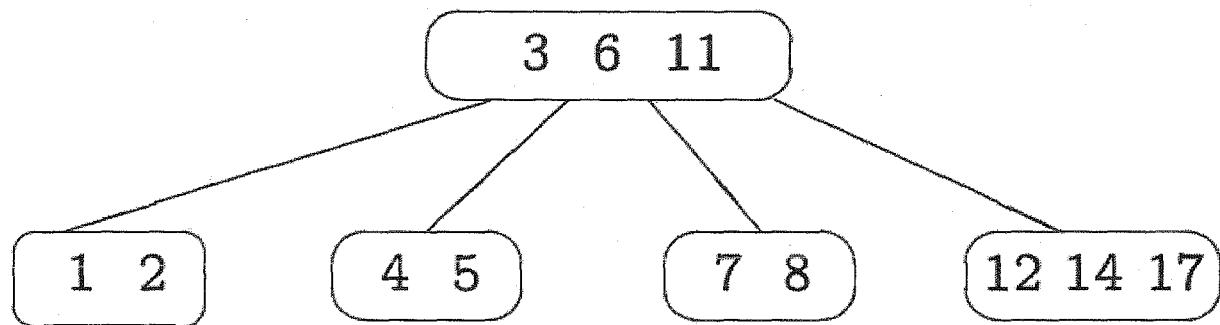
Ein möglicher Algorithmus zur Klassifikation der Weinflaschen besteht darin, diese zuerst (möglichst effizient) nach ihrem Wert zu sortieren. Dann wird solange die jeweils wertvollste Flasche zur Kategorie A hinzugefügt, bis 60% des Wertes oder  $n/\log n$  Flaschen erreicht sind. Anschließend wird  $\lfloor 0,6n \rfloor$  mal die Flasche mit dem geringsten Wert in die Kategorie C eingereiht. Die verbleibenden Flaschen bilden Kategorie B.

- a) Konstruieren Sie ein Beispiel mit  $n = 8$  Flaschen, denen Sie Werte so zuordnen, dass mehr als 60% des Gesamtwertes in Kategorie A fallen.
- b) Geben Sie einen Algorithmus zur Klassifikation an, der eine asymptotisch bessere Laufzeit im worst-case hat, und begründen Sie dies. Sie können bekannte Algorithmen und Datenstrukturen verwenden.

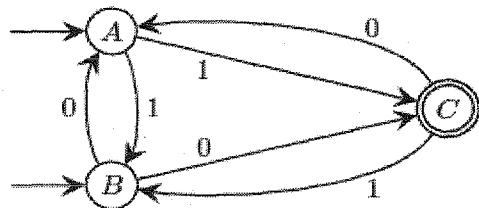
**Fortsetzung nächste Seite!**

**Aufgabe 4: B-Bäume**

- a) Was ist ein B-Baum vom Grad m? Geben Sie alle Bedingungen an, die an so einen Baum gestellt werden.
- b) Wofür werden B-Bäume verwendet?
- c) Fügen Sie in folgenden B-Baum vom Grad 2 das Element 42 ein und zeichnen Sie den entstehenden Baum.



- d) Löschen Sie aus obigem Baum die 4 und die 5 und zeichnen Sie das Resultat.

**Aufgabe 5**

Der endliche Automat N mit diesem Zustandsgraphen ist nichtdeterministisch. Konstruieren Sie einen deterministischen endlichen Automaten M, der dieselbe Sprache  $L \subseteq \{0, 1\}^*$  akzeptiert wie N.

*Vergessen Sie nicht, Start- und Endzustände von M anzugeben.*

**Fortsetzung nächste Seite!**

**Aufgabe 6**

Für  $z \in \Sigma^*$  sei  $|z|_a$  die Anzahl der a und  $|z|_b$  die Anzahl der b und  $|z|_c$  die Anzahl der c im Wort z.

Sei  $\Sigma = \{a, b, c\}$  und  $L = \{ z \in \Sigma^* \mid |z|_a < |z|_b + |z|_c \}$ .

Beweisen Sie, dass L nicht vom Typ 3 (regulär) ist.

*Hinweis: Es gibt verschiedene Möglichkeiten für den Beweis.*

**Aufgabe 7**

Sei  $G = (V, \Sigma, P, S)$  mit Startsymbol  $S$  und  $V = \{S, T, A, B\}$  und  $\Sigma = \{a, b\}$  und

$P = \{S \rightarrow AB, \quad S \rightarrow BT, \quad A \rightarrow BA, \quad B \rightarrow TT, \quad A \rightarrow a, \quad B \rightarrow b,$   
 $T \rightarrow AB, \quad T \rightarrow a\}$

Sei  $L = L(G)$  die von dieser Grammatik erzeugte Sprache.

Sei  $w = baaab$

a) Wenden Sie den Algorithmus von Cocke, Younger und Kasami (CYK) auf  $w$  an.

$b$	$a$	$a$	$a$	$b$
$B$	$A, T$	$A, T$	$A, T$	$B$

Es gilt  $w \in L$ . Woran sieht man das?

b) Mit dem Verfahren kann man zeigen, ob es einen Syntaxbaum mit Wurzel  $S$  für ein gegebenes Wort gibt. Tatsächlich hat  $w$  sogar mehrere verschiedene Syntaxbäume mit Wurzel  $S$ . Das Verfahren kann so modifiziert werden, dass es solche Fälle erkennt.

*Beschreiben Sie*

- i) eine *sehr kleine* dafür geeignete Modifikation des Verfahrens,
- ii) das Kriterium, mit dem man im modifizierten Verfahren sieht, ob es *mehrere* Syntaxbäume mit Wurzel S für ein gegebenes Wort gibt.
- iii) Geben Sie alle echten Teilwörter von w an, die mindestens zwei Syntaxbäume mit Wurzel S haben.

### Aufgabe 8

Gegeben sei

$$\begin{array}{rcl} h : \mathbb{N} \times \mathbb{N} & \rightarrow & \mathbb{N} \\ (y, x) & \mapsto & \text{absdiff}(x, \text{mult}(y, x)) \end{array}$$

Multiplikation	absolute Differenz
$\text{mult} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ $(x, y) \mapsto x \cdot y$	$\text{absdiff} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ $(x, y) \mapsto  x - y $

*Sie können voraussetzen, dass mult und absdiff primitiv rekursiv sind.*

Zeigen Sie:

h ist primitiv rekursiv.

Geben Sie zum Nachweis eine Definition von h an, die strikt nach den syntaktischen

Vorgaben des Kompositions- und/oder Rekursionsschemas für primitive Rekursion aufgebaut ist.

## Thema Nr. 2

### 1. Aufgabe (reguläre Sprachen und endliche Automaten)

Die Elemente einer regulären Sprache können durch deterministische oder nicht-deterministische endliche Automaten erkannt werden.

Betrachten Sie folgenden nicht-deterministischen endlichen Automaten  $A_1 = (\{q_1, q_2, q_3, q_4, q_5\}, \{a, b\}, \delta, q_1, \{q_4\})$  mit Zustandsmenge  $\{q_1, q_2, q_3, q_4, q_5\}$ , Eingangsalphabet  $\{a, b\}$ , Anfangszustand  $q_1$  und Endzustandsmenge  $\{q_4\}$ .

Die Übergangsfunktion  $\delta$  sei durch folgende Tabelle definiert, wobei  $\epsilon$  das leere Wort bezeichnet:

$\delta$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$
a	$\{q_1, q_2\}$	$\{q_4\}$	$\emptyset$	$\{q_4\}$	$\emptyset$
b	$\{q_1\}$	$\emptyset$	$\{q_5\}$	$\{q_4\}$	$\{q_4\}$
$\epsilon$	$\{q_3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

- a) Zeichnen Sie das Übergangsdiagramm des Automaten mit Zuständen und Übergangskanten.
- b) Beschreiben Sie die von  $A_1$  erkannte reguläre Sprache  $L_1$ , indem Sie eine mathematisch exakte Definition der Menge der erkannten Worte über  $\{a, b\}$  angeben. Geben Sie einen möglichst kurzen regulären Ausdruck an, der die Sprache  $L_1$  beschreibt.
- c) Wandeln Sie den nicht-deterministischen endlichen Automaten  $A_1$  in einen deterministischen endlichen Automaten  $A_2$  um, indem Sie die Teilmengenkonstruktion anwenden.
- d) Geben Sie eine Definition der Äquivalenz von Zuständen in deterministischen endlichen Automaten.
- e) Bestimmen Sie alle äquivalenten Zustände von  $A_2$ . Bauen Sie dazu die vollständige Tabelle mit Zustandspaaren schrittweise auf und markieren Sie, ob die jeweiligen Zustände unterscheidbar sind. Erläutern Sie jeden durchgeführten Schritt. Fassen Sie anschließend die äquivalenten Zustände zusammen und konstruieren Sie den resultierenden deterministischen endlichen Automaten  $A_3$ , indem Sie für  $A_3$  ein Übergangsdiagramm und eine tabellenförmige Darstellung der Übergangsfunktion angeben.

Fortsetzung nächste Seite!

## 2. Aufgabe (kontextfreie Sprachen und Kellerautomaten)

Betrachten Sie die kontextfreie Sprache  $L = \{ww^R; w \in \{0,1\}^*\}$ . Dabei bezeichnet  $w^R$  die Umkehrung des Wortes  $w$ , d.h. für  $w = a_1 \dots a_n$  ist  $w^R = a_n \dots a_1$ .

- Beweisen Sie durch Anwendung des Pumping-Lemmas für reguläre Sprachen, dass die Sprache  $L$  nicht regulär ist. Begründen Sie die jeweiligen Schritte.
- Geben Sie eine kontextfreie Grammatik  $G$  mit Terminalsymbolen, Nichtterminalsymbolen und Produktionen an, die  $L$  erzeugt, d.h.  $L = L(G)$ . Geben Sie die Schritte zur Ableitung des Wortes  $011110 \in L$  mit dieser Grammatik an.
- Konstruieren Sie einen nicht-deterministischen Kellerautomaten  $K = (Q, \Sigma, \Gamma, \delta, q_0, z_0, f)$ , der  $L$  erkennt, d.h.  $L = L(K)$ . Geben Sie eine genaue Definition aller Elemente des Kellerautomaten mit einer mathematisch exakten Definition der Übergangsrelation  $\delta$ . Erläutern Sie die Arbeitsweise des Kellerautomaten und begründen Sie, warum  $K$  alle Worte aus  $L$  erkennt.
- Erläutern Sie den Unterschied zwischen nicht-deterministischen und deterministischen Kellerautomaten durch Angabe der exakten Definitionen. Welche Unterschiede in den Verarbeitungsschritten gibt es?
- Gibt es Sprachen, für die ein nicht-deterministischer Kellerautomat  $K_1$  konstruiert werden kann, der die jeweilige Sprache erkennt, für die aber kein deterministischer Kellerautomat existiert, der die Sprache erkennt? Verwenden Sie die Sprache  $L$  für ihre Argumentation. Begründen Sie Ihre Antwort ausführlich.
- Konstruieren Sie einen deterministischen Kellerautomaten für die Sprache  $L' = \{wcw^R \mid w \in \{0,1\}^*\}$  mit separatem Markierungszeichen  $c \notin \{0,1\}$ . Geben Sie eine mathematisch exakte Definition der Übergangsfunktion und erläutern Sie die Arbeitsweise des Kellerautomaten.

## 3. Aufgabe (Turingmaschinen)

- Konstruieren Sie eine deterministische Turingmaschine, die eine Eingabe  $x \in \{0,1\}^*$  als Binärzahl interpretiert und die Binärdarstellung der Zahl produziert, die durch Addition von 1 entsteht. Erläutern Sie die Rolle der Zustände der von Ihnen konstruierten Turingmaschine und geben Sie die Übergangsfunktion in Tabellenform an. Illustrieren Sie die Arbeitsweise der von Ihnen konstruierten Maschine, indem Sie die Berechnungsschritte für die Eingabe  $x = 101$  als Konfigurationsübergänge angeben.
- Erkennen deterministische Turingmaschinen dieselbe Sprachklasse wie nicht-deterministische Turingmaschinen oder gibt es Sprachen, die zwar von nicht-deterministischen, nicht aber von deterministischen Turingmaschinen erkannt werden können? Geben Sie eine ausführliche Begründung Ihrer Antwort. Geben Sie entweder eine Sprache an, die zwar von nicht-deterministischen, nicht aber von deterministischen Turingmaschinen erkannt werden kann, oder beschreiben Sie, wie eine beliebige nicht-deterministische Turingmaschine durch eine deterministische Turingmaschine simuliert werden kann.

**4. Aufgabe (Algorithmen und Datenstrukturen)**

Rot-Schwarz-Bäume sind balancierte Bäume, die im Gegensatz zu unbalancierten Bäumen effiziente Sucheigenschaften auch für beliebige Einfügereihenfolgen garantieren.

- (a) Geben Sie eine genaue Definition von Rot-Schwarz-Bäumen an, aus der insbesondere hervorgeht, wann ein Suchbaum ein Rot-Schwarz-Baum ist.
- (b) Beweisen Sie, dass in einem Rot-Schwarz-Baum folgende Eigenschaft gilt: Ein Rot-Schwarz-Baum mit  $n$  inneren Knoten hat höchstens die Höhe  $2 \cdot \log_2(n + 1)$ .
- (c) Erläutern Sie das effiziente Einfügen von Elementen in einem Rot-Schwarz-Baum mit Rotationen, indem Sie eine Prozedur in Pseudocodenotation angeben, die in einen beliebigen Rot-Schwarz-Baum ein Element einfügt, so dass der resultierende Baum weiterhin ein Rot-Schwarz-Baum ist.
- (d) Illustrieren Sie die Arbeitsweise Ihrer in (c) entworfenen Einfügemethode, indem Sie in einen anfangs leeren Rot-Schwarz-Baum schrittweise die Zahlen 41, 38, 31, 12, 19, 8 in dieser Reihenfolge einfügen. Geben Sie den nach jedem Schritt resultierenden Rot-Schwarz-Baum an und erläutern Sie den Ablauf des Einfügens.
- (e) Analysieren Sie die Laufzeit der von Ihnen angegebenen Einfügemethode.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2011**

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

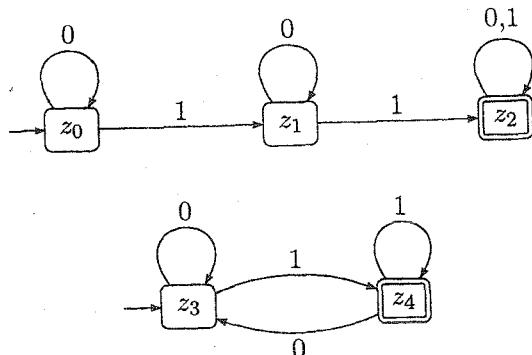
Gegeben ist die Grammatik  $G = (\{S, A, B, C\}, \{a, b\}, \Phi, S)$  mit

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

- 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  $M_1$  und  $M_2$ :



- 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  $\gamma$  an, der diese Sprache beschreibt, so dass  $\gamma$  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.

### 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  $\Phi = \{S \rightarrow aT, S \rightarrow aTbT, T \rightarrow aT, T \rightarrow aTbT, T \rightarrow \epsilon\}$  gegeben.

- a) Zeigen Sie, dass die Grammatik mehrdeutig ist.
- b) Welche Sprache wird von dieser Grammatik erzeugt?
- c) Geben Sie eine eindeutige,  $\epsilon$ -freie Grammatik an, die dieselbe Sprache erzeugt.

### Aufgabe 3: Berechenbarkeit

#### 3.1 Rekursive Aufzählbarkeit

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

#### 3.2 Entscheidbarkeit

Beweisen Sie: Wenn eine nicht-leere Menge  $M \subseteq \mathbb{N}$  eine monoton wachsende Aufzählungsfunktion  $\alpha_M$  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.

## 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;
}
```

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

- Skizzieren Sie kurz die Funktionsweise des *Heap-Sort-Algorithmus*. Stellen Sie dabei insbesondere dar, welche beiden Phasen bei diesem Algorithmus unterschieden werden.
- 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  
 $A = \{n \in \mathbb{N} \mid \text{es existieren Primzahlen } p \text{ und } q \text{ mit } n = p + q\}$ .
- b) Beweisen Sie die Aufzählbarkeit der Menge  
 $B = \{n \in \mathbb{N} \mid \text{es existieren Primzahlen } p \text{ und } q \text{ mit } n = p - q\}$ .

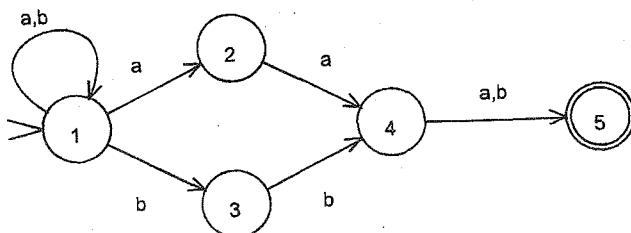
**Aufgabe 2:**

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

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

**Aufgabe 3:**

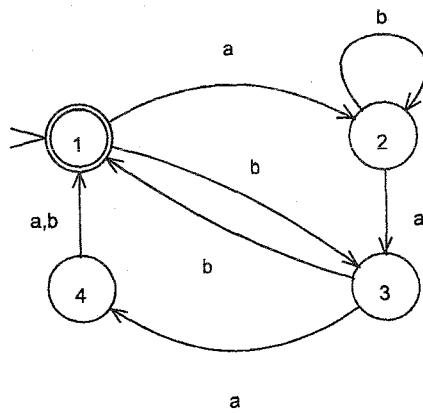
Sei  $D$  die von dem folgenden endlichen Automaten akzeptierte Sprache.



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

$$F = \{(a, b, c, d) \in \mathbb{N}^4 \mid \exists x, y \in \mathbb{N} \text{ mit } ax^3y + bx^2y^2 + cxy^3 = d\}$$

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  $w: E \rightarrow \mathbb{R}_{>0}$  eine Abbildung, die den Kanten von  $G$  ein positives Gewicht zuordnet. Dann ist ein Matching  $M_{\text{opt}}$  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.

- Zeigen Sie, dass Professor Ydeergs Algorithmus zwar immer ein Matching liefert – aber nicht immer ein schwerstes.
- 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  $M_{\text{opt}}$  in  $G$ . Was passiert, wenn Professor Ydeergs Algorithmus eine Kante  $\{u, v\}$  auswählt und die zu  $u$  und  $v$  inzidenten Kanten löscht?
- Welche Laufzeit hat der Algorithmus in Abhängigkeit von der Anzahl  $n$  der Knoten und der Anzahl  $m$  der Kanten von  $G$ ?
- Kann man diese Laufzeit verbessern, wenn man weiß, dass die Gewichtsfunktion  $w$  die Kantenmenge  $E$  in die Menge  $\{1, 2, \dots, m\}$  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.

- 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?

- 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.

- 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.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2012**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl:		
Kennwort:		66115
Arbeitsplatz-Nr.:		

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**

**— Prüfungsaufgaben —**

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 9

Bitte wenden!

Thema Nr. 1  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

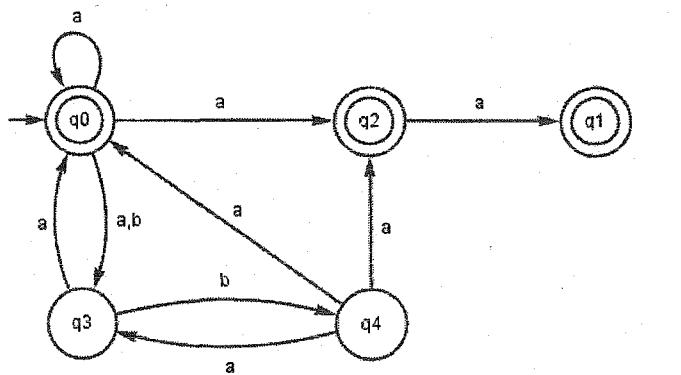
**Aufgabe 1:**

Sei  $M_0, M_1, \dots$  eine Gödelisierung der Registermaschinen (Random-Access-Maschinen, RAMs). Bestimmen Sie, ob folgende Mengen aufzählbar sind. Bestimmen Sie außerdem, ob die Mengen entscheidbar sind. Beweisen Sie Ihre Aussagen!

$$\begin{aligned} A_1 &= \{(i, t) \in \mathbb{N} \times \mathbb{N} \mid M_i \text{ hält auf Eingabe } i \text{ nicht innerhalb von } t \text{ Rechenschritten}\} \\ A_2 &= \{(i, x) \in \mathbb{N} \times \mathbb{N} \mid M_i \text{ hält nicht auf Eingabe } x\} \\ A_3 &= \{i \in \mathbb{N} \mid \text{für die von } M_i \text{ berechnete Funktion } f : \mathbb{N} \rightarrow \mathbb{N} \text{ gilt } \exists x \in \mathbb{N}, f(x) = x\} \end{aligned}$$

**Aufgabe 2:**

Gegeben ist der folgende nichtdeterministische, endliche Automat  $M$ . Konstruieren Sie einen endlichen Automaten  $M'$  mit  $L(M') = \{a, b\}^* \setminus L(M)$ .



**Aufgabe 3:**

Beweisen Sie, dass folgende Sprache kontextfrei, aber nicht regulär ist.

$$C = \{a^n b^m \mid n \geq m \geq 1\}$$

**Aufgabe 4:**

Gegeben ist die kontextfreie Grammatik  $G = (\Sigma, N, S, R)$  mit  $\Sigma = \{a, b\}$ ,  $N = \{S, A, B\}$  und  $R = \{S \rightarrow A, S \rightarrow B, A \rightarrow aAb, B \rightarrow AA, B \rightarrow bBa, A \rightarrow a\}$ . Geben Sie eine äquivalente Grammatik in Chomsky-Normalform an.

**Aufgabe 5:**

Gegeben sind zwei Varianten des Teilsummenproblems (sum of subset, SOS). Beweisen Sie:  $S_1$  ist auf  $S_2$  in Polynomialzeit reduzierbar.

$$S_1 = \{(a_1, \dots, a_m, c) \mid m, a_1, \dots, a_m, c \in \mathbb{N} \text{ und } \exists b_1, \dots, b_m \in \{0, 1, 2\} \text{ mit } \sum_{i=1}^m b_i \cdot a_i = c\}$$

$$S_2 = \{(a_1, \dots, a_m, c) \mid m, a_1, \dots, a_m, c \in \mathbb{N} \text{ und } \exists I \subseteq \{1, \dots, m\} \text{ mit } \sum_{i \in I} a_i = c\}$$

**Aufgabe 6:****Binäre Suchbäume**

- a) Gegeben sei eine Folge von  $n$  Zahlen, über deren Verteilung nichts bekannt ist. Es soll ein binärer Suchbaum konstruiert werden, der die Zahlen in dieser Folge speichert. Argumentieren Sie, warum es dafür keinen Algorithmus mit linearer Worst-Case-Laufzeit geben kann. (Um für zwei Zahlen  $a$  und  $b$  zu entscheiden, ob  $a \leq b$ , gehen wir davon aus, dass eine Methode  $\text{compare}(a, b)$  verwendet werden muss. Diese liefert in  $O(1)$  Zeit `true`, falls  $a \leq b$  und sonst `false`.)
- b) Angenommen die Zahlenfolge sei aufsteigend sortiert. Geben Sie nun einen Linearzeitalgorithmus an, der die Zahlen in dieser Folge in einem binären Suchbaum speichert.
- c) Erweitern oder modifizieren Sie Ihren Algorithmus so, dass er in Linearzeit einen *balancierten* Binärbaum, also etwa einen Rot-Schwarz-Baum oder einen AVL-Baum ausgibt, der die Zahlen in der gegebenen Folge enthält.

Zeigen Sie, dass der von Ihrem Algorithmus konstruierte Baum tatsächlich die Eigenschaften eines Rot-Schwarz- oder AVL-Baums besitzt.

### Aufgabe 7:

#### Kürzeste Kreise

Mit der *Länge* eines Pfads oder eines Kreises bezeichnen wir die Anzahl der Kanten, aus denen der Pfad bzw. der Kreis besteht. Bekanntlich kann man Breitensuche verwenden, um für zwei gegebene Knoten  $s$  und  $t$  die Länge eines kürzesten  $s-t$ -Wegs zu berechnen. Im folgenden geht es um die Berechnung kürzester Kreise.

- a) Für einen Graphen  $G$  und einen Knoten  $v$  von  $G$  berechnet  $\text{KK}(G, v)$  (siehe Abbildung 1) die Länge des kürzesten Kreises in  $G$ , der durch  $v$  geht.

Analysieren Sie die Laufzeit von  $\text{KK}$  in Abhängigkeit von der Anzahl  $n$  der Knoten von  $G$ , von der Anzahl  $m$  der Kanten von  $G$  und vom Grad  $\deg(v)$  des übergebenen Knotens  $v$ .

- b) Wenn man den Algorithmus  $\text{KK}$  für jeden Knoten eines Graphen  $G$  aufruft, kann man die Länge eines kürzesten Kreises in  $G$  berechnen. Welche Laufzeit hat der resultierende Algorithmus in Abhängigkeit von  $n$  und  $m$ ?
- c) Geben Sie einen Algorithmus  $\text{KKschnell}(G, v)$  an, der in  $O(n+m)$  Zeit die Länge des kürzesten Kreises in  $G$  berechnet, der durch  $v$  geht.

Argumentieren Sie, warum ihr Algorithmus korrekt ist.

Abbildung 1

$\text{KK}(\text{ungerichteter ungewichteter Graph } G = (V, E), \text{Knoten } v)$

```

1  $L := \infty$ 
2  $\text{Adj}[v] := \{w \in V \mid \{v, w\} \in E\}$ 
3 foreach  $w \in \text{Adj}[v]$  do
4   Sei  $G'$  der Graph  $G$  ohne die Kante  $\{v, w\}$ .
5   Sei  $\ell$  die Länge eines kürzesten  $v-w$ -Wegs in  $G'$ .
6   if  $\ell < L$  then
7      $L := \ell$ 
8 return  $L$ 

```

Thema Nr. 2  
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Für natürliche Zahlen  $a, b$  mit  $b > 0$  bezeichne  $\text{div}(a, b)$  den Quotienten und  $\text{mod}(a, b)$  den Rest der ganzzahligen Division von  $a$  durch  $b$ , d.h. es gilt

$$a = b \cdot \text{div}(a, b) + \text{mod}(a, b) \quad \text{mit } 0 \leq \text{mod}(a, b) < b,$$

und die Divisionseigenschaft der ganzen Zahlen besagt, dass  $\text{div}(a, b)$  und  $\text{mod}(a, b)$  durch diese Beziehung eindeutig bestimmt sind. Für  $b = 0$  setzen wir  $\text{div}(a, 0) = \text{mod}(a, 0) = 0$  fest.

Mit  $\text{ggT}(a, b)$  wird der grösste gemeinsame Teiler von  $a$  und  $b$  bezeichnet, wobei  $\text{ggT}(a, 0) = a$  ist (vereinbarungsgemäß auch für den Fall  $a = 0$ ).

- a) Zeigen Sie, dass  $\text{mod}$  eine primitiv-rekursive Funktion ist.
- b) Zeigen Sie, dass  $\text{div}$  eine primitiv-rekursive Funktion ist.
- c) Geben Sie ein LOOP-Programm zur Berechnung von  $\text{ggT}$  an.

**Aufgabe 2:**

Welche folgenden Behauptungen über Sprachen  $A, B \subseteq \Sigma^*$  sind korrekt und welche sind falsch.  
Begründen Sie Ihre Antwort!

$A \setminus B$  bezeichnet die Mengendifferenz.

- a) Sind  $A$  und  $B$  regulär, so ist auch  $A \setminus B$  regulär.
- b) Sind  $A$  und  $B$  kontextfrei, so ist auch  $A \setminus B$  kontextfrei.
- c) Sind  $A$  und  $B$  entscheidbar, so ist auch  $A \setminus B$  entscheidbar.
- d) Sind  $A$  und  $B$  partiell-entscheidbar, so ist auch  $A \setminus B$  partiell-entscheidbar.
- e) Sind  $A$  und  $B$  polynomiell-entscheidbar, d.h. in der Komplexitätsklasse P, so ist auch  $A \setminus B$  polynomiell-entscheidbar.

**Aufgabe 3:**

Es sei  $\Sigma$  ein endliches Alphabet. Die Menge  $\Gamma$  bestehe aus allen kontextfreien Grammatiken mit Terminalalphabet  $\Sigma$ . Das Problem  $L \subseteq \Gamma \times \Sigma^*$  bestehe aus allen Paaren  $(G, w) \in \Gamma \times \Sigma^*$  mit  $G \vdash^* w$ , d. h.  $w$  ist mit Hilfe von  $G$  ableitbar.

$IS$  bezeichne das Problem INDEPENDENTSET - ein bekanntes NP-vollständiges Problem.

Kommentieren Sie die beiden folgenden Reduktions-Behauptungen unter der Hypothese, dass  $P \neq NP$  gilt:

1.  $IS \leq_p L$ ,
2.  $L \leq_p IS$ .

Welche der Behauptungen treffen zu bzw. treffen nicht zu? Mit Begründung! Hinweis: In welcher Komplexitätsklasse liegt  $L$ ?

**Aufgabe 4:**

Die drei Symbolmengen

$$\Sigma_0 = \{a, b, c, d, e, f\}, \Sigma_1 = \{-\}, \Sigma_2 = \{+, *, /\}$$

ergeben das Alphabet  $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ . Die Elemente von  $\Sigma_i$  stellen  $i$ -stellige Funktionssymbole dar ( $i \in \{0, 1, 2\}$ ). Die nullstelligen Funktionssymbole  $a, b, c, d, e, f$  sind symbolische Konstanten.

Die Menge der *Terme*  $\text{Te}$  ist folgendermaßen definiert:

- jedes  $x \in \Sigma_0$  gehört zu  $\text{Te}$ ;
- für jedes  $y \in \Sigma_1$  und jeden Term  $t \in \text{Te}$  gehört  $y t$  zu  $\text{Te}$ ;
- für jedes  $z \in \Sigma_2$  und je zwei Terme  $s, t \in \text{Te}$  gehört  $z s t$  zu  $\text{Te}$ .

- Geben Sie eine kontextfreie Grammatik  $G$  an, von der die Menge der Terme generiert wird:  
 $\mathcal{L}(G) = \text{Te}$ .
- Geben Sie einen Term  $\text{xsol} \in \text{Te}$  an, der die  $x$ -Komponente der Lösung eines linearen Gleichungssystems

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned}$$

darstellt. Zeichen aus  $\Sigma_1 \cup \Sigma_2$  haben ihre übliche Bedeutung.

Zur Erinnerung: für eine Lösung  $(x, y)$  gilt in üblicher Notation

$$\begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - eb}{ad - bc} \end{aligned}$$

- c) Zeigen Sie  $G \vdash^* \text{xsol}$  durch Angabe einer Ableitung.  
d) Eine Funktion  $h : \Sigma^* \rightarrow \mathbb{Z}$  wird eindeutig definiert durch:

- $h(\epsilon) = 0$
- $\forall x \in \Sigma_i : h(x) = i - 1 \ (i \in \{0, 1, 2\})$
- $\forall u, v \in \Sigma^+ : h(uv) = h(u) + h(v)$

Beispiele:

$$\begin{array}{llll} s = +a - a \in L & t = +a + b \notin L & u = /a * a - b \in L & v = +a - a * b \notin L \\ h(s) = -1 & h(t) = 0 & h(u) = -1 & h(v) = -1 \end{array}$$

Beweisen Sie die Richtung  $\Rightarrow$  in der folgenden Äquivalenz:

$$(\mathcal{L}) \quad \forall w \in \Sigma^* : w \in \text{Te} \Leftrightarrow \begin{cases} h(w) = -1 \quad \text{und} \\ \forall u, v : w = uv \wedge v \neq \epsilon \Rightarrow h(u) \geq 0 \end{cases}$$

- e) Verwenden Sie das Kriterium  $(\mathcal{L})$ , um einen Kellerautomaten zu konstruieren, der die Sprache  $\text{Te}$  akzeptiert.

**Aufgabe 5:**

Folgendes Codefragment stellt einen Sortieralgorithmus dar:

```

1  static void sortiere(int p, int q, int[] a) {
2      int m, x, y, t, i;
3      if (p>=q) {System.out.print("Fehler!");}
4      m=a[(p+q)/2];
5      x=p; y=q;
6
7      do {
8          while(a[x]<m) {x++;}
9          while(a[y]>m) {y--;}
10         if(x<y) {
11             t=a[x];
12             a[x]=a[y];
13             a[y]=t;
14             x++; y--;
15         } else break;
16     while (x<=y);
17
18     if (x==y) {x++; y--;}
19     if (y>p) {sortiere(p, y, a);}
20     if (x<q) {sortiere(x, q, a);}

```

- a) Tragen Sie in einer Tabelle (s. u.) ein, wie oft jede der Programmzeilen 7, 8, 16, 17 und 18 im besten sowie im schlechtesten Fall ausgeführt wird. Nehmen Sie an, dass Variablen korrekt vereinbart wurden. Das zu sortierende Feld enthält 5 Elemente.

Programmzeile	bester Fall	schlechtester Fall
7	v-mal	w-mal

- b) Geben Sie jeden Zwischenschritt bis zur sortierten Liste für folgenden Aufruf an:  
*sortiere(0, 4, [4, 6, 3, 1, 2])*
- c) Geben Sie eine worst-case Abschätzung des Aufwandes der Methode *sortiere* in O-Notation an.
- d) Bei obigem Algorithmus wird die Zahlenliste im Datentyp Feld gespeichert. Alternativ könnte auch eine einfach verkettete Liste verwendet werden. Vergleichen Sie in maximal drei Sätzen den Einsatz dieser beiden Datentypen bei der Verwendung des obigen Sortierverfahrens hinsichtlich Speicherbedarf und Gesamlaufzeit.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2012**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl:		
Kennwort:	<b>Herbst 2012</b>	<b>66115</b>
Arbeitsplatz-Nr.:		

---

**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: **7**

---

**Bitte wenden!**

**Thema Nr. 1**  
**(Aufgabengruppe)**

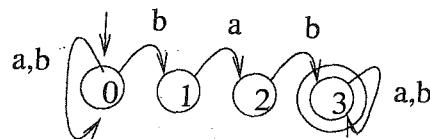
Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Wir fixieren das Alphabet  $\Sigma = \{a, b\}$  und definieren  $L \subseteq \Sigma^*$  durch

$$L = \{w \mid \text{in } w \text{ kommt das Teilwort } bab \text{ vor}\}$$

Z.B. ist  $babaabb \in L$ , aber  $baabaabb \notin L$ . Der folgende nichtdeterministische Automat  $A$  erkennt  $L$ :



- a) Wenden Sie die Potenzmengenkonstruktion auf  $A$  an und geben Sie den resultierenden deterministischen Automaten an. Nicht erreichbare Zustände sollen nicht dargestellt werden.
- b) Konstruieren Sie aus dem so erhaltenen deterministischen Automaten den Minimalautomaten für  $L$ . Beschreiben Sie dabei die Arbeitsschritte des verwendeten Algorithmus in nachvollziehbarer Weise.
- c) Geben Sie die Äquivalenzklassen der Myhill-Nerode Äquivalenz von  $L$  durch Repräsentanten an. (Diese Äquivalenz ist definiert durch  $x \sim_L y \iff \forall u. xu \in L \iff yu \in L$ .)

Tipp: Die Vereinigung aller Klassen muss  $\{a, b\}^*$  ergeben und ihre Anzahl entspricht der Zustanzahl des Minimalautomaten.

**Aufgabe 2:**

Wir fixieren wieder das Alphabet  $\Sigma = \{a, b\}$  und betrachten die Sprache  $L = \{(aba)^n a (baa)^n \mid n \geq 1\}$ .

- a) Obwohl es auf den ersten Blick nicht so aussieht, ist diese Sprache regulär. Begründen Sie das durch Angabe eines regulären Ausdrucks für  $L$ .
- b) Professor Plem versucht fälschlicherweise mithilfe des Pumpinglemmas nachzuweisen, dass  $L$  nicht regulär sei. Er schreibt:

*Nehmen wir widerspruchshalber an,  $L$  sei regulär. Dann gäbe es eine Pumpingzahl  $n$ . Wir betrachten  $w = (aba)^n a (baa)^n$ . Sei jetzt  $w = xyz$  eine Zerlegung derart, dass  $|xy| \leq n$  und  $|y| \geq 1$ . Laut Pumpinglemma wäre nun aber  $xz$  auch in  $L$ . Das ist ein Widerspruch, da  $xy$  nach Annahme vollständig im  $(aba)^n$ -Block von  $w$  liegt und somit  $xz$  nicht in  $L$  sein kann.*

Begründen Sie detailliert, an welcher Stelle Professor Plem irrt und geben Sie eine Pumpingzahl  $n$  für  $L$  an. Legen Sie konkret dar, wie jedes Wort  $w \in L$  mit  $|w| \geq n$  so zerlegt werden kann, wie es vom Pumpinglemma garantiert wird.

**Aufgabe 3:**

Das 0-1-Integer Linear Programming Problem (0-1ILP) ist wie folgt definiert:

- a) GEGEBEN: Eine Liste  $V$  von Variablen  $x_1, \dots, x_m$  und eine Liste  $U$  von Ungleichungen  $t_1 \geq 0, \dots, t_n \geq 0$ , wobei die  $t_i$  lineare Terme mit den Variablen  $x_1, \dots, x_n$  sind. Eine konkrete Instanz wäre zum Beispiel  $V : x, y, z$  und  $U : x + y + z - 1 \geq 0, x + (1 - y) - 2 \geq 0$ .
- b) GESUCHT: Eine Belegung der Variablen mit Werten aus  $\{0, 1\}$  (jede Variable ist entweder 0 oder 1), sodass alle Ungleichungen erfüllt sind.

Weisen Sie nach, dass 0-1-ILP NP-vollständig ist. Für die eine Richtung bietet sich eine Reduktion von 3SAT an.

**Aufgabe 4:**

Gegeben ist ein Array  $a$  von ganzen Zahlen der Länge  $n$ , z.B.:

$i$	0	1	2	3	4	5	6	7	8	9
$a_i$	5	-6	4	2	-5	7	-2	-7	3	5

Im Beispiel ist also  $n = 10$ . Es soll die maximale Teilsumme berechnet werden, also der Wert des Ausdrucks

$$\max_{i,j \leq n} \sum_{k=i}^{j-1} a_k$$

Im Beispiel ist dieser Wert 8 und wird für  $i = 8, j = 10$  erreicht. Entwerfen Sie ein Divide-And-Conquer Verfahren, welches diese Aufgabenstellung in Zeit  $\tilde{O}(n \log n)$  löst. Skizzieren Sie Ihre Lösung hinreichend detailliert.

Tipp: Sie sollten ein geringfügig allgemeineres Problem lösen, welches neben der maximalen Teilsumme auch noch die beiden "maximalen Randsummen" berechnet. Die werden dann bei der Endausgabe verworfen.

## Thema Nr. 2 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

### Aufgabe 1: endliche Automaten

Konstruieren Sie einen deterministischen endlichen Automaten (DFA) A  
(durch Angabe von A oder eines Diagramms für A)  
für die folgende Sprache L:

L besteht aus der Menge aller Binärzahlen ohne führende Nullen  
bei denen die Anzahl der Einsen ungerade und die Anzahl der Nullen gerade ist.

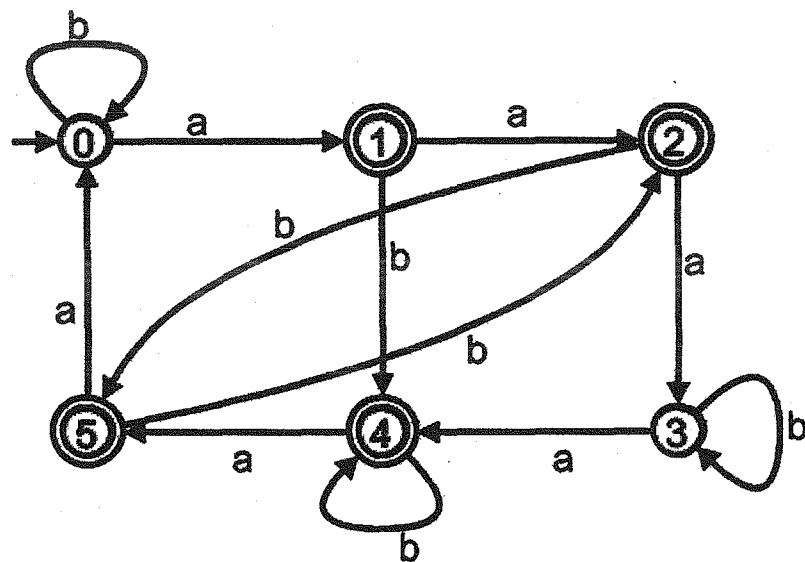
### Aufgabe 2: reguläre Sprachen

Zeigen oder widerlegen Sie, dass die folgende Sprache L regulär ist.  
Zeigen: durch Angabe eines endlichen Automaten oder eines regulären Ausdrucks  
Widerlegen: mit dem Pumping Lemma

$L = \{w c w^{\text{rev}} \mid w \in \{a,b\}^*, w^{\text{rev}}$  ist die Spiegelung von  $w\}$   
L ist die Menge der Palindrome mit der Markierung c.

### Aufgabe 3: Minimierung DFA

Minimieren Sie den folgenden deterministischen Automaten mit den Zuständen {0,1,2,3,4,5}  
und den Endzuständen {1,2,4,5}. 0 ist der Startzustand.



**Aufgabe 4: Berechenbarkeit und Komplexität**

Gegeben sei die Sprache  $L = \{w c w \mid w \in \{a,b\}^*\}$  (Duplikate von Worten)

- a) Geben Sie eine Turingmaschine  $M$  mit  $L(M) = L$  an.  
Beschreiben Sie in Worten, wie Ihre Turingmaschine arbeitet.
- b) Welche Zeit- und welche Speicherkomplexität in O-Notation hat Ihre Turingmaschine?  
Erläutern Sie dies anhand Ihrer in a) gegebenen Beschreibung
- c) Skizzieren Sie, warum  $L$  eine  $O(\log n)$  speicher-beschränkte Sprache ist.

**Aufgabe 5: Komplexität**

Die Kinder in einem Kindergarten sollen einen Kreis bilden, bei dem sich nebeneinander stehende Kinder die Hand reichen. Es gibt aber Kinder, die sich nicht mögen und sich nicht die Hand reichen wollen. Diese dürfen im Kreis nicht nebeneinander stehen. Nur dann ist der Kreis zulässig.

Formal:

Gegeben sei eine Menge von  $n$  Kindern  $K = \{k_1, \dots, k_n\}$  und eine symmetrische binäre Relation  $E$  bestehend aus allen Paaren  $\{k, k'\}$  mit der Eigenschaft dass sich  $k$  und  $k'$  nicht mögen.

Problem:

Kann man einen zulässigen Kreis bilden?

Begründen Sie warum dieses Problem NP-vollständig ist.

Hinweis:

Das Hamilton Problem ist NP-vollständig. Dies können Sie bei Ihrer Begründung verwenden.  
Das Hamilton (genauer Hamilton Kreis) Problem in einem ungerichteten Graph  $G = (V, E)$  ist die Frage, ob es einen Weg  $w$  (Rundreise) genau einmal durch jeden Knoten gibt, so dass Anfangs- und Endknoten zusammenfallen, siehe auch Informatik Duden unter den Stichworten NP und Königsberger Brückenproblem

**Aufgabe 6: O-Notation**

Gegeben seien die Funktionen  $f: N \rightarrow N$  und  $g: N \rightarrow N$ , wobei  $f(n) = (n-1)^3$  und  $g(n) = (2n+3)(3n+2)$ .  
Geben Sie an, welche der folgenden Aussagen gelten. Beweisen Sie Ihre Angaben.

- a)  $f(n) \in O(g(n))$
- b)  $g(n) \in O(f(n))$

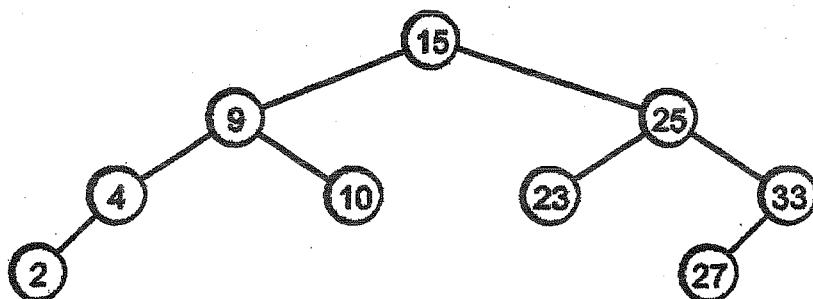
**Aufgabe 7: Heap und binärer Suchbaum**

- a) Fügen Sie nacheinander die Zahlen 3,5,1,2,4
  - (i) in einen leeren binären Suchbaum ein
  - (ii) in einen leeren Heap einGeben Sie die Ergebnisse an (Zeichnung)
- b) Geben Sie zwei Merkmale an, bei denen sich Heaps und binäre Suchbäume wesentlich unterscheiden. Ein wesentlicher Unterschied zwischen Bubblesort und Mergesort ist z.B. die worst case Laufzeit mit  $O(n^2)$  für Bubblesort und  $O(n \log n)$  für Mergesort.

**Aufgabe 8: AVL-Bäume**

Gegeben sei der folgende AVL-Baum T. Führen Sie auf T folgende Operationen durch.

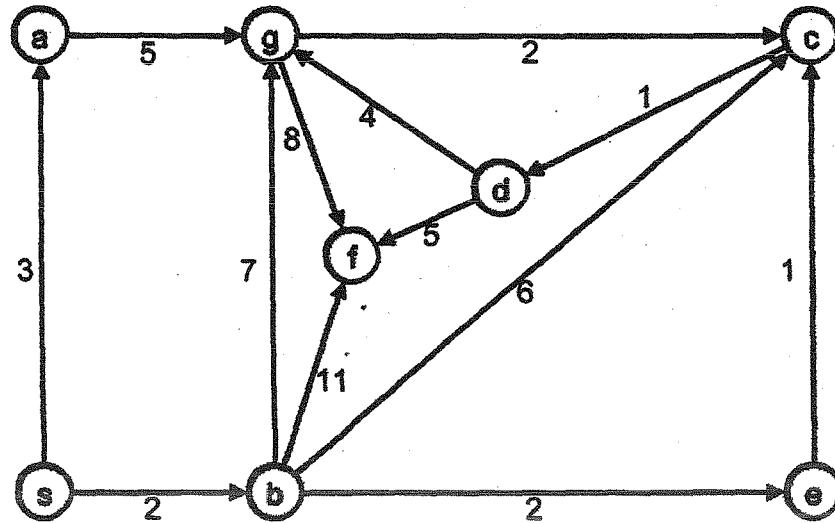
- (a) Fügen Sie den Wert 1 in T ein. Balanceieren Sie falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.  
Fügen Sie nun den Wert 28 in T ein. Balanceieren Sie falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.
- (b) Löschen Sie aus T den Wert 15. Balanceieren Sie falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.



**Aufgabe 9: Dijkstra**

Gegeben sei der unten stehende gerichtete Graph  $G=(V, E)$  mit positiven Kantenlängen  $l(e)$  für jede Kante  $e \in E$ .

In welcher Reihenfolge werden die Knoten von  $G$  durch den Dijkstra-Algorithmus bei der Berechnung der kürzesten Wege von Knoten  $s$  ausgehend endgültig bearbeitet?



**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2013**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl:		
Kennwort:		
Arbeitsplatz-Nr.:		

**Frühjahr  
2013**

**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: **9**

**Bitte wenden!**

**Thema Nr. 1****Aufgabe 1 Endliche Automaten und reguläre Sprachen**

Gegeben sei folgender nicht-deterministischer endlicher Automat  $M$ :

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \delta, \{0,1\}, \{q_0\}, \{q_4\})$$

mit dem Eingabealphabet  $\Sigma = \{0,1\}$ , dem Startzustand  $q_0$ , dem Endzustand  $q_4$  und der Übergangsfunktion  $\delta$  definiert durch

$$\delta(q_0, 0) = \{q_0, q_1\},$$

$$\delta(q_0, 1) = \{q_0\},$$

$$\delta(q_1, 1) = \{q_2\},$$

$$\delta(q_2, 1) = \{q_3\},$$

$$\delta(q_3, 0) = \{q_4\},$$

$$\delta(q_3, 1) = \{q_2\},$$

$$\delta(q_4, 0) = \delta(q_4, 1) = \{q_4\}$$

- a) Geben Sie einen regulären Ausdruck für die von  $M$  akzeptierte Sprache  $L(M)$  an.
- b) Geben Sie eine reguläre,  $\varepsilon$ -freie Grammatik  $G$  an, die die Sprache  $L(M)$  erzeugt.
- c) Wandeln Sie den nicht-deterministischen endlichen Automaten  $M$  durch Teilmengekonstruktion in einen deterministischen endlichen Automaten  $N$  um und minimieren Sie den Automaten  $N$  bzw. weisen Sie nach, dass der Automat  $N$  bereits minimal ist.

**Fortsetzung nächste Seite!**

**Aufgabe 2 Kontextfreie Grammatiken**

Gegeben sei die Grammatik  $G = (V, \Sigma, P, S)$  mit  $V = \{S, A, B, C\}$ ,  $\Sigma = \{a, b\}$ , dem Startsymbol  $S$  und

$$P = \{S \rightarrow AB, S \rightarrow CS, A \rightarrow BC, A \rightarrow BB, A \rightarrow a, B \rightarrow AC, B \rightarrow b, C \rightarrow AA, C \rightarrow BA\}.$$

$L = L(G)$  ist die von  $G$  erzeugte Sprache.

- a) Zeigen Sie, dass  $G$  mehrdeutig ist.
- b) Entscheiden Sie mithilfe des Algorithmus von Cocke, Younger und Kasami (CYK), ob das Wort  $w = babaaa$  zur Sprache  $L$  gehört. Begründen Sie Ihre Entscheidung.
- c) Geben Sie eine Ableitung für  $w = babaaa$  an.

**Aufgabe 3 Turing-Maschinen**

Konstruieren Sie eine deterministische Turing-Maschine  $M$ , die die Sprache

$$L = L(\gamma) \text{ mit } \gamma = (01|10)^+$$

entscheidet.

Geben Sie die Übergangsfunktion als Tabelle an und erläutern Sie die Bedeutung der Zustände in der von Ihnen konstruierten Maschine.

**Aufgabe 4 Algorithmen**

Die Potenz  $y$  einer Zahl  $x$  für zwei Zahlen  $x$  und  $y$  ( $y \in \mathbb{N}_0, x \in \mathbb{Z}$ ) kann als fortgesetzte Multiplikation realisiert werden:

$$x^y = \exp(x, y) = \begin{cases} x \cdot x^{y-1} & \text{falls } y > 0 \\ 1 & \text{falls } y = 0 \end{cases}$$

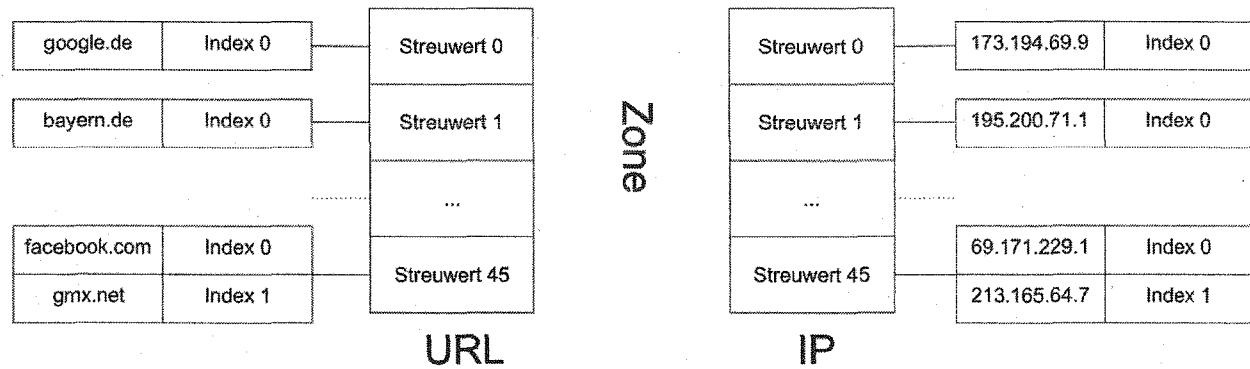
- a) Schreiben Sie eine rekursive Implementierung für diese Funktion in einer Programmiersprache Ihrer Wahl.
- b) Implementieren Sie diese Funktion nun in der Programmiersprache Ihrer Wahl in iterativer Form.
- c) Geben Sie für beide Algorithmen jeweils die Speicher- und Laufzeitkomplexität an.

### Aufgabe 5 Doppelt verkettete Listen und Sortieralgorithmen

- Implementieren Sie in einer objektorientierten Programmiersprache Ihrer Wahl eine doppelt verkettete Liste für ganze Zahlen. Jedes Listenelement soll die Methoden `get()` und `set()` der Attribute `value`, `predecessor` und `successor` enthalten.
- Implementieren Sie die Methoden `head()` und `tail()`, die das erste und letzte Element der Liste ausgeben.
- Implementieren Sie eine Methode `sort()`, die die Liste nach den Werten von `value` aufsteigend sortiert.
- Welche Komplexität hat Ihr Sortieralgorithmus beim Sortieren der doppelt verketteten Liste aus c jeweils im besten und schlechtesten Fall?

### Aufgabe 6 Streutabellen (Hash-Tables)

Um die URL (zum Beispiel `google.de`) und die zugehörige IP des Servers (hier `173.194.69.9`) zu verwalten, werden Streutabellen verwendet, die eine bestimmte Zone von Adressen abbilden. Die Streutabellen werden als zwei dynamische Arrays (in Java: `ArrayLists`) realisiert. Kollisionen innerhalb einer Zone werden ebenfalls in dynamischen Arrays verwaltet.



Um zu einer URL die IP zu finden, berechnet man zunächst mittels der Funktion `hash()` den entsprechenden Streuwert, entnimmt dann den Index der Tabelle URL und sucht schließlich an entsprechender Stelle in der Tabelle IP die IP-Adresse.

- a) Erläutern Sie am vorgestellten Beispiel, wie ein Hash-Verfahren zum Speichern großer Datenmengen prinzipiell funktioniert und welche Voraussetzungen und Bedingungen daran geknüpft sind.
- b) Nun implementieren Sie Teile dieser IP- und URL-Verwaltung in einer objektorientierten Sprache Ihrer Wahl. Verwenden Sie dabei die folgende Klasse (die Vorgaben sind in der Sprache Java gehalten):

```
class Zone {  
    private ArrayList<ArrayList<String>> urlList =  
        new ArrayList<ArrayList<String>>();  
    private ArrayList<ArrayList<String>> ipList =  
        new ArrayList<ArrayList<String>>();  
    public int hash(String url) /* calculates hash-value h, >=0 */  
}
```

- i) Prüfen Sie in einer Methode `boolean exists(int h)` der Klasse `Zone`, ob bereits mindestens ein Eintrag für einen gegebenen Streuwert vorhanden ist. Falls `h` größer ist als die derzeitige Größe der Streutabelle, existiert der Eintrag nicht.
- ii) Die Methode `int getIndex(string url, ArrayList<String> urlList)` soll den Index einer URL in der Kollisionsliste berechnen. Ist die URL in der Kollisionsliste nicht vorhanden, soll `-1` zurückgeliefert werden.
- iii) Ergänzen Sie die Klasse `Zone` um eine Methode `String lookup(String url)`, die in der Streutabelle die IP-Adresse zur `url` zurückgibt. Wird eine nicht vorhandene Adresse abgerufen, wird eine Fehlermeldung zurückgegeben.

## Thema Nr. 2

1. Es bezeichne  $G = \langle V, T, P, A \rangle$  die kontextfreie Grammatik mit dem Variablenalphabet  $V = \{A, B\}$ , dem Terminalalphabet  $T = \{a, b\}$ , dem Startsymbol  $A$  und der Menge  $P$  von Produktionen:

$$P = \{ A \rightarrow aBa \mid bBa, B \rightarrow aBa \mid bBa \mid \lambda \}$$

wobei  $\lambda$  für das leere Wort steht.

- (a) Welches ist die von  $G$  generierte Sprache  $L(G)$ ? Geben Sie eine Beschreibung von  $L(G)$ , die nicht auf die Grammatik  $G$  Bezug nimmt. Beweisen Sie Ihre Behauptung!
- (b) Zeigen Sie dass die Sprache  $L(G)$  nicht regulär ist.
- (c) Transformieren Sie die Grammatik  $G$  in eine äquivalente Grammatik  $G'$  in Chomsky-Normalform und geben Sie in  $G'$  eine Linksableitung für das Wort  $ab^2a^5$  an.

2. Es sei  $\Sigma = \{0, 1\}$ . Das Alphabet  $\Sigma_2$  bestehe aus allen Paaren von Elementen aus  $\Sigma$ , geschrieben als Spaltenvektoren der Länge 2 über  $\Sigma$ , also

$$\Sigma_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

Ein Wort  $w = w_1w_2 \dots w_n \in \Sigma_2^n$  mit  $w_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$ ,  $1 \leq i \leq n$ , kann aufgefasst werden als ein Paar  $(x, y) \in \Sigma^n \times \Sigma^n$  mit  $x = x_1x_2 \dots x_n$ ,  $y = y_1y_2 \dots y_n$ , d.h.

$$w = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \dots \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$

Jedes Wort  $a = a_1a_2 \dots a_{n-1}a_n \in \Sigma^n$  stellt die natürliche Zahl

$$\text{bin}(a) = a_1 \cdot 2^{n-1} + a_2 \cdot 2^{n-2} + \dots + a_{n-1} \cdot 2 + a_n$$

dar (binäre Zahldarstellung).

- (a) Die Sprache der "Multiplikation mit 3" ist

$$M3 := \left\{ w = \begin{bmatrix} x \\ y \end{bmatrix} \in \Sigma_2^* ; \text{bin}(y) = 3 \cdot \text{bin}(x) \right\}$$

Es gilt also beispielsweise

$$\begin{bmatrix} 0011 \\ 1001 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \in M3, \begin{bmatrix} 011 \\ 010 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \notin M3.$$

Zeigen Sie, dass die Sprache  $M3$  regulär ist.

Hinweis: Sie können – falls Ihnen das hilfreich erscheint – hier die Tatsache verwenden, dass eine Sprache  $L$  genau dann regulär ist, wenn die gespiegelte Sprache  $L^R = \{w^R ; w \in L\}$  regulär ist. Dabei ist  $(w_1w_2 \dots w_{n-1}w_n)^R = w_nw_{n-1} \dots w_2w_1$ .

- (b) Zeigen Sie, dass die Sprache

$$\left\{ w = \begin{bmatrix} x \\ y \end{bmatrix} \in \Sigma_2^* ; y = x^R \right\}$$

nicht regulär ist.

3. Es sei  $X = \{x_1, x_2, \dots, x_n\}$  eine Variablenmenge und es bezeichne  $\text{AL}(X)$  die Menge der aussagenlogischen Formeln über  $X$  mit Konnektiven für die Konjunktion ( $\wedge$ ), die Disjunktion ( $\vee$ ) und die Negation ( $\neg$ ). Es bezeichne  $\text{Sat}(X)$  die Menge der erfüllbaren Formeln und  $\text{Taut}(X)$  die Menge der Tautologien aus  $\text{AL}(X)$ . Eine aussagenlogische Formel heisse **2-erfüllbar**, wenn es mindestens zwei verschiedene Bewertungen ihrer Variablen gibt, die diese Formel wahr machen.  $\text{Sat}_2(X)$  bezeichne die Menge der 2-erfüllbaren aussagenlogischen Formeln über  $X$ .

- (a) Geben Sie eine induktive Definition für  $\text{AL}(X)$  in Form einer Grammatik an.
- (b) Geben Sie für den Fall  $X = \{x_1, x_2, x_3\}$  Beispiele für Formeln folgender Art an:
  - i. eine Formel  $F_1$ , die unerfüllbar ist, d.h.  $F_1 \in \text{AL}(X) \setminus \text{Sat}(X)$ ;
  - ii. eine Formel  $F_2$ , die erfüllbar, aber nicht 2-erfüllbar ist,  
d.h.  $F_2 \in \text{Sat}(X) \setminus \text{Sat}_2(X)$ ;
  - iii. eine Formel  $F_3$ , die 2-erfüllbar ist, aber keine Tautologie,  
d.h.  $F_3 \in \text{Sat}_2(X) \setminus \text{Taut}(X)$ ;
  - iv. eine Tautologie  $F_4$ , d.h.  $F_4 \in \text{Taut}(X)$ .
- (c) Zeigen Sie, dass die Frage nach der Zugehörigkeit einer Formel zu  $\text{Sat}_2(X)$  (für beliebige Variablenmengen  $X$ ) ein NP-vollständiges Problem ist. Konstruieren Sie hierfür eine polynomielle Reduktion zwischen dem bekannten Erfüllbarkeitsproblem  $\text{Sat}$  und dem  $\text{Sat}_2$ -Problem, die dies beweist.

Falls Sie die verlangte Reduktion nicht finden, so geben Sie möglichst genau an, aus welchen Daten eine solche besteht und was zu beweisen wäre.

*Hinweis: Verwenden Sie zur Formulierung von Algorithmen bzw. Datentypen eine gängige höhere Programmiersprache oder einen entsprechenden Pseudocode. Erläutern Sie Ihre Lösung ausgiebig durch Kommentare.*

**Aufgabe 4**

Für das bekannte rekursive Sortierverfahren Mergesort soll eine nicht-rekursive Variante nrMergesort entwickelt werden. Die Grundidee des Verfahrens besteht dabei darin, auf den rekursiven Abstieg zu verzichten und davon auszugehen, dass zu Beginn sortierte einelementige Listen hintereinander in einem Feld vorliegen. Im Verlauf des Verfahrens werden immer länger werdende, sortierte Teillisten miteinander solange gemischt, bis die gesamte Liste sortiert ist.

- a) Geben Sie eine Implementierung des nicht-rekursiven Mergesort-Verfahrens an, welche zwei Felder verwendet.
- b) Werden bei diesem Verfahren die gleichen Mischoperationen ausgeführt, wie beim rekursiven Mergesort? Begründen Sie Ihre Antwort.
- c) Ist folgende Behauptung wahr oder falsch: Die Laufzeit von Mergesort (rekursive Variante) hängt nicht vom Wert der Schlüssel in der Eingabedatei ab. Begründen Sie Ihre Antwort.

**Aufgabe 5**

Drei Missionare und drei Kannibalen befinden sich am Ufer eines Flusses und möchten diesen überqueren. Dazu steht ihnen ein Boot zur Verfügung, das ein oder zwei Personen befördern kann. Verbleiben an einem Ufer mehr Kannibalen als Missionare, so werden die Missionare verspeist. Die Frage besteht nun darin, wie *alle* Personen unversehrt auf die andere Seite des Flusses gelangen.

Im Anfangszustand befinden sich alle Personen und das Boot auf einer Seite des Flusses. Nehmen Sie an, es sei die linke Seite des Flusses. Im Zielzustand befinden sich alle Personen und das Boot auf der anderen Seite des Flusses. Jeden Zustand kann man durch folgendes Fünftupel beschreiben:

$$\text{Missionare}_{\text{links}} \times \text{Kannibalen}_{\text{links}} \times \text{Missionare}_{\text{rechts}} \times \text{Kannibalen}_{\text{rechts}} \times \text{Bootposition}$$

Dabei werden die Elemente für Missionare und Kannibalen durch natürliche Zahlen und die Bootposition durch einen der Strings „links“ oder „rechts“ modelliert. Der Anfangszustand wäre somit also  $(3, 3, 0, 0, \text{„links“})$  und der Zielzustand  $(0, 0, 3, 3, \text{„rechts“})$ .

Schreiben Sie Algorithmen zur Lösung dieses Suchproblems und retten Sie die Missionare! Es ist empfehlenswert (aber nicht zwingend), hierzu eine funktionale Programmiersprache zu verwenden. Gehen Sie im Einzelnen vor, wie folgt:

- a.) Schreiben Sie eine Funktion, die alle möglichen Überfahrten von links nach rechts oder umgekehrt modelliert, d. h. die zu einem gegebenen Zustand die Liste der möglichen Folgezustände im Sinne der o. g. Regeln berechnet. Gehen Sie dazu von allen möglichen Überfahrten aus und überprüfen Sie für jede konkrete Überfahrt mittels einer geeigneter Funktion, ob diese zu einem zulässigen Zustand führt. Zustände, die die Missionare nicht überleben, gelten im Sinne des Rettungsvorhabens ebenfalls als nicht zulässig. Nicht zulässige Zustände werden nicht in die Liste der möglichen Folgezustände eingefügt.
- b.) Geben Sie eine Funktion an, die feststellt, ob ein Zyklus vorliegt, d. h. ob ein Zustand in einer Liste bereits besuchter Zustände schon enthalten ist.
- c.) Verwenden Sie Ihre Ergebnisse aus a.) und b.), um eine Funktion anzugeben, die dieses Suchproblem mittels Breitensuche löst. (Sie können die Funktionen aus a.) und b.) hier auch dann verwenden, wenn Sie diese Teilaufgaben nicht vollständig gelöst haben.) Die Funktion erhält als Eingabe einen Start- und einen Zielzustand und liefert als Ergebnis die erste gefundene Liste von Zuständen, die das Problem löst.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2013**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl:		
Kennwort:	<b>Herbst</b>	<b>66115</b>
Arbeitsplatz-Nr.:	<b>2013</b>	

---

## **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: **7**

---

**Bitte wenden!**

## Thema Nr. 1

**Teilaufgabe I** Es sei  $\Sigma = \{1, 2\}$  und  $L \subseteq \Sigma^*$  die Menge aller Wörter  $x_1x_2\dots x_n$ , derart, dass  $\sum x_i$  ein Vielfaches von drei ist. So ist, z.B.  $12112 \notin L$ , denn  $1+2+1+1+2=7$ , aber  $1221 \in L$ , denn  $1+2+2+1=6$ . Definitionsgemäß ist auch das leere Wort  $\epsilon \in L$ .

1. Konstruieren Sie einen deterministischen endlichen Automaten für  $L$ .
2. Minimieren Sie Ihren Automaten oder begründen Sie, dass er bereits minimal ist.
3. Leiten Sie aus diesem Automaten einen regulären Ausdruck für  $L$  ab. Dokumentieren Sie Ihre Arbeitsschritte geeignet.
4. Geben Sie zwei unterschiedliche Wörter an, die im Sinne der Myhill-Nerode Äquivalenz  $\sim_L$  äquivalent sind. Erinnerung:  $u \sim_L v \iff \forall w. uw \in L \Leftrightarrow vw \in L$ . Begründen Sie Ihre Antwort.
5. Beweisen oder widerlegen Sie folgende Aussage:  $L(a(a+b)^*) = L((ab^*)^*ab^*)$ , wobei  $L(\alpha)$  die durch den regulären Ausdruck  $\alpha$  definierte Sprache bezeichnet.

Ist die Aussage falsch, so müssen Sie ein konkretes Wort angeben, welches auf einen der beiden regulären Ausdrücke passt und auf den anderen nicht. Ist sie wahr, so müssen Sie beide Richtungen  $\subseteq$  und  $\supseteq$  zeigen.

**Teilaufgabe II** Wie man weiß ist die Sprache  $L = \{a^n b^n c^n \mid n \geq 0\}$  über dem Alphabet  $\Sigma = \{a, b, c\}$  nicht kontextfrei. Dies dürfen Sie im folgenden ohne Begründung voraussetzen.

1. Finden Sie zwei kontextfreie Sprachen  $L_1$  und  $L_2$ , sodass  $L = L_1 \cap L_2$ . Geben Sie jeweils Grammatiken für diese Sprachen an. Hinweis:  $L_1$  sorgt dafür, dass die Zahlen der  $a$ 's und  $b$ 's übereinstimmen.
2. Warum folgt daraus, dass die Klasse der kontextfreien Sprachen nicht unter Komplement abgeschlossen ist?
3. Bekanntlich ist der Schnitt einer kontextfreien Sprache mit einer regulären Sprache wieder kontextfrei. Folgern Sie, dass die Sprache  $L' = \{w \mid |w|_a = |w|_b = |w|_c\}$  nicht kontextfrei ist (hier bezeichnet  $|w|_x$  die Anzahl der Vorkommen des Buchstabens  $x$  in  $w$ ).

**Teilaufgabe III** Wir betrachten das wie folgt definierte Problem DOPP:

GEGEBEN: Eine deterministische Turingmaschine  $M$ , eine Eingabe  $x$  (für  $M$ ), ein Zustand  $q$  (von  $M$ ).

GEFRAGT: Wird der Zustand  $q$  bei der Berechnung von  $M$  auf  $x$  mindestens zweimal besucht?

1. Zeigen Sie durch Angabe einer Reduktion vom Halteproblem, dass DOPP unentscheidbar ist.
2. Begründen Sie, dass DOPP rekursiv aufzählbar (semi-entscheidbar) ist.

**Teilaufgabe IV** Ein ungerichteter Graph  $G = (V, E)$  ist zusammenhängend, wenn je zwei Knoten in  $G$  durch einen Pfad verbunden sind.

1. Der Graph  $G = (V, E)$  sei durch Adjazenzlisten gegeben. Beschreiben Sie, wie man in Zeit  $O(|E| + |V|)$  prüfen kann, ob  $G$  zusammenhängend ist.
2. Beweisen oder widerlegen Sie: Hat jeder Knoten mindestens  $|V|/2$  Nachbarn (ganzzahlige Division), so ist  $G$  zusammenhängend.

## Thema Nr. 2

### 1. Aufgabe: Endliche Automaten

Zeigen Sie durch Angabe eines regulären Ausdrucks, dass die Sprache L regulär ist.

L besteht aus der Menge aller natürlichen Zahlen in Dezimalzahldarstellung ohne führende Nullen, die durch zwei oder durch fünf teilbar sind.

Konstruieren Sie einen deterministischen endlichen Automaten A mit  $L(A) = L$ .

### 2. Aufgabe: Reguläre Sprachen

Zeigen oder widerlegen Sie, dass die folgende Sprache L regulär ist.

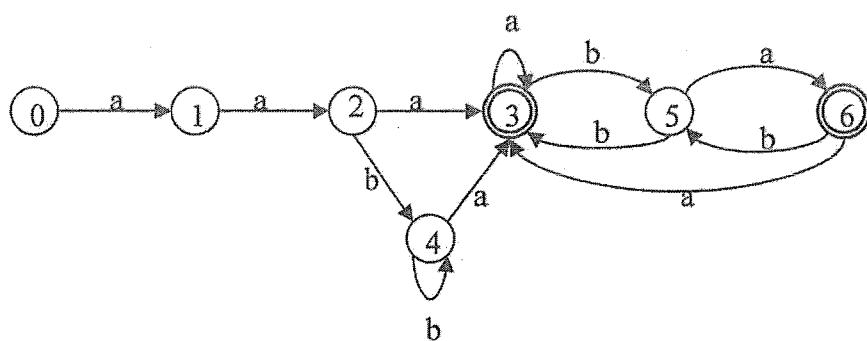
$$L = \{a^n b^m \mid n, m \geq 1, m = 2^n\}.$$

Hinweis: Um zu zeigen, dass L regulär ist, geben Sie einen endlichen Automaten oder einen regulären Ausdruck an. Soll dies widerlegt werden, verwenden Sie das Pumping Lemma.

### 3. Aufgabe: Minimierung DFA

Minimieren Sie den folgenden deterministischen Automaten mit den Zuständen  $\{0, 1, 2, 3, 4, 5, 6\}$ , dem Startzustand 0 und den Endzuständen  $\{3, 6\}$ .

Geben Sie z.B. durch die Bezeichnung an, welche Zustände zusammengefasst wurden.



Fortsetzung nächste Seite!

**4. Aufgabe: Berechenbarkeit und Komplexität**

Gegeben sei die Sprache  $L = \{ a^n b^m c^k \mid n, m, k \geq 1, n=m \text{ und } n \neq k \}$ .

- a) Geben Sie eine Turingmaschine  $M$  an, die  $L$  erkennt.  
Beschreiben Sie in Worten, wie ihre Turingmaschine arbeitet.
- b) Welche Zeitkomplexität in O-Notation hat ihre Turingmaschine?  
Erläutern Sie dies anhand Ihrer in a) gegebenen Beschreibung.

**5. Aufgabe: Komplexität**

Beim BIN PACKING Problem hat man eine Menge von Elementen  $\{x_1, \dots, x_n\}$ .

Jedes Element  $x$  hat ein nicht negatives Gewicht  $w(x)$ .

Außerdem gibt es Kisten, die je höchstens mit dem Gewicht  $G$  beladen werden dürfen.

Das Problem ist, die Elemente so in Kisten zu packen, dass keine Kiste überladen wird.

Reichen dafür  $k$  Kisten?

- 1) Geben Sie eine formale Beschreibung dieses Problems an (mit Mengen, Summen, etc.).
- 2) Warum ist dieses Problem in NP?

Fortsetzung nächste Seite!

**6. Aufgabe: O-Notation**

Gegeben seien die Funktionen  $f: \mathbb{N} \rightarrow \mathbb{N}$  und  $g: \mathbb{N} \rightarrow \mathbb{N}$ , wobei

$$f(n) = 2(n-1)^2 + 3n - 1 \text{ und}$$

$$g(n) = 10n \log n + 20n + 30$$

Geben Sie an, welche der folgenden Aussagen gelten. Beweisen Sie Ihre Behauptungen.

- a)  $f(n) \in O(g(n))$
- b)  $g(n) \in O(f(n))$

**7. Aufgabe: Heap und binärer Suchbaum**

a)

- (i) Fügen Sie nacheinander die Zahlen 7, 1, 12, 8, 10, 3, 5 in einen leeren binären Suchbaum ein und zeichnen Sie den Suchbaum nach „8“ und nach „5“.
- (ii) Löschen Sie die „1“ aus dem in (i) erstellten Suchbaum und zeichnen Sie den Suchbaum.
- (iii) Fügen Sie 7, 1, 12, 8, 10, 3, 5 in einen leeren MIN-Heap ein, der bzgl. „ $\leq$ “ angeordnet ist. Geben Sie den Heap nach jedem Element an.

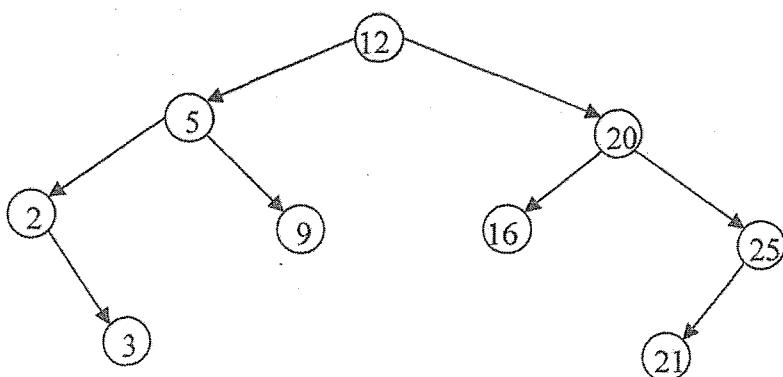
b)

- Was ist die worst-case Laufzeit in O-Notation für das Einfügen eines Elements in einen Heap der Größe  $n$ ? Begründen Sie ihre Antwort.

**8. Aufgabe: AVL-Bäume**

Gegeben sei der folgende AVL-Baum T. Führen Sie auf T folgende Operationen durch.

- (a) Fügen Sie den Wert 22 in T ein. Balancieren Sie falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.
- (b) Löschen Sie danach die 5. Balancieren Sie T falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.



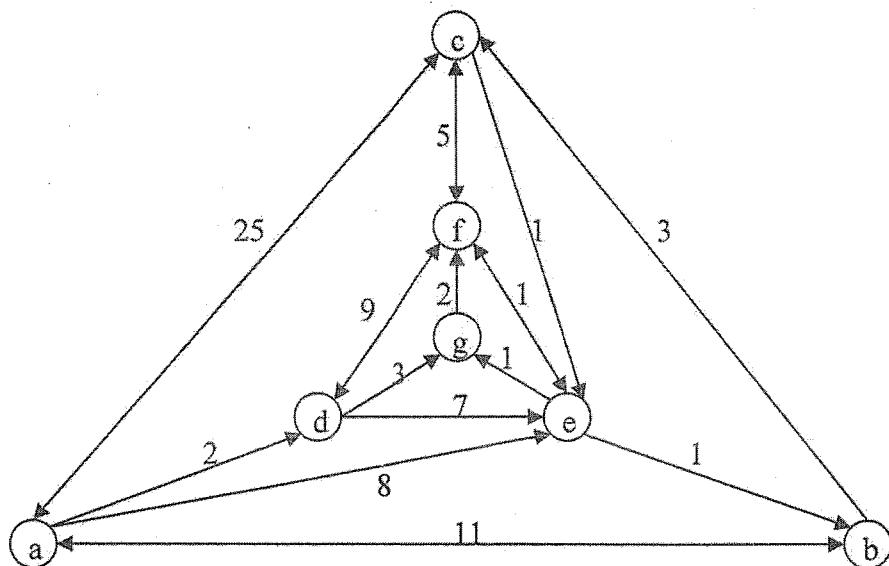
### 9. Aufgabe: Dijkstra

Gegeben sei der unten stehende gerichtete Graph  $G=(V, E)$  mit positiven Kantenlängen  $l(e)$  für jede Kante  $e \in E$ . Kanten mit Doppelpitzen können in beide Richtungen durchlaufen werden.

In welcher Reihenfolge werden die Knoten von  $G$  ab dem Knoten  $a$  durch den Dijkstra-Algorithmus bei der Berechnung der kürzesten Wege endgültig bearbeitet?

Berechnen Sie die Länge des kürzesten Weges von  $a$  zu jedem Knoten.

Geben Sie einen kürzesten Weg von  $a$  nach  $c$  an.



### 10. Aufgabe: Minimaler Spannbaum

- Betrachten Sie den obigen Graphen ohne die Kantenrichtung.  
Konstruieren Sie dann einen minimalen Spannbaum für  $G$ .
- Zeigen Sie durch ein Beispiel, dass ein minimaler Spannbaum eines ungerichteten Graphen nicht eindeutig ist.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2014**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
--------------------	----------------	----------------------

Kennzahl: \_\_\_\_\_

Kennwort: \_\_\_\_\_

Arbeitsplatz-Nr.: \_\_\_\_\_

**Frühjahr**

**2014**

**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: **13**

---

**Bitte wenden!**

## Thema Nr. 1

### Aufgabe 1: „Rekursion und Induktion“

- a) Gegeben sei die Methode **BigInteger lfBig(int n)** zur Berechnung der eingeschränkten Linksfakultät:

$$!n = \begin{cases} n \cdot !(n-1) - (n-1) \cdot !(n-2) & \text{falls } 1 < n < 32767 \\ 1 & \text{falls } n = 1 \\ 0 & \text{sonst} \end{cases}$$

```

import java.math.BigInteger;
import static java.math.BigInteger.*;

public class LeftFactorial {
    // returns the left factorial !n
    BigInteger lfBig(int n) {
        if (n <= 0 || n >= Short.MAX_VALUE) {
            return ZERO;
        } else if (n == 1) {
            return ONE;
        } else {
            return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
        }
    }
}

```

Implementieren Sie unter Verwendung des Konzeptes der *dynamischen Programmierung* die Methode **BigInteger dp(int n)**, die jede  $!n$  auch bei mehrfachem Aufrufen mit dem gleichen Parameter höchstens einmal rekursiv berechnet. Sie dürfen der Klasse **LeftFactorial** genau ein Attribut beliebigen Datentyps hinzufügen und die in **lfBig(int)** verwendeten Methoden und Konstanten ebenfalls nutzen.

**Fortsetzung nächste Seite!**

- b) Betrachten Sie nun die Methode `lfLong (int)` zur Berechnung der vorangehend definierten Linksfakultät ohne obere Schranke. Nehmen Sie im Folgenden an, dass der Datentyp `long` unbeschränkt ist und daher kein Überlauf auftritt.

```
long lfLong(int n) {
    if (n <= 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
    }
}
```

Beweisen Sie formal mittels *vollständiger Induktion*:

$$\forall n \geq 0 : lfLong(n) \equiv \sum_{k=0}^{n-1} k !$$

Fortsetzung nächste Seite!

**Aufgabe 2: „Binäre Bäume“**

Implementieren Sie in einer objekt-orientierten Sprache Ihrer Wahl eine Klasse namens **BinBaum**, deren Instanzen binäre Bäume mit ganzzahligen Datenknoten darstellen, nach folgender Spezifikation:

a) Beginnen Sie zunächst mit der Datenstruktur selbst:

- Mit Hilfe des Konstruktors soll ein neuer Baum erstellt werden, der aus einem einzelnen Knoten besteht, in dem der dem Konstruktor als Parameter übergebene Wert (ganzzahlig, in Java z.B. `int`) gespeichert ist.
- Die Methoden `setLeft(int value)` bzw. `setRight(int value)` sollen den linken bzw. rechten Teilbaum des aktuellen Knotens durch einen jeweils neuen Teilbaum ersetzen, der seinerseits aus einem einzelnen Knoten besteht, in dem der übergebene Wert `value` gespeichert ist. Sie haben keinen Rückgabewert.
- Die Methoden `getLeft()` bzw. `getRight()` sollen den linken bzw. rechten Teilbaum zurückgeben (bzw. `null`, wenn keiner vorhanden ist).
- Die Methode `int getValue()` soll den Wert zurückgeben, der im aktuellen Wurzelknoten gespeichert ist.

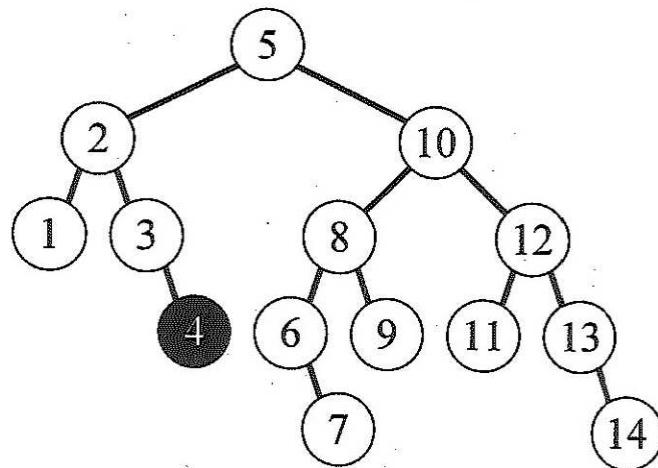
- b) Implementieren Sie nun die Methoden `preOrder()` bzw. `postOrder()`. Sie sollen die Knoten des Baumes mit Tiefensuche traversieren, die Werte dabei in pre-order bzw. post-order Reihenfolge in eine Liste (z.B. `List<Integer>`) speichern und diese Ergebnisliste zurückgeben. Die Tiefensuche soll dabei zuerst in den linken und dann in den rechten Teilbaum absteigen.
- c) Ergänzen Sie schließlich die Methode `isSearchTree()`. Sie soll überprüfen, ob der Binärbaum die Eigenschaften eines *binären Suchbaums* erfüllt. Beachten Sie, dass die Laufzeit-Komplexität Ihrer Implementierung linear zur Anzahl der Knoten im Baum sein muss.

Fortsetzung nächste Seite!

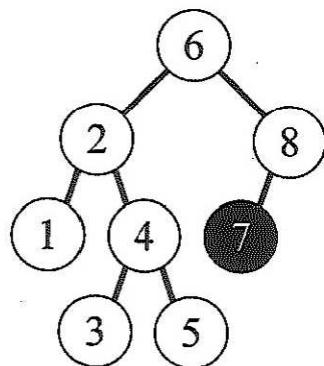
**Aufgabe 3: „AVL-Bäume“**

- a) Fügen Sie die Zahlen (7, 6, 2, 1, 5, 3, 8, 4) in dieser Reihenfolge in einen anfangs leeren AVL Baum ein. Stellen Sie die AVL Eigenschaft ggf. nach jedem Einfügen mit geeigneten Rotationen wieder her. Zeichnen Sie den AVL Baum einmal vor und einmal nach jeder einzelnen Rotation.
- b) Entfernen Sie den jeweils markierten Knoten aus den folgenden AVL-Bäumen. Stellen Sie die AVL-Eigenschaft ggf. durch geeignete Rotationen wieder her. Zeichnen Sie nur den resultierenden Baum.

i) Baum 1:

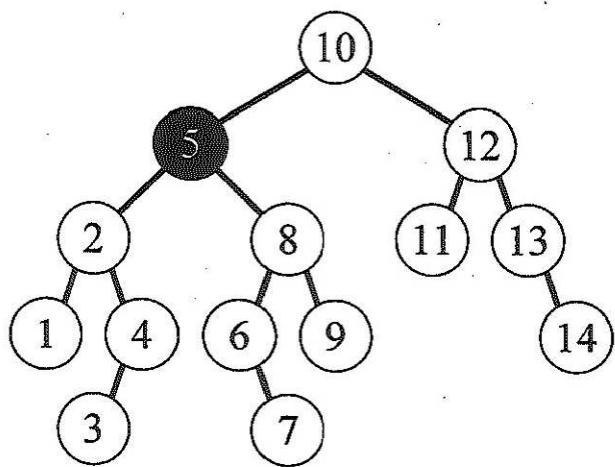


ii) Baum 2:



Fortsetzung nächste Seite!

iii) Baum 3:



Fortsetzung nächste Seite!

**Aufgabe 4:**

Zeigen oder widerlegen Sie die folgenden Aussagen (die jeweiligen Beweise sind sehr kurz):

- a) Sei  $L \subseteq \Sigma^*$ . Ist  $L$  regulär, so gilt das Pumping-Lemma für kontextfreie Sprachen auch für  $L$ .
- b) Sei  $H$  das (allgemeine) Halteproblem (kodiert mit Zeichen über  $\Sigma$ ), und sei  $\overline{H} = \Sigma^* \setminus H$  das Komplement des (allgemeinen) Halteproblems. Dann kann  $\overline{H}$  auf  $H$  reduziert werden, d. h. es gilt:  $\overline{H} \leq H$ .
- c) Der Standard-Beweis dafür, dass CLIQUE NP-schwer ist, zeigt, dass  $SAT \leq_p CLIQUE$ . Stimmt es auch, dass  $CLIQUE \leq_p SAT$  ist?

Schreiben Sie zuerst zur Aussage „Stimmt“ oder „Stimmt nicht“ und dann Ihre Begründung.

**Fortsetzung nächste Seite!**

**Aufgabe 5:**

a) Definieren Sie die zum Halteproblem für Turing-Maschinen bei fester Eingabe  $m \in \mathbb{N}_0 = \{0, 1, 2, \dots\}$  gehörende Menge  $H_m$ .

b) Gegeben sei das folgende Problem  $E$ :

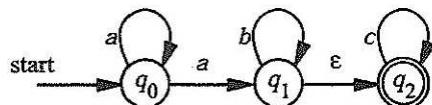
- Entscheide, ob es für die deterministische Turing-Maschine mit der Gödelnummer  $n$  eine Eingabe  $w \in \mathbb{N}_0$  gibt, so dass  $w$  eine *gerade* Zahl ist und die Maschine  $n$  gestartet mit  $w$  hält.

Zeigen Sie, dass  $E$  nicht entscheidbar ist. Benutzen Sie, dass  $H_m$  aus (a) für jedes  $m \in \mathbb{N}_0$  nicht entscheidbar ist.

c) Zeigen Sie, dass das Problem  $E$  aus (b) partiell-entscheidbar (= rekursiv aufzählbar) ist.

**Aufgabe 6:**

a) Gegeben sei der folgende nichtdeterministische endliche Automat  $N$ :



Konstruieren Sie zu  $N$  mit den Potenzmengen-Konstruktionen einen äquivalenten deterministischen endlichen Automaten  $A$ . Zeichnen Sie nur die vom Startzustand erreichbaren Zustände ein, die aber *alle*. Die Zustandsnamen von  $A$  müssen erkennen lassen, wie sie zustande gekommen sind. Führen Sie *keine* „Vereinfachungen“ durch.

*Hinweis:* In einem deterministischen endlichen Automaten darf es keine  $\epsilon$ -Übergänge geben, und es muss an jedem Zustand für jedes Zeichen einen Übergang geben.

b) Geben Sie einen regulären Ausdruck  $\alpha(N)$  für die Sprache, die der nichtdeterministische endliche Automat  $N$  aus (a) akzeptiert, an.

c) Beweisen oder widerlegen Sie die folgende Behauptung.

Beh.: Ist die Sprache  $L$  nicht regulär, dann ist auch jede echte Obermenge von  $L$  nicht regulär.

d) Sei  $A$  ein deterministischer endlicher Automat mit Zustandsmenge

$Q = \{q_1, \dots, q_n\}$ , Startzustand  $q_1$ , Alphabet  $\Sigma$  und Endzustandsmenge  $F$ .

Im Zusammenhang der Umwandlung endlicher Automaten in reguläre Ausdrücke wird die Menge  $R_{ij}^k = \{w \in \Sigma^* \mid \delta(q_i, w) = q_j, \text{ kein echter Zwischenzustand hat Index } > k\}$  betrachtet.

d1) Wie lautet die Rekursionsformel für die  $R_{ij}^k$ , die die Grundlage des Umwandlungsalgorithmus ist?

Hinweis: Es werden die drei Fälle unterschieden: (i)  $k = 0$ ,  $i \neq j$ , (ii)  $k = 0$ ,  $i = j$  und (iii) der Sonst-Fall.

d2) Wie ergibt sich aus den Mengen  $R_{ij}^k$  dann die Sprache  $L(A)$  des Automaten?

**Aufgabe 7:**

a) Sei SAT das Erfüllbarkeitsproblem, und sei  $H_m$  das Halteproblem bei fester Eingabe  $m$  (siehe Aufgabe 5).

Zeigen Sie: SAT kann in *polynomieller* Zeit auf  $H_m$  reduziert werden. (Als Relation:  $\text{SAT} \leq_p H_m$ )

b) Angenommen, es wurde gezeigt, dass  $P = NP$  ist. Zeigen Sie, dass dann *jede* Sprache  $L \in P$  über dem Alphabet  $\Sigma$  mit  $L \neq \emptyset$  und  $L \neq \Sigma^*$  sogar NP-vollständig ist.

## Thema Nr. 2

### Teilaufgabe I

Seien  $\Sigma, \Delta$  Alphabete. Ein Homomorphismus ist eine Funktion  $f : \Sigma^* \rightarrow \Delta^*$  sodass  $f(uv) = f(u)f(v)$  für alle  $u, v \in \Sigma^*$ . Es ist klar, dass dann  $f$  schon eindeutig durch die Werte  $f(a)$  für  $a \in \Sigma$  bestimmt ist.

Seien  $f, g : \Sigma^* \rightarrow \Delta^*$  Homomorphismen. Der *Differenzkern*  $E(f, g) \subseteq \Sigma^*$  ist definiert durch

$$E(f, g) = \{w \mid f(w) = g(w)\}$$

Ist zum Beispiel  $f(a) = 00$ ,  $f(b) = 0$ ,  $g(a) = 0$  und  $g(b) = 00$ , so gilt für  $w = ababab$  dass  $f(w) = 00\ 0\ 00\ 0\ 00\ 0 = 0\ 00\ 0\ 00\ 0\ 00 = g(w)$  und es ist hier  $E(f, g) = \{w \mid 2|w|_a + |w|_b = |w|_a + 2|w|_b\} = \{w \mid |w|_a = |w|_b\}$ . Wie üblich ist  $|w|_x$  die Zahl der Buchstaben  $x$  im Wort  $w$ .

Von den folgenden vier Aussagen ist genau eine falsch, die anderen drei sind wahr. Identifizieren Sie die falsche Aussage, erklären Sie, warum sie falsch ist und zeigen Sie, dass die anderen drei Aussagen wahr sind.

1. Seien  $f, g$  Homomorphismen. Falls  $E(f, g)$  durch einen nichtdeterministischen Kellerautomaten (PDA) erkannt wird, so ist  $E(f, g)$  kontextfrei.
2. Für alle  $f, g$  kann  $E(f, g)$  durch einen PDA wie folgt erkannt werden: Sei die Eingabe  $aw$  mit  $a \in \Sigma$ ,  $w \in \Sigma^*$ . Man vergleiche  $f(a)$  mit  $g(a)$ . Falls keines von beiden Präfix des anderen ist, so verwirfe man; anderenfalls lege man die Differenz der beiden auf den Keller, merke sich, auf welcher Seite noch etwas fehlt und fahre fort, wobei dann natürlich der Kellerinhalt auch noch zu berücksichtigen ist.
3. Die Sprache  $L = \{w \mid |w|_a = |w|_b = |w|_c\}$  ist nicht kontextfrei.
4. Für  $L$  aus 3) gilt  $L = E(f, g)$  mit  $f, g : \Sigma^* \rightarrow \{0, 1\}^*$  definiert durch  $f(a) = 00$ ,  $f(b) = 1$ ,  $f(c) = 1$ ,  $g(a) = 0$ ,  $g(b) = 0$ ,  $g(c) = 11$ .

## Teilaufgabe II

Wir definieren das Problem INTPOLY wie folgt:

GEGEBEN: Eine Liste von Variablen  $(x_1, \dots, x_n)$  und ein Polynom  $p(x_1, \dots, x_n)$  in diesen Variablen und mit ganzzahligen Koeffizienten.

GEFRAGT: Hat dieses Polynom eine ganzzahlige Nullstelle, d.h. existieren ganze Zahlen  $z_1, \dots, z_n$ , sodass  $p(z_1, \dots, z_n) = 0$ .

So sind beispielsweise  $((x, y, z), x^2 + y^2 - z^2)$  und  $((x, y, z), x^2 - 2xy + y^2 + 1)$  beides Instanzen, aber nur die erste ist eine JA Instanz, also formal ein Element von INTPOLY. Es ist z.B.  $3^2 + 4^2 - 5^2 = 0$ , aber  $x^2 - 2xy + y^2 + 1 = (x - y)^2 + 1 > 0$  für alle  $x, y, z$ .

1. Zeigen Sie durch Reduktion von 3SAT, dass INTPOLY NP-schwierig ist. Zeigen Sie also, dass  $\text{3SAT} \leq_P \text{INTPOLY}$ . Legen Sie zunächst dar, aus welchen Daten so eine Reduktion besteht und was über diese Daten zu zeigen ist.

Weitere Hinweise:  $p(\vec{x})q(\vec{x}) = 0 \iff p(\vec{x}) = 0 \vee q(\vec{x}) = 0$  und  $p(\vec{x})^2 + q(\vec{x})^2 = 0 \iff p(\vec{x}) = 0 \wedge q(\vec{x}) = 0$ . Für jede Boole'sche Variable  $A$  bietet es sich an, zwei ganzzahlige Variablen  $x_A$  und  $x_{\neg A}$  einzuführen.

2. Interessanterweise ist INTPOLY ein unentscheidbares Problem (das wurde in den 1970er Jahren recht aufwendig bewiesen). Nun behauptet aber jemand, INTPOLY sei in NP, denn, wenn  $\vec{x}$  und  $p(\vec{x})$  gegeben sind, dann könne man ja nichtdeterministisch ganze Zahlen  $z_1, \dots, z_n$  raten und dann prüfen, ob  $p(\vec{z}) = 0$  ist. Beschreiben Sie genau, an welcher Stelle diese Person irrt.

3. Gibt es unentscheidbare Probleme in NP ?

## Teilaufgabe III

Gegeben sei ein Array  $A[1], \dots, A[n]$  von Zahlen. Es soll ein zweidimensionales Array  $B[1, 1] \dots B[n, n]$  bestimmt werden, sodass gilt

$$B[i, j] = A[i] + A[i + 1] + \dots + A[j]$$

falls  $i \leq j$  und  $B[i, j] = 0$ , falls  $i > j$ . Folgender Algorithmus leistet das Verlangte:

```
for i := 1, ..., n
    for j := 1, ..., n
        B[i,j]:=0;
        for t := i ... j
            B[i,j]:=B[i,j]+A[t];
```

- Geben Sie eine Funktion  $f(n)$  an, sodass die Laufzeit dieses Algorithmus  $\Theta(f(n))$  ist. Sollte Ihnen die  $\Theta$ -Notation nicht geläufig sein, so können Sie alternativ eine *möglichst schwach wachsende* Funktion  $f$  bestimmen, sodass die Laufzeit  $O(f(n))$  ist.
- Finden Sie einen Algorithmus für dieselbe Problemstellung mit einer echt besseren Laufzeit. Gesucht ist also ein Algorithmus für unser Problem mit einer Laufzeit  $O(g(n))$  wobei  $\lim_{n \rightarrow \infty} g(n)/f(n) = 0$ . Geben Sie sowohl den Algorithmus, als auch die neue Laufzeitschranke  $g(n)$  an.

[Nach Kleinberg-Tardos. Algorithm Design. Pearson 2005]

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2014**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	<b>Herbst 2014</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

---

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**  
**— Prüfungsaufgaben —**

---

Fach: **Informatik (vertieft studiert)**

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!

**Aufgabe 1:**

Wir fixieren das Alphabet  $\Sigma = \{0, 1\}$  und definieren  $L \subseteq \Sigma^*$  durch

$$L = \{w \mid \text{in } w \text{ kommt das Teilwort } 0010 \text{ höchstens einmal vor}\}$$

Es ist also  $000 \in L$ ,  $00010 \in L$ , aber  $00010000100 \notin L$  und auch  $0010010 \notin L$ .

- a) Zeigen Sie, dass  $L$  regulär ist.
- b) Vervollständigen Sie durch Hinzufügen eines weiteren Zustandes, sowie von Kanten und der Angabe der Endzustände folgendes Diagramm zu einem deterministischen Automaten für  $L$ . Übertragen Sie dazu das Diagramm auf Ihr Arbeitspapier.

$$\rightarrow q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_4 \xrightarrow{0} q_5 \xrightarrow{1} q_6 \xrightarrow{0} q_7$$

- c) Zeigen Sie durch Ausführung des Minimierungsalgorithmus, dass dieser Automat minimal ist.
- d) Geben Sie die Äquivalenzklassen der Myhill-Nerode Äquivalenz von  $L$  durch Repräsentanten an. (Diese Äquivalenz ist definiert durch  $x \sim_L y \iff \forall u. xu \in L \iff yu \in L$ .)

Geben Sie zu zwei Klassen Ihrer Wahl neben dem gewählten Repräsentanten noch ein weiteres Element an.

Geben Sie für die Klasse, in der sich das leere Wort befindet, einen regulären Ausdruck an.

**Aufgabe 2:**

Ordnen Sie die folgenden formalen Sprachen bestmöglich in die Chomsky-Hierarchie ein und geben Sie eine ausreichende Begründung an:

- a)  $L_1 = \{(ab)^n a (ab)^n \mid n \geq 1\}$
- b)  $L_2 = \{(ab)^n (ab)^n \mid n \geq 1\}$
- c)  $L_3 = \{(ab)^n (ab)^n (ab)^n \mid n \geq 1\}$
- d)  $L_4 = \{(ab)^n (ab)^n a (ab)^n \mid n \geq 1\}$
- e)  $L_5 = \{(ab)^n a (ab)^n a (ab)^n \mid n \geq 1\}$ . (Bei dieser Teilaufgabe ist keine Begründung notwendig.)
- f)  $L_6$  sei die Menge aller syntaktisch korrekten Java-Programme, die ohne Eingabe nicht terminieren.
- g)  $L_7$  sei die Menge aller wohlgeformten Klammerausdrücke, also  $((), (((), ((())() \in L_7$ , aber  $), (), ((()) \notin L_7$ .

**Aufgabe 3:**

Bekanntlich sind beim RUCKSACK-Problem natürliche Zahlen  $a_1, \dots, a_n$  und eine Zielzahl  $b$  gegeben und es ist gefragt, ob eine Teilmenge  $I \subseteq \{1, \dots, n\}$  existiert, sodass  $\sum_{i \in I} a_i = b$ . Dieses Problem ist bekanntermaßen NP-vollständig.

- a) Zeigen Sie, dass die Variante GRUCKSACK, bei der die Zahlen  $a_i$  und  $b$  allesamt gerade sein müssen, ebenfalls NP-vollständig ist.
- b) Zeigen Sie, dass die Variante BZRUCKSACK, bei der die Zahl  $b$  eine Zweierpotenz sein muss, ebenfalls NP-vollständig ist.
- c) In welche Komplexitätsklasse fällt die Variante ZARUCKSACK, bei der es nur zwei verschiedene Arten von Gewichten gibt, also zwei Zahlen  $u, v$  existieren, sodass  $a_i \in \{u, v\}$  für  $i = 1, \dots, n$ ?

**Aufgabe 4:**

Es seien natürliche Zahlen  $a_1, \dots, a_n$  in Form eines Arrays gegeben. Die Zahlen  $a_i$  steigen bis zu einem gewissen Punkt an und fallen anschließend wieder. Formal existiert also  $m \leq n$ , sodass Folgendes gilt: Ist  $1 \leq i < m$ , so ist  $a_i < a_{i+1}$ . Ist  $m \leq i < n$ , so ist  $a_i > a_{i+1}$ .

- a) Entwerfen Sie einen Algorithmus, der den größten Eintrag des Arrays, also  $\max_i a_i$  in Zeit  $O(\log n)$  bestimmt. Genügt die Eingabe nicht der Spezifikation, so darf sich Ihr Algorithmus völlig beliebig verhalten.
- b) Jetzt erlauben wir auch die Möglichkeit, dass die Arraywerte "liegenbleiben", also dass  $m < n$  existiert, sodass gilt: Ist  $1 \leq i < m$ , dann ist  $a_i \leq a_{i+1}$ . Ist  $m \leq i < n$ , dann ist  $a_i \geq a_{i+1}$ .

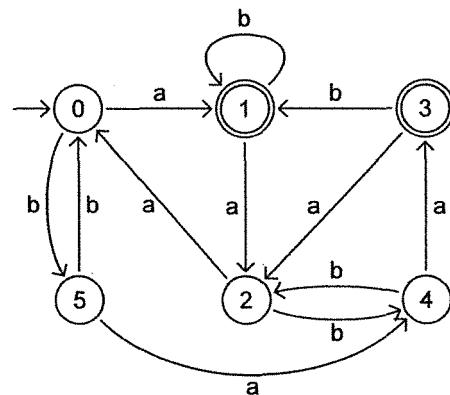
Begründen Sie, dass kein Algorithmus mit Laufzeit  $O(\log n)$  existiert, der entscheidet, ob ein Array von der oben beschriebenen Form ist. Hilfe: Lassen Sie einen hypothetischen Algorithmus auf der Eingabe  $a_i = 0$  für  $i = 1, \dots, n$  laufen.

Thema Nr. 2  
 (Aufgabengruppe)

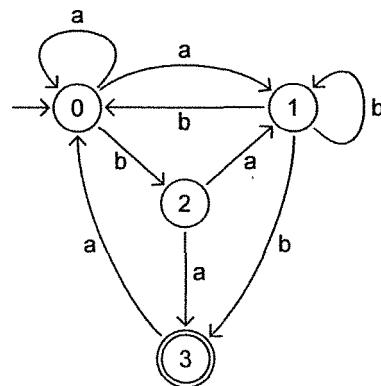
Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

- a) Minimieren Sie folgenden deterministischen, endlichen Automaten.



- b) Bestimmen Sie einen regulären Ausdruck für die von dem folgenden nichtdeterministischen, endlichen Automaten akzeptierte Sprache.



**Aufgabe 2:**

Beweisen Sie folgende Aussagen.

- a)  $L_1 = \{w \in \{0,1\}^* \mid \text{die Länge von } w \text{ ist durch 2 oder 3 teilbar}\}$  ist regulär.  
 b)  $L_2 = \{w \in \{0,1\}^* \mid w = 0^i 1^j 0^k \text{ mit } i, j, k \in \mathbb{N} \text{ und } j = i + k\}$  ist kontextfrei.

**Aufgabe 3:**

- a) Definieren Sie eine berechenbare Funktion  $f: \mathbb{N} \rightarrow \mathbb{N}$  mit entscheidbarem Definitionsbereich und unentscheidbarem Wertebereich. Begründen Sie Ihre Aussagen.
- b) Beweisen oder widerlegen Sie folgende Aussagen.
- Jede berechenbare Funktion  $h: \mathbb{N} \rightarrow \mathbb{N}$  mit endlichem Wertebereich besitzt einen entscheidbaren Definitionsbereich.
  - Jede berechenbare Funktion  $h: \mathbb{N} \rightarrow \mathbb{N}$  mit endlichem Definitionsbereich besitzt einen entscheidbaren Wertebereich.

**Aufgabe 4:**

- a) Die Fibonacci-Funktion  $fib: \mathbb{N} \rightarrow \mathbb{N}$  ist definiert durch

$$fib(x) = \begin{cases} x, & \text{falls } x \in \{0, 1\} \\ f(x-1) + f(x-2) & \text{sonst.} \end{cases}$$

Beweisen Sie, dass  $fib$  nicht in polynomieller Zeit berechenbar ist.

- b) Sei  $f: \mathbb{N} \rightarrow \mathbb{N}$  eine totale, in polynomieller Zeit berechenbare Funktion mit  $f(x) \geq x$  für alle  $x \in \mathbb{N}$ .  $W_f$  bezeichne den Wertebereich der Funktion  $f$ . Beweisen Sie, dass  $W_f \in NP$ .

**Aufgabe 5:**

Gegeben sei eine Standarddatenstruktur Stapel (Stack) mit den Operationen

- void Push(Element e),
- Element Pop(),
- Boolean Empty().

sowie dem Standardkonstruktor Stapel(), der einen leeren Stapel zur Verfügung stellt.

- a) Geben Sie eine Methode Stapel Merge(Stapel  $S$ , Stapel  $T$ ) an, die einen aufsteigend geordneten Stapel zurückgibt, unter der Bedingung, dass die beiden übergebenen Stapel aufsteigend sortiert sind, d.h.  $S.Pop()$  liefert das größte Element in  $S$  zurück und  $T.Pop()$  liefert das größte Element in  $T$  zurück. Als Hilfsdatenstruktur dürfen Sie nur Stapel verwenden, keine Felder oder Listen.
- b) Analysieren Sie die Laufzeit Ihrer Methode.

**Aufgabe 6:**

Gegeben sei ein einfacher Sortieralgorithmus, der ein gegebenes Feld  $A$  dadurch sortiert, dass er das Minimum  $m$  von  $A$  findet, dann das Minimum von  $A$  ohne das Element  $m$  usw.

- a) Geben Sie den Algorithmus in Pseudocode an. Implementieren Sie den Algorithmus *in situ*, d.h. so, dass er außer dem Eingabefeld nur konstanten Extraspeicher benötigt.
- b) Analysieren Sie die Laufzeit Ihres Algorithmus.
- c) Geben Sie eine Datenstruktur an, mit der Sie Ihren Algorithmus beschleunigen können.

**Aufgabe 7:**

Gegeben sei folgende Adjazenzmatrix des Graphen G („ $\infty$ “ bedeutet, dass zwischen den beiden Knoten keine direkte Verbindung existiert):

	Q	A	B	C	D	E	F
Q	0	1	3	$\infty$	5	$\infty$	$\infty$
A	1	0	$\infty$	$\infty$	3	1	$\infty$
B	3	$\infty$	0	4	1	$\infty$	$\infty$
C	$\infty$	$\infty$	4	0	2	$\infty$	2
D	5	3	1	2	0	3	1
E	$\infty$	1	$\infty$	$\infty$	3	0	8
F	$\infty$	$\infty$	$\infty$	2	1	8	0

- a) Bestimmen Sie mit dem Algorithmus von *Dijkstra* den kürzesten Weg von der Quelle Q bis zu den Senken C und F. Verwenden Sie zur Lösung eine Tabelle nach unten stehendem Muster, markieren Sie in jeder Zeile den jeweils als nächstes zu betrachtenden Knoten und führen Sie die Prioritätsliste der noch zu betrachtenden Knoten in der letzter Spalte (der nächste Knoten steht links):

Q	A	B	C	D	E	F	Queue
0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	Q

- b) Geben Sie den vorangehend ermittelten, kürzesten Weg von Q zu F als Knotenfolge an.  
 c) Bestimmen Sie mit Hilfe des Verfahrens nach *Kruskal* den kleinsten Spannbaum des Graphen G. Geben Sie die Kanten (Knotenpaare) in der Reihenfolge an, in der Sie sie dem minimalen Spannbaum gemäß Kruskal hinzufügen würden. Wie groß ist die Kantengewichtssumme im Spannbaum insgesamt?

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2015**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	Frühjahr	
Kennwort: _____	2015	66115
Arbeitsplatz-Nr.: _____		

---

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**  
**— Prüfungsaufgaben —**

---

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 7

---

Bitte wenden!

**Thema Nr. 1**  
**(Aufgabengruppe)**

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Die Sprache  $L$  über dem Alphabet  $\Sigma = \{0, 1\}$  enthält alle Wörter, bei denen beim Lesen von links nach rechts der Unterschied in der Zahl der 0en und 1en stets höchstens 3 ist. Also ist  $w \in L$  genau dann, wenn für alle  $u, v$  mit  $w = uv$  gilt  $||u|_0 - |u|_1| \leq 3$ . Erinnerung:  $|w|_a$  bezeichnet die Zahl der  $a$ 's im Wort  $w$ .

- Begründen Sie, dass  $L$  regulär ist.
- Jemand behauptet, diese Sprache sei nicht regulär und gibt folgenden "Beweis" dafür an: *Wäre  $L$  regulär, so sei  $n$  eine entsprechende Pumping-Zahl. Nun ist  $w = (01)^n \in L$ . Zerlegt man nun  $w = uxv$ , wobei  $u = 0$ ,  $x = 1$ ,  $v = (01)^{n-1}$ , so ist zum Beispiel  $ux^5v \notin L$ , denn es ist  $ux^5v = 011111010101\dots$*   
 Legen Sie genau dar, an welcher Stelle dieser "Beweis" fehlerhaft ist.
- Sei  $A = (Q, \delta, q_0, E)$  ein nichtdeterministischer endlicher Automat für  $L$ . Es sei  $w_1 = 111$ ,  $w_2 = 11$ ,  $w_3 = 1$ ,  $w_4 = \epsilon$ ,  $w_5 = 0$ ,  $w_6 = 00$ ,  $w_7 = 000$ . Machen Sie sich klar, dass der Automat jedes dieser Wörter verarbeiten können muss. Foltern Sie, dass der Automat mindestens sieben Zustände haben muss. Schreiben Sie Ihre Argumentation schlüssig und vollständig auf.
- In anderen Fällen können nichtdeterministische endliche Automaten echt kleiner sein als die besten deterministischen Automaten. Ein Beispiel ist die Sprache  $L_2 = \Sigma^* 1 \Sigma$  aller Wörter, deren vorletztes Symbol 1 ist. Geben Sie einen nichtdeterministischen Automaten mit nur drei Zuständen an, der  $L_2$  erkennt.
- Führen Sie auf Ihrem Automaten die Potenzmengenkonstruktion und anschließend den Minimierungsalgorithmus durch. Wie viele Zustände muss ein deterministischer Automat für  $L_2$  also mindestens haben?

**Aufgabe 2:**

Gegeben sind eine Menge  $A$  von Angestellten und für jede Angestellte  $a$  eine Menge  $F(a)$  von Fähigkeiten, die diese mitbringt. Außerdem gibt es eine Menge  $Z$  von Paaren von Angestellten, die nicht gut miteinander zusammenarbeiten. Für eine gegebene Menge  $P$  ("Projekt") von Fähigkeiten soll nun entschieden werden, ob eine Teilmenge  $T$  ("Team") der Angestellten existiert, sodass  $P \subseteq \bigcup_{a \in T} F(a)$  und für kein Paar  $(a, a') \in Z$  sowohl  $a$ , als auch  $a'$  in  $P$  sind. Alle Mengen sind endlich und durch explizite Aufzählung gegeben.

- Begründen Sie, dass dieses Problem in NP liegt.
- Zeigen Sie, dass das Problem NP-vollständig ist, indem Sie (zum Beispiel) eine Reduktion von 3SAT angeben.

**Aufgabe 3:**

- a) Geben Sie für jede der Chomsky-Sprachklassen I-IV eine Sprache an, die in der jeweiligen Klasse liegt, aber nicht in der nächstniedrigeren.
- b) Nennen Sie auch eine Sprache, die nicht einmal in der Klasse IV liegt.

**Aufgabe 4:**

In der automatischen Programmverifikation kommt folgende graphentheoretische Grundaufgabe häufig vor.

Gegeben ist ein gerichteter Graph  $G = (V, E)$ , zwei Teilmengen  $A, B \subseteq V$  und ein Startknoten  $v_0 \in V$ . Ein "Lasso" ist eine Folge von Knoten  $v_0, v_1, v_2, \dots, v_m, v_{m+1}, \dots, v_n$  beginnend beim Startknoten mit  $m < n$  und  $(v_i, v_{i+1}) \in E$  für  $i = 0, \dots, n - 1$  und  $v_n = v_m$ . Gefragt ist, ob ein Lasso existiert, in dem kein Knoten aus  $A$  vorkommt und dessen Schleife einen Knoten aus  $B$  enthält. Es soll also  $A \cap \{v_0, \dots, v_n\} = \emptyset$  und  $B \cap \{v_m, v_{m+1}, \dots, v_n\} \neq \emptyset$  sein.

(Bemerkung: Der Graph könnte die Zustände und Zustandsübergänge eines nebenläufigen Systems repräsentieren.)

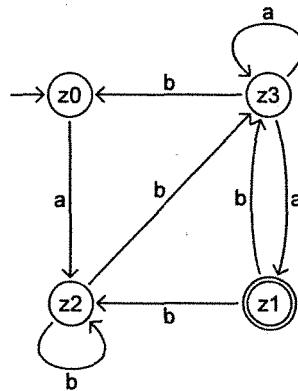
- a) Beschreiben Sie, wie man dieses Problem unter Zuhilfenahme der Tiefensuche lösen kann.
- b) Schätzen Sie die Laufzeit Ihrer Lösung als Funktion von  $|V|$  und  $|E|$  unter Verwendung der O-Notation ab und begründen Sie Ihre Abschätzung. Ihre Lösung sollte mindestens die Schranke  $O(|V|(|V| + |E|))$  erfüllen.
- c) Man kann diese Aufgabe auch in Zeit  $O(|V| + |E|)$  lösen. Diskutieren Sie mögliche Verbesserungsansätze Ihrer Lösung. Sie müssen das  $O(|V| + |E|)$ -Verfahren nicht finden oder kennen und dürfen in diesem Teil auch nicht vollständig durchgeführte Ideen ansprechen. Sollten Sie bereits oben ein  $O(|V| + |E|)$ -Verfahren gefunden haben, entfällt diese Teilaufgabe.

**Thema Nr. 2**  
**(Aufgabengruppe)**

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Gegeben ist der folgende nichtdeterministische, endliche Automat  $A$ , der Wörter über dem Alphabet  $\{a, b\}$  verarbeitet. Die von  $A$  akzeptierte Sprache bezeichnen wir mit  $L(A)$ .



- Geben Sie eine Typ-3-Grammatik (auch bekannt unter dem Namen *reguläre Grammatik*) für die Sprache  $L(A)$  an. Dazu müssen Sie die Menge der Terminalen, die Menge der Nichtterminale, das Startsymbol und die Regelmenge angeben.
- Konstruieren Sie einen deterministischen, endlichen Automaten, der die Sprache  $L(A)$  akzeptiert.

**Aufgabe 2:**

Beweisen Sie, dass folgende Menge nicht regulär ist.

$$B = \{w \in \{0\}^* \mid \text{die Länge von } w \text{ ist eine Zweierpotenz}\}$$

**Aufgabe 3:**

Für ein Wort  $w$  und einen Buchstaben  $e$  bezeichnet  $|w|_e$  die Anzahl der Buchstaben  $e$  im Wort  $w$ .

$$C = \{w \in \{a, b, c\}^* \mid |w|_a + |w|_b = |w|_c\}$$

- Geben Sie einen nichtdeterministischen Kellerautomaten an, der die Sprache  $C$  akzeptiert.
- Erklären Sie die Arbeitsweise Ihres Kellerautomaten.

**Aufgabe 4:**

Sei  $M_0, M_1, \dots$  eine Gödelisierung aller Registermaschinen (RAMs). Geben Sie für jede der Mengen  $D_1, D_2, D_3$  an, ob sie entscheidbar ist und ob sie aufzählbar ist. Beweisen Sie Ihre Behauptungen, wobei Sie die Aufzählbarkeit und Unentscheidbarkeit des speziellen Halteproblems

$K_0 = \{x \in \mathbb{N} \mid M_x \text{ hält bei Eingabe } x\}$  verwenden dürfen.

$$\begin{aligned} D_1 &= \{x \in \mathbb{N} \mid x < 9973 \text{ und } M_x \text{ hält bei Eingabe } x\} \\ D_2 &= \{x \in \mathbb{N} \mid x \geq 9973 \text{ und } M_x \text{ hält bei Eingabe } x\} \\ D_3 &= \{x \in \mathbb{N} \mid M_x \text{ hält nicht bei Eingabe } x\} \end{aligned}$$

### Aufgabe 5:

Gegeben seien die Standarddatenstrukturen Stapel (Stack) und Schlange (Queue) mit folgenden Standardoperationen:

Stapel	Schlange
Boolean Empty()	Boolean Empty()
Push(Zahl e)	Enqueue(Zahl e)
Zahl Pop()	Zahl Dequeue()
Zahl Top()	Zahl Head()

Beim Stapel gibt die Operation `Top()` das gleiche Element wie `Pop()` zurück, bei der Schlange gibt `Head()` das gleiche Element wie `Dequeue()` zurück. Im Unterschied zu `Pop()`, beziehungsweise `Dequeue()`, wird das Element bei `Top()` und `Head()` nicht aus der Datenstruktur entfernt.

- Geben Sie in Pseudocode einen Algorithmus `Sort(Stapel S)` an, der als Eingabe einen Stapel  $S$  mit  $n$  Zahlen erhält und die Zahlen in  $S$  sortiert. (Sie dürfen die Zahlen wahlweise entweder aufsteigend oder absteigend sortieren.) Verwenden Sie als Hilfsdatenstruktur ausschließlich eine Schlange  $Q$ . Sie erhalten volle Punktzahl, wenn Sie außer  $S$  und  $Q$  keine weiteren Variablen benutzen. Sie dürfen annehmen, dass alle Zahlen in  $S$  verschieden sind.
- Analysieren Sie die Laufzeit Ihrer Methode in Abhängigkeit von  $n$ .

### Aufgabe 6:

Ein binäres Feld (ein Feld über den Zahlen 0 und 1) sei genau dann ein *gutes* Feld, wenn die Methode `Boolean IstGut(Feld A)` als Ausgabe `true` zurückgibt.

Die Laufzeit von `IstGut(Feld A)` ist linear zu der Anzahl der Zahlen in  $A$ . Sie dürfen in den folgenden Teilaufgaben die Methode `IstGut(Feld A)` aufrufen.

- Geben Sie einen Algorithmus `int AnzahlGute(int n)` in Pseudocode an, der die Anzahl aller guten binären Felder der Länge  $n$  zurückgibt. Was ist die Laufzeit Ihres Algorithmus in Abhängigkeit von  $n$ ?

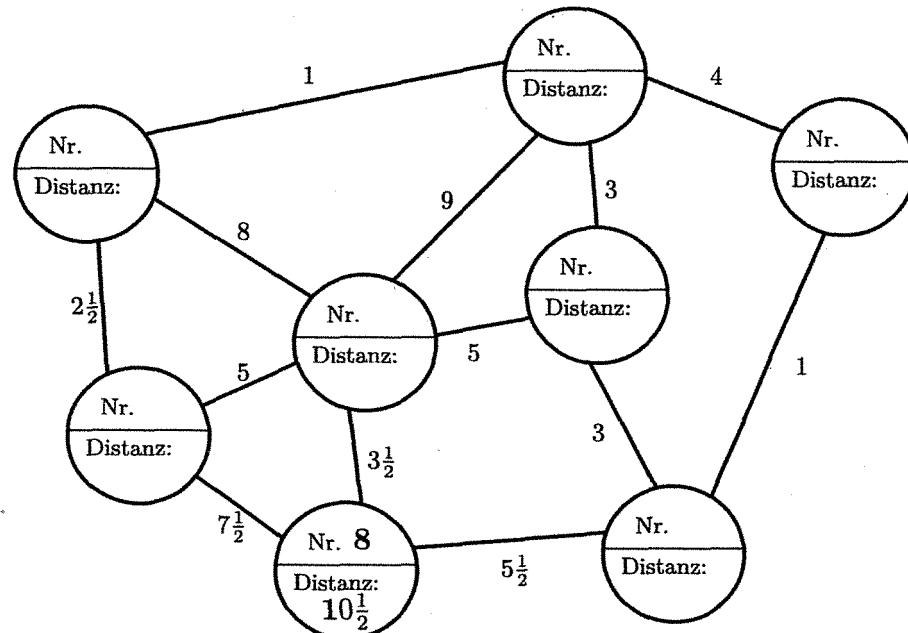
- b) Geben Sie einen *iterativen* Algorithmus `int AnzahlGute(int n)` in Pseudocode an, der die Anzahl aller guten binären Felder der Länge  $n$  zurückgibt, die genau zwei Einser enthalten. Die Laufzeit soll in  $O(n^3)$  sein. Sie brauchen die Laufzeit nicht zu zeigen.
- c) Geben Sie einen *rekursiven* Algorithmus `int AnzahlGuteRek(int n, int k)` in Pseudocode an, der die Anzahl aller guten binären Felder der Länge  $n$  zurückgibt, die genau  $k$  Einser enthalten. Die Laufzeit soll in  $O(n \binom{n}{k})$  sein. Sie brauchen die Laufzeit nicht zu zeigen.

### Aufgabe 7:

Auf folgendem ungerichteten, gewichteten Graphen wurde der Dijkstra-Algorithmus (wie auf der nächsten Seite beschrieben) ausgeführt, doch wir wissen lediglich, welcher Knoten als letztes schwarz (*black*) wurde (Nr. 8) und was seine Distanz zum Startknoten (Nr. 1) ist. Die Gewichte der Kanten sind angegeben.

Finden Sie zunächst den Startknoten, nummerieren Sie anschließend die Knoten in der Reihenfolge, in der sie schwarz wurden, und geben Sie in jedem Knoten die Distanz zum Startknoten an.

Hinweis: Der Startknoten ist eindeutig.



Dijkstra(WeightedGraph  $G$ , Vertex  $s$ )

```
Initialize( $G, s$ ) ;
 $S = \emptyset$  ;
 $Q = \text{new PriorityQueue}(V, d)$  ;
while not  $Q.Empty()$  do
     $u = Q.ExtractMin()$  ;
     $S = S \cup \{u\}$  ;
    foreach  $v \in Adj[u]$  do
         $\quad \text{Relax}(u, v; w)$ ;
         $u.color = black$ ;
```

Initialize(Graph  $G$ , Vertex  $s$ )

```
foreach  $u \in V$  do
     $u.color = white$  ;
     $u.d = \infty$  ;
 $s.color = gray$  ;
 $s.d = 0$  ;
```

Relax( $u, v; w$ )

```
if  $v.d > u.d + w(u, v)$  then
     $v.color = gray$  ;
     $v.d = u.d + w(u, v)$  ;
     $Q.DecreaseKey(v, v.d)$ 
```

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2015**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	<b>Herbst 2015</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**  
**— Prüfungsaufgaben —**

---

Fach: **Informatik (verieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 10

---

Bitte wenden!

Thema Nr. 1  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Die Sprache  $L$  über dem Alphabet  $\Sigma = \{a, b\}$  enthält alle Wörter, in denen das Wort  $bab$ , oder das Wort  $aba$ , oder beide vorkommen. Also ist z.B.  $babba \in L$ , aber  $bbaabb \notin L$ .

- Begründen Sie, dass  $L$  regulär ist.
- Geben Sie einen nichtdeterministischen Automaten für  $L$  mit 7 Zuständen an. Der Automat soll zu Beginn nichtdeterministisch entscheiden, ob nach  $bab$  oder  $aba$  gesucht wird.
- Führen Sie auf diesem Automaten die Potenzmengenkonstruktion durch und minimieren Sie anschließend den resultierenden deterministischen Automaten.
- Geben Sie ein Beispiel (mit Begründung!) einer Sprache an, welche nicht durch einen *deterministischen* endlichen Automaten mit nur einem Endzustand erkannt werden kann.
- Begründen Sie, dass jede Sprache  $L \subseteq \{a, b\}^+$  (NB:  $\epsilon \notin L$ ) von einem *nichtdeterministischen* Automaten mit nur einem Endzustand erkannt werden kann. Hinweis: Sie können die Sprachen  $U = \{w \mid wa \in L\}$  und  $V = \{w \mid wb \in L\}$  verwenden.

**Aufgabe 2:**

Gegeben ist ein gerichteter Graph  $G = (V, E)$ , der eine Web-Site repräsentieren soll. Die Knoten des Graphen sind die einzelnen Seiten; eine Kante  $(v, v') \in E$  bedeutet, dass die Seite  $v'$  auf der Seite  $v$  verlinkt ist. Weiterhin sind eine Menge  $P$  von "Trails", also typischen Besuchssequenzen gegeben. Jeder solche Trail ist einfach ein Pfad in  $G$ , also eine Folge  $(v_1, \dots, v_n)$  von Knoten mit  $(v_i, v_{i+1}) \in E$  für  $i = 1 \dots n - 1$ . Diese Trails werden durch statistische Analyse des Surfverhaltens der Besucher der Web-Site ermittelt.

Nun soll ein Algorithmus gefunden werden, der Werbeanzeigen möglichst effizient platziert; hierzu soll zu gegebener Zahl  $k$ , eine Auswahl  $W$  von  $k$  Seiten (also Knoten von  $G$ ) ermittelt werden, sodass jeder Trail mindestens eine Seite in  $W$  enthält.

Zeigen Sie durch Reduktion von einem geeigneten Problem aus dem Informatikduden oder aus der von Ihnen besuchten Vorlesung, dass es NP-vollständig ist zu entscheiden, ob bei vorgelegten  $G, P$  und  $k$  eine solche Auswahl  $W$  existiert.

Beachten Sie, dass Sie sowohl die Zugehörigkeit zu NP, als auch die NP-Härte (NP-Schwere) zeigen müssen.

**Aufgabe 3:**

Beim *Postischen Korrespondenzproblem* (PCP) ist bekanntlich eine Liste von Paaren  $(u_1, v_1), \dots, (u_n, v_n)$  mit  $u_i, v_i \in \Sigma^*$  für ein Alphabet  $\Sigma$  gegeben. Zum Beispiel  $(u_1, v_1) = (b, bab)$  und  $(u_2, v_2) = (ba, aa)$  und  $(u_3, v_3) = (abb, bb)$ , wobei hier also  $n = 3$  und  $\Sigma = \{a, b\}$  sind. Gefragt ist, ob es eine Indexfolge  $i_1, i_2, \dots, i_N$ , wobei  $i_k \leq n$  gilt, so dass gilt  $u_{i_1} u_{i_2} \dots u_{i_N} = v_{i_1} v_{i_2} \dots v_{i_N}$ . Im Beispiel wäre  $1, 3, 2, 3$  solch eine Lösung, also genauer  $N = 4$  und  $i_1 = 1, i_2 = 3, i_3 = 2, i_4 = 3$ , denn es ist hier  $u_1 u_3 u_2 u_3 = b \text{ abb } ba \text{ abb} = bab \text{ bb } aa \text{ bb} = v_1 v_3 v_2 v_3$ .

Es ist bekannt, dass das PCP ein unentscheidbares Problem ist.

Zeigen Sie durch Reduktion von PCP, dass es unentscheidbar ist, von einer gegebenen kontextfreien Grammatik  $G = (T, N, S, \rightarrow)$  festzustellen, ob sie eindeutig ist, d.h. ob es für jedes Wort  $w \in L(G)$  genau eine Linksableitung  $S \xrightarrow{*} w$  in  $G$  gibt. Hinweis: Zu einer gegebenen Instanz des PCP betrachten Sie die (kontextfreie!) Sprache  $\{u_{i_1} \dots u_{i_N} i_N \dots i_1 \mid N \in \mathbb{N}, i_1, \dots, i_N \leq n\} \cup \{v_{i_1} \dots v_{i_N} i_N \dots i_1 \mid N \in \mathbb{N}, i_1, \dots, i_N \leq n\}$ .

**Aufgabe 4:**

Sie sollen entlang eines vielbefahrenen Autobahnabschnitts Reklametafeln positionieren. Die möglichen Positionen für solche Tafeln sind  $x_1, \dots, x_n$ , wobei  $x_i$  den Abstand der Position in Kilometern vom Anfang des Abschnitts bezeichnet. Sie können also eine Tafel "bei Autobahnkilometer"  $x_i$  für  $i = 1, \dots, n$  platzieren. Diese Werte  $x_i$  sind aufsteigend sortiert. Außerdem sind Werte  $r_1, \dots, r_n$  gegeben, wobei  $r_i$  den zu erwartenden Gewinn (in 100TEUR) angibt, wenn Sie eine Tafel bei  $x_i$  platzieren.

Allerdings gibt es die Vorschrift, dass zwei aufgestellte Tafeln mehr als 5km Abstand voneinander haben müssen. Platzieren Sie also Tafeln bei  $x_i$  und  $x_j$ , so muss  $|x_i - x_j| > 5$  sein.

Beispiel:  $(x_1, x_2, x_3, x_4) = (6, 7, 12, 14)$  und  $(r_1, r_2, r_3, r_4) = (5, 6, 5, 1)$ . Hier könnten Sie bei km 7 und km 14 werben, was einen Gewinn von 700TEUR liefert, oder aber bei km 6 und 12, was einen Gewinn 1MEUR liefert. Dass das Aufstellen der Tafeln auch Kosten verursacht, bleibt hier außer Betracht.

1. Für jedes  $i \leq n$  sei  $f(i)$  die größte Zahl unterhalb von  $i$ , sodass  $x_i - x_{f(i)} > 5$ . Der Wert  $f(i)$  sei undefiniert, wenn keine solche Zahl existiert. Im Beispiel ist  $f(4) = 2, f(3) = 1$  und  $f(2), f(1)$  sind undefiniert. Beschreiben Sie, wie die Werte  $f(n), f(n-1), \dots, f(1)$  zusammen in Zeit  $O(n)$  bestimmt werden können.
2. Sei  $W_i$  der maximal erzielbare Gewinn bei ausschließlicher Verwendung der Positionen  $x_1, \dots, x_i$ . Überlegen Sie sich, wie man  $W_i$  aus den Werten  $W_j$  für  $j < i$  berechnen kann.
3. Beschreiben Sie, wie  $W_n$  mit dynamischer Programmierung in Zeit  $O(n)$  berechnet werden kann.

Thema Nr. 2  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

"Bäume und Rekursion"

Gegeben sei folgende Klasse **Node** für binäre Bäume:

```
class Node implements Comparable<Node> {
    char c;          // irrelevant for inner nodes
    int f;           // frequency (number of occurrences of c)
    Node zero;      // child
    Node one;       // child

    // creates a leaf
    public Node(char c, int f) {
        this.c = c; this.f = f;
    }

    // creates an inner node
    public Node(Node zero, Node one) {
        this.f = zero.f + one.f;
        this.zero = zero; this.one = one;
    }

    // natural ordering based on the frequency
    @Override
    public int compareTo(Node that) {
        return this.f - that.f;
    }
}
```

Aus der Java-API dürfen Sie auch folgende Methoden verwenden:

HashMap<K,V>: ▶ V put(K key, V value)  
 ▶ V get(Object key)

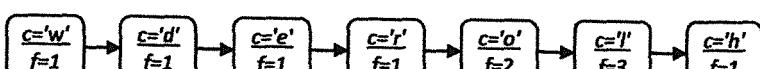
Collections: ▶ static <T extends Comparable<? super T>> void sort(List<T> list)

LinkedList<E>: ▶ int size()  
 ▶ E removeFirst()  
 ▶ void addFirst(E e)

String: ▶ char charAt(int index)  
 ▶ String substring(int beginIndex)

- a) Ergänzen Sie die Methode **count**, welche die HashMap **map** von Node-Objekten anlegt. Für jedes Zeichen **c** des übergebenen Strings **s** soll **map** ein Node-Objekt für dieses Zeichen in **Node.c** enthalten. Am Ende der Schleife gibt **Node.f** jeweils an, wie oft das Zeichen **c** in **s** vorkommt. Das in der letzten Code-Zeile erzeugte Ergebnis von **count** ist die Liste der Node-Objekte aus **map**.

**Beispiel:** Ergebnis für  
**count("helloworld")**

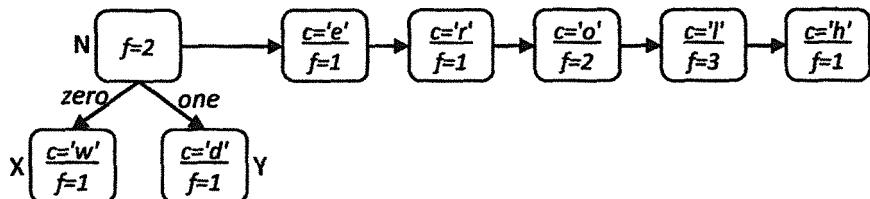


```
LinkedList<Node> count(String s) {
    assert s != null : new IllegalArgumentException();
    HashMap<Character, Node> map = new HashMap<>();
    for (char c : s.toCharArray()) {
        // ToDo: Code hier ergänzen
    }
    return new LinkedList<>(map.values());
}
```

- b) Vervollständigen Sie die Methode `merge`, die zuerst die Node-Liste mit der API-Methode `Collections.sort` sortiert, damit die *kleinsten* Knoten X und Y (bzgl. `compareTo`) am Anfang der Liste stehen. Dann verschmilzt sie X und Y zu einem neuen inneren Knoten N, wobei X zum `zero`-Kind und Y zum `one`-Kind von N wird. Anschließend werden X und Y aus der Liste entfernt und N hinzugefügt.

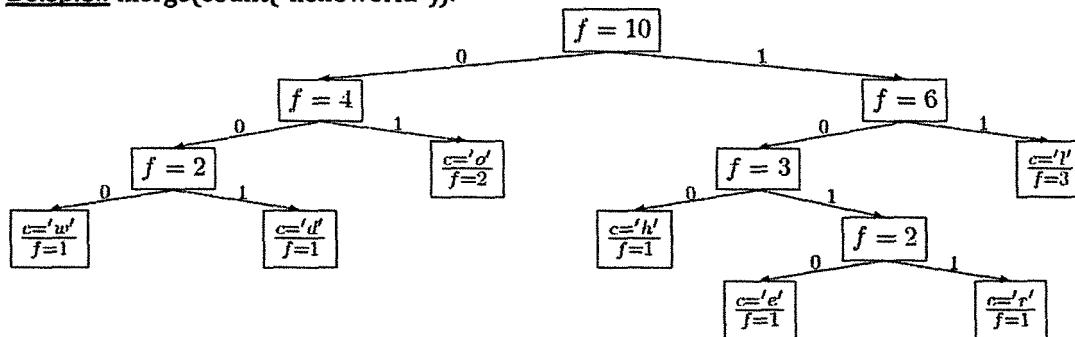
Beispiel:

Erster Durchlauf von `merge(count(s))` für `s = "helloworld"`



Schließlich wiederholt `merge` den vorangehend beschriebenen Vorgang, bis nur noch ein Knoten (die Wurzel des Baums) in der Liste übrig bleibt.

Beispiel: `merge(count("helloworld")):`



```
void merge(LinkedList<Node> nodes) {
    // ToDo: Code hier ergänzen
```

- c) Der Pfad von der Wurzel des resultierenden Baums zu einem Zeichen `c` stellt dessen Huffman-Codierung dar:

$$b \leftrightarrow \overbrace{100}^h \overbrace{1010}^e \overbrace{11}^l \overbrace{11}^l \overbrace{01}^o \overbrace{000}^w \overbrace{01}^o \overbrace{1011}^r \overbrace{11}^l \overbrace{001}^d$$

Ergänzen Sie nun die rekursive Methode `dec`, die eine als String aus 0 und 1 gespeicherte Bitfolge `b` wieder zur ursprünglichen Zeichenkette dekodiert. Dazu traversiert sie den Baum wiederholt entsprechend der Folge `b`: Sobald ein Blatt erreicht wird, ergänzt sie das Zwischenergebnis mit dem zugehörigen Zeichen und beginnt die Dekodierung der restlichen Folge wieder an der Wurzel des Baums. Falls die Folge vorzeitig (vor Erreichen eines Blatts) endet, dann kann nicht entpackt werden und die Methode muss eine `IllegalArgumentException` werfen.

```
String decode(Node root, String b) {
    assert (root != null && b != null) : new IllegalArgumentException();
    return dec(root, root, b);
}
```

```
String dec(Node root, Node node, String b) {
    // ToDo: Code hier ergänzen
```

**Aufgabe 2:**"Haldensortierung"

Gegeben sei folgende Klasse:

```
class W {
    int t;
    String f;
    // ...
```

Dazu gibt es verschiedene **Comparatoren**, zum Beispiel:

```
// ascending order for field W.t
class ComparatorAscByFieldT implements Comparator<W> {
    // Returns a negative integer, zero, or a positive integer as the
    // first argument is less than, equal to, or greater than the second.
    @Override
    public int compare(W o1, W o2) { // ...
```

Außerdem steht Ihnen die vorgegebene Methode **swap** zur Verfügung:

```
void swap(W[] w, int a, int b) { // ...
```

- a) Phase 1: Die Haldensortierung beginnt mit der Herstellung der Max-Heap-Eigenschaft von rechts nach links. Diese ist für alle Feldelemente im dunklen Bereich bereits erfüllt. Geben Sie die Positionen (IDs) derjenigen Elemente des Feldes an, die das Verfahren im „Versickerschritt“ für das nächste Element mit Hilfe des **ComparatorAscByFieldT** miteinander vergleicht:

IDs angeben >	0	1	2	3	4	5	6	< IDs angeben
	6	7	0	3	1	5	2	

Nach dem Vergleichen werden gegebenenfalls Werte mit **swap** vertauscht. Geben Sie das Resultat (in obiger Array-Darstellung) nach diesem Schritt an.

- b) Phase 2: Das folgende Feld enthält den bereits vollständig aufgebauten Max-Heap:

0	1	2	3	4	5	6
7	6	5	3	1	0	2

Die Haldensortierung verschiebt das maximale Element in den sortierten (dunklen) Bereich:

0	1	2	3	4	5	6
2	6	5	3	1	0	7

Geben Sie das Ergebnis des nachfolgenden „Versickerns“ (erneut in derselben Array-Darstellung) an, bei dem die Heap-Eigenschaft wiederhergestellt wird.

- c) Ergänzen Sie die rekursive Methode **reheap**, die die Max-Heap-Eigenschaft im Feld **w** zwischen den Indizes **i** und **k** (jeweils einschließlich) in  $O(\log(k-i))$  gemäß **Comparator<W>** **c** wiederherstellt, indem sie das Element **w[i]** „versickt“. **k** bezeichnet das Ende des unsortierten Bereichs.

```
// restores the max-heap property in w[i to k] using c
void reheap(W[] w, Comparator<W> c, int i, int k) {
    int leftId = 2 * i + 1;
    int rightId = leftId + 1;
    int kidId;
} // ToDo: Code hier ergaenzen
```

- d) Implementieren Sie nun die eigentliche Haldensortierung. Sie dürfen hier die Methode `reheap` verwenden.

```
// sorts w in-situ according to the order imposed by c
void heapSort(W[] w, Comparator<W> c) {
    int n = w.length;

    // Phase 1: Max-Heap-Eigenschaft herstellen
    //          (siehe Teilaufgabe a)
    // ToDo: Code hier ergaenzen

    // Phase 2: jeweils Maximum entnehmen und sortierte Liste am Ende aufbauen
    //          (siehe Teilaufgabe b)
    // ToDo: Code hier ergaenzen
}
```

**Aufgabe 3:**“Laufzeitanalyse mittels Landau-0-Kalkül“

Gegeben seien die folgenden Methoden, in denen bestimmte Stellen mit Kommentaren der Form `/** x */` markiert sind. Geben Sie zunächst für jede solche Stelle einer Methode eine geschlossene Formel an, die exakt berechnet, wie oft die jeweiligen Stellen, in Abhängigkeit von den Eingangsgrößen  $n$  und  $m$ , durchlaufen werden.

Ermitteln Sie anschließend für diese Quellcode-Fragmente jeweils die kleinste obere Schranke für den Laufzeitaufwand im 0-Kalkül (Landau-Notation) abhängig von den Parametern. Skizzieren Sie in wenigen Sätzen, wie Sie die Aufwandklasse abgeschätzt haben – eine Beweisführung ist *nicht* verlangt.

a) `matrixVectorMul`

```
static int[] matrixVectorMul(int[][] mat, int[] vec) {
    int hoehe = mat.length; /* -> m */
    if (hoehe <= 0)
        return new int[0];
    int breite = mat[0].length; /* -> n */
    int[] erg = new int[hoehe];
    for (int zeile = 0; zeile < hoehe; ++zeile) {
        /** 1 */
        for (int spalte = 0; spalte < breite /** 2 **/; ++spalte) {
            erg[zeile] += mat[zeile][spalte] * vec[spalte];
            /** 3 */
        }
    }
    return erg;
}
```

b) `zaehlen`

```
static int zaehlen(int n) {
    int count = 0;
    while (n > 0) {
        if (n % 2 == 1) {
            ++count;
        }
        n /= 2;
        /** 1 */
    }
    return count;
}
```

c) `f`:

```
static int f(int n) {
    if (n <= 0) {
        /** 1 */
        return 0;
    } else {
        for (int i = 0; i < n; ++i) {
            /* Operationen mit konstantem Aufwand */
        }
        return f(n / 2);
    }
}
```

**Aufgabe 4:**

Zeigen oder widerlegen Sie die folgenden Aussagen (die jeweiligen Beweise sind sehr kurz):

- Sei  $L \subseteq \Sigma^*$ . Ist  $L$  regulär, so ist jede Teilmenge  $U$  von  $L$  auch regulär.
- Sei  $L$  eine Sprache über dem Alphabet  $\Sigma$ , die rekursiv aufzählbar (= partiell-entscheidbar), aber nicht entscheidbar ist. Sei  $\overline{L} = \Sigma^* \setminus L$  das Komplement von  $L$ . Dann ist  $\overline{L}$  rekursiv aufzählbar.
- Seien  $L_1$  und  $L_2$  beliebige kontextfreie Sprachen über dem Alphabet  $\Sigma$ . Dann ist  $L_1 \cap L_2$  entscheidbar.
- Alle Probleme in NP sind entscheidbar.

Schreiben Sie zuerst zur Aussage „Stimmt“ oder „Stimmt nicht“ und dann Ihre Begründung.

**Aufgabe 5:**

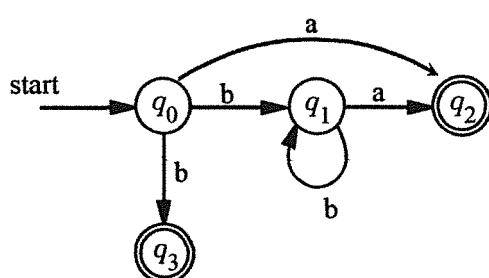
- Sei  $m \in \mathbb{N}_0 = \{0, 1, 2, \dots\}$ . Definieren Sie formal die Menge  $H_m$  der Gödelnummern der Turing-Maschinen, die gestartet mit  $m$  halten.
- Gegeben sei das folgende Problem  $E$ :
  - Entscheide, ob es für die deterministische Turing-Maschine  $M$  mit der Gödelnummer  $\langle M \rangle$  mindestens zwei Eingaben  $w_1, w_2 \in \mathbb{N}_0$ ,  $w_1 \neq w_2$ , gibt, so dass die Maschine  $M$  gestartet mit  $w_1$  hält und dass  $M$  gestartet mit  $w_2$  hält.

Zeigen Sie, dass  $E$  nicht entscheidbar ist. Benutzen Sie, dass  $H_m$  aus (a) für jedes  $m \in \mathbb{N}_0$  nicht entscheidbar ist.

- Zeigen Sie, dass das Problem  $E$  aus (b) partiell-entscheidbar (= rekursiv aufzählbar) ist.

**Aufgabe 6:**

- Gegeben sei der folgende nichtdeterministische endliche Automat  $N$ :



- Geben Sie einen regulären Ausdruck  $\alpha(N)$  für die Sprache, die der nichtdeterministische endliche Automat  $N$  aus (a) akzeptiert, an.

- c) Sei  $L = \{a^k b^k \mid k \in \mathbb{N}\}$ . Jemand behauptet, einen deterministischen endlichen Automaten mit Zustandsmenge  $Q = \{q_0, \dots, q_{n-1}\}$ , Startzustand  $q_0$  und Endzustandsmenge  $F$  konstruiert zu haben mit  $L = L(A)$ .

Geben Sie in Abhängigkeit von  $A$  ein Wort  $z \in L$  an, das folgende Eigenschaft besitzt: Aus einer akzeptierenden Rechnung von  $A$  für  $z$  können Sie ein Wort  $\hat{z}$  konstruieren mit der Eigenschaft: (i)  $A$  akzeptiert  $\hat{z}$  und (ii)  $\hat{z} \notin L$ .

Beweisen Sie konkret die Eigenschaften (i) und (ii) für Ihr Wort  $z$ .

Nachtrag zur Einzelprüfungnr. 66115, Thema 2, Seite 9, Aufgabe 6, Buchstabe a)

Bitte fügen Sie nach dem abgebildeten nichtdeterministischen endlichen Automaten folgende Fragestellung ein:

„Konstruieren Sie zu  $N$  mit der Potenzmengen-Konstruktion einen äquivalenten deterministischen endlichen Automaten  $A$ . Zeichnen Sie die nur vom Startzustand erreichbaren Zustände ein, diese aber *alle*. Die Zustandsnamen von  $A$  müssen erkennen lassen, wie sie zustande gekommen sind. Führen Sie *keine „Vereinfachungen“* durch!“

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2016**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	<b>Frühjahr 2016</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

---

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**  
**— Prüfungsaufgaben —**

---

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

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

Anzahl der Druckseiten dieser Vorlage: **9**

---

Bitte wenden!

Thema Nr. 1  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

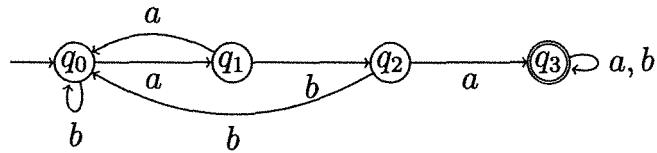
### 1. Reguläre Sprachen

- a) Geben Sie einen möglichst einfachen regulären Ausdruck für die Sprache  $L_1 = \{a_1a_2 \cdots a_n \mid n \geq 3, a_i \in \{a, b\} \text{ für alle } i = 1, \dots, n \text{ und } a_1 \neq a_n\}$  an.
- b) Geben Sie einen möglichst einfachen regulären Ausdruck für die Sprache

$$L_2 = \{w \in \{a, b\}^* \mid w \text{ enthält genau ein } b \text{ und ist von ungerader Länge}\}$$

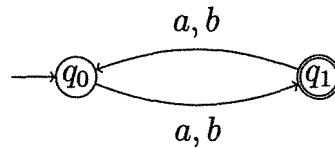
an.

- c) Beschreiben Sie die Sprache des folgenden Automaten  $A_1$



möglichst einfach und präzise in ihren eigenen Worten.

- d) Betrachten Sie folgenden Automaten  $A_2$ :



Konstruieren Sie einen endlichen Automaten, der die Schnittmenge der Sprachen  $L(A_1)$  und  $L(A_2)$  akzeptiert.

## 2. Kontextfreie Sprachen

Betrachten Sie die folgende Grammatik:  $G = (V, \Sigma, S, P)$  mit

- $V = \{S, A\}$ ,
- $\Sigma = \{0, 1, 2\}$ ,
- $P$ :

$$S \rightarrow 0S0 \mid 1S1 \mid 2A2 \mid 0 \mid 1 \mid \varepsilon$$

$$A \rightarrow A2$$

Konstruieren Sie für die Grammatik  $G$  schrittweise eine äquivalente Grammatik in Chomsky-Normalform. Geben Sie für jeden einzelnen Schritt des Verfahrens das vollständige Zwischenergebnis an und erklären Sie kurz was in dem Schritt getan wurde.

## 3. Klassifizierung

Im Folgenden bezeichne  $\#_\sigma(w)$  die Anzahl der Vorkommen von  $\sigma$  in  $w$ . Sei  $\mathbb{N}^+ = \{1, 2, 3, \dots\}$ . Welche der folgenden Sprachen über dem Alphabet  $\{a, b\}$  sind

- a) regulär?
  - b) kontextfrei, aber nicht regulär?
  - c) nicht kontextfrei?
  - d) in  $P$ ?
- 1)  $L_1 = \{w_1 w_2 \in \{a, b\}^* \mid w_1 \neq w_2, |w_1| = |w_2|\}$
  - 2)  $L_2 = \{a^n b^n b^n a^n \mid n \in \mathbb{N}\}$
  - 3)  $L_3 = \{(ab)^i a (ba)^j \mid i, j \geq 0 \text{ und } i \leq j\}$

Geben Sie zu Ihrer Lösung jeweils einen Beweis an. 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. Um zu zeigen, dass eine Sprache in  $P$  liegt, genügt die Angabe eines Algorithmus in Pseudocode.

#### 4. Turingmaschinen

- a) Geben Sie eine deterministische 2-Band Turingmaschine  $M$  an, die die Funktion

$$f_M(a^n) = a^n b^n$$

berechnet. Die Maschine  $M$  nimmt somit immer einen String der Form  $a^n$  (ein String, der aus  $n$   $a$ 's für beliebiges  $n \in \mathbb{N}$  besteht) als Eingabe und produziert anschließend auf Band 2 als Ausgabe den String  $a^n b^n$  (ein String aus  $n$   $a$ 's gefolgt von  $n$   $b$ 's).

Beschreiben Sie außerdem die Idee hinter ihrer Konstruktion.

- b) Geben Sie die Konfigurationsfolge der Turingmaschine aus (a) für die Eingabe  $aa$  an.

#### 5. Komplexität

Das Problem  $k$ -COL ist wie folgt definiert:

**Gegeben:** Ein ungerichteter Graph  $G = (V, E)$ .

**Frage:** Kann man jedem Knoten  $v$  in  $V$  eine Zahl  $z(v) \in \{1, \dots, k\}$  zuordnen, so dass für alle Kanten  $(u_1, u_2) \in E$  gilt:  $z(u_1) \neq z(u_2)$ ?

Zeigen Sie, dass man 3-COL in polynomieller Zeit auf 4-COL reduzieren kann. Beschreiben Sie dazu die Reduktion und zeigen Sie anschließend ihre Korrektheit.

#### 6. Sortieren

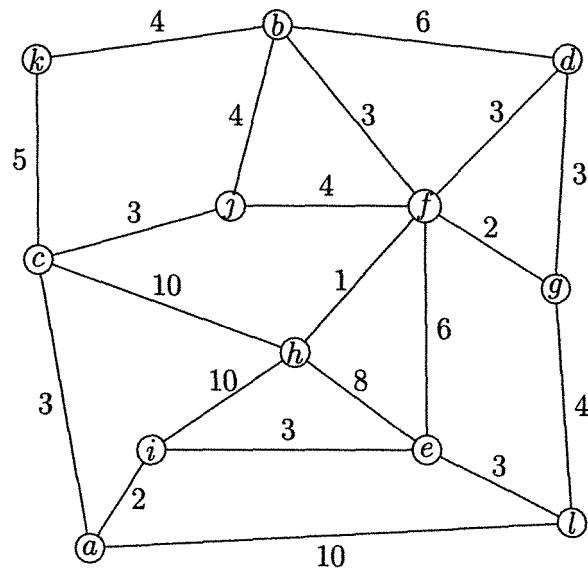
Sortieren Sie die Werte

1   45   8   53   9   2   17   10

mit Quicksort.

**7. Kürzeste Pfade**

Berechnen Sie mit Hilfe des Algorithmus von Dijkstra den kürzesten Pfad von Knoten  $a$  nach  $h$ .



Thema Nr. 2  
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

### **1. Verständnis formale Sprachen**

Beantworten Sie kurz, präzise und mit Begründung folgende Fragen: (Die Begründungen müssen keine formellen mathematischen Beweise sein).

- a) Welche Möglichkeiten gibt es, eine formale Sprache vom Typ 3 zu definieren?
- b) Was ist die Komplexität des Wortproblems für Typ-3 Sprachen und wieso ist das so?
- c) Sind Syntaxbäume zu einer Grammatik immer eindeutig? Falls nicht, geben Sie ein Gegenbeispiel.
- d) Wie kann man die Äquivalenz zweier Typ-3 Sprachen nachweisen?
- e) Wie kann man das Wortproblem für das Komplement einer Typ-3 Sprache lösen?
- f) Weshalb gilt das Pumping-Lemma für Typ 3 Sprachen?
- g) Ist der Nachweis, dass das Typ-3 Pumping-Lemma für eine gegebene Sprache gilt, ausreichend, um zu zeigen, dass die Sprache vom Typ 3 ist? Falls nicht, geben Sie ein Gegenbeispiel, mit Begründung.
- h) Geben Sie ein Beispiel, an dem deutlich wird, dass deterministische und nicht-deterministische Typ-2 Sprachen unterschiedlich sind.
- i) Worin macht sich der Unterschied zwischen Typ 0 und 1 bemerkbar, wenn man Turingmaschinen benutzt, um das Wortproblem vom Typ 0 oder 1 zu lösen. Warum ist das so?

### **2. Verständnis Berechenbarkeitstheorie**

Beantworten Sie kurz, präzise und mit Begründung folgende Fragen: (Die Begründungen müssen keine formellen mathematischen Beweise sein)

- a) Warum genügt es, sich auf Funktionen zu beschränken, die natürliche Zahlen auf natürliche Zahlen abbilden, wenn man untersuchen will, was heutige Computer im Prinzip berechnen können?
- b) Was besagt die Church-Turing These? Könnte man sie beweisen oder widerlegen?
- c) Für reelle Zahlen, wie z.B.  $\pi$ , lässt sich die Dezimaldarstellung durch entsprechende Programme beliebig genau approximieren. Gilt das für alle reellen Zahlen, d.h. lässt sich für jede reelle Zahl die Dezimaldarstellung mit entsprechenden Programmen beliebig genau approximieren?

- d) Was ist für die Berechnungskraft der wesentliche Unterschied zwischen While-Berechenbarkeit und Loop-Berechenbarkeit.
- e) Die Ackermannfunktion ist ein Beispiel einer totalen Funktion, die While-berechenbar, aber nicht Loop-berechenbar ist. Sie verallgemeinert die Idee, dass Multiplikation die wiederholte Addition ist, Exponentiation die wiederholte Multiplikation, Hyperexponentiation die wiederholte Exponentiation usw. Die Stufe dieser hyper-hyper ... Exponentiation ist ein Parameter der Ackermannfunktion. Generieren Sie aus dieser Idee ein Argument, das illustriert, warum die Ackermannfunktion nicht Loop-berechenbar ist.
- f) Geben Sie ein Beispiel einer Menge an, die abzählbar, aber nicht rekursiv aufzählbar ist, und begründen Sie es.
- g) Wie ist der Zusammenhang zwischen rekursiv aufzählbar und semi-entscheidbar?

### 3. Verständnis Komplexitätstheorie

Beantworten Sie kurz, präzise und mit Begründung folgende Fragen: (Die Begründungen müssen keine formellen mathematischen Beweise sein)

- a) In der O-Notation insbesondere für die Zeitkomplexität von Algorithmen lässt man i.A. konstante Faktoren oder kleinere Terme weg. Z.B. schreibt man anstelle  $O(3n_2 + 5)$  einfach nur  $O(n_2)$ . Warum macht man das so?
- b) Was ist die typische Vorgehensweise, wenn man für ein neues Problem die NP-Vollständigkeit untersuchen will?
- c) Was könnte man tun, um P=NP zu beweisen?
- d) Sind NP-vollständige Problem mit Loop-Programmen lösbar? (Antwort mit Begründung!)
- e) Wie zeigt man aus der NP-Härte des SAT-Problems die NP-Härte des 3SAT-Problems? (3SAT ist ein SAT-Problem wobei alle Klauseln maximal 3 Literale haben.)

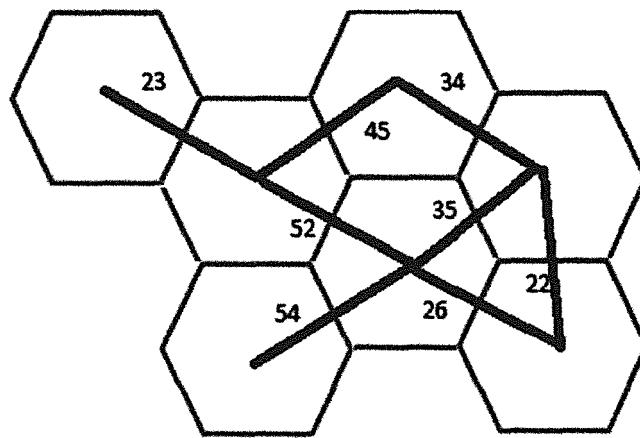
### 4. Hashing

Betrachte eine Hashtabelle der Größe  $m = 10$ .

- a) Welche der folgenden Hashfunktionen ist für Hashing mit verketteten Listen am besten geeignet? Begründen Sie Ihre Wahl.
1.  $h_1(x) = (4x+3) \bmod m$
  2.  $h_2(x) = (3x+3) \bmod m$ .
- b) Welche der folgenden Hashfunktionen ist für Hashing mit *offener Adressierung* am besten geeignet. Begründen Sie Ihre Wahl.
1.  $h_1(x, i) = (7x+i \bmod m) \bmod m$
  2.  $h_2(x, i) = (7x+i \bmod (m-1)) \bmod m$ .

## 5. Minimaler Spannbaum

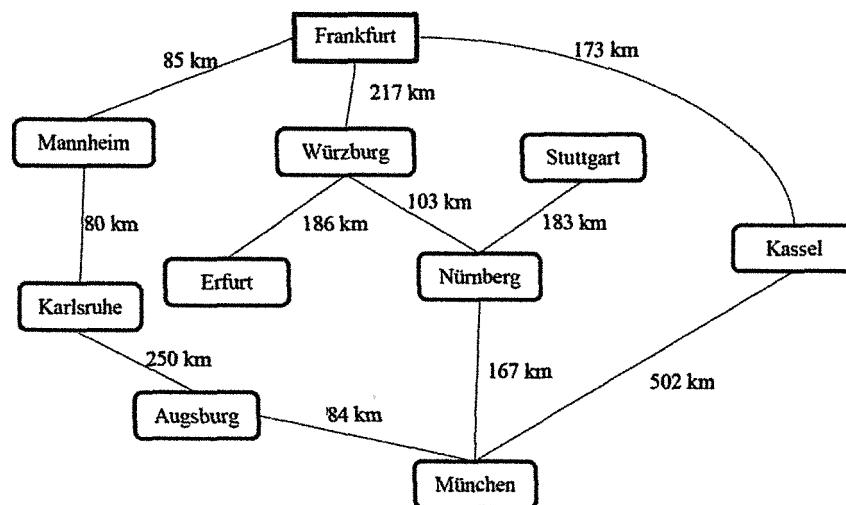
Eine Schule möchte alle seine Räume mit Internetanschlüssen versehen, hat aber wenig Geld. Sie haben sich bereit erklärt, eine Verkabelung mit möglichst wenig Kabeln zu planen. Dabei können Sie vorhandene Kabelkanäle nutzen. Der schematische Raumplan mit den ebenfalls schematisch eingezeichneten Kabelkanälen sieht folgendermaßen aus (Maße in Dezimetern).



- Beschreiben Sie das Verfahren, nach dem Sie die optimale Verkabelung wählen.
- Welche Komplexität hat das Verfahren? Woher röhrt die Komplexität?
- Zeichnen Sie den Graphen mit der minimalen Verkabelung.
- Wieviel Meter Kabel benötigen Sie?

## 6. Dijkstra Algorithmus

- a) Berechnen Sie für folgenden Graphen den kürzesten Weg von Karlsruhe nach Kassel und dokumentieren Sie den Berechnungsweg:



- b) Könnte man den Dijkstra Algorithmus auch benutzen, um das Travelling-Salesman Problem zu lösen?

## 7. Verständnis Suchbäume

Wofür eignen sich die folgenden Baum-Datenstrukturen im Vergleich zu den anderen angeführten Baumstrukturen am besten, und warum. Sprechen Sie auch die Komplexität der wesentlichen Operationen und die Art der Speicherung an.

- a) Rot-Schwarz Baum
- b) AVL-Baum
- c) Binärer-Heap
- d) B-Baum
- e) R-Baum

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2016**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
--------------------	----------------	----------------------

Kennzahl: \_\_\_\_\_

**Herbst  
2016**

\_\_\_\_\_

**66115**

Kennwort: \_\_\_\_\_

Arbeitsplatz-Nr.: \_\_\_\_\_

---

**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

$\delta$	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).

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

**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  $i = 1, \dots, k$  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  $A_1, \dots, A_k$  und eine endliche Menge  $P$  von Paaren gegeben.

Gefragt ist, ob eine Auswahl von Elementen  $y_1 \in A_1, \dots, y_n \in A_k$  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  $\{1, \dots, n\}$  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:

- \*  $0 \leq i \leq h < j < n$ , 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

$$\text{ZI}([2, 3, 8, 6, 1]) = ([1, 2, 3, 6, 8], 5)$$

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!

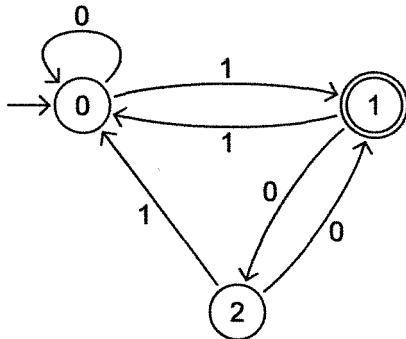
**Aufgabe 1:**

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.

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

- c) Sei  $\mathbb{N}^+ = \{1, 2, 3, \dots\}$  die Menge der positiven natürlichen Zahlen und sei  $g : \mathbb{N}^+ \rightarrow \mathbb{N}^+$  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  $M_0, M_1, \dots$  eine Gödelisierung aller Registermaschinen (RAMs). Beantworten Sie folgende Fragen zur Aufzählbarkeit und Entscheidbarkeit. Beweisen Sie Ihre Antworten.

- a) Ist folgende Menge entscheidbar?

$$A = \{x \in \mathbb{N} \mid x \geq 100 \text{ oder } M_x \text{ hält bei Eingabe } x\}$$

- b) Ist folgende Menge entscheidbar?

$$B = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid M_x \text{ hält bei Eingabe } x \text{ genau dann, wenn } M_y \text{ bei Eingabe } y \text{ hält}\}$$

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.

- 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  $\ell < r$  then
   $m = \text{Partition}(A, \ell, r)$ 
  QuickSort(A,  $\ell, m - 1$ )
  QuickSort(A,  $m + 1, r$ )
```

**Algorithm:** `Partition(A,  $\ell, r$ )`

```
 $pivot = A[r]$ 
 $i = \ell$ 
for  $j = \ell$  to  $r - 1$  do
  if  $A[j] \leq pivot$  then
     $\text{Swap}(A, i, j)$ 
     $i = i + 1$ 
 $\text{Swap}(A, i, r)$ 
return  $i$ 
```

**Algorithm:** `Swap(A,  $i, j$ )`

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

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2017**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
--------------------	----------------	----------------------

Kennzahl: \_\_\_\_\_

Kennwort: \_\_\_\_\_

Arbeitsplatz-Nr.: \_\_\_\_\_

**Frühjahr  
2017**

**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: **14**

---

**Bitte wenden!**

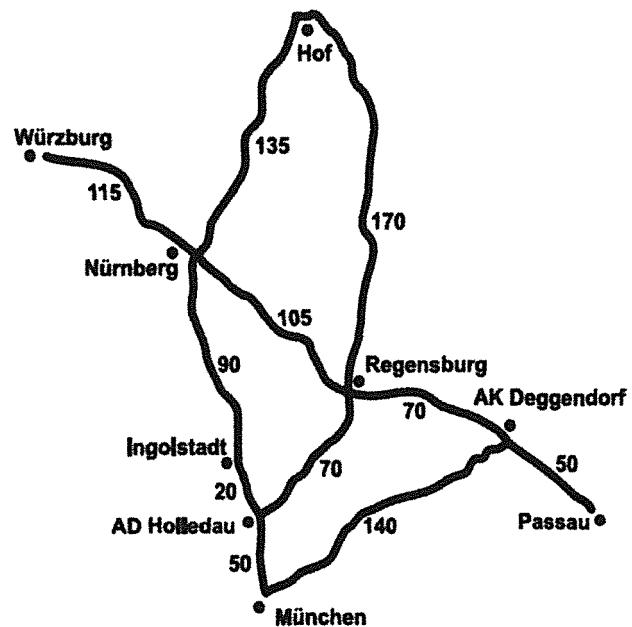
**Thema Nr. 1**  
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1 (Graphalgorithmen)**

Die folgende Abbildung zeigt die wichtigsten bayerischen Autobahnen zusammen mit einigen anliegenden Orten und die Entfernungen zwischen diesen.

Entfernungstabelle		km
von	nach	
Würzburg	Nürnberg	115
Nürnberg	Regensburg	105
Regensburg	AK Deggendorf	70
AK Deggendorf	Passau	50
Hof	Nürnberg	135
Nürnberg	Ingolstadt	90
Ingolstadt	AD Holledau	20
AD Holledau	München	50
München	AK Deggendorf	140
Hof	Regensburg	170
Regensburg	AD Holledau	70



- a) Bestimmen Sie mit dem Algorithmus von *Dijkstra* den kürzesten Weg von Ingolstadt zu allen anderen Orten. Verwenden Sie zur Lösung eine Tabelle gemäß folgendem Muster und markieren Sie in jeder Zeile den jeweils als nächstes zu betrachtenden Ort. Setzen Sie für die noch zu bearbeitenden Orte eine Prioritätswarteschlange ein, d.h. bei gleicher Entfernung wird der ältere Knoten gewählt.

Ingolstadt	Hof	Würzburg	Nürnberg	Regensburg	AK Deggendorf	AD Holledau	Passau	München
0	8	8	8	8	8	8	8	8
⋮								
<i>Ergebnis:</i>								

Fortsetzung nächste Seite!

- b) Die bayerische Landesregierung hat beschlossen, die eben betrachteten Orte mit einem breitbandigen Glasfaser-Backbone entlang der Autobahnen zu verbinden. Dabei soll aus Kostengründen so wenig Glasfaser wie möglich verlegt werden. Identifizieren Sie mit dem Algorithmus von Kruskal diejenigen Strecken, entlang welcher Glasfaser verlegt werden muss. Geben Sie die Ortspaare (Autobahnsegmente) in der Reihenfolge an, in der Sie sie in Ihre Verkabelungsliste aufnehmen.
- c) Um Touristen den Besuch aller Orte so zu ermöglichen, dass sie dabei jeden Autobahnabschnitt genau einmal befahren müssen, bedarf es zumindest eines sogenannten offenen Eulerzugs. Zwischen welchen zwei Orten würden Sie eine Autobahn bauen, damit das bayerische Autobahnnetz mindestens einen Euler-Pfad enthält?

**Fortsetzung nächste Seite!**

### Aufgabe 2 (Sortierverfahren)

In dieser Aufgabe sei vereinfachend angenommen, dass sich Top-Level-Domains (TLD) ausschließlich aus zwei oder drei der 26 Kleinbuchstaben des deutschen Alphabets ohne Umlaute zusammensetzen. Im Folgenden sollen TLDs lexikographisch aufsteigend sortiert werden, d.h. eine TLD ( $s_1, s_2$ ) mit zwei Buchstaben (z. B. „co“ für Kolumbien) wird also vor einer TLD ( $t_1, t_2, t_3$ ) der Länge drei (z. B. „com“) einsortiert, wenn  $s_1 < t_1 \vee (s_1 = t_1 \wedge s_2 \leq t_2)$  gilt.

- a) Sortieren Sie zunächst die Reihung [„de“, „com“, „uk“, „org“, „co“, „net“, „fr“, „ee“] schrittweise unter Verwendung des *Radix-Sortierverfahrens* (Bucketsort). Erstellen Sie dazu eine Tabelle wie das folgende Muster und tragen Sie dabei in das Feld „Stelle“ die Position des Buchstabens ein, nach dem im jeweiligen Durchgang sortiert wird (das Zeichen am TLD-Anfang habe dabei die „Stelle“ 1).

Stelle	Reihung								
–	de	com	uk	org	co	net	fr	ee	
...									

- b) Sortieren Sie nun die gleiche Reihung wieder schrittweise, diesmal jedoch unter Verwendung des *Mergesort*-Verfahrens (Sortieren durch Mischen). Erstellen Sie dazu eine Tabelle wie das folgende Muster und vermerken Sie in der ersten Spalte jeweils welche Operation durchgeführt wurde: Wenn Sie die Reihung geteilt haben, schreiben Sie in die linke Spalte ein T und markieren Sie die Stelle, an der Sie die Reihung geteilt haben, mit einem senkrechten Strich „|“. Wenn Sie zwei Teilreihungen durch Mischen zusammengeführt haben, schreiben Sie ein M in die linke Spalte und unterstreichen Sie die zusammengemischten Einträge. Beginnen Sie mit dem rekursiven Abstieg immer in der linken Hälfte einer (Teil-)Reihung.

Op.	Reihung
T	de, com, uk, org   co, net, fr, ee
...	

- c) Implementieren Sie das Sortierverfahren *Quicksort* für String-TLDs in einer gängigen Programmiersprache Ihrer Wahl. Ihr Programm (Ihre Methode) wird mit drei Parametern gestartet: dem String-Array mit den zu sortierenden TLDs selbst sowie jeweils der Position des ersten und des letzten zu sortierenden Eintrags im Array.

**Aufgabe 3 (Rekursion und Dynamische Programmierung)**

Gegeben seien die folgenden Formeln zur Berechnung der *ersten* Fibonacci-Zahlen:

$$fib_n = \begin{cases} 1 & \text{falls } n \leq 2 \\ fib_{n-1} + fib_{n-2} & \text{sonst} \end{cases}$$

sowie der Partialsumme der Fibonacci-Quadrat:

$$sos_n = \begin{cases} fib_n & \text{falls } n = 1 \\ fib_n^2 + sos_{n-1} & \text{sonst} \end{cases}$$

Sie dürfen im Folgenden annehmen, dass die Methoden nur mit  $1 \leq n \leq 46$  aufgerufen werden, so dass der Datentyp `long` zur Darstellung aller Werte ausreicht.

- a) Implementieren Sie die obigen Formeln zunächst rekursiv (ohne Schleifenkonstrukte wie `for` oder `while`) und ohne weitere Optimierungen („naiv“) in Java als:

`long fibNaive(int n) {`

bzw.

`long sosNaive(int n) {`

- b) Offensichtlich ist die naive Umsetzung extrem ineffizient, da viele Zwischenergebnisse wiederholt rekursiv ausgewertet werden müssen. Die Dynamische Programmierung (DP) erlaubt es Ihnen, die Laufzeit auf Kosten des Speicherbedarfs zu reduzieren, indem Sie alle einmal berechneten Zwischenergebnisse speichern und bei erneutem Bedarf „direkt abrufen“. Implementieren Sie obige Formeln nun rekursiv aber mittels DP in Java als:

`long fibDP(int n) {`

bzw.

`long sosDP(int n) {`

- c) Am “einfachsten“ und bzgl. Laufzeit [in  $\mathcal{O}(n)$ ] sowie Speicherbedarf [in  $\mathcal{O}(1)$ ] am effizientesten ist sicherlich eine iterative Implementierung der beiden Formeln. Geben Sie eine solche in Java an als:

`long fIter(int n) {`

bzw.

`long sosIter(int n) {`

**Fortsetzung nächste Seite!**

**Aufgabe 4 (Formale Verifikation)**

Sie dürfen im Folgenden davon ausgehen, dass keinerlei Under- oder Overflows auftreten (der Datentyp `long` also einen beliebig großen Wertebereich hat).

Gegeben sei folgendes rekursives Programmfragment in der Sprache Java für  $n \geq 0$ :

```
long sumOfSquares(long n) {
    if (n == 0)
        return 0;
    else
        return n * n + sumOfSquares(n - 1);
}
```

- a) Beweisen Sie *formal* mittels *vollständiger Induktion*:

$$\forall n \in \mathbb{N}_0 : \text{sumOfSquares}(n) = \frac{n(n+1)(2n+1)}{6}$$

- b) Beweisen Sie die Terminierung von `sumOfSquares(n)` für alle  $n \geq 0$ .

**Aufgabe 5 (Aussagen)**

Zeigen oder widerlegen Sie die folgenden **Aussagen** (die jeweiligen Beweise sind sehr kurz):

- a) Alle regulären Sprachen liegen in NP.
- b) Es gibt Sprachen A, B mit  $A \subseteq B$ , sodass B regulär und A kontextfrei, aber nicht regulär ist.
- c) Es gibt unentscheidbare Sprachen  $L$  über dem Alphabet  $\Sigma$ , so dass sowohl  $L$  als auch das Komplement  $\bar{L} = \Sigma^* \setminus L$  rekursiv aufzählbar (= partiell-entscheidbar) sind.
- d) Sei  $L$  eine beliebige kontextfreie Sprache über dem Alphabet  $\Sigma$ . Dann ist das Komplement  $\bar{L} = \Sigma^* \setminus L$  entscheidbar.

Schreiben Sie zuerst zur Aussage „Stimmt“ oder „Stimmt nicht“ und dann Ihre Begründung.

**Fortsetzung nächste Seite!**

**Aufgabe 6 (Turing)**

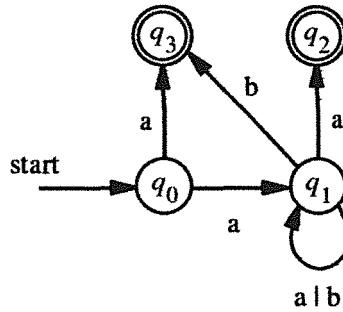
Es sei  $E$  die Menge aller (geeignet codierten) Turingmaschinen  $M$  mit folgender Eigenschaft: Es gibt eine Eingabe  $w$ , so dass  $M$  gestartet auf  $w$  mindestens 1000 Schritte rechnet und dann irgendwann hält.

Das Halteproblem auf leerer Eingabe  $H_0$  ist definiert als die Menge aller Turingmaschinen, die auf leerer Eingabe gestartet, irgendwann halten.

- Zeigen Sie, dass  $E$  unentscheidbar ist (etwa durch Reduktion vom Halteproblem  $H_0$ ).
- Begründen Sie, dass  $E$  partiell entscheidbar ist.
- Geben Sie ein Problem an, welches nicht einmal partiell entscheidbar ist.

**Aufgabe 7 (Automaten)**

- Gegeben sei der folgende nichtdeterministische endliche Automat  $N$ :



Konstruieren Sie zu  $N$  mit Hilfe der Potenzmengen-Konstruktion einen äquivalenten deterministischen endlichen Automaten  $A$ . Zeichnen Sie nur die vom Startzustand erreichbaren Zustände ein, die aber alle. Die Zustandsnamen von  $A$  müssen erkennen lassen, wie sie zustande gekommen sind. Führen Sie keine „Vereinfachungen“ durch!

*Hinweis:* In einem deterministischen endlichen Automaten muss es an jedem Zustand für jedes Zeichen einen Übergang geben.

- Geben Sie einen regulären Ausdruck  $\alpha(N)$  für die Sprache, die der nichtdeterministische endliche Automat  $N$  aus (a) akzeptiert, an.
- Sei  $L = \{a^k b^\ell \mid k, \ell \in \mathbb{N}, k > \ell\}$ . Jemand behauptet, einen deterministischen endlichen Automaten  $A$  mit Zustandsmenge  $Q = \{q_0, \dots, q_{n-1}\}$ , Startzustand  $q_0$  und Endzustandsmenge  $F$  konstruiert zu haben mit  $L = L(A)$ . Geben Sie in Abhängigkeit von  $A$  ein Wort  $z \in L$  an, das folgende Eigenschaft besitzt: Aus einer akzeptierenden Rechnung von  $A$  für  $z$  können Sie ein Wort  $\hat{z}$  konstruieren mit der Eigenschaft:  
 (i)  $A$  akzeptiert  $\hat{z}$  und (ii)  $\hat{z} \notin L$ .  
 Beweisen Sie konkret die Eigenschaften (i) und (ii) für Ihr Wort  $z$ .

**Aufgabe 8 (Halteproblem und  $P = NP$ )**

Sei  $L$  die durch den regulären Ausdruck  $(10)^* \cup ((101 \cup 11)^* \cup 111)^*$  beschriebene Sprache.  
[Alternative Schreibweise:  $(10)^* + ((101 + 11)^* + 111)^*$ ]

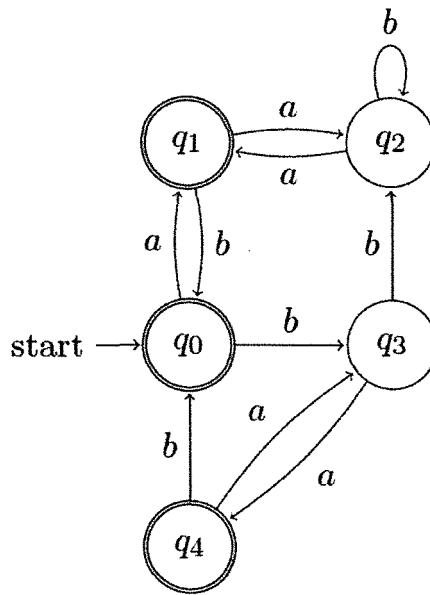
- a) Sei  $H_m$  das Halteproblem bei fester Eingabe  $m$ .  
Zeigen Sie:  $L$  kann auf  $H_m$  reduziert werden. (Als Relation:  $L \leq H_m$ )
- b) Angenommen, es wurde gezeigt, dass  $P = NP$  ist. Zeigen Sie, dass  $L$  dann NP-vollständig ist.

**Thema Nr. 2**  
**(Aufgabengruppe)**

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1: Reguläre Sprachen**

1. Sei  $\Sigma = \{a, b\}$  und  $L_1$  die Sprache aller Wörter über  $\Sigma$ , in denen das Wort  $ab$  nicht vorkommt. Sei weiterhin  $L_2$  die Sprache aller Wörter über  $\Sigma$ , in denen  $b$  genau zwei mal vorkommt.
  - a) Geben Sie einen (möglicherweise nichtdeterministischen) endlichen Automaten an, der  $L_1$  akzeptiert.
  - b) Geben Sie einen (möglicherweise nichtdeterministischen) endlichen Automaten an, der  $L_2$  akzeptiert.
  - c) Geben Sie einen regulären Ausdruck für  $L_1 \cap L_2$  an.
2. Gegeben sei der unten aufgeführte deterministische endliche Automat  $A$ . Geben Sie einen minimalen deterministischen Automaten für die von  $A$  akzeptierte Sprache an.



**Fortsetzung nächste Seite!**

**Aufgabe 2: Kontextfreie Sprachen**

1. Gegeben sei die kontextfreie Grammatik  $G = (V, \Sigma, P, S)$  mit Sprache  $L(G)$ , wobei  $V = \{S, T, U\}$  und  $\Sigma = \{a, b, c, d, e\}$ .  $P$  bestehe aus den folgenden Produktionen:

$$S \rightarrow U \mid SbU \quad T \rightarrow dSe \mid a \quad U \rightarrow T \mid UcT$$

- a) Zeigen Sie  $acdae \in L(G)$ .
  - b) Bringen Sie  $G$  in Chomsky-Normalform.
2. Geben Sie eine kontextfreie Grammatik für  $L = \{a^i b^k c^i \mid i, k \in \mathbb{N}\}$  an.
3. Zeigen Sie, dass  $L = \{a^i b^k c^i \mid i, k \in \mathbb{N} \wedge i < k\}$  nicht kontextfrei ist, indem Sie das Pumping-Lemma für kontextfreie Sprachen anwenden.

**Aufgabe 3: Berechen- und Entscheidbarkeit**

1. Primitiv rekursive Funktionen

- a) Zeigen Sie, dass die folgendermaßen definierte Funktion  $\text{if} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  primitiv rekursiv ist.

$$\text{if } (b, x, y) = \begin{cases} x & \text{falls } b=0 \\ y & \text{sonst} \end{cases}$$

- b) Wir nehmen eine primitiv rekursive Funktion  $p : \mathbb{N} \rightarrow \mathbb{N}$  an und definieren  $g(n)$  als die Funktion, welche die größte Zahl  $i \leq n$  zurückliefert, für die  $p(i) = 0$  gilt. Falls kein solches  $i$  existiert, soll  $g(n) = 0$  gelten:

$$g(n) = \max (\{i \leq n \mid p(i) = 0\} \cup \{0\})$$

Zeigen Sie, dass  $g : \mathbb{N} \rightarrow \mathbb{N}$  primitiv rekursiv ist. (Sie dürfen obige Funktion  $\text{if}$  als primitiv rekursiv voraussetzen.)

2. Sei  $\Sigma = \{a, b, c\}$  und  $L \subseteq \Sigma^*$  mit  $L = \{a^i b^i c^i \mid i \in \mathbb{N}\}$ .

- a) Beschreiben Sie eine Turingmaschine, welche die Sprache  $L$  entscheidet. Eine textuelle Beschreibung der Konstruktionsidee ist ausreichend.
- b) Geben Sie Zeit- und Speicherkomplexität (abhängig von der Länge der Eingabe) Ihrer Turingmaschine an.

3. Sei  $\Sigma = \{0, 1\}$ . Jedes  $w \in \Sigma^*$  kodiert eine Turingmaschine  $M_w$ . Die von  $M_w$  berechnete Funktion bezeichnen wir mit  $\varphi_w$ .

- a) Warum ist  $\{w \in \Sigma^* \mid \exists x : \varphi_w(x) = xx\}$  nicht entscheidbar?
- b) Warum ist  $\{w \in \Sigma^* \mid \exists x : w = xx\}$  entscheidbar?

**Aufgabe 4: Komplexitätstheorie**

Eine Menge  $U \subseteq V$  heißt *Knotenüberdeckung* eines ungerichteten Graphen  $G = (V, E)$ , wenn jede Kante des Graphen mit mindestens einem Knoten aus  $U$  verbunden ist:

$$\forall (u, v) \in E : u \in U \vee v \in U$$

Das Problem **KNOTENÜBERDECKUNG** fragt, ob zu einem gegebenen ungerichteten Graphen und einer natürlichen Zahl  $k$  eine aus  $k$  Knoten bestehende Knotenüberdeckung existiert. Für  $G = (V, E)$  und  $k \in \mathbb{N}$  gilt also:

$$\begin{aligned} (G, k) \in \text{KNOTENÜBERDECKUNG} \\ \Leftrightarrow \\ \exists U \subseteq V : U \text{ ist Knotenüberdeckung von } G \text{ und } |U| = k \end{aligned}$$

1. Begründen Sie, dass **KNOTENÜBERDECKUNG** in NP liegt.

Eine Menge  $C \subseteq V$  heißt *Clique* eines ungerichteten Graphen  $G = (V, E)$ , wenn alle Paare verschiedener Knoten der Clique durch eine Kante des Graphen verbunden sind:

$$\forall u \in C \quad \forall v \in C : u \neq v \Rightarrow (u, v) \in E$$

Das Problem **CLIQUE** fragt, ob zu einem gegebenen ungerichteten Graphen und einer natürlichen Zahl  $k$  eine aus  $k$  Knoten bestehende Clique existiert. Für  $G = (V, E)$  und  $k \in \mathbb{N}$  gilt also:

$$(G, k) \in \text{CLIQUE} \Leftrightarrow \exists C \subseteq V : C \text{ ist Clique und } |C| = k$$

Wir definieren  $\bar{E} := \{(u, v) \in V \times V \mid (u, v) \notin E\}$ .

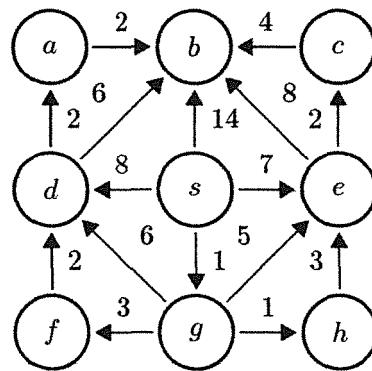
2. Zeigen Sie:  $C$  ist Clique von  $G = (V, E)$ , genau dann wenn  $V \setminus C$  Knotenüberdeckung von  $(V, \bar{E})$  ist.
3. Warum ist  $r : ((V, E), k) \mapsto ((V, \bar{E}), |V| - k)$  eine polynomiale Reduktion von **CLIQUE** auf **KNOTENÜBERDECKUNG**?

**Fortsetzung nächste Seite!**

**Aufgabe 5: Algorithmen und Datenstrukturen I**

Führen Sie den Algorithmus von Dijkstra mit Startknoten  $s$  auf dem folgenden Graphen durch, um einen Kürzeste-Wege-Baum zu finden. Übernehmen Sie dazu die Berechnungstabelle auf Ihr Lösungsblatt und füllen Sie dort die Zeilen der Schritte 2. bis 9. aus. Markieren Sie zum Schluss alle Kanten, die zum berechneten Kürzeste-Wege-Baum gehören.

*Anmerkung:* Für den  $i$ -ten Schritt enthält die Spalte *Aktueller Knoten* den im  $i$ -ten Schritt betrachteten aktuellen Knoten zusammen mit seinem Distanzwert.  $(s, 0)$  im ersten Schritt bedeutet also, dass der Knoten  $s$  die Distanz 0 zu  $s$  hat. Für den  $i$ -ten Schritt enthält die Spalte *Inhalt der Priority-Queue* Paare bestehend aus Knoten und zugehöriger Priorität.  $(g, 1)$  bedeutet also beispielsweise, dass Knoten  $g$  mit Priorität 1 in der Queue ist.



Schritt	Aktueller Knoten	Vorgänger-Array								Inhalt der Priority-Queue
		a	b	c	d	e	f	g	h	
1.	$(s, 0)$		$s$		$s$	$s$		$s$		$(g, 1), (e, 7), (d, 8), (b, 14)$
2.										
3.										
4.										
5.										
6.										
7.										
8.										
9.										

**Aufgabe 6: Algorithmen und Datenstrukturen II**

Für ein Array  $A$  bezeichne  $A[i]$  das  $i$ -te Element von  $A$ . Die Elemente eines Arrays der Länge  $n$  haben die Indizes 1 bis  $n$ .

Das Maximum Subarray Sum Problem (kurz: MSAS Problem) ist wie folgt definiert:

**Gegeben:** Ein nichtleeres Array  $A$  der Länge  $n \in \mathbb{N}$  von ganzen Zahlen (d.h. Zahlen aus  $\mathbb{Z}$ ).

**Aufgabe:** Finden Sie die größte Zahl  $s \in \mathbb{Z}$ , sodass  $s$  die Summe der Einträge eines nichtleeren Teilarrays von  $A$  ist. D.h., finden Sie  $s = \max \left\{ \sum_{k=i}^j A[k] ; 1 \leq i \leq j \leq n \right\}$ .

1. Betrachten Sie Algorithmus 1.

**Algorithmus 1 : MSAS ( $A, i, j$ )**

```

1  $\ell \leftarrow j - i + 1$ 
2 if  $\ell = 1$  then
3   | return  $A[i]$ 
4  $sA \leftarrow 0$ 
5 for  $r \leftarrow i$  to  $j$  do
6   |  $sA \leftarrow sA + A[r]$ 
7  $sLinks \leftarrow \text{MSAS}(A, i, j - 1)$ 
8  $sRechts \leftarrow \text{MSAS}(A, i + 1, j)$ 
9 return max { $sA, sLinks, sRechts$ }
```

- Begründen Sie weshalb dieser Algorithmus mit dem Aufruf  $\text{MSAS}(A, 1, n)$  das MSAS Problem löst.
  - Analysieren Sie die Laufzeit des Algorithmus.
2. Algorithmus 1 ist relativ ineffizient, da einige Teilarrays zu oft untersucht werden.  
Betrachten Sie Algorithmus 2.

Hierbei ist  $B$  ein zweidimensionales *globales* Array, dessen Einträge mit  $-\infty$  vorinitialisiert sind.  
Analysieren Sie die Laufzeit dieses Algorithmus.

**Algorithmus 2 : MSAS ( $A, i, j$ )**

```

1 if  $B[i][j] \neq -\infty$  then
2   | return  $B[i][j]$ 
3  $\ell \leftarrow j - i + 1$ 
4 if  $\ell = 1$  then
5 return  $A[i]$ 
6  $sA \leftarrow 0$ 
7 for  $r \leftarrow i$  to  $j$  do
8   |  $sA \leftarrow sA + A[r]$ 
9  $sLinks \leftarrow \text{MSAS}(A, i, j - 1)$ 
10  $sRechts \leftarrow \text{MSAS}(A, i + 1, j)$ 
11  $B[i][j] \leftarrow \max \{sA, sLinks, sRechts\}$ 
12 return  $B[i][j]$ 
```

Fortsetzung nächste Seite!

3. Eine bessere Idee, als alle Teilarrays zu betrachten, ist die folgende. Für ein Array  $B$  bezeichne  $s(B)$  die größte Zahl, die die Summe eines nichtleeren Teilarrays von  $B$  ist. Außerdem bezeichne  $sRechts(B)$  die größte Zahl, die die Summe eines nichtleeren Teilarrays von  $B$  ist, das das letzte Element von  $B$  enthält. Sei  $A_i$  das Teilarray von  $A$ , das aus den Elementen  $A[1], \dots, A[i]$  gebildet wird.

Angenommen, wir kennen für ein  $i \in \{1, \dots, n\}$  bereits die Zahlen  $s(A_i)$  und  $sRechts(A_i)$ . Dann können wir auf sehr einfache Weise die Zahlen  $s(A_{i+1})$  und  $sRechts(A_{i+1})$  bestimmen.

Geben Sie einen Algorithmus in Pseudo-Code an, der keine rekursiven Aufrufe verwendet und mit obiger Idee das MSAS Problem löst. Der Algorithmus soll eine möglichst gute Laufzeit besitzen. (Lineare Laufzeit ist möglich.)

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2017**

---

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	<b>Herbst</b>	
Kennwort: _____	<b>2017</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

---

**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: **14**

---

Bitte wenden!

Thema Nr. 1  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

**Myhill Nerode**

Für zwei Wörter  $x, y \in \Sigma^*$  und eine Sprache  $L \subseteq \Sigma^*$  gilt  $x \equiv_L y$  genau dann, wenn für jedes Wort  $z \in \Sigma^*$  gilt: ( $xz \in L \iff yz \in L$ ). Die Relation  $\equiv_L$  ist eine Äquivalenzrelation.

Der Satz von Myhill und Nerode sagt, dass eine Sprache genau dann regulär ist, wenn  $\equiv_L$  endlich viele Äquivalenzklassen hat.

- a) Wir bezeichnen mit  $\#_a(w)$  die Anzahl Vorkommen des Alphabetsymbols  $a$  in der Zeichenkette  $w$ . Zeigen Sie mit Hilfe des Satzes von Myhill und Nerode, dass die Sprache

$$L_1 = \{w \mid \#_a(w) = \#_b(w)\}$$

über dem Alphabet  $\{a, b\}$  nicht regulär ist.

- b) Geben Sie alle Äquivalenzklassen der Äquivalenzrelation  $\equiv_{L_2}$  an. Hier ist  $L_2 = L((a + b)^*a)$  (über dem Alphabet  $\{a, b\}$ ). Begründen Sie auch, warum es keine anderen Äquivalenzklassen gibt.  
 c) Ändert sich die Anzahl der Äquivalenzklassen der Sprache  $L_2$ , wenn Sie stattdessen Wörter über dem Alphabet  $\{a, b, c\}$  betrachten? Begründen Sie Ihre Antwort.

**Aufgabe 2:**

**Kontextfreie Sprachen**

Betrachten Sie die Sprache  $L_1 = \{a^nbc^n \mid n \in \mathbb{N}\} \cup \{ab^mc^m \mid m \in \mathbb{N}\}$ .

- a) Geben Sie für  $L_1$  eine kontextfreie Grammatik an.  
 b) Ist Ihre Grammatik aus a) eindeutig? Begründen Sie Ihre Antwort.

Betrachten Sie die Sprache  $L_2 = \{a^{2^n} \mid n \in \mathbb{N}\}$ .

- c) Zeigen Sie, dass  $L_2$  nicht kontextfrei ist.

### Aufgabe 3: Komplexität

Betrachten Sie die folgenden Probleme:

#### 3SAT

Gegeben: Eine aussagenlogische Formel  $\varphi$  in konjunktiver Normalform  
(drei Literale pro Klausel).

Frage: Ist  $\varphi$  erfüllbar?

#### NAE-3SAT

Gegeben: Eine aussagenlogische Formel  $\varphi$  in konjunktiver Normalform  
(drei Literale pro Klausel).

Frage: Gibt es eine Belegung, die in jeder Klausel  
mindestens ein Literal wahr und  
mindestens ein Literal falsch macht?

Wir erlauben, dass NAE-3SAT-Formeln Literale der Form `false` haben, die immer falsch sind.  
So ist

$$(x_1 \vee \text{false} \vee \text{false}) \wedge (\neg x_1 \vee x_1 \vee x_1)$$

in NAE-3SAT (setze  $x_1$  wahr).

- a) Zeigen Sie, dass sich 3SAT in polynomieller Zeit auf NAE-3SAT reduzieren lässt.
- b) Was können Sie aus a) folgern, wenn Sie wissen, dass 3SAT NP-vollständig ist?
- c) Was können Sie aus a) folgern, wenn Sie wissen, dass NAE-3SAT NP-vollständig ist?

### Aufgabe 4: Berechenbarkeit

Betrachten Sie die folgenden Sprachen:

- $H = \{w\$x \mid M_w \text{ hält bei Eingabe } x\}$  über Alphabet  $\{0, 1, \$\}$
- $H_\epsilon = \{w \mid M_w \text{ hält bei Eingabe } \epsilon\}$  über Alphabet  $\{0, 1\}$

Dabei sei  $x \in \{0, 1\}^*$  und bezeichnet  $M_w$  die von  $w \in \{0, 1\}^*$  kodierte Turingmaschine.

- a) Zeigen Sie, dass es eine Reduktion von  $H_\epsilon$  auf  $H$  gibt.
- b) Zeigen Sie, dass es eine Reduktion von  $H$  auf  $H_\epsilon$  gibt.

Diese Reduktionen dürfen mehr als polynomielle Zeit benötigen.

**Aufgabe 5:****Die Begründung zählt**

Bewerten Sie die folgenden Aussagen und geben Sie eine kurze Begründung (ca. ein bis drei Sätze) Ihrer Bewertung an. Nur Ihre Begründung wird bewertet.

- Eine universelle Turingmaschine muss ein unendliches Eingabealphabet haben, um beliebige Turingmaschinen simulieren zu können.
- Für beliebige reguläre Ausdrücke  $\alpha, \beta$  gilt:  $L(\alpha \cdot (\alpha + \beta)) = L(\alpha) \cup L(\alpha + \beta)$ .
- Sei  $L_1$  eine Sprache und  $L_2$  eine reguläre Sprache. Dann ist  $L_1 \cap L_2$  immer entscheidbar.
- Es gibt LOOP-Programme, die auch WHILE-Programme sind.
- Mit dem Satz von RICE kann man nachweisen, dass es unentscheidbar ist, ob eine gegebene Turingmaschine mehr als 10 Zustände hat.
- Es ist NP-schwer zu entscheiden, ob die Sprache eines endlichen Automaten leer ist.

**Aufgabe 6:****Sortieren und das Gegenteil davon**

- Sei  $S = (42, 4711, 98, 103, 1001, -13, 8, 98, 5, 13, 45, 64, 42, 98, 0, 17)$ . Zeichnen Sie den Rekursionsbaum der Sortierung von  $S$  mittels MergeSort. Geben Sie für jeden Knoten sowohl die Ein- als auch die Ausgabe des entsprechenden Aufrufs an.
- MergeSort benötigt zum Sortieren von  $n$  Elementen  $O(n \log n)$  Schritte im Worst-Case. Welche Eigenschaft muss eine Eingabe haben, sodass diese Laufzeit tatsächlich erreicht wird?
- Geben Sie einen Algorithmus in Pseudocode an, der eine Folge von  $n$  Elementen aus der Menge  $U = \{0, 1\}$  in  $O(n)$  Zeit sortiert.

Wir sagen eine Methode  $M(S)$  permutiert eine Folge  $S = (x_1, \dots, x_n)$  von  $n$  paarweise verschiedenen Elementen, wenn sie die Reihenfolge der Elemente von  $S$  potentiell verändert (die ursprüngliche Reihenfolge darf auch beibehalten werden). Eine gute Permutationsmethode erzeugt jede mögliche Permutation von  $S$  mit gleicher Wahrscheinlichkeit.

Betrachten Sie die folgende Permutationsmethode:

**1 Algorithmus : CleverShuffle( $S$ )**

**Eingabe :** Eine Folge  $S = (x_1, \dots, x_n)$

**Ergebnis :**  $S$  wurde permutiert.

**2 begin**

```

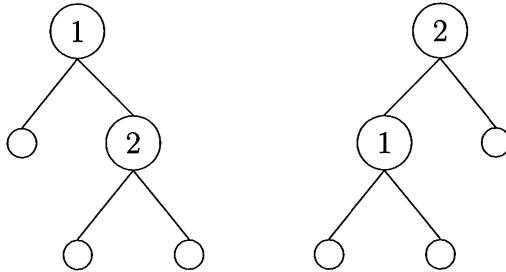
3   |   for  $i \leftarrow 1$  to  $n$  do
4   |   |    $k \leftarrow \text{Random}(i, n);$ 
5   |   |   vertausche  $S[i] \leftrightarrow S[k];$ 
```

Die Methode  $\text{Random}(i, j)$  wählt dabei zufällig eine ganze Zahl aus dem Intervall  $[i, j]$ , jede mit Wahrscheinlichkeit  $\frac{1}{j+1-i}$ . Man kann zeigen, dass CleverShuffle jede mögliche Permutation von  $S$  erzeugen kann, und jede mögliche Permutation mit gleicher Wahrscheinlichkeit erzeugt.

- Was ist die asymptotische Laufzeit von CleverShuffle, wenn Random eine Laufzeit von  $\Theta(1)$  hat?

**Aufgabe 7:****Zählen von binären Suchbäumen**

Für  $n \geq 0$  betrachten wir die Folge  $X_n = (1, \dots, n)$  der ersten  $n$  natürlichen Zahlen ( $X_0$  ist also die *leere Folge*). Wir wollen einen effizienten Algorithmus entwerfen, der berechnet, wie viele *verschiedene* binäre Suchbäume für (die Elemente von)  $X_n$  existieren. Wir bezeichnen diese Zahl mit  $b(n)$ . Für  $n = 2$  gibt es zwei verschiedene binäre Suchbäume (d.h.  $b(2) = 2$ ):



1. Begründen Sie, dass  $b(0) = 1$  und  $b(1) = 1$ .
2. Warum ist die Anzahl der verschiedenen binären Suchbäume für die Folge  $(4, 5)$  genau  $b(2)$ ?
3. Begründen Sie, dass  $b(3) = 5$ .
4. Welchen (arithmetischen) Zusammenhang erkennen Sie zwischen der Anzahl der verschiedenen binären Suchbäume für  $X_5$  die in der Wurzel den Schlüssel 3 speichern, und der Anzahl der verschiedenen binären Suchbäume für die Folgen  $(1, 2)$  bzw.  $(4, 5)$ ? Begründen Sie diesen.
5. Begründen Sie, dass

$$b(n) = \sum_{0 \leq l \leq n-1} b(l) \cdot b(n-1-l) \text{ für } n > 1 \\ \text{und } b(n) = 1 \text{ für } n \leq 1.$$

6. Wie sieht eine Reihenfolge aus, in der die  $b(n)$  bottom-up berechnet werden können, die mit der Rekursionsgleichung für  $b(n)$  verträglich ist? Begründen Sie Ihre Antwort.
7. Geben Sie ein *dynamisches Programm* in Pseudocode an, das den Wert  $b(n)$  iterativ statt rekursiv berechnet.
8. Warum ist Ihr Algorithmus korrekt? Begründen Sie Ihre Antwort.
9. Was ist die asymptotische Laufzeit Ihres Algorithmus? Was ist der asymptotische Speicherbedarf Ihres Algorithmus? Begründen Sie Ihre Antworten.

*Hinweis:* Wenn Sie die obige Aufgabe nicht gelöst haben, verwenden Sie folgenden Algorithmus zur Beantwortung dieser Frage (*Achtung:* Das ist nicht die Lösung obiger Aufgabe!):

```

1 Algorithmus : ExAlg( $n$ )
2 begin
3   Initialize Array  $a[0,..,n]$ ;
4    $a[0] \leftarrow 1$ ;
5    $a[1] \leftarrow 2$ ;
6   for  $l \leftarrow 2$  to  $n$  do
7      $s \leftarrow 1$ ;
8     for  $r \leftarrow 1$  to  $l$  do
9        $| s \leftarrow s \cdot (a[r-1] + a[n-r])$ ;
10       $a[l] \leftarrow s$ ;
11  return  $a[n]$ ;
  
```

**Aufgabe 8:****Greedy-Färben von Intervallen**

Sei  $X = \{I_1, I_2, \dots, I_n\}$  eine Menge von  $n$  (geschlossenen) Intervallen über den reellen Zahlen  $\mathbb{R}$ . Das Intervall  $I_j$  sei dabei gegeben durch seine linke Intervallgrenze  $l_j \in \mathbb{R}$  sowie seine rechte Intervallgrenze  $r_j \in \mathbb{R}$  mit  $r_j > l_j$ , d.h.  $I_j = [l_j, r_j]$ .

Wir nehmen in dieser Aufgabe der Einfachheit halber an, dass die Zahlen  $l_1, l_2, \dots, l_n, r_1, r_2, \dots, r_n$  alle paarweise verschieden sind.

Zwei Intervalle  $I_j, I_k$  überlappen sich gdw. sie mindestens einen Punkt gemeinsam haben, d.h. gdw. falls für (o.B.d.A.)  $l_j < l_k$ , auch  $l_k < r_j$  gilt. Eine *gültige Färbung* von  $X$  mit  $c \in \mathbb{N}$  Farben ist eine Funktion  $F : X \rightarrow \{1, 2, \dots, c\}$  mit der Eigenschaft, dass für jedes Paar  $I_j, I_k$  von überlappenden Intervallen  $F(I_j) \neq F(I_k)$  gilt.

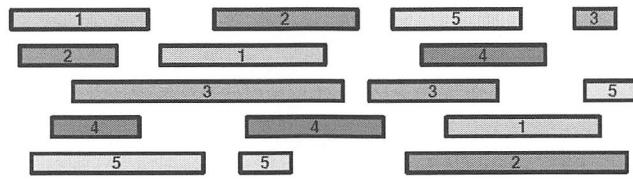


Abbildung 1: Eine gültige Färbung von  $X$

Eine *minimale gültige Färbung* von  $X$  ist eine gültige Färbung mit einer minimalen Anzahl an Farben. Die Anzahl von Farben in einer minimalen gültigen Färbung von  $X$  bezeichnen wir mit  $\chi(X)$ .

Wir gehen im Folgenden davon aus, dass für  $X$  eine minimale gültige Färbung  $F^*$  gefunden wurde.

1. Nehmen wir an, dass aus  $X$  alle Intervalle einer bestimmten Farbe von  $F^*$  gelöscht werden. Ist die so aus  $F^*$  entstandene Färbung der übrigen Intervalle in jedem Fall immer noch eine minimale gültige Färbung? Begründen Sie Ihre Antwort.
2. Nehmen wir an, dass aus  $X$  ein beliebiges Intervall gelöscht wird. Ist die so aus  $F^*$  entstehende Färbung der übrigen Intervalle in jedem Fall immer noch eine minimale gültige Färbung? Begründen Sie Ihre Antwort.
3. Mit  $\omega(X)$  bezeichnen wir die maximale Anzahl von Intervallen in  $X$ , die sich paarweise überlappen. Zeigen Sie, dass  $\chi(X) \geq \omega(X)$  ist.

Wir betrachten nun folgenden Algorithmus, der die Menge  $X = \{I_1, I_2, \dots, I_n\}$  von  $n$  Intervallen einfärbt:

- Zunächst sortieren wir die Intervalle von  $X$  aufsteigend nach ihren *linken* Intervallgrenzen. Die Intervalle werden jetzt in *dieser Reihenfolge* nacheinander eingefärbt; ist ein Intervall dabei erst einmal eingefärbt, ändert sich seine Farbe nie wieder. Angenommen die sortierte Reihenfolge der Intervalle sei  $I_{\sigma(1)}, \dots, I_{\sigma(n)}$ .
- Das erste Intervall  $I_{\sigma(1)}$  erhält die Farbe 1. Für  $1 < i \leq n$  verfahren wir im  $i$ -ten Schritt zum Färben des  $i$ -ten Intervalls  $I_{\sigma(i)}$  wie folgt:

Bestimme die Menge  $C_i$  aller Farben der bisher schon eingefärbten Intervalle die  $I_{\sigma(i)}$  überlappen. Färbe  $I_{\sigma(i)}$  dann mit der Farbe  $c_i = \min(\{1, 2, \dots, n\} \setminus C_i)$ .

4. Begründen Sie, warum der Algorithmus immer eine gültige Färbung von  $X$  findet (*Hinweis: Induktion*).
5. Zeigen Sie, dass die Anzahl an Farben, die der Algorithmus für das Einfärben benötigt, mindestens  $\omega(X)$  ist.
6. Zeigen Sie, dass die Anzahl an Farben, die der Algorithmus für das Einfärben benötigt, höchstens  $\omega(X)$  ist.
7. Begründen Sie mit Hilfe der o.g. Eigenschaften, warum der Algorithmus korrekt ist, d.h. immer eine minimale gültige Färbung von  $X$  findet.
8. Wir betrachten folgenden Implementierung des Algorithmus in Pseudocode:

```

1 Algorithmus : ColoringNumber( $X_L[1, \dots, n], X_R[1, \dots, n]$ )
Eingabe : Felder  $X_R$  und  $X_L$  mit den rechten und linken Intervallgrenzen.
Ergebnis : Minimale gültige Färbung der Intervalle.

2 begin
3   sortiere  $X_L$  (und passe  $X_R$  an);                                /*
/* color[i] ist die Farbe des Intervalls  $i$                                 */
4   initialisiere Array  $color[1,..,n]$ ;      // mit Nullen
/* lastintervalofcolor[c] ist der Index des letzten Intervalls das mit  $c$  gefärbt
wurde                                */
5   initialisiere Array  $lastintervalofcolor[1,..,n]$ ;    // mit Nullen
6    $maxcolor \leftarrow 1$ ;
7    $freecolor \leftarrow maxcolor$ ;
8    $color[1] \leftarrow freecolor$ ;
9    $lastintervalofcolor[freecolor] \leftarrow 1$ ;
10  for  $i \leftarrow 2$  to  $n$  do
11     $freecolorfound \leftarrow false$ ;
12    for  $c \leftarrow 1$  to  $maxcolor$  do
13       $i_c \leftarrow lastintervalofcolor[c]$ ;
14      if  $X_L[i] > X_R[i_c]$  then
/* i schneidet kein Interval der Farbe c                                */
15         $freecolorfound \leftarrow true$ ;
16         $freecolor \leftarrow c$ ;
17        break;
/* i schneidet ein Interval der Farbe c                                */
18      if  $!freecolorfound$  then
19         $maxcolor \leftarrow maxcolor + 1$ ;
20         $freecolor \leftarrow maxcolor$ ;
21         $color[i] \leftarrow freecolor$ ;
22         $lastintervalofcolor[freecolor] \leftarrow i$ ;
23  return  $color$ ;

```

Was ist die asymptotische Laufzeit dieses Algorithmus? Was ist der asymptotische Speicherbedarf dieses Algorithmus? Begründen Sie Ihre Antworten.

Thema Nr. 2  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1:**

Zeigen oder widerlegen Sie die folgenden **Aussagen** (die jeweiligen Beweise sind sehr kurz):

- a) Bezeichne  $SAT = \{\langle\Phi\rangle \mid \Phi \text{ ist erfüllbare Booleschen Formel in konjunktiver Normalform}\}$  das NP-vollständige Erfüllbarkeitsproblem.

Es gibt eine Grammatik  $G$  vom Typ Chomsky-0, die genau die  $\langle\Phi\rangle$  der **erfüllbaren** Booleschen Formeln  $\Phi$  in konjunktiver Normalform erzeugt, d.h.  $L(G) = SAT$ .

*Hinweis:*  $\langle\Phi\rangle$  bezeichnet lediglich eine Kodierung der Formel  $\Phi$ .

- b) Sei  $L$  eine beliebige kontextfreie Sprache über dem Alphabet  $\Sigma$ . Dann ist das Komplement  $\overline{L} = \Sigma^* \setminus L$  entscheidbar.
- c) Es ist entscheidbar, ob eine durch einen deterministischen endlichen Automaten gegebene Sprache unendlich viele Wörter enthält.
- d) Seien  $L_1$  und  $L_2$  beliebige Sprachen über dem Alphabet  $\Sigma = \{0, 1\}$  mit  $L_1 \neq L_2$ . Ist  $L_1 \cap L_2$  entscheidbar, dann ist mindestens eine der beiden Sprachen  $L_1$  und  $L_2$  entscheidbar.

Schreiben Sie zuerst zur Aussage „Stimmt“ oder „Stimmt nicht“ und dann Ihre Begründung.

**Aufgabe 2:**

Sei  $M_0, M_1, \dots$  eine Gödelisierung aller Registermaschinen (RAMs). Das spezielle Halteproblem ist definiert als  $K_0 = \{x \in \mathbb{N} \mid M_x \text{ hält bei Eingabe } x\}$ . Gesucht ist eine *totale*, berechenbare Funktion  $g : \mathbb{N} \rightarrow \mathbb{N}$  mit  $W_g = K_0$ . Dabei bezeichnet  $W_g$  den Wertebereich von  $g$ , d.h.  $W_g = \{g(x) \mid x \in \mathbb{N}\}$ . Gehen Sie wie folgt vor:

- a) Definieren Sie eine Funktion  $g : \mathbb{N} \rightarrow \mathbb{N}$ .
- b) Beweisen Sie, dass  $g$  total und berechenbar ist.
- c) Beweisen Sie  $W_g \subseteq K_0$ .
- d) Beweisen Sie  $W_g \supseteq K_0$ .

**Aufgabe 3:**

Sei  $A$  die durch den regulären Ausdruck  $(a + b + c)^* \cdot c \cdot b^* \cdot a \cdot (a + b)^* \cdot c \cdot b^*$  beschriebene Sprache.

- a) Geben Sie einen nichtdeterministischen, endlichen Automaten  $N$  an, der  $A$  akzeptiert.
- b) Wandeln Sie  $N$  in einen äquivalenten deterministischen, endlichen Automaten um.

**Aufgabe 4:**

Sei  $L$  die durch den regulären Ausdruck  $(10 \cup 11)^*((11 \cup \varepsilon)^*1)^*$  beschriebene Sprache. [Alternative Schreibweise:  $(10 + 11)^*((11 + \varepsilon)^*1)^*$ ]

- a) Sei SAT das Erfüllbarkeitsproblem (siehe Aufgabe 1 a) zur Definition).

Zeigen Sie:  $L$  kann in polynomieller Zeit auf SAT reduziert werden (als Relation:  $L \leq_p \text{SAT}$ ).

- b) Angenommen, es wurde für das NP-vollständige Problem SAT gezeigt, dass  $\text{SAT} \in \text{P}$  ist.  
Zeigen Sie, dass  $L$  dann NP-vollständig ist.

**Aufgabe 5:**

Sei  $G = (V, \Sigma, P, S)$  eine kontextfreie Grammatik ( $V$ : Variablenmenge;  $\Sigma$ : Menge der Terminalsymbole;  $S$ : Startsymbol;  $P$ : Menge der Produktionen) in Chomsky-Normalform, und sei  $w = w_1 \dots w_n$  ein Wort aus  $n$  Zeichen aus  $\Sigma$ . Der Algorithmus von Cocke/Younger/Kasami (CYK-Algorithmus) berechnet für alle  $i, j \in \{1, \dots, n\}$ ,  $i \leq j$ , die Variablenmenge  $V(i, j) = \{A \in V \mid A \xrightarrow{*} w_i \dots w_j\}$ .

Sei  $G = (V, \Sigma, P, S)$  die kontextfreie Grammatik in Chomsky-Normalform mit  $V = \{S, A, B, C\}$ ,  $\Sigma = \{a, b\}$ , Startsymbol  $S$  und der Menge  $P$  der Produktionen:

$$S \rightarrow AB \mid BC \quad B \rightarrow CC \mid b$$

$$A \rightarrow BA \mid a \quad C \rightarrow AB \mid a$$

Sei  $w = baaab$ . Folgende Tabelle entsteht durch Anwendung des CYK-Algorithmus. Z. B. bedeutet  $B \in V(3, 5)$ , dass aus der Variablen  $B$  das Teilwort  $w_3 w_4 w_5 = aab$  hergeleitet werden kann. Drei Einträge wurden weggelassen.

$b$	$a$	$a$	$a$	$b$
$V(1,1)$ $\{B\}$	$V(2,2)$ $\{A, C\}$	$V(3,3)$ $\{A, C\}$	$V(4,4)$ $\{A, C\}$	$V(5,5)$ $\{B\}$
$V(1,2)$ 	$V(2,3)$ $\{B\}$	$V(3,4)$ $\{B\}$	$V(4,5)$ $\{S, C\}$	
$V(1,3)$ 	$V(2,4)$ $\{S, A, C\}$	$V(3,5)$ $\{B\}$		
$V(1,4)$ $\{S, A, C\}$	$V(2,5)$ $\{S, C\}$			
$V(1,5)$ 				

- i) Bestimmen Sie die Mengen  $V(1, 2)$ ,  $V(1, 3)$  und  $V(1, 5)$ .  
ii) Wie entnehmen Sie dieser Tabelle, dass  $w \in L(G)$  ist?

**Aufgabe 6:****Wissen**

Ordnen Sie die unten stehenden Aussagen entsprechend ihres Wahrheitsgehalts in einer Tabelle der folgenden Form ein:

<i>Kategorie</i>	<i>WAHR</i>	<i>FALSCH</i>
X	X1, X3	X2
Y	Y2	Y1
...	...	...

**A – Datenstrukturen**

- A1 Eine Streutabelle, die Kollisionen mit Hilfe einer verketteten Liste auflöst, kann beliebig viele Elemente speichern.
- A2 Wird der Datentyp `Set<T>` (zur Darstellung von Mengen) mit einer doppelt verketteten Liste ohne Sortierung implementiert, dann haben das Einfügen eines Wertes in eine bestehende Menge mit  $n$  Werten und das anschließende Löschen eines anderen Wertes aus dieser Menge zusammen eine Laufzeitkomplexität von  $\mathcal{O}(\log_2(n))$ .
- A3 Eine doppelt verkettete Liste der Länge  $n$  ohne wahlfreien Zugriff erlaubt das Löschen des Listenkopfes ebenso wie des Listenendes in  $\mathcal{O}(1)$  (konstanter Aufwand, unabhängig von  $n$ ).
- A4 Eine doppelt verkettete Liste ohne wahlfreien Zugriff kann zur Umsetzung von Warteschlangen nach dem FIFO-Prinzip verwendet werden.

**B – Laufzeitkomplexität:**

Gegeben seien zwei Methoden `f(int n)` und `g(int n)` mit den jeweiligen Laufzeiten  $\mathcal{O}_f(\log(n))$  bzw.  $\mathcal{O}_g(n)$ . Bei welchen der folgenden Schleifen stimmt die Laufzeitangabe im Kommentar?

- B1 `for (int i = 0; i < n; i++) { f(n); } //  $\mathcal{O}(n)$`
- B2 `for (int i = 0; i < n; i++) { g(i); } //  $\mathcal{O}(n^2)$`
- B3 `for (int i = 0; i < n; i *= 2) { f(i); g(i); } //  $\mathcal{O}(n^3)$`
- B4 `for (int i = n; i > 0; i /= 2) { f(n); } //  $\mathcal{O}(\log^2(n))$`

## C – Graphen:

Ein minimaler Spannbaum eines zusammenhängenden Graphen mit  $n$  Knoten ...

- |    |   |
|----|---|
| C1 | ... ist stets eindeutig (zu einem Graphen kann es also keine verschiedenen Spannbäume geben). |
| C2 | ... muss nicht immer zusammenhängend sein.  |
| C3 | ... enthält höchstens $n - 2$ Kanten.   |
| C4 | ... enthält genau $n - 1$ Kanten.   |

## D – Bäume:

- |    |   |
|----|---|
| D1 | Das Einfügen eines Elements in eine <i>Halde (Heap)</i> mit $n$ Einträgen hat konstante Laufzeit.   |
| D2 | Das Einfügen von $n$ Werten in einen <i>binären Suchbaum</i> hat eine Laufzeit von $\mathcal{O}(n)$ .   |
| D3 | Sowohl das Einfügen und das Löschen, als auch das Suchen haben in einem AVL-Baum einen Laufzeitaufwand von maximal $\mathcal{O}(\log_2(n))$ .     |
| D4 | In einem AVL-Baum entspricht der <i>Balancefaktor</i> eines Knotens der Summe der Höhe des linken Unterbaums und der Höhe des rechten Unterbaums. |

**Aufgabe 7:****Rekursion**

Die *Potenzmenge*  $\mathcal{P}(n)$  sei die Menge aller Teilmengen der Zahlen von 1 bis  $n$  (jeweils einschließlich), wobei die leere Menge  $\emptyset$  auch zu den Teilmengen gehört, z. B.  $\mathcal{P}(3) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ . Ergänzen Sie die Methode *potenzmenge*, die rekursiv  $\mathcal{P}(n)$  bestimmen soll:

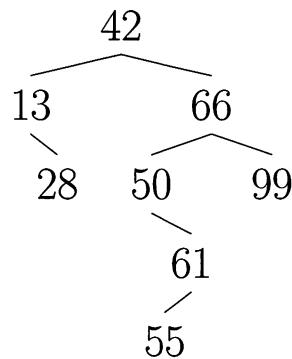
*Hinweise zur API der Klasse ArrayList<E>:*

- `public ArrayList(Collection<? extends E> c): Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.`
- `public boolean add(E e): Appends the specified element to the end of this list.`

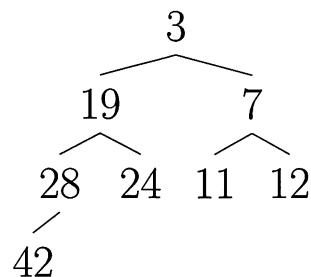
```
static List<List<Long>> potenzmenge(long n) {
    // Rueckgabe pm ist Potenzmenge der Zahlen von 1 bis n
    List<List<Long>> pm = new ArrayList<>();
    ...
}
```

**Aufgabe 8:****Bäume**

- a) Fügen Sie die Zahlen **13, 12, 42, 3, 11** in der gegebenen Reihenfolge in einen zunächst leeren *binären Suchbaum* mit aufsteigender Sortierung ein. Stellen Sie nur das Endergebnis dar.
- b) Löschen Sie den Wurzelknoten mit Wert **42** aus dem folgenden *binären Suchbaum* mit aufsteigender Sortierung und ersetzen Sie ihn dabei durch einen geeigneten Wert aus dem rechten Teilbaum. Lassen Sie möglichst viele Teilbäume unverändert und erhalten Sie die Suchbaum-eigenschaft.

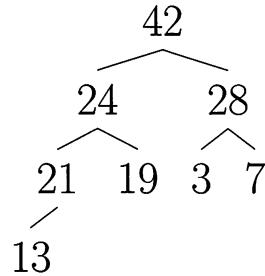


- c) Fügen Sie einen neuen Knoten mit dem Wert **13** in die folgende *Min-Halde* ein und stellen Sie anschließend die Halden-Eigenschaft vom neuen Blatt aus beginnend wieder her, wobei möglichst viele Knoten der Halde unverändert bleiben und die Halde zu jedem Zeitpunkt links-vollständig sein soll. Geben Sie nur das Endergebnis an.



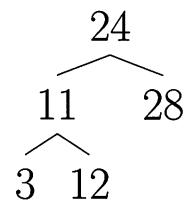
- d) Geben Sie für die *ursprüngliche Min-Halde* aus Teilaufgabe c) (d.h. ohne den neu eingefügten Knoten mit dem Wert 13) die Feld-Einbettung (Array-Darstellung) an.

- e) Löschen Sie den Wurzelknoten mit Wert **42** aus der folgenden Max-Halde und stellen Sie anschließend die Halden-Eigenschaft ausgehend von einer neuen Wurzel wieder her, wobei möglichst viele Knoten der Halde unverändert bleiben und die Halde zu jedem Zeitpunkt links-vollständig sein soll. Geben Sie nur das Endergebnis an.

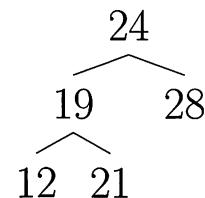


- f) Fügen Sie in jeden der folgenden *AVL-Bäume* mit aufsteigender Sortierung jeweils einen neuen Knoten mit dem Wert **13** ein und führen Sie anschließend *bei Bedarf* die erforderliche(n) Rotation(en) durch. Zeichnen Sie den Baum vor und nach den Rotationen.

i) AVL-Baum  $\mathcal{A}$ :

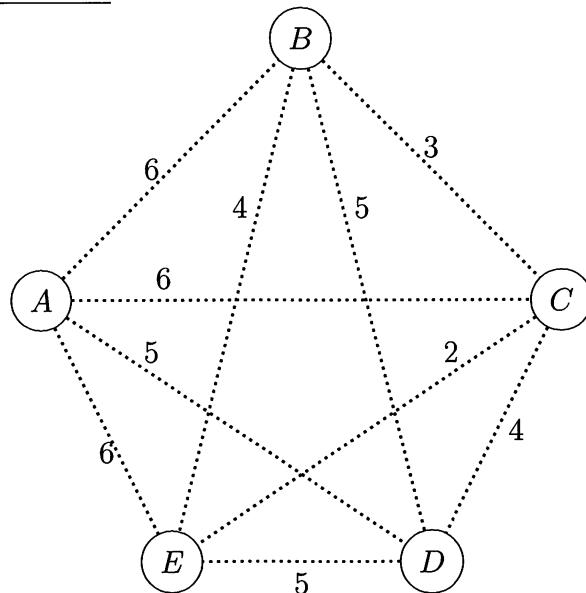


ii) AVL-Baum  $\mathcal{B}$ :



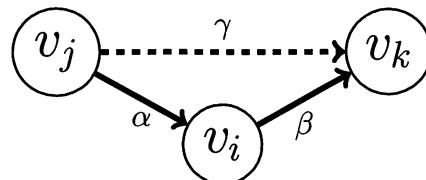
**Aufgabe 9:****Graphen**

- a) Wenden Sie den Algorithmus von *Kruskal* auf den Graphen  $\mathcal{F}$  an und geben Sie die Kanten in der Reihenfolge an, in welcher das Verfahren sie *in den Spannbaum aufnimmt*:

Graph  $\mathcal{F}$ :

*Was Sie in den Graphen zeichnen  
wird **nicht** bewertet!*

- b) Für einen beliebigen Graphen  $\mathcal{G} = (V, E)$  mit Kantenanzahl  $e := |E|$  und Knotenzahl  $v := |V|$ , welche Laufzeitkomplexitäten (in Landau- $\mathcal{O}$ -Notation) haben effiziente Implementierungen der Algorithmen von *Prim* und *Kruskal* im allgemeinen Fall?
- c) Ergänzen Sie den Algorithmus von *Floyd* für Graphen  $\mathcal{G}$ , dargestellt durch die Adjazenzmatrix  $g$ . Der Eintrag  $g[j][k]$  der Adjazenzmatrix  $g$  enthält das (positive) Gewicht der Kante  $(j, k)$  bzw. 0, falls es keine solche Kante gibt oder falls  $j = k$ . Die Implementierung arbeitet hier *in-situ* und ergänzt auch indirekte Kanten direkt in  $g$ .



```
// betrachte alle Kanten-Tripel  $v_j \Rightarrow v_i \Rightarrow v_k$ ,  
// vergleiche diese Weglänge mit  $v_j \Rightarrow v_k$  (sofern vorhanden)  
// und aktualisiere ggf. die Weglänge  $v_j \Rightarrow v_k$   
public void floydify(double [][] g) {  
    int n = g.length;  
    // durchlaufe alle Knoten  $v_i$  und  $v_j$ :  
    for (int i = 0; i < n; i++) {  
        ...  
    }  
}
```

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2018**

---

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	Frühjahr	
Kennwort: _____	2018	66115
Arbeitsplatz-Nr.: _____		

---

## 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: 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

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

- Führen Sie für diesen NEA die Potenzmengenkonstruktion durch; geben Sie alle acht entstehenden Zustände mit ihren Transitionen an, nicht nur die erreichbaren.
- 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  $c$ s entstehen. Ist zum Beispiel  $L = \{acb, acbcc, abbaccc\}$ , so ist  $L' = \{ab, abba\}$ . Zeigen Sie, dass  $L'$  regulär ist.
- 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.
- 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.
- 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.

- 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.
- 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.

**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  $l + 7 \geq r$ 
2    then
3      INSERTIONSORT( $A, l, r$ )      //sortiere  $A$  von  $l$  bis  $r$  mit InsertionSort
4    else
5       $m := \lfloor (r - l + 1)/4 \rfloor$ 
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.

- 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.
- 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  $M_0, M_1, \dots$  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?  
 $D_3 = \{x \in \mathbb{N} \mid \text{es existiert ein } y \in \mathbb{N}, \text{ sodass } M_x \text{ bei Eingabe } y \text{ hält}\}$

**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.

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

Beachten Sie, dass die in der Definition von NAE-SAT genannte Belegung die Formel  $\varphi$  erfüllt. Z.B. gilt  $(x \vee y) \wedge (\neg x \vee \neg y) \in \text{NAE-SAT}$  (belege z.B.  $x$  mit 1 und  $y$  mit 0) und  $(x \vee y) \wedge (x \vee \neg y) \notin \text{NAE-SAT}$  (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  $T(n) = \begin{cases} d \in \Theta(1), & \text{falls } n \leq k \\ a \cdot T\left(\frac{n}{b}\right) + g(n), & \text{sonst} \end{cases}$  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  
 $a \cdot g\left(\frac{n}{b}\right) \leq c \cdot g(n)$  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)
    }
}
```

- 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].

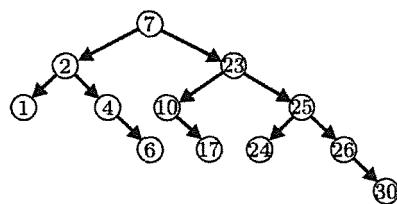
Sortieren Sie das Array mittels Quicksort *in-situ aufsteigend* von links nach rechts. Geben Sie die (Teil-)Arrays nach jeder Swap-Operation (auch wenn Elemente mit sich selber getauscht werden) und am Anfang jedes Aufrufs der rekursiven Methode an. Verwenden Sie als Pivotelement jeweils das *rechteste* Element im Teilarray und markieren Sie dieses entsprechend. Teilarrays der Länge  $\leq 2$  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 Worst-Case-Laufzeit benötigt (ohne Begründung).

### **Aufgabe 8:**

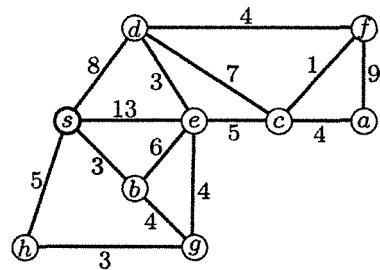
Bearbeiten Sie folgende Aufgaben zu AVL-Suchbäumen. Geben Sie jeweils bei jeder einzelnen Operation zum Einfügen, Löschen, sowie jeder elementaren Operation zum Wiederherstellen der AVL-Baumeigenschaften den entstehenden Baum als Baumzeichnung an. Geben Sie zur Darstellung der elementaren Operation auch vorübergehend ungültige AVL-Bäume an und stellen Sie Doppelrotationen in zwei Schritten dar. Dabei sollen die durchgeföhrten 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.



**Aufgabe 9:**

Gegeben sei folgender Graph  $G$ .



- 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  $(v, p, \delta)$  mit  $v$  als Knotenname,  $p$  als aktueller Vorgängerknoten und  $\delta$  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  $(v, p, \delta)$  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.
- Der Dijkstra-Algorithmus liefert bekanntlich auf Graphen mit negativen Kantengewichten unter Umständen ein falsches Ergebnis.
  - Geben Sie einen Graphen mit negativen Kantengewichten an, sodass der Dijkstra-Algorithmus ausgehend von einem von Ihnen ausgezeichneten Startknoten ein korrektes Ergebnis liefert.
  - 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

- 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	-

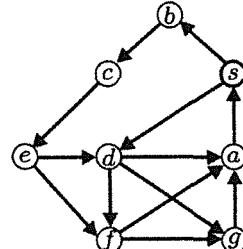
Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte denjenigen Knoten  $v$ , der vom Algorithmus als nächstes in den Ergebnisbaum aufgenommen wird (dieser sog. „schwarze“ Knoten ist damit fertiggestellt), als Tripel  $(v, p, \delta)$  mit  $v$  als Knotenname,  $p$  als aktueller Vorgängerknoten und  $\delta$  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  $(v, p, \delta)$  auf.

Zeichnen Sie anschließend den entstandenen Spannbaum und geben sein Gewicht an.

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

- Tiefensuche (DFS),
- 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.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2018**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	<b>Herbst 2018</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

---

**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: **12**

---

Bitte wenden!

**Thema Nr. 1**  
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1 (Aussagen)**

[24 PUNKTE]

Zeigen oder widerlegen Sie die folgenden Aussagen (die jeweiligen Beweise sind sehr kurz).

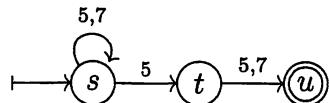
Schreiben Sie zunächst zur Aussage „Richtig“ oder „Falsch“ und dann Ihre Begründung.

- (a) [6 PUNKTE] Seien  $L, L' \subseteq \Sigma^*$  formale Sprachen. Falls  $L$  regulär und  $L'$  semi-entscheidbar (= rekursiv aufzählbar) ist, dann ist der Durchschnitt  $L \cap L'$  regulär.
- (b) [6 PUNKTE] Die Menge aller semi-entscheidbaren (= rekursiv-aufzählbaren) Sprachen ist überabzählbar unendlich.
- (c) [6 PUNKTE] Angenommen es gilt  $\mathcal{P} \neq \mathcal{NP}$ . Dann ist jede formale Sprache  $L \in \mathcal{NP} \setminus \mathcal{P}$  sicher nicht regulär.
- (d) [6 PUNKTE] Das Halteproblem liegt in der Klasse  $\mathcal{NP}$ .

**Aufgabe 2 (Reguläre Sprachen)**

[35 PUNKTE]

- (a) [10 PUNKTE] Es sei  $L \subseteq \{5, 7\}^*$  die von dem folgenden nichtdeterministischen Automaten akzeptierte Sprache. Geben Sie einen Automaten für das Komplement  $\{5, 7\}^* \setminus L$  der Sprache  $L$  an.



- (b) [15 PUNKTE] Minimieren Sie den deterministischen Automaten

$$A = (\{q, r, s, t, u, v\}, \{0, 1\}, \delta, q, \{q, s, u\})$$

mit der folgenden tabellarisch gegebenen Zustandsübergangsfunktion:

$\delta$	0	1
$q$	$s \quad v$	
$r$	$q \quad s$	
$s$	$q \quad v$	
$t$	$r \quad t$	
$u$	$r \quad q$	
$v$	$r \quad t$	

Machen Sie dabei Ihren Rechenweg deutlich.

Fortsetzung nächste Seite!

- (c) [10 PUNKTE] Beweisen Sie, dass die folgende formale Sprache über  $\Sigma = \{a, b\}$  nicht regulär ist:

$$L = \{b^n a^m \mid 2n \neq m\}$$

**Aufgabe 3 (Kontextfreie Sprachen)**

[25 PUNKTE]

- (a) [10 PUNKTE] Entwerfen Sie eine kontextfreie Grammatik für die folgende kontextfreie Sprache über dem Alphabet  $\Sigma = \{a, b, c\}$ :

$$L = \{wb^{3k}c^{2k+1}v \mid k \in \mathbb{N}, |w|_c = |v|_a\}.$$

(Hierbei bezeichnet  $|u|_x$  die Anzahl des Zeichens  $x$  in dem Wort  $u$ , und es gilt  $0 \in \mathbb{N}$ .) Erklären Sie den Zweck der einzelnen Nichtterminale (Variablen) und der Grammatikregeln Ihrer Grammatik.

- (b) [10 PUNKTE] Betrachten Sie die folgende kontextfreie Grammatik

$$G = (\{S, X, Y, Z\}, \{x, y\}, P, S)$$

mit den Produktionen

$$P : \quad S \rightarrow ZX \mid y$$

$$X \rightarrow ZS \mid SS \mid x$$

$$Y \rightarrow SX \mid YZ$$

$$Z \rightarrow XX \mid XS$$

Benutzen Sie den Algorithmus von Cocke-Younger-Kasami (CYK) um zu zeigen, dass das Wort  $xxxxx$  zu der von  $G$  erzeugten Sprache  $L(G)$  gehört.

- (c) [5 PUNKTE] Geben Sie eine Ableitung des Wortes  $xxxxx$  mit  $G$  an.

**Aufgabe 4 (Entscheidbarkeit)**

[36 PUNKTE]

- (a) [8 PUNKTE] Benutzen Sie den Satz von Rice um zu beweisen, dass das folgende Problem nicht entscheidbar ist:

**Eingabe:** eine (geeignet codierte) Turingmaschine  $M$ , die eine (möglicherweise partielle) Funktion  $f_M : \mathbb{N} \rightarrow \mathbb{N}$  berechnet;

**Aufgabe:** entscheiden, ob es ein  $n \in \mathbb{N}$  mit  $f_M(n) = 42$  gibt.

- (b) [8 PUNKTE] Zeigen Sie, dass das Problem aus (a) semi-entscheidbar (= rekursiv-aufzählbar) ist.

- (c) [20 PUNKTE] Zeigen Sie mit Hilfe einer Reduktion, dass das folgende Problem nicht entscheidbar ist:

**Eingabe:** zwei (geeignet codierte) Turingmaschinen  $M_1$  und  $M_2$ ;

**Aufgabe:** entscheiden, ob für jedes Eingabewort  $w$   $M_1$  gestartet auf  $w$  genau dann hält, wenn  $M_2$  gestartet auf  $w$  hält.

(Für die Reduktion kann benutzt werden, dass das Problem zu entscheiden, ob eine gegebene Turingmaschine auf jedes Eingabewort hält, unentscheidbar ist.)

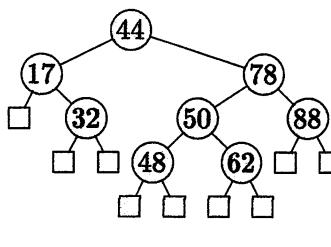
### Aufgabe 5 (AVL Bäume)

[54 PUNKTE]

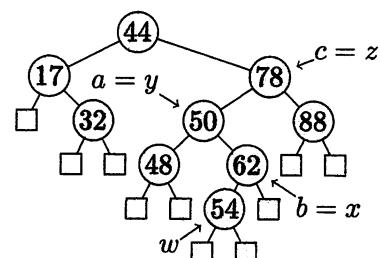
Hinweis: Wir betrachten in dieser Aufgabe binäre Suchbäume, bei denen jeder innere Knoten genau zwei Kinder hat. Schlüssel werden nur in den inneren Knoten gespeichert - die Blätter speichern keinerlei Informationen.

- 5.1 [2 PUNKTE] Welche Eigenschaften muss ein binärer Suchbaum haben, damit er ein AVL-Baum ist?
- 5.2 [8 PUNKTE] Mit  $n(h)$  bezeichnen wir die *minimale* Anzahl innerer Knoten eines AVL-Baums der Höhe  $h$ .
  - (a) [2 PUNKTE] Begründen Sie, dass  $n(1) = 1$  und  $n(2) = 2$ .
  - (b) [3 PUNKTE] Begründen Sie, dass  $n(h) = 1 + n(h-1) + n(h-2)$ .
  - (c) [3 PUNKTE] Folgern Sie, dass  $n(h) > 2^{\frac{h}{2}-1}$ .
- 5.3 [2 PUNKTE] Warum ist die Höhe jedes AVL-Baums mit  $n$  inneren Knoten  $O(\log n)$ ?
- 5.4 [16 PUNKTE] Fügen Sie die Elemente  $(45, 16, 79, 31, 51, 87, 49, 61)$  in der angegebenen Reihenfolge in einen anfangs leeren binären Suchbaum ein (*ohne Rebalancierungen*). Zeichnen Sie den resultierenden Suchbaum nach jeder Einfügeoperation.
- 5.5 [2 PUNKTE] Ist der resultierende Suchbaum aus Teilaufgabe 5.4 ein AVL-Baum? Begründen Sie Ihre Antwort.
- 5.6 [12 PUNKTE] Das Einfügen in einen AVL-Baum funktioniert (zunächst) wie beim binären Suchbaum durch Erweitern eines äußeren Knotens  $w$ :

vor dem Einfügen von 54



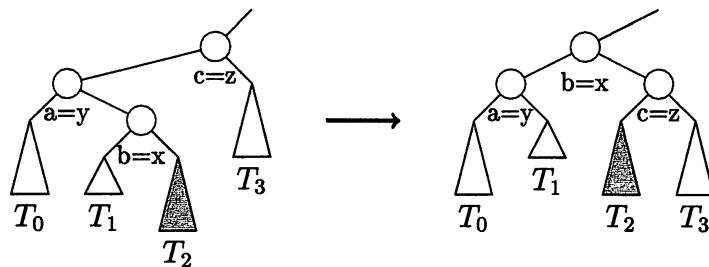
nach dem Einfügen von 54



Fortsetzung nächste Seite!

Anschließend wird die AVL-Baum Eigenschaft (falls notwendig) durch eine (Doppel-)Rotation wiederhergestellt: Wenn  $z$  der erste Knoten auf dem Pfad  $P$  von  $w$  zur Wurzel ist, der nicht balanciert ist,  $y$  das Kind von  $z$  auf  $P$  und  $x$  das Kind von  $y$  auf  $P$ , und wenn  $(a, b, c)$  die Inorder-Reihenfolge von  $x, y, z$  ist, dann führen wir die Rotation aus, die benötigt wird, um  $b$  zum obersten Knoten der drei zu machen.

Die folgende Illustration zeigt den Fall, dass  $\text{key}(y) < \text{key}(x) < \text{key}(z)$ , d.h.  $(a, b, c) = (y, x, z)$ , wobei  $w$  ein Knoten in  $T_2$  ist.



Sei  $h$  die Höhe des Teilbaums  $T_3$ . Für  $i = 0, 1, 2$  sei  $h_i$  die Höhe des Teilbaums  $T_i$  und für  $v = x, y, z$  sei  $h_v$  die Höhe des Teilbaums mit der Wurzel  $v$  vor der Restrukturierung. Begründen Sie, dass

- (a) [2 PUNKTE]  $h_0 = h$
  - (b) [2 PUNKTE]  $h_1 = h - 1$
  - (c) [2 PUNKTE]  $h_2 = h$
  - (d) [2 PUNKTE]  $h_x = h + 1$
  - (e) [2 PUNKTE]  $h_y = h + 2$
  - (f) [2 PUNKTE]  $h_z = h + 3$
- 5.7 [3 PUNKTE] Welche Höhe haben die Teilbäume mit den Wurzeln  $x, y, z$  nach der Restrukturierung? Begründen Sie Ihre Antworten.
- 5.8 [4 PUNKTE] Begründen Sie, dass die oben gezeigte Doppelrotation die AVL-Baum-Eigenschaft wiederherstellt.
- 5.9 [3 PUNKTE] Beschreiben Sie, wie ein binärer Baum der Höhe  $h$  in einem Array repräsentiert werden kann. Wie viel Speicherplatz ist für so eine Darstellung erforderlich?
- 5.10 [2 PUNKTE] Warum verwendet man bei der Implementierung von AVL-Bäumen eine verzweigte Struktur und nicht eine Array-basierte Repräsentation?

**Aufgabe 6 (Ein Spiel mit Zahlen)**

[51 PUNKTE]

Wir spielen folgendes Spiel: Gegeben sei eine Folge  $S = (x_1, \dots, x_n)$  von  $n$  Zahlen. In einer *Runde* dürfen wir in  $S$  zwei beliebige *benachbarte* Zahlen  $x_k, x_{k+1}$  entfernen und durch die Zahl  $2 \cdot (x_k + x_{k+1})$  ersetzen, ohne die Reihenfolge der anderen Zahlen in  $S$  zu verändern. Wir erhalten so eine neue Folge  $S' = (x'_1, \dots, x'_{n-1})$  von  $n - 1$  Zahlen, wobei  $x'_i = x_i$  für  $1 \leq i < k$ ,  $x'_k = 2 \cdot (x_k + x_{k+1})$ , und  $x'_i = x_{i+1}$  für  $k < i < n$ . In der nächsten Runde spielen wir mit  $S'$  weiter. Nach  $n - 1$  Runden erhalten wir eine einelementige Folge  $S^* = (x_1^*)$ . Die Zahl  $x_1^*$  ist unser *Gewinn*.

Ein mögliches Spiel auf der Folge  $(2, 1, 3, 4, 1, 2, 3)$  sieht so aus:

$$(2, \underline{1}, 3, 4, 1, 2, 3) \rightarrow (\underline{6}, 3, 4, \underline{1}, 2, 3) \rightarrow (\underline{6}, \underline{3}, \underline{4}, 6, 3) \rightarrow \\ (\underline{6}, \underline{14}, \underline{6}, \underline{3}) \rightarrow (\underline{6}, \underline{14}, 18) \rightarrow (\underline{40}, 18) \rightarrow (116).$$

Ein Spiel können wir auch als eine *Klammerung* der Elemente von  $S$  interpretieren: Anstatt in einer Runde  $x_k, x_{k+1}$  zu entfernen und durch  $2 \cdot (x_k + x_{k+1})$  zu ersetzen, ersetzen wir die beiden Zahlen durch den geklammerten Ausdruck  $(x_k * x_{k+1})$ . In unserem Beispiel sieht das so aus:

$$(2, \underline{1}, 3, 4, 1, 2, 3) \rightarrow ((2 * 1), 3, 4, \underline{1}, 2, 3) \rightarrow ((2 * 1), \underline{3}, \underline{4}, (1 * 2), 3) \rightarrow \\ ((2 * 1), (3 * 4), \underline{(1 * 2) * 3}) \rightarrow ((\underline{2} * 1), (3 * 4), ((1 * 2) * 3)) \rightarrow \\ (((2 * 1) * (3 * 4)), ((1 * 2) * 3)) \rightarrow \\ (((((2 * 1) * (3 * 4)) * ((1 * 2) * 3))).$$

Nach  $n - 1$  Runden entsteht ein Klammerausdruck  $K$  auf den Elementen von  $S$ . Wenn wir den Operator  $*$  definieren als  $x * y := 2 \cdot (x + y)$ , dann ist der Wert  $w(K)$  dieses letzten Klammerausdrucks gerade der Gewinn des Spiels:  $w(((2 * 1) * (3 * 4)) * ((1 * 2) * 3))) = 116$ .

Einen Klammerausdruck der den *größten* Gewinn erzielt nennen wir *optimal*. Für  $1 \leq i \leq j \leq n$  betrachten wir die Teilfolge  $S[i, j] := (x_i, \dots, x_j)$  von  $S$ . Mit  $W(i, j)$  bezeichnen wir den Wert eines optimalen Klammerausdrucks für  $S[i, j]$ .

- 6.1 [5 PUNKTE] Angenommen  $K$  ist ein *optimaler* Klammerausdruck für  $S[i, j]$ , dessen *letzte*  $*$ -Operation zwischen  $x_l$  und  $x_{l+1}$  liegt, d.h.  $K = L * R$ , wobei  $L$  ein Klammerausdruck für die Teilfolge  $S[i, l]$  und  $R$  ein Klammerausdruck für die Teilfolge  $S[l + 1, j]$  ist.

- (a) [1 PUNKT] Warum ist  $w(K) = 2 \cdot (w(L) + w(R))$ ?
- (b) [4 PUNKTE] Begründen Sie, dass  $L$  und  $R$  *optimale* Klammerausdrücke für  $S[i, l]$  bzw.  $S[l + 1, j]$  sind.

- 6.2 [7 PUNKTE] Begründen Sie, dass

- (a) [1 PUNKT]  $W(i, i) = x_i$  für  $1 \leq i \leq n$
- (b) [6 Punkte] und dass für  $1 \leq i < j \leq n$  folgende Rekursionsgleichung gilt:

$$W(i, j) = \max_{i \leq l < j} \{2 \cdot (W(i, l) + W(l + 1, j))\}$$

- 6.3 [6 PUNKTE] Formulieren Sie auf Basis obiger Rekursionsgleichung einen *rekursiven* Algorithmus  $\text{WINGAMEREC}(i, j)$  zur Berechnung von  $W(i, j)$  in Pseudocode. Die Laufzeit  $T(n)$  Ihres Algorithmus beim Aufruf  $\text{WINGAMEREC}(1, n)$  sollte folgender asymptotischer Rekursionsungleichung genügen:

$$T(n) = \sum_{1 \leq l < n} (T(l) + T(n - l) + \Theta(1)), \text{ für } n > 1$$

(mit  $T(1) = \Theta(1)$ )

Begründen Sie, dass dem so ist.

- 6.4 [6 PUNKTE] Finden Sie eine (möglichst kurze) Eingabefolge  $S$  an der deutlich wird, dass sich die Teilprobleme der Rekursion zur Berechnung von  $W(1, n)$  überlappen. Illustrieren Sie das Phänomen, indem Sie den Rekursionsbaum zeichnen und die mehrfach berechneten Teilprobleme markieren.

- 6.5 [3 PUNKTE] Begründen Sie, dass  $T(n) = 2T(n - 1) + \Omega(1)$  für  $n > 1$ .

- 6.6 [4 PUNKTE] Begründen Sie, dass  $T(n) = \Omega(2^n)$ .

- 6.7 [4 PUNKTE] Wie sieht eine Reihenfolge aus, in der die  $W(i, j)$  bottom-up berechnet werden können, die mit obiger Rekursionsgleichung kompatibel ist? Begründen Sie Ihre Antwort.

- 6.8 [6 PUNKTE] Geben Sie ein *dynamisches Programm*  $\text{WINGAMEDP}(S)$  in Pseudocode an, das den Wert  $W(1, n)$  in *polynomieller Zeit* berechnet.

*Hinweis:* Wenn Sie die vorige Aufgabe nicht lösen konnten, verwenden Sie die lexikographische Reihenfolge zur bottom-up Berechnung der  $W(i, j)$ .

- 6.9 [6 PUNKTE] Begründen Sie, warum Ihr Algorithmus in Teilaufgabe 6.8 korrekt ist.

- 6.10 [4 PUNKTE] Bestimmen Sie die asymptotische Laufzeit und den Speicherplatzbedarf Ihres Algorithmus aus Teilaufgabe 6.8 möglichst genau.

**Thema Nr. 2**  
**(Aufgabengruppe)**

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1 (Reguläre Sprachen)****[22 PUNKTE]**

- (a) [6 PUNKTE] Geben Sie alle Nerode-Äquivalenzklassen von  $L((aa)^*)$  über dem Alphabet  $\{a, b\}$  an. Geben Sie für jede Klasse 3 Wörter an, die zu dieser gehören.
- (b) [6 PUNKTE] Geben Sie einen DEA mit maximal 4 Zuständen an, zu dem es keinen äquivalenten DEA mit höchstens einem akzeptierenden Zustand gibt. Verwenden Sie ein Alphabet mit höchstens zwei Symbolen. Warum erfüllt Ihr DEA diese Eigenschaft?
- (c) [10 PUNKTE] Ist  $L_1 = \{a^i b^j c^\ell \mid i + j = \ell\} \cup L((a + c)^*) \cup L((b + c)^*)$  eine reguläre Sprache? Geben Sie einen Beweis für Ihre Antwort. Um Regularität zu zeigen reicht die Angabe eines Automaten oder regulären Ausdrucks, der die Sprache akzeptiert.

**Aufgabe 2 (Kontextfreie Sprachen)****[22 PUNKTE]**

- (a) [6 PUNKTE] Entscheiden Sie mit Hilfe des CYK-Algorithmus, ob das Wort *acbaabaa* in der Sprache der nachfolgenden Grammatik enthalten ist.

$$S \rightarrow c \mid BA$$

$$A \rightarrow a \mid b \mid CB$$

$$B \rightarrow AA \mid AS$$

$$C \rightarrow SA$$

- (b) [4 PUNKTE] Seien  $L_1$  und  $L_2$  kontextfreie Sprachen. Ist  $L_1 \cup L_2$  eine entscheidbare Sprache? Ist  $L_1 \cap L_2$  eine entscheidbare Sprache? Begründen Sie Ihre Antworten.
- (c) [6 PUNKTE] Sei  $L_3$  die Komplementsprache von  $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$  über Alphabet  $\{a, b, c\}$ , d.h.,  $L_3 = \{w \in \{a, b, c\}^* \mid w \notin L\}$ . Ist  $L_3$  kontextfrei? Geben Sie einen Beweis für Ihre Antwort. Um Kontextfreiheit zu zeigen reicht die Angabe eines Automaten oder einer Grammatik, die die Sprache akzeptiert. Bitte erläutern Sie dann Ihre Konstruktionsidee. Sie müssen nicht formal beweisen, dass Ihr Automat oder Ihre Grammatik korrekt ist.
- (d) [6 PUNKTE] Sei  $L_4$  die Sprache  $\{a^{n^2} \mid n \in \mathbb{N}\}$  über Alphabet  $\{a\}$ . Ist  $L_4$  kontextfrei? Geben Sie einen Beweis für Ihre Antwort. Um Kontextfreiheit zu zeigen reicht die Angabe eines Automaten oder einer Grammatik, die die Sprache akzeptiert. Bitte erläutern Sie dann Ihre Konstruktionsidee. Sie müssen nicht formal beweisen, dass Ihr Automat oder Ihre Grammatik korrekt ist.

**Aufgabe 3 (Entscheidbarkeit)**

[16 PUNKTE]

Wir bezeichnen mit  $M_w$  die Turingmaschine, die von einem Wort  $w$  kodiert wird.

(a) [8 PUNKTE] Ist die Sprache  $\{w \mid M_w \text{ hält auf jeder Eingabe nach maximal } 42 \text{ Schritten an}\}$  entscheidbar?

(b) [8 PUNKTE] Ist die Sprache  $\{w \mid M_w \text{ hält auf Eingabe } \varepsilon \text{ nach einer geraden Anzahl von Schritten an}\}$  entscheidbar?

Beweisen Sie Ihre Antwort.

**Aufgabe 4 (Komplexität)**

[10 PUNKTE]

Wir betrachten ungerichtete Graphen  $G = (V, E)$ , wo  $E$  eine Teilmenge  $E_e$  hat, die wir *exklusive Kanten* nennen. Eine *beschränkte Überdeckung* von  $G$  ist eine Teilmenge  $U$  von  $V$ , so dass

- jeder Knoten einen Nachbarknoten in  $U$  hat oder selbst in  $U$  liegt (für jeden Knoten  $u \in V \setminus U$  gibt es einen Knoten  $v \in U$  mit  $(u, v) \in E$ ) und
- für jede exklusive Kante  $(u, v) \in E_e$  genau einer der Knoten  $u, v$  in  $U$  liegt.

Betrachten Sie nun die folgenden Entscheidungsprobleme:

## 3SAT

Gegeben: Aussagenlogische Formel  $\varphi$  in 3KNF

Gefragt: Hat  $\varphi$  eine erfüllende Belegung?

## BÜ

Gegeben: Graph  $G = (V, E)$ , exklusive Kantenmenge  $E_e \subseteq E$  und  $k \in \mathbb{N}$

Gefragt: Hat  $G$  eine *beschränkte Überdeckung*  $U$  mit  $|U| \leq k$ ?

Beweisen Sie, dass BÜ NP-vollständig ist. Sie dürfen dabei annehmen, dass 3SAT NP-vollständig ist.

**Aufgabe 5 (Studienzeitoptimierung)**

[45 PUNKTE]

Für den Bachelorstudiengang Informatik an Ihrer Hochschule werden die Module als Knoten eines gerichteten Graphen  $G = (V, E)$  modelliert. Es führt eine Kante  $(u, w)$  von Modul  $u$  zu Modul  $w$  genau dann, wenn der (erfolgreiche) Abschluss von Modul  $u$  notwendige Voraussetzung für die Teilnahme an Modul  $w$  ist.

Der Einfachheit halber machen wir zwei Annahmen:

Fortsetzung nächste Seite!

1. Jedes Modul dauert genau ein Semester.
2. In jedem Semester werden alle Module des Studiengangs angeboten.

Wenn  $k$  die Länge des *längsten Pfades* in  $G$  ist, dann ist die Mindeststudiendauer  $k + 1$  Semester. Ihre Aufgabe ist es nun, den längsten Pfad in  $G$  zu berechnen und dafür einen möglichst effizienten Algorithmus zu entwickeln.

Zunächst folgen einige Tipps, dann drei Fragen als Vorüberlegung. Anschließend geht es zur Implementierung.

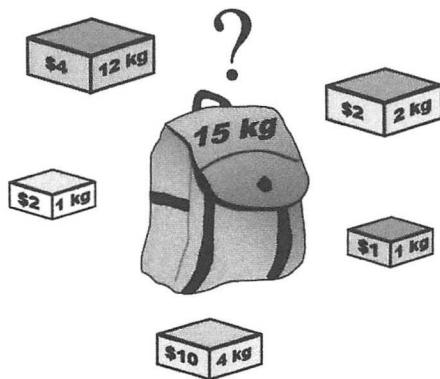
Tipps:

1. Der Graph ist zyklusfrei.
  2. Der Graph ist kein gerichteter Wald, d.h. es kann Knoten geben, die mehrere Vorgänger haben.
- (a) [5 PUNKTE] Von welchen Knoten aus sollte die Suche nach dem längsten Pfad beginnen? Begründen Sie Ihre Antwort.
- (b) [5 PUNKTE] Beschreiben Sie, wie man *induktiv* für einen beliebigen Knoten  $v$  die Länge des längsten Teilpfades bis zu diesem Knoten aus der maximalen Pfadlänge zu allen Vorgängerknoten berechnen kann. Beschreiben Sie die Umstände der *Induktionsverankerung*.
- (c) [5 PUNKTE] Welches ist eine sinnvolle Reihenfolge, in der die Knoten von  $G$  bei der Durchführung des Algorithmus abgearbeitet werden? Begründen Sie Ihre Antwort.
- (d) [25 PUNKTE] Formulieren Sie einen Algorithmus, der Ihnen die minimale Studiendauer berechnet. Geben Sie zudem die asymptotische Laufzeit Ihres Algorithmus mittels O-Notation an.
- (e) [5 PUNKTE] Wie ändert sich die Problemstellung, wenn es Module gibt, die nur im Sommer- bzw. nur im Wintersemester angeboten werden? Müssen Sie den Algorithmus dafür abändern? Begründen Sie Ihre Antwort. (Annahme: Das Studium kann sowohl zum Sommersemester als auch zum Wintersemester aufgenommen werden.)

### Aufgabe 6 (Backtracking)

[30 PUNKTE]

Ein sehr bekanntes Optimierungsproblem ist das sogenannte Rucksackproblem: Gegeben ist ein Rucksack mit der Tragfähigkeit  $B$ . Weiterhin ist eine endliche Menge von Gegenständen mit Werten und Gewichten gegeben. Nun soll eine Teilmenge der Gegenstände so ausgewählt werden, dass ihr Gesamtwert maximal ist, aber ihr Gesamtgewicht die Tragfähigkeit des Rucksacks nicht überschreitet.



Mathematisch exakt kann das Rucksackproblem wie folgt formuliert werden:

Gegeben ist eine endliche Menge von Objekten  $U$ . Durch eine Gewichtsfunktion  $w : U \rightarrow \mathbb{R}^+$  wird den Objekten ein Gewicht und durch eine Nutzenfunktion  $v : U \rightarrow \mathbb{R}^+$  ein festgelegter Nutzwert zugeordnet.

Des Weiteren gibt es eine vorgegebene Gewichtsschranke  $B \in \mathbb{R}^+$ . Gesucht ist eine Teilmenge  $K \subseteq U$ , die die Bedingung  $\sum_{u \in K} w(u) \leq B$  einhält und die Zielfunktion  $\sum_{u \in K} v(u)$  maximiert.

Das Rucksackproblem ist NP-vollständig (Problemgröße ist die Anzahl der Objekte), sodass es an dieser Stelle wenig Sinn macht, über eine effiziente Lösung nachzudenken. Lösen Sie das Rucksackproblem daher mittels *Backtracking* und formulieren Sie einen entsprechenden Algorithmus. Gehen Sie davon aus, dass die Gewichtsschranke  $B$  sowie die Anzahl an Objekten  $N$  beliebig, aber fest vorgegeben sind.

Das Programm soll folgende Ausgaben liefern:

1. Maximaler Nutzwert, der durch eine Objektauswahl unter Einhaltung der Gewichtsschranke  $B$  erreicht werden kann.
2. Das durch die maximierende Objektmenge erreichte Gesamtgewicht.
3. Diejenigen Objekte (Objektnummern) aus  $U$ , die zur Maximierung des Nutzwerts beigetragen haben.

### Aufgabe 7 (O-Nation)

[20 PUNKTE]

- (a) Beweisen Sie die folgenden Aussagen formal nach den Definitionen der O-Notation oder widerlegen Sie sie.
- i. [5 PUNKTE]  $O(n \cdot \log_2 n) \subseteq O(n \cdot (\log_2 n)^2)$
  - ii. [5 PUNKTE]  $2^{n+1} \in O(2^n)$
- (b) Bestimmen Sie eine asymptotische Lösung (in  $\Theta$ -Schreibweise) für die folgende Rekursionsgleichung:

$$[10 \text{ PUNKTE}] T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2$$

Fortsetzung nächste Seite!

**Aufgabe 8 (Sortierverfahren)****[25 PUNKTE]**

Gegeben sei das folgende Feld  $A$  mit 7 Schlüsseln:

$$[15, 4, 10, 7, 1, 8, 10]$$

- (a) [10 PUNKTE] Sortieren Sie das Feld mittels des Sortierverfahrens *Bubblesort*. Markieren Sie jeweils, welche zwei Feldwerte verglichen werden und geben Sie den Zustand des gesamten Feldes jeweils neu an, wenn Sie eine Vertauschung durchgeführt haben.
- (b) [10 PUNKTE] Sortieren Sie das Feld mittels des Sortierverfahrens *Selectionsort*. Markieren Sie jeweils, welche zwei Feldwerte verglichen werden und geben Sie den Zustand des gesamten Feldes jeweils neu an, wenn Sie eine Vertauschung durchgeführt haben.
- (c) [5 PUNKTE] Vergleichen Sie beide Sortierverfahren hinsichtlich ihres Laufzeitverhaltens im *best case*. Welches Verfahren ist in dieser Hinsicht besser, wenn das zu sortierende Feld anfangs bereits sortiert ist? Begründen Sie Ihre Antwort.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Frühjahr 2019**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	Frühjahr	
Kennwort: _____	2019	66115
Arbeitsplatz-Nr.: _____		

**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: 8

Bitte wenden!

**Thema Nr. 1**  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

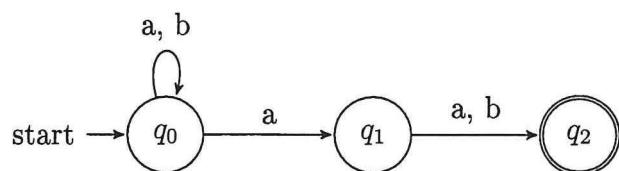
### Aufgabe 1

Antworten Sie auf die folgenden Behauptungen mit Wahr/Falsch und geben Sie eine kurze Begründung an.

- (a) Wenn  $L_2$  regulär ist und  $L_1 \subseteq L_2$  gilt, dann ist  $L_1$  auch regulär.
- (b)  $Q = \{a^q \mid \exists i \in \mathbb{N}. q = i^2\}$  ist bekanntlich nicht regulär.  
 Behauptung:  $Q^*$  ist ebenfalls nicht regulär.
- (c) Wenn  $L \subseteq \Sigma^*$  entscheidbar ist, dann ist auch das Komplement  $\bar{L} = \Sigma^* \setminus L$  entscheidbar.
- (d) Jedes  $\mathcal{NP}$ -vollständige Problem ist entscheidbar.

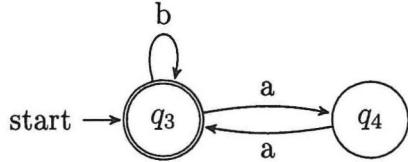
### Aufgabe 2

- (a) Gegeben sei der nichtdeterministische endliche Automat  $A$  über dem Alphabet  $\Sigma = \{a, b\}$  wie folgt:



Konstruieren Sie einen deterministischen endlichen Automaten, der das Komplement  $\overline{L(A)} = \{w \in \Sigma^* \mid w \notin L(A)\}$  der von  $A$  akzeptierten Sprache  $L(A)$  akzeptiert.

- (b) Gegeben sei zudem der nichtdeterministische Automat  $B$  über dem Alphabet  $\Sigma = \{a, b\}$ :



Konstruieren Sie einen endlichen Automaten (möglicherweise mit  $\epsilon$ -Übergängen), der die Sprache  $(L(A)L(B))^* \subseteq \Sigma^*$  akzeptiert ( $A$  aus der vorigen Aufgabe). Erläutern Sie auch Ihre Konstruktionsidee.

### Aufgabe 3

Gegeben sei die kontextfreie Grammatik  $G = (V, \Sigma, P, S)$  mit Sprache  $L(G)$ , wobei  $V = \{S, T, U\}$  und  $\Sigma = \{a, b\}$ .  $P$  bestehe aus den folgenden Produktionen:

$$S \rightarrow TUUT$$

$$T \rightarrow aT \mid \epsilon$$

$$U \rightarrow bUb \mid a$$

- (a) Geben Sie fünf verschiedene Wörter  $w \in \Sigma^*$  mit  $w \in L(G)$  an.

- (b) Geben Sie eine explizite Beschreibung der Sprache  $L(G)$  an.

- (c) Bringen Sie  $G$  in Chomsky-Normalform und erklären Sie Ihre Vorgehensweise.

### Aufgabe 4

Wir betrachten die Turingmaschine  $M = (Q, q_0, F, \Sigma, \Gamma, \square, \delta)$ . Hierbei ist die Zustandsmenge  $Q = \{q_0, q_a, q_b, q_L, q_f\}$  mit Startzustand  $q_0$  und akzeptierenden Zuständen  $F = \{q_f\}$ . Das Eingabealphabet ist  $\Sigma = \{a, b, \$\}$ , das Bandalphabet ist  $\Gamma = \Sigma \cup \{\square\}$  mit Blank-Zeichen  $\square$  für leeres Feld. Die Übergangsfunktion  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, N, R\}$ , wobei der Schreib-Lese-Kopf mit  $L$  nach links, mit  $N$  nicht und mit  $R$  nach rechts bewegt wird, ist durch folgende Tabelle gegeben (bspw. ist  $\delta(q_0, a) = (q_a, \square, R)$ ):

	$a$	$b$	$\$$	$\square$
$q_0$	$(q_a, \square, R)$	$(q_b, \square, R)$	$(q_f, \$, N)$	$(q_f, \square, N)$
$q_a$	$(q_a, a, R)$	$(q_a, b, R)$	$(q_a, \$, R)$	$(q_L, a, L)$
$q_b$	$(q_b, a, R)$	$(q_b, b, R)$	$(q_b, \$, R)$	$(q_L, b, L)$
$q_L$	$(q_L, a, L)$	$(q_L, b, L)$	$(q_L, \$, L)$	$(q_0, \square, R)$

- (a) Die Notation  $(v, q, aw)$  beschreibt eine Konfiguration der Turingmaschine: der interne Zustand ist  $q$ , der Schreib-Lesekopf steht auf einem Feld mit  $a \in \Gamma$ , rechts vom Schreib-Lesekopf steht  $w \in \Gamma^*$ , links vom Schreib-Lesekopf steht  $v \in \Gamma^*$ .

Vervollständigen Sie die Folge von Konfigurationen, die die Turingmaschine bei Eingabe  $ab\$\$$  bis zum Erreichen des Zustands  $q_f$  durchläuft. Sie können auch Ihre eigene Notation zur Darstellung von Konfigurationen verwenden.

$$\begin{aligned} (\varepsilon, q_0, ab\$) &\longrightarrow \\ (\varepsilon, q_a, b\$) &\longrightarrow \\ (b, q_a, \$) &\longrightarrow \\ \dots \end{aligned}$$

- (b) Sei  $w \in \{a, b\}^*$  beliebig. Mit welchem Bandinhalt terminiert die Turingmaschine bei Eingabe von  $w\$$ ? Geben Sie auch eine kurze Begründung an.

## Aufgabe 5

Wir betrachten das Behälterproblem BEHAELTER. Gegeben ist eine Menge von  $k \in \mathbb{N}$  Behältern, die jeweils ein Fassungsvermögen der Größe  $b \in \mathbb{N}$  haben. Gegeben sind weiterhin  $n$  Objekte mit jeweiligen Größen  $a_1, \dots, a_n$ . Gesucht ist eine Zuordnung der  $n$  Objekte auf die  $k$  Behälter, sodass keiner der Behälter überläuft.

Formal sind Instanzen des Behälterproblems BEHAELTER durch Tupel  $(k, b, a_1, \dots, a_n)$  gegeben, die wie folgt zu interpretieren sind:

- $k \in \mathbb{N}$  steht für eine Anzahl von Behältern.
- Jeder Behälter hat ein Fassungsvermögen von  $b \in \mathbb{N}$ .
- Die  $a_i$  stehen für die jeweiligen Größen von  $n$  Objekten.

Zuordnungen von Objekten zu Behältern geben wir durch eine Funktion  $v$  an, wobei  $v(j) = i$  wenn das  $j$ -te Objekt (mit Größe  $a_j$ ) dem  $i$ -ten Behälter zugeordnet wird.

$(k, b, a_1, \dots, a_n)$  ist eine JA-Instanz von BEHAELTER, wenn es eine Zuordnung  $v$  von Objekten auf Behälter ( $v : [1; n] \rightarrow [1; k]$ ) gibt, die sicherstellt, dass kein Behälter überläuft:

$$(k, b, a_1, \dots, a_n) \in \text{BEHAELTER} \iff (\exists v : [1; n] \rightarrow [1; k]. \forall i \leq k. \sum_{j=v(i)} a_j \leq b)$$

Wir betrachten auch das modifizierte Problem GERADEBEHAELTER. Instanzen von GERADEBEHAELTER tragen die zusätzliche Einschränkung, dass alle  $a_j$  gerade (durch zwei teilbar) sein müssen.

- (a) Warum ist sowohl BEHAELTER  $\in \mathcal{NP}$  als auch GERADEBEHAELTER  $\in \mathcal{NP}$ ?
- (b) Beweisen Sie, dass das Problem BEHAELTER auf das Problem GERADEBEHAELTER in polynomieller Zeit reduzierbar ist.
- (c) BEHAELTER ist  $\mathcal{NP}$ -vollständig. Begründen Sie, was obige Reduktion für die Komplexität von GERADEBEHAELTER bedeutet.

**Aufgabe 6** (Algorithmen und Datenstrukturen)

Aus dem Känguru-Wettbewerb 2017 – Klassenstufen 3 und 4.

Luna hat für den Kuchenbasar Muffins mitgebracht: 10 Apfelmuffins, 18 Nussmuffins, 12 Schokomuffins und 9 Blaubeermuffins. Sie nimmt immer 3 verschiedene Muffins und legt sie auf einen Teller. Welches ist die kleinste Zahl von Muffins, die dabei übrig bleiben können?

A: 1, B: 3, C: 4, D: 7, E: 8

- (a) Geben Sie die richtige Antwort auf die im Känguru-Wettbewerb gestellte Frage und begründen Sie sie.
- (b) Lunas Freundin empfiehlt den jeweils nächsten Teller immer aus den drei aktuell häufigsten Muffinsorten zusammenzustellen. Leiten Sie aus dieser Idee einen effizienten Greedy-Algorithmus her, der die Fragestellung für beliebige Anzahlen von Muffins löst (nach wie vor soll es nur vier Sorten und je drei pro Teller geben). Skizzieren Sie in geeigneter Form, wie Ihr Algorithmus die Beispielinstantz von oben richtig löst.
- (c) Beschreiben Sie eine mögliche und sinnvolle Verallgemeinerung Ihrer Lösung auf  $n$  Muffinsorten und  $k$  Muffins pro Teller für  $n > 4$  und  $k > 3$ .
- (d) Diskutieren Sie, wie man die Korrektheit des Greedy-Algorithmus zeigen könnte, also dass er tatsächlich immer eine optimale Lösung findet. Ein kompletter, rigoroser Beweis ist nicht verlangt.

**Thema Nr. 2**  
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

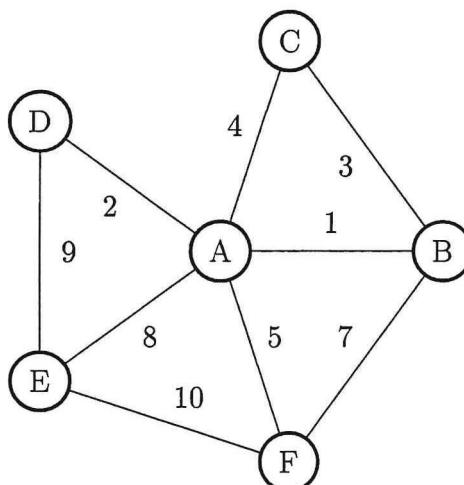
**Aufgabe 1 ( $k$ -kleinste Elemente)**

Gegeben sei eine unsortierte Liste von  $n$  verschiedenen natürlichen Zahlen. Das  $k$ -kleinste Element ist das Element, das größer als genau  $k - 1$  Elementen der Liste ist.

- (a) Geben Sie einen Algorithmus mit Laufzeit  $\mathcal{O}(n \log n)$  an, um das  $k$ -kleinste Element zu berechnen.
- (b) Gegeben sei nun ein Algorithmus  $\mathcal{A}$ , der den Median einer unsortierten Liste von  $n$  Zahlen in  $\mathcal{O}(n)$  Schritten berechnet. Nutzen Sie Algorithmus  $\mathcal{A}$  um einen Algorithmus  $\mathcal{B}$  anzugeben, welcher das  $k$ -kleinste Element in  $\mathcal{O}(n)$  Schritten berechnet.  
Argumentieren Sie auch, dass der Algorithmus die gewünschte Laufzeit besitzt.
- (c) Geben Sie einen Algorithmus an, der für alle  $i = 1, \dots, \lfloor n/k \rfloor$  das  $i \cdot k$ -kleinste Element berechnet. Die Laufzeit Ihres Algorithmus sollte  $\mathcal{O}(n \log(n/k))$  sein. Sie dürfen weiterhin Algorithmus  $\mathcal{A}$ , wie in Teilaufgabe (b) beschrieben, nutzen.

**Aufgabe 2 (Spannbäume)**

- (a) Berechnen Sie zu dem angegebenen Graphen einen minimalen Spannbaum. Nutzen Sie den Algorithmus von Kruskal. Geben Sie tabellarisch mit jedem Schritt des Algorithmus an, welche Kanten dem Baum bereits hinzugefügt wurden.



- (b) Der *Durchmesser* eines Spannbaums ist die Länge des längsten einfachen Pfads im Baum. Geben Sie ein möglichst effizientes Verfahren an, den minimalen Spannbaum mit Durchmesser 2 zu bestimmen, falls ein solcher existiert. Analysieren Sie die asymptotische worst-case Laufzeit Ihres Algorithmus in Abhängigkeit der Kanten- und Knotenzahl.
- (c) Geben Sie nun ein effizientes Verfahren an, den minimalen Spannbaum mit Durchmesser 3 zu bestimmen, falls ein solcher existiert. Analysieren Sie die asymptotische worst-case Laufzeit Ihres Algorithmus.

### Aufgabe 3 (Rotierende Bäume)

- (a) Ein Binärbaum ist eine rechtslineare Kette, falls kein Knoten im Baum ein linkes Kind besitzt.  
Zeigen Sie, wie ein beliebiger Binärbaum durch höchstens  $n$  ( $n$ =Anzahl der Elemente im Baum) Rotationen in eine rechtslineare Kette umgewandelt werden kann.
- (b) Gegeben sei ein Algorithmus wie in Teil (a) gefordert. Zeigen Sie, dass ein Binärbaum durch  $\mathcal{O}(n)$  Rotationen in einen beliebigen anderen Binärbaum gleicher Knotenzahl umgewandelt werden kann.
- (c) Zeigen Sie, dass es nicht möglich ist, einen binären Heap in  $\mathcal{O}(n)$  Operationen, die nicht notwendigerweise Rotationen sind, in einen binären Suchbaum umzuwandeln.

**Hinweis:** Ihre Lösung sollte nicht länger als 4 Sätze sein.

### Aufgabe 4 (O-Notation)

Zeigen oder widerlegen Sie die folgenden Aussagen zur O-Notation.

- (a)  $2^n \in \Theta(3^n)$ .
- (b)  $2^{2\log_2 n} \in \mathcal{O}(n)$ .

### Aufgabe 5 (Formale Sprachen und Komplexität)

Wie Sie wissen, ist der Schnitt zweier kontextfreier Sprachen nicht notwendigerweise selbst kontextfrei und es ist unentscheidbar, ob der Schnitt zweier durch Grammatiken gegebener kontextfreier Sprachen leer ist.

Das Shuffle-Produkt  $L_1 \star L_2$  zweier Sprachen  $L_1, L_2$  über dem Alphabet  $\Sigma$  ist bekanntlich definiert durch

$$L_1 \star L_2 = \{u_1 v_1 u_2 v_2 \dots u_n v_n \mid u_1 u_2 \dots u_n \in L_1, v_1 v_2 \dots v_n \in L_2 \text{ mit } u_1, \dots, u_n, v_1, \dots, v_n \in \Sigma^*\}$$

Die Sprache  $L_1 \star L_2$  enthält also alle Wörter, die man durch Verzähnen eines Wortes in  $L_1$  mit einem Wort in  $L_2$  erhalten kann.

Beispiel:

$$\{aab, abab\} \star \{aa\} = \{aaaab, aaaba, aabaa, aaabab, aabaab, aababa, abaaab, abaaba, ababaa\}$$

Shuffle-Produkte spielen bei der Verifikation nebenläufiger Programme eine wichtige Rolle.

- (a) Geben Sie einen regulären Ausdruck für das Shuffle-Produkt der (selbst durch reguläre Ausdrücke gegebenen) Sprache  $a^*b^*$  mit der Sprache  $c^*d^*$  an.
- (b) Zeigen Sie unter Verwendung des o.a. Resultats über Schnitte kontextfreier Sprachen, dass folgendes Problem unentscheidbar ist: Gegeben sind zwei kontextfreie Sprachen  $L_1, L_2$  über dem Alphabet  $\Sigma = \{a, b\}$  durch Grammatiken, sowie eine reguläre Sprache  $L$  über  $\Sigma$  durch regulären Ausdruck. Gefragt ist, ob  $(L_1 \star L_2) \cap L$  leer ist.

Es bietet sich an, zu einer Sprache  $L$  über  $\Sigma$  die Sprache  $L'$  aller Wörter über dem von  $\Sigma$  disjunkten Alphabet  $\Sigma' = \{a', b'\}$  zu betrachten, die man durch Ersetzen von  $a$  durch  $a'$  und  $b$  durch  $b'$  erhält.

- (c) Dieses Resultat legt nahe, dass das Shuffle-Produkt zweier kontextfreier Sprachen nicht selbst kontextfrei sein muss. Hier soll ein rigoroser Beweis erbracht werden:  
Seien  $L_1 = \{a^n b^n \mid n \geq 1\}$  und  $L_2 = \{c^n d^n \mid n \geq 1\}$ . Zeigen Sie mithilfe des Pumpinglemmas für kontextfreie Sprachen, dass  $L_1 \star L_2$  nicht kontextfrei ist.

**66115**

**Theoretische Informatik / Algorithmen (vertieft)**

**Herbst 2019**

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	<b>Herbst 2019</b>	<b>66115</b>
Arbeitsplatz-Nr.: _____		

---

**Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen**  
**— Prüfungsaufgaben —**

---

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

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!

**Aufgabe 1 (Reguläre Sprachen)**

[36 PUNKTE]

- (a) [10 Punkte] Betrachten Sie den regulären Ausdruck

$$\alpha = ((a+b)^*ab) + ((a+b)^*bb) .$$

Geben Sie einen minimalen DEA für die von diesem Ausdruck beschriebene Sprache  $L(\alpha)$  an.

- (b) [14 Punkte] Betrachten Sie die folgenden Sprachen über dem Alphabet  $\Sigma = \{a, b\}$ :

$$L_1 = \{w \in \{a, b\}^* \mid \text{es existieren } u, v \in \{a, b\}^* \text{ mit } |u| = |v| \text{ und } w = uav\}$$

$$L_2 = \{w \in \{a\}^* \mid \text{es existieren } u, v \in \{a\}^* \text{ mit } |u| = |v| \text{ und } w = uav\}$$

Geben Sie für die Sprachen jeweils an, ob sie regulär sind oder nicht, und beweisen Sie Ihre Antwort. Für den Beweis von Regularität reicht die Angabe eines Automaten oder regulären Ausdrucks für die jeweilige Sprache.

- (c) [12 Punkte] Wir definieren die Menge der Permutationen eines Wortes  $w$ . Sei  $w = a_1 \cdots a_n$  mit  $a_i \in \Sigma$  für alle  $i = 1, \dots, n$ . Dann ist

$$\text{Perm}(w) := \{a_{\pi(1)} \cdots a_{\pi(n)} \mid \pi \text{ ist eine Permutation von } \{1, \dots, n\}\} .$$

So ist z.B.  $\text{Perm}(aab) = \{aab, aba, baa\}$ . Weiter definieren wir  $\text{Perm}(L)$  für eine Sprache  $L$  als  $\bigcup_{w \in L} \text{Perm}(w)$ .

Zeigen oder widerlegen Sie: Falls  $L$  eine reguläre Sprache ist, dann ist auch  $\text{Perm}(L)$  eine reguläre Sprache.

**Aufgabe 2 (Kontextfreie Sprachen)**

[24 PUNKTE]

- (a) [10 Punkte] Geben Sie einen (deterministischen oder nichtdeterministischen) Kellerautomaten für die Sprache

$$\{a^i b^j c^k \mid i, j, k \in \mathbb{N} \text{ und } i + k < j\}$$

an.

- (b) [14 Punkte] Sei

$$L_3 = \{wc^i w \mid w \in \{a, b\}^* \text{ und } i \geq 1\}$$

eine Sprache über dem Alphabet  $\{a, b, c\}$ . Zeigen Sie, dass  $L_3$  nicht kontextfrei ist.

**Aufgabe 3 (Entscheidbarkeit)**

[30 PUNKTE]

- (a) [18 Punkte] Wir betrachten eine Gödelisierung von Turingmaschinen und bezeichnen mit  $M_w$  die Turingmaschine, die gemäß dieser Kodierung vom Binärwort  $w$  kodiert wird. Außerdem bezeichnen wir mit  $M_w(x)$  die Berechnung der Maschine  $M_w$  bei Eingabe  $x$ . Betrachten Sie die Sprache

$$L_4 = \{w \mid M_w(0) \text{ hält nach weniger Schritten an als } M_w(1)\}$$

Hier sind Wörter  $w$  bei denen sowohl  $M_w(0)$  als auch  $M_w(1)$  nicht anhalten, nicht in  $L$ . Beweisen Sie, dass  $L_4$  unentscheidbar ist.

- (b) [12 Punkte] Das CYK-Verfahren ist ein Algorithmus, der für eine bestimmte kontextfreie Grammatik  $G$  in polynomieller Zeit entscheidet, ob ein gegebenes Wort  $w$  in  $L(G)$  liegt. So- mit ist jede kontextfreie Sprache entscheidbar. Zeigen oder widerlegen Sie folgende Aussage: Seien  $K_1$ ,  $K_2$  und  $K_3$  kontextfreie Sprachen. Sei  $L_5$  die Sprache, für die  $L_5 = (K_1 \cap K_2) \setminus K_3$  gilt. Dann ist  $L_5$  entscheidbar.

**Aufgabe 4 (Komplexität)**

[30 PUNKTE]

Betrachten Sie die folgenden Probleme:

**SAT**

Gegeben: Aussagenlogische Formel  $F$  in KNF.

Frage: Gibt es mindestens eine erfüllende Belegung für  $F$ ?

**DOPPELSAT**

Gegeben: Aussagenlogische Formel  $F$  in KNF.

Frage: Gibt es mindestens eine erfüllende Belegung für  $F$ , in der mindestens zwei Literale pro Klausel wahr sind?

- (a) [24 Punkte] Führen Sie eine polynomielle Reduktion von SAT auf DOPPELSAT durch.

- (b) [6 Punkte] Zeigen Sie, dass DOPPELSAT NP-vollständig ist.

**Aufgabe 5 (Sortierverfahren)**

[40 PUNKTE]

In der folgenden Aufgabe soll ein Feld A von ganzen Zahlen aufsteigend sortiert werden. Das Feld habe n Elemente A[0] bis A[n-1]. Der folgende Algorithmus (in der Notation des Informatik-Duden) sei gegeben:

```

1 procedure quicksort(links , rechts : integer)
2 var i , j , x : integer;
3 begin
4   i := links;
5   j := rechts;
6   if j > i then begin
7     x := A[links];
8     repeat
9       while A[i] < x do i := i+1;
10      while A[j] > x do j := j-1;
11      if i ≤ j then begin
12        tmp := A[i]; A[i] := A[j]; A[j] := tmp;
13        i := i+1; j := j-1;
14      end
15      until i > j;
16      quicksort(links , j);
17      quicksort(i , rechts);
18    end
19 end

```

Der initiale Aufruf der Prozedur lautet:

quicksort (0,n-1)

- (a) [18 Punkte] Sortieren Sie das folgende Feld der Länge 7 mittels des Algorithmus. Notieren Sie jeweils alle Aufrufe der Prozedur quicksort mit den konkreten Parameterwerten. Geben Sie zudem für jeden Aufruf der Prozedur den Wert des in Zeile 7 gewählten Elements an.

Index	0	1	2	3	4	5	6
Wert	27	32	3	6	17	44	42

- (b) [5 Punkte] Angenommen, die Bedingung  $j > i$  in Zeile 6 des Algorithmus wird ersetzt durch die Bedingung  $j \geq i$ . Ist der Algorithmus weiterhin korrekt? Begründen Sie Ihre Antwort.
- (c) [5 Punkte] Angenommen, die Bedingung  $i \leq j$  in Zeile 11 des Algorithmus wird ersetzt durch die Bedingung  $i < j$ . Ist der Algorithmus weiterhin korrekt? Begründen Sie Ihre Antwort.
- (d) [5 Punkte] Wie muss das Feld A gestaltet sein, damit der Algorithmus mit der geringsten Anzahl von Schritten terminiert? Betrachten Sie dazu vor allem Zeile 7. Begründen Sie Ihre Antwort und geben Sie ein Beispiel.

- (e) [7 Punkte] Die rekursiven Aufrufe in den Zeilen 16 und 17 des Algorithmus werden zur Laufzeit des Computers auf dem Stack verwaltet. Die Anzahl der Aufrufe von quicksort auf dem Stack abhängig von der Eingabegröße  $n$  sei mit  $s(n)$  bezeichnet. Geben Sie die Komplexitätsklasse von  $s(n)$  für den schlimmsten möglichen Fall an. Begründen Sie Ihre Antwort.

**Aufgabe 6 (O-Notation)**

[40 PUNKTE]

- (a) [8 Punkte] Sortieren Sie die unten angegebenen Funktionen der O-Klassen  $O(a(n))$ ,  $O(b(n))$ ,  $O(c(n))$ ,  $O(d(n))$  und  $O(e(n))$  bezüglich ihrer Teilmengenbeziehungen. Nutzen Sie ausschließlich die echte Teilmenge  $\subset$  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):

$$O(f_4(n)) \subset O(f_3(n)) = O(f_5(n)) \subset O(f_1(n)) = O(f_2(n))$$

Die angegebenen Beziehungen müssen weder bewiesen noch begründet werden.

- $a(n) = n^2 \cdot \log_2(n) + 42$
- $b(n) = 2^n + n^4$
- $c(n) = 2^{2 \cdot n}$
- $d(n) = 2^{n+3}$
- $e(n) = \sqrt{n^5}$

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

- (i) [8 Punkte]  $O(n \cdot \log_2 n) \subseteq O(n \cdot (\log_2 n)^2)$   
(ii) [8 Punkte]  $2^{n+1} \in O(2^n)$

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

- (i) [8 Punkte]  $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2$   
(ii) [8 Punkte]  $T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$

**Aufgabe 7 (Bäume)**

[40 PUNKTE]

Gegeben sei die folgende Realisierung von binären Bäumen (in einer an Java angelehnten Notation):

```

1 class Node {
2     Node left , right ;
3     int value ;
4 }
```

- (a) [8 Punkte] Beschreiben Sie in möglichst wenigen Worten, was die folgende Methode foo auf einem nicht-leeren binären Baum berechnet.

```

1 int foo( Node node ){
2     int b = node.value ;
3     if ( b < 0 ) {
4         b = -1 * b ;
5     }
6     if ( node.left != null ) {
7         b = b + foo( node.left );
8     }
9     if ( node.right != null ) {
10        b = b + foo( node.right );
11    }
12    return b ;
13 }
```

- (b) [6 Punkte] Die Laufzeit der Methode foo(tree) ist linear in  $n$ , der Anzahl von Knoten im übergebenen Baum tree. Begründen Sie kurz, warum foo(tree) eine lineare Laufzeit hat.

- (c) [14 Punkte] Betrachten Sie den folgenden Algorithmus für nicht-leere, binäre Bäume. Beschreiben Sie in möglichst wenigen Worten die Wirkung der Methode magic(tree). Welche Rolle spielt dabei die Methode max?

```

1 void magic( Node node ){
2     Node m = max( node );
3     if (m.value > node.value ) {
4         // Werte von m und node vertauschen
5         int tmp = m.value ;
6         m.value = node.value ;
7         node.value = tmp ;
8     }
9     if (node.left != null )
10        magic( node.left );
11     if (node.right != null )
12        magic( node.right );
13 }
14 Node max( Node node ){
15     Node max = node ;
16     if ( node.left != null ){
```

```
17     Node tmp = max( node.left );
18     if (tmp.value > max.value ) max = tmp ;
19 }
20 if ( node.right != null ){
21     Node tmp = max( node.right );
22     if (tmp.value > max.value ) max = tmp ;
23 }
24 return max ;
25 }
```

- (d) [12 Punkte] Geben Sie in Abhängigkeit von  $n$ , der Anzahl von Knoten im übergebenen Baum tree, jeweils eine Rekursionsgleichung für die asymptotische Best-Case-Laufzeit ( $B(n)$ ) und Worst-Case-Laufzeit ( $W(n)$ ) des Aufrufs `magic(tree)` sowie die entsprechende Komplexitätsklasse ( $\Theta$ ) an. Begründen Sie Ihre Antwort.

Hinweis: Überlegen Sie, ob die Struktur des übergebenen Baumes Einfluss auf die Laufzeit hat. Die lineare Laufzeit von `max(t)` in der Anzahl der Knoten des Baumes `t` darf vorausgesetzt werden.

Thema Nr. 2  
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

**Aufgabe 1 (Turing-Maschine)**

[26 PUNKTE]

Gesucht ist eine Turing-Maschine mit genau einem beidseitig unendlichen Band, die die Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  mit  $f(x) = 3x$  berechnet. Zu Beginn der Berechnung steht die Eingabe binär codiert auf dem Band, wobei der Kopf auf die linkste Ziffer (most significant bit) zeigt. Am Ende der Berechnung soll der Funktionswert binär codiert auf dem Band stehen, wobei der Kopf auf ein beliebiges Feld zeigen darf.

- (a) Beschreiben Sie zunächst in Worten die Arbeitsweise Ihrer Maschine.
- (b) Geben Sie dann das kommentierte Programm der Turing-Maschine an und erklären Sie die Bedeutung der verwendeten Zustände.

**Aufgabe 2 (Zwei Abschlusseigenschaften von REG und CFL)**

[24 PUNKTE]

Es sei  $\Sigma = \{a, b\}$  das Alphabet. Die folgenden Aussagen A und B werden betrachtet.

- A: Für jede reguläre Sprache  $L_1 \subseteq \Sigma^*$  ist auch die Sprache  $L'_1$  regulär, wobei  $L'_1 = \{w \in \Sigma^* \mid \text{für alle Wörter } w' \in \Sigma^* \text{ gilt } ww' \in L_1\}$ .
- B: Für jede kontextfreie Sprache  $L_2 \subseteq \Sigma^*$  ist auch die Sprache  $L'_2$  kontextfrei, wobei  $L'_2 = \{w \in \{a, b, \#\}^* \mid \exists n \geq 1 \exists u_1, \dots, u_n, v_1, \dots, v_n \in L_2 \text{ mit } w = u_1 \cdots u_n \# v_1 \cdots v_n\}$ . D.h.  $L'_2$  besteht aus den Wörtern  $u\#v$ , sodass sich  $u$  und  $v$  aus der gleichen Anzahl von Wörtern aus  $L_2$  bilden lassen.

- (a) Zeigen Sie die Aussage A. Es genügt die Beschreibung der Konstruktion, die aus einem endlichen Automaten für die Sprache  $L_1$  einen endlichen Automaten für die Sprache  $L'_1$  erzeugt.
- (b) Zeigen Sie die Aussage B. Es genügt die Beschreibung der Konstruktion, die aus einer kontextfreien Grammatik für die Sprache  $L_2$  eine kontextfreie Grammatik für die Sprache  $L'_2$  erzeugt.

**Aufgabe 3 (Einordnung von Mengen in Komplexitätsklassen)**

[42 PUNKTE]

Sei  $M_0, M_1, \dots$  eine Gödelisierung der Registermaschinen (RAMs). Weiter bezeichne  $\text{bin}(x)$  die Binärdarstellung der Zahl  $x \in \mathbb{N}$ . Wir betrachten folgende Klassen von Sprachen über dem Alphabet  $\Sigma = \{0, 1, \#\}$ .

- REG =  $\{L \subseteq \Sigma^* \mid L \text{ ist regulär}\}$
- P =  $\{L \subseteq \Sigma^* \mid L \text{ ist in deterministischer Polynomialzeit entscheidbar}\}$
- NP =  $\{L \subseteq \Sigma^* \mid L \text{ wird von einer nichtdeterministischen Polynomialzeit-Maschine akzeptiert}\}$
- REC =  $\{L \subseteq \Sigma^* \mid L \text{ ist entscheidbar}\}$
- RE =  $\{L \subseteq \Sigma^* \mid L \text{ ist aufzählbar}\}$
- ALL =  $\{L \mid L \subseteq \Sigma^*\}$

Fortsetzung nächste Seite!

Es gilt

$$\text{REG} \subsetneq \text{P} \subseteq \text{NP} \subsetneq \text{REC} \subseteq \text{RE} \subsetneq \text{ALL}. \quad (*)$$

Weiter seien folgende Sprachen über dem Alphabet  $\Sigma = \{0, 1, \#\}$  gegeben.

- $L_1 = \{w \in \Sigma^* \mid \exists n \in \mathbb{N}, w = 0^n 1^n 0^n\}$
- $L_2 = \{\text{bin}(x) \mid x \in \mathbb{N} \text{ und } M_x \text{ hält nicht bei Eingabe } x\}$
- $L_3 = \{\text{bin}(x) \mid x \in \mathbb{N} \text{ und } x \text{ ist durch 4 teilbar}\}$
- $L_4 = \{\text{bin}(x) \mid x \in \mathbb{N} \text{ und die von } M_x \text{ berechnete Funktion } \mathbb{N} \rightarrow \mathbb{N} \text{ ist nicht injektiv}\}$
- $L_5 = \{\text{bin}(a_1)\#\text{bin}(a_2)\#\cdots\#\text{bin}(a_n)\#\text{bin}(b) \mid a_1, \dots, a_n, b \in \mathbb{N} \text{ und } \exists I \subseteq \{1, \dots, n\}, \sum_{i \in I} a_i = b\}$

Geben Sie für jedes  $L_i$  Folgendes an:

- (a) die (nach aktuellem Kenntnisstand) kleinste Klasse  $\mathcal{C}$  aus der Inklusionskette  $(*)$  mit der Eigenschaft  $L_i \in \mathcal{C}$ ;
- (b) eine kurze Begründung für  $L_i \in \mathcal{C}$  (ca. 2-3 Zeilen Begründung, kein ausführlicher Beweis);
- (c) eine kurze Begründung dafür, dass  $L_i$  (nach aktuellem Kenntnisstand) nicht in der nächst kleineren Klasse liegt (ca. 2-3 Zeilen Begründung, kein ausführlicher Beweis).

#### Aufgabe 4 (Reduktion von 3-FARB auf 7-FARB)

[28 PUNKTE]

Im Folgenden werden endliche, ungerichtete Graphen betrachtet, wobei ein endlicher ungerichteter Graph  $G$  ein Paar  $(V, E)$  ist, sodass

- $V$  eine nichtleere, endliche Menge ist (die Knotenmenge) und
- $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$  (die Kantenmenge), d.h.  $E$  ist eine Menge zweielementiger Teilmengen von  $V$ .

Für  $k \geq 3$  betrachten wir das Problem

$$k\text{-FARB} = \{G \mid G \text{ ist ein } k\text{-färbbarer ungerichteter Graph}\},$$

wobei ein Graph  $G = (V, E)$  genau dann  $k$ -färbbbar heißt, falls seine Knoten so mit  $k$  Farben eingefärbt werden können, dass alle durch eine Kante verbundenen Knoten verschiedenfarbig sind (d.h. es gibt ein totales  $c : V \rightarrow \{1, \dots, k\}$  mit  $\forall \{u, v\} \in E$  gilt  $c(u) \neq c(v)$ ).

Zeigen Sie, dass 3-FARB in polynomieller Zeit auf 7-FARB reduzierbar ist (d.h.  $3\text{-FARB} \leq 7\text{-FARB}$ ). Gehen Sie dazu wie folgt vor:

- (a) Geben Sie eine totale, in Polynomialzeit berechenbare Reduktionsfunktion  $f$  an.  
(Es ist hier kein Nachweis der Totalität und Polynomialzeit-Berechenbarkeit von  $f$  gefordert.)
- (b) Zeigen Sie: Falls  $G$  in 3-FARB, so ist  $f(G)$  in 7-FARB.
- (c) Zeigen Sie: Falls  $f(G)$  in 7-FARB, so ist  $G$  in 3-FARB.

**Aufgabe 5 ( $\mathcal{O}$ -Notation)**

[35 PUNKTE]

- (a) [5 Punkte] Geben Sie eine formale Definition von  $\mathcal{O}(f)$ .
- (b) [10 Punkte] Zeigen oder widerlegen Sie:  $\mathcal{O}(\log(n!)) = \mathcal{O}(n \log n)$ .
- (c) [20 Punkte] Zeigen oder widerlegen Sie jeweils für Funktionen

$$f(n) = 10^4 \cdot \sqrt{n} \text{ und } g(n) = \begin{cases} 10, & \text{für } n \bmod 2 = 0 \\ \frac{1}{2}n, & \text{sonst} \end{cases}$$

- (i)  $f \in \mathcal{O}(g)$ .
- (ii)  $g \in \mathcal{O}(f)$ .

**Aufgabe 6 (Mastertheorem)**

[20 PUNKTE]

Der Hauptsatz der Laufzeitfunktionen ist bekanntlich folgendermaßen definiert:

$$\text{Sei } T(n) = \begin{cases} d \in \Theta(1), & \text{falls } n \leq k \\ a \cdot T\left(\frac{n}{b}\right) + g(n), & \text{sonst} \end{cases} \quad \text{mit } k \in \mathbb{N}, a \geq 1 \text{ und } b > 1$$

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  $a \cdot g\left(\frac{n}{b}\right) \leq c \cdot g(n)$  für fast alle  $n$  und ein  $c$  mit  $0 < c < 1$   
 $\Rightarrow T(n) \in \Theta(g(n))$

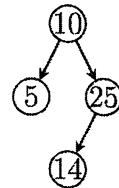
Bestimmen und begründen Sie formal mit Hilfe dieses Satzes welche Komplexität folgende Laufzeitfunktionen haben.

- (a) [12 Punkte]  $T(n) = 8T\left(\frac{n}{3}\right) + 5n^2$
- (b) [8 Punkte]  $T(n) = 9T\left(\frac{n}{3}\right) + 5n^2$

**Aufgabe 7 (AVL-Bäume)**

[32 PUNKTE]

Fügen Sie (manuell) nacheinander die Zahlen 20, 31, 2, 17, 7 in folgenden AVL-Baum ein. Löschen Sie anschließend aus dem entstandenen Baum nacheinander 14 und 25.

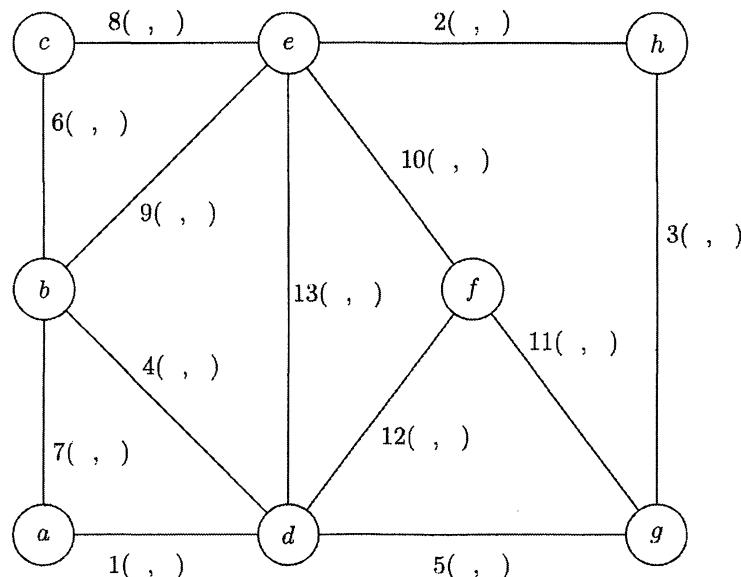


Zeichnen Sie jeweils direkt nach jeder einzelnen Operation zum Einfügen oder Löschen eines Knotens, sowie nach jeder elementaren Rotation den entstehenden Baum. Insbesondere sind evtl. anfallende Doppelrotationen in zwei Schritten darzustellen. Geben Sie zudem an jedem Knoten die Balancewerte an.

**Aufgabe 8 (Minimaler Spannbaum)**

[21 PUNKTE]

Gegeben Sei der folgende ungerichtete Graph mit Kantengewichten.



(a) [7 Punkte] Zeichnen Sie den (hier eindeutigen) minimalen Spannbaum.

(b) [14 Punkte] Geben Sie sowohl für den Algorithmus von Jarník-Prim als auch für den Algorithmus von Kruskal die Reihenfolge an, in der die Kanten hinzugefügt werden. Starten Sie für den Algorithmus von Jarník-Prim beim Knoten  $a$ .

Übernehmen Sie den Graph auf Ihre Bearbeitung und füllen Sie hierzu das Tupel jeder Kante  $e$  aus dem MST in der Form  $(n, m)$  aus, wobei die Kante  $e$  vom Algorithmus von Jarník-Prim als  $n$ 'te Kante und vom Algorithmus von Kruskal als  $m$ 'te Kante hinzugefügt wird. Lassen Sie andere Tupel unausgefüllt.

**Aufgabe 9 (Hashing)**

[12 PUNKTE]

Verwenden Sie die Hashfunktion  $h(k, i) = (h'(k) + i^2) \bmod 11$  mit  $h'(k) = k \bmod 13$ , um die Werte 12, 29 und 17 in die folgende Hashtabelle einzufügen. Geben Sie zudem jeweils an auf welche Zellen der Hashtabelle zugegriffen wird.

0	1	2	3	4	5	6	7	8	9	10
			16		5				22	