

**Sammlung aller Staatsexamensaufgaben der
Prüfungsnummer**

46115

**Theoretische Informatik / Algorithmen /
Datenstrukturen (nicht vertieft)**

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2010

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

**Frühjahr
2010**

46115

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

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Theoret. Informatik, Algorith./Datenstr**

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!

1. Betrachten Sie über dem Alphabet $\Sigma = \{a, b\}$ die Sprache L , die durch folgendes Regelsystem vollständig gegeben ist:

- Das Wort a gehört zu L .
- Gehört ein Wort w zu L und endet w mit einem a , so gehört auch wb zu L , d.h. für $u \in \Sigma^*$ gilt $ua \in L \Rightarrow uab \in L$.
- Gehört w zu L , so gilt auch $ww \in L$.
- Folgen in einem Wort $w \in L$ drei a 's unmittelbar aufeinander, so gehört auch das Wort zu L , das man aus w erhält, wenn man aaa durch b ersetzt, d.h. für $u, v \in \Sigma^*$ gilt $uaav \in L \Rightarrow ubv \in L$.
- Folgen in einem Wort $w \in L$ zwei b 's unmittelbar aufeinander, so gehört auch das Wort zu L , das man aus w erhält, wenn man bb streicht, d.h. für $u, v \in \Sigma^*$ gilt $ubbv \in L \Rightarrow uv \in L$.

Beispielsweise gehört das Wort $baab$ zu L , was man an der Ableitung

$$a \vdash aa \vdash aaaa \vdash ba \vdash bab \vdash babbab \vdash baab$$

sieht. Da das Regelsystem sowohl verlängernde als auch verkürzende Regeln enthält, ist die Zugehörigkeit eines Worts $w \in \Sigma^*$ zur Sprache L scheinbar nicht einfach zu entscheiden. Da das Regelsystem keine reguläre Grammatik ist, ist auch nicht klar, ob die Sprache L regulär ist.

- a) Zeigen Sie, dass alle Wörter $w \in L$ die Eigenschaft haben:

(*) Die Anzahl $|w|_a$ der Vorkommen von a in w ist nicht durch 3 teilbar.

- b) Tatsächlich charakterisiert die Eigenschaft (*) die Sprache L , d.h. es gilt

$$L = \{w \in \Sigma^* ; 3 \nmid |w|_a\}.$$

Verwenden Sie diese Aussage (die Sie nicht beweisen sollen!), um einen minimalen vollständigen DFA zu konstruieren, der die Sprache L akzeptiert.

- c) Beweisen Sie, dass Ihr Automat tatsächlich minimal ist!

2. Betrachten Sie die Funktion des ganzzahligen Quadratwurzelziehens:

$$\text{sqrt} : \mathbb{N} \rightarrow \mathbb{N} : x \mapsto \lfloor \sqrt{x} \rfloor$$

- a) Geben Sie ein Programm für `sqrt` in einer Ihnen geläufigen Programmiersprache an.
- b) Geben Sie ein WHILE-Programm für `sqrt` an.
- c) Geben Sie ein LOOP-Programm für `sqrt` an.
- d) Zeigen Sie, dass `sqrt` eine primitiv-rekursive Funktion ist, indem Sie eine entsprechende Definition angeben.

Hinweise:

- Sie können, falls Ihnen das hilfreich erscheint, die Tatsache verwenden, dass $1 + 3 + 5 + \dots + (2n - 1) = n^2$ ist.
 - $\lfloor x \rfloor$ bezeichnet die größte ganze Zahl, die kleiner oder gleich x ist, also z.B. $\lfloor 3,1 \rfloor = 3$.
3. Beweisen oder widerlegen Sie die folgenden Behauptungen über Sprachen $A, B \subseteq \Sigma^*$. $A\Delta B = (A \setminus B) \cup (B \setminus A)$ bezeichne dabei die symmetrische Mengendifferenz.
- Sind A und B regulär, so ist auch $A\Delta B$ regulär.
 - Sind A und B kontextfrei, so ist auch $A\Delta B$ kontextfrei.
 - Sind A und B entscheidbar, so ist auch $A\Delta B$ entscheidbar.
 - Sind A und B partiell-entscheidbar, so ist auch $A\Delta B$ partiell-entscheidbar.
 - Sind A und B polynomiell-entscheidbar, so ist auch $A\Delta B$ polynomiell-entscheidbar.
4. ALICE und BOB verfügen jeder über einen DFA über dem Alphabet $\Sigma = \{0, 1\}$. Der Automat von ALICE sei \mathfrak{A} , der Automat von BOB sei \mathfrak{B} . Beide Automaten haben die gleiche Anzahl n von Zuständen. ALICE und BOB wollen die Frage klären, ob es ein Wort $w \in \Sigma^*$ gibt, das von beiden Automaten akzeptiert wird.

ALICE argumentiert: "Diese Frage können wir gar nicht entscheiden, denn dafür müssten wir unendlich viele Wörter aus Σ^* durchprobieren."

BOB hält dem entgegen: "Das geht doch ganz einfach: Wenn ein Automat n Zustände hat, können alle akzeptierenden Zustände in höchstens $n - 1$ Schritten erreicht werden. Es genügt also, Wörter der Länge $< n$ zu testen, ob eines von ihnen von beiden Automaten \mathfrak{A} und \mathfrak{B} akzeptiert wird."

Kommentieren Sie die beiden Aussagen! Sollten Sie weder ALICE noch BOB zustimmen, so zeigen Sie einen Weg auf, die Frage zu beantworten. Geben Sie ein Entscheidungsverfahren an oder weisen Sie nach, dass es sich um ein unentscheidbares Problem handelt.

5. Datenstrukturen und Algorithmen: Binäre Suchbäume und AVL-Bäume

- a) Geben Sie jeweils eine Definition für *binäre Suchbäume* und *AVL-Bäume* an.
- b) Zeichnen Sie einen AVL-Baum, der die folgenden Schlüssel enthält: 11, 1, 5, 37, 17, 29, 31, 3.
- c) Welche Zeitkomplexität haben die Operationen *Einfügen*, *Löschen* und *Suchen* auf AVL-Bäumen? Begründen Sie Ihre Antwort.
- d) Implementieren Sie die Datenstruktur AVL-Baum mit Schlüsseln vom Typ `int` (für Java oder entsprechende Datentypen für die anderen Programmiersprachen) in entweder Java, C++, Smalltalk oder Eiffel. Ihre Implementierung muss als einzige Operation die Methode `suche` bereitstellen, die einen Schlüssel als Parameter bekommt und
 - `true` zurückliefert, wenn der gesuchte Schlüssel im Baum enthalten ist,
 - ansonsten `false`.

Ihre Implementierung muss keine Konstruktoren oder andere Methoden zur Initialisierung bereitstellen. Des Weiteren soll die Implementierung nur Schlüsselknoten berücksichtigen.

Kommentieren Sie Ihre Implementierung ausführlich. Geben Sie an, welche Programmiersprache Sie gewählt haben.

Thema Nr. 2
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

1. Gegeben ist die folgende Sprache L_1 über dem Alphabet $\Sigma = \{a, b\}$:

$$L_1 = \{w \in \Sigma^* \mid \text{die Anzahl der } a \text{ in } w \text{ ist gerade und } b \text{ kommt in } w \text{ genau einmal vor}\}.$$

- a) Geben Sie einen deterministischen endlichen Automaten an, der die Sprache L_1 akzeptiert.
- b) Geben Sie einen regulären Ausdruck an, der die Sprache L_1 beschreibt.

Die folgende Sprache L_2 ist eine Erweiterung von L_1 :

$$L_2 = \{w \in \Sigma^* \mid \text{die Anzahl der } a \text{ in } w \text{ ist gerade und die Anzahl der } b \text{ in } w \text{ ist ungerade}\}.$$

- c) Geben Sie einen deterministischen endlichen Automaten an, der die Sprache L_2 akzeptiert.
- d) Geben Sie eine rechtslineare Grammatik an, die die Sprache L_2 erzeugt.

2. Gegeben ist die folgende Sprache L über dem Alphabet $\Sigma = \{0, 1\}$:

$$L = \{w \in \Sigma^* \mid w \neq \epsilon, w \text{ enthält gleich viele } 0\text{-en wie } 1\text{-en und in jedem Anfangswort von } w \text{ kommen mindestens soviele } 0\text{-en wie } 1\text{-en vor}\}.$$

- a) Geben Sie alle Wörter $w \in L$ mit Länge $|w| \leq 4$ an.
- b) Geben Sie eine kontextfreie Grammatik an, die die Sprache L erzeugt.
- c) Zeigen Sie, dass L nicht regulär ist. Der Beweis kann ohne Anwendung des Pumping Lemmas unter Verwendung der beiden folgenden Tatsachen geführt werden:
 - i) Der Durchschnitt zweier regulärer Sprachen ist regulär.
 - ii) Die Sprache $L' = \{0^j 1^j \mid j > 0\}$ ist nicht regulär.

Wir betrachten nun für alle natürlichen Zahlen $n \in \mathbb{N}$ die folgenden Teilsprachen L_n von L :

$$L_n = \{w \in L \mid \text{in jedem Anfangswort von } w \text{ kommen höchstens } n \text{ mehr } 0\text{-en vor als } 1\text{-en}\}.$$

- d) Für alle natürlichen Zahlen n ist L_n regulär. Zeigen Sie dies für den Fall $n = 3$.
- e) Wie kann man aus den Tatsachen, dass L nicht regulär ist und L_n für alle $n \in \mathbb{N}$ regulär ist, schließen, dass die Vereinigung abzählbar unendlich vieler regulärer Mengen nicht notwendigerweise regulär ist?

3. Zeigen Sie, dass die Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$, mit $f(n) = 2^{(n \text{ div } 2)}$ für alle $n \in \mathbb{N}$, primitiv rekursiv ist. Dabei bezeichnet $n \text{ div } 2$ die ganzzahlige Division durch 2.

Beim Beweis ist das Schema der primitiven Rekursion zu verwenden, d.h. es sind geeignete primitiv rekursive Funktionen $g : \mathbb{N} \rightarrow \mathbb{N}$ und $h : \mathbb{N}^2 \rightarrow \mathbb{N}$ anzugeben, so dass gilt:

$$f(0) = g, f(n+1) = h(n, f(n)) \text{ für alle } n \in \mathbb{N}.$$

Es kann vorausgesetzt werden, dass die Multiplikation und die Fallunterscheidung, ob eine natürliche Zahl gerade ist oder nicht, primitiv rekursiv sind.

4. Entscheiden Sie, welche der folgenden Aussagen richtig und welche falsch sind und geben Sie jeweils eine kurze Begründung für Ihre Antwort an. Unter einer totalen Funktion wird im Folgenden, wie üblich, eine auf allen Argumenten definierte Funktion verstanden.
- Jede μ -rekursive totale Funktion ist primitiv rekursiv.
 - Jede primitiv rekursive Funktion ist eine totale μ -rekursive Funktion.
 - Das Komplement jeder primitiv rekursiven Teilmenge M der natürlichen Zahlen ist primitiv rekursiv. (Eine Menge heißt primitiv rekursiv, wenn ihre charakteristische Funktion primitiv rekursiv ist.)
 - Das Komplement jeder entscheidbaren Teilmenge M der natürlichen Zahlen ist entscheidbar.
 - Das Komplement jeder rekursiv aufzählbaren Teilmenge M der natürlichen Zahlen ist rekursiv aufzählbar.
5. a) Beschreiben Sie die Datenstruktur HEAP, insbesondere die zugehörigen Operationen, Aufbau und Update einschließlich Aufwandsbetrachtungen.
- b) Stellen Sie mindestens zwei unterschiedliche Sortierverfahren einander gegenüber und vergleichen Sie diese Verfahren mit Heap-Sort.
- c) Beschreiben Sie die Suchstrategie „Tiefensuche (Depth First Search)“ und zeigen Sie Anwendungen auf bei der Bestimmung von Zusammenhangskomponenten in Graphen.

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2011

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

**Frühjahr
2011**

46115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen
— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Theoret. Informatik, Algorith./Datenstr.**

Anzahl der gestellten Themen (Aufgaben): **1**

Anzahl der Druckseiten dieser Vorlage: **6**

Bitte wenden!

Thema Nr. 1

Teilaufgabe I Wir fixieren das Alphabet $\Sigma = \{(0), (0), (1), (1)\}$ und definieren $L \subseteq \Sigma^*$ als die Sprache aller Wörter w mit der folgenden Eigenschaft:

- Es sei $w_1 \in \{0, 1\}^*$ das Wort bestehend aus den „oberen“ Einträgen von w und es sei $w_2 \in \{0, 1\}^*$ das aus den „unteren“ Einträgen von w bestehende Wort. Wenn etwa $w = (0)(1)(1)$, dann ist $w_1 = 011$ und $w_2 = 010$.
 - Das Wort w ist in L genau dann, wenn w_1 als Binärzahl aufgefasst, um eins größer als der Wert von w_2 ist. So ist also das obige Beispielwort in L , da 011 den Wert 3 hat und 010 den Wert 2 hat.
1. Konstruieren Sie einen endlichen Automaten (egal, ob deterministisch oder nicht) für L^{rev} , also für die Sprache aller Wörter über Σ , welche von rechts nach links gelesen in L liegen. Hinweis: Man addiert eins zu einer Binärzahl, indem man die erste Null von rechts gesehen zur Eins umändert und alle davor (von rechts her gesehen) liegenden Einsen zu Nullen ändert.
 2. Konstruieren Sie nun einen nichtdeterministischen endlichen Automaten für L selbst. Es bietet sich an, den Automaten aus Aufgabenteil 1 zu verwenden, Sie müssen das aber nicht tun.
 3. Beschreiben Sie die Potenzmengenkonstruktion zur Konversion eines nichtdeterministischen in einen deterministischen Automaten.
 4. Konstruieren Sie einen deterministischen endlichen Automaten für L .
 5. Ermitteln Sie, ob Ihr Automat minimal ist. Falls ja, begründen Sie die Minimalität, falls nein, konstruieren Sie den Minimalautomaten. Wenn Sie Aufgabenteil 4 nicht lösen konnten, beschreiben Sie allgemein, wie man den Minimalautomaten konstruiert und wie man die Minimalität eines vorgelegten Automaten zeigen kann.

Teilaufgabe II

1. Begründen Sie, dass folgende Menge unentscheidbar ist:

$$M = \{e \mid \text{die durch } e \text{ bezeichnete Turingmaschine hält bei Eingabe 37 nicht.}\}$$

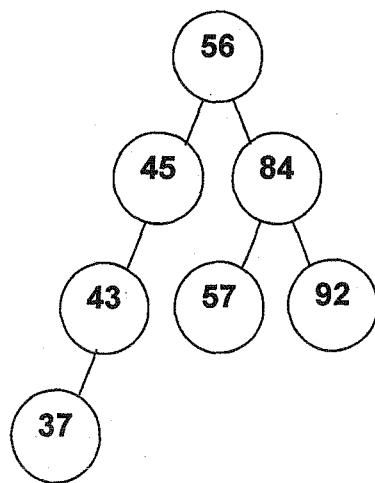
2. Ist M partiell entscheidbar (Synonym: semi-entscheidbar, rekursiv aufzählbar)? (Begründung!)
3. Zeigen Sie durch Angabe eines Gegenbeispiels, dass folgende Aussage unwahr ist: Wenn L_1 und L_2 unentscheidbar sind, dann ist auch $L_1 \cup L_2$ unentscheidbar.

Teilaufgabe III

1. Fügen Sie in einen anfangs leeren 2-3-4 Baum (B-Baum der Ordnung 4) der Reihe nach die folgenden Schlüssel ein: 1, 2, 3, 5, 7, 8, 9, 4, 11, 12, 13, 6. Dokumentieren Sie die Zwischenschritte so, dass die Entstehung des Baumes und nicht nur das Endergebnis nachvollziehbar ist.
2. Zeichnen Sie einen Rot-Schwarz-Baum *oder* einen AVL-Baum, der dieselben Einträge enthält.
3. Geben Sie eine möglichst gute untere Schranke (in Ω -Notation) für die Anzahl der Schlüssel in einem 2-3-4-Baum der Höhe h an. Hinweis: Überlegen Sie sich, wie ein 2-3-4 Baum mit Höhe h und möglichst wenigen Schlüsseln aussieht.
4. Geben Sie eine möglichst gute obere Schranke (in O -Notation) für die Anzahl der Schlüssel in einem 2-3-4-Baum der Höhe h an.
5. Für welche $a \in \mathbb{R}$ gilt: $4^{an} = O(2^n)$?
6. Für welche $a \in \mathbb{R}$ gilt: $n^2 = \Omega(44an^a)$?

Thema Nr. 2**Aufgabe 1**

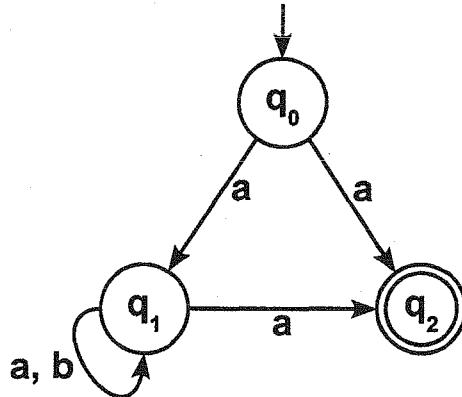
Als spezielle binäre Bäume werden in der Informatik oft AVL-Bäume eingesetzt. Gegeben sei folgender AVL-Baum, in den zuletzt der Knoten mit dem Wert 37 eingefügt wurde.



- Erläutern Sie, warum jetzt die AVL-Eigenschaft des Baumes verletzt ist, und stellen Sie diese Eigenschaft durch geeignete Maßnahmen wieder her. Erläutern Sie Ihr Vorgehen.
- Fügen Sie nun schrittweise Knoten mit den Werten 21, 42 und 41 in dieser Reihenfolge in den AVL-Baum ein und stellen Sie gegebenenfalls jeweils die AVL-Eigenschaft durch geeignete Maßnahmen wieder her.

Aufgabe 2

Gegeben sei folgender endlicher Automat:



- Überführen Sie diesen nichtdeterministischen Automaten in einen deterministischen endlichen Automaten.
- Geben Sie die Sprache, die dieser Automat erkennt, als möglichst kurzen regulären Ausdruck an.

Aufgabe 3

Sei $L = \{a^{2n}b^{3n} | n > 0\}$ eine Sprache über dem Alphabet $\Sigma = \{a, b\}$.

- Geben Sie eine kontextfreie Grammatik G an, die L erzeugt.
- Geben Sie eine Ableitung des Wortes aaaabbbbb in der Grammatik G an.
- Begründen Sie ausführlich, dass es keine reguläre Grammatik gibt, die L erzeugt.

Betrachten Sie die Sprachen $L_1 = \{ww^R | w \in \Sigma^*\}$ und $L_2 = \{ww | w \in \Sigma^*\}$ über dem Alphabet Σ .

- Ordnen Sie die Sprachen L_1 und L_2 möglichst exakt in die Chomsky Hierarchy ein.
- Beschreiben Sie informell die Arbeitsweise der L_1 und L_2 akzeptierenden Automaten.
- Begründen Sie insbesondere, warum L_1 und L_2 nicht in die jeweils niedrigere Hierarchiestufe eingeordnet werden können (Typ 3 sei die niedrigste, Typ 0 die höchste Stufe).

Begründen oder widerlegen Sie die folgenden Aussagen:

- Die Schnittmenge zweier semi-entscheidbarer Sprachen ist ebenfalls semi-entscheidbar.
- Die Vereinigung zweier semi-entscheidbarer Sprachen ist ebenfalls semi-entscheidbar.
- Das Komplement einer semi-entscheidbaren Sprache ist ebenfalls semi-entscheidbar.
- Die Differenz zweier semi-entscheidbarer Sprachen ist ebenfalls semi-entscheidbar.

Aufgabe 4

Eine Datenstruktur zu *augmentieren* bedeutet, in der Datenstruktur zusätzliche Information zu speichern, um neue Typen von Anfragen effizient beantworten zu können. In dieser Aufgabe geht es darum, Ihnen bekannte Datenstrukturen so zu augmentieren, dass man schnell den Median der gespeicherten Menge von Zahlen bestimmen kann. Zur Erinnerung: Der Median einer Menge $\{a_1, a_2, \dots, a_n\}$ von Zahlen mit $a_1 < a_2 < \dots < a_n$ ist das Element $a_{\lfloor n/2 \rfloor}$.

Gehen Sie in beiden Teilaufgaben der Einfachheit halber davon aus, dass eine Zahl zu jedem Zeitpunkt höchstens einmal in der Liste gespeichert ist.

- a) Gehen Sie davon aus, dass Sie eine korrekte Implementierung einer *sortierten*, doppelt verketteten Liste vorliegen haben. Die Liste besitzt einen Zeiger *head*, der auf das erste Element der Liste zeigt. Jedes Element verfügt über einen Schlüssel *key* – die zu speichernde Zahl – und über zwei Zeiger. Der Zeiger *next* zeigt auf das folgende Element, falls ein solches existiert; ansonsten ist *next* der Nullzeiger *nil*. Entsprechend zeigt *prev* auf das vorhergehende Element, falls ein solches existiert; ansonsten ist *prev* = *nil*.

Die Liste sucht beim Einfügen einer Zahl ihren Platz in der Sortierung und gibt dann einen Zeiger auf das neu angelegte Listenelement zurück, in dem die Zahl gespeichert wurde. Mithilfe dieses Zeigers kann der Benutzer die Zahl später in konstanter Zeit aus der Liste löschen.

Ihre Aufgabe besteht darin anzugeben, welche zusätzliche Information wo und wie aufrechterhalten werden soll, wenn der Benutzer Zahlen in die Liste einfügt oder aus der Liste löscht. Geben Sie dazu in Pseudocode-Schreibweise an, welche Anweisungen im Anschluss an die Einfüge- bzw. die Löschoperation der zugrundeliegenden Liste ausgeführt werden sollen. Stellen Sie sicher, dass eine Löschoperation auch mit Ihrer Augmentierung nur konstante Zeit dauert. Das Einfügen darf lineare Zeit dauern.

- b) Zeigen Sie, wie man eine andere Datenstruktur augmentieren kann, um nach wie vor schnellen Zugriff auf den Median einer sich ändernden Menge von Zahlen zu haben – nun aber so, dass die Einfügeoperation asymptotisch schneller ist als bei der Liste in Teilaufgabe (a). Genauer gesagt soll die augmentierte Datenstruktur die drei Operationen Einfügen, Löschen und Zugriff auf den Median in jeweils sublinearer Zeit ermöglichen.

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2012

Prüfungsteilnehmer

Prüfungstermin

Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

**Frühjahr
2012**

46115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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

Anzahl der Druckseiten dieser Vorlage: **7**

Bitte wenden!

Thema Nr. 1

Aufgabe 1: Turingmaschinen

Die Turingmaschine „Reverse“ spiegelt das gegebene Eingabewort. Aus dem Wort FUNKTION wird so zum Beispiel das Wort NOITKNUF.

- Beschreiben Sie die Arbeitsweise der Turingmaschine „Reverse“!
- Geben Sie die Turingtafel von „Reverse“ an! Verwenden Sie nach Möglichkeit nicht mehr als fünf Zustände!
- Geben Sie den Zeitaufwand einer Berechnung von „Reverse“ in Abhängigkeit von der Eingabelänge an!

Aufgabe 2: Chomsky-Hierarchie

Sei $L = \{a^n b^n c^n | n \geq 0\}$!

- Beweisen Sie mittels Pumping-Lemma, dass L nicht kontextfrei ist!
- Zeigen Sie, dass das Komplement von L kontextfrei ist!
- Zeigen Sie, dass das Komplement von L nicht regulär ist!

Aufgabe 3: Entscheidbarkeit

Das Wortproblem lautet: Kann für eine Sprache immer entschieden werden, ob ein Wort ein Element der Sprache ist oder nicht? Begründen Sie, für welche der folgenden Sprachklassen der Chomsky-Hierarchie das Wortproblem entscheidbar ist!

- Reguläre Sprachen (Typ 3)
- Kontextfreie Sprachen (Typ 2)
- Kontextsensitive Sprachen (Typ 1)
- Rekursiv aufzählbare Sprachen (Typ 0)

Aufgabe 4: Komplexitätsklassen

Ordnen Sie das Wortproblem (siehe Aufgabe 3) in Komplexitätsklassen ein, sofern es für den betreffenden Sprachtyp entscheidbar ist. Skizzieren Sie zur Begründung das jeweilige Entscheidungsverfahren.

- a) Reguläre Sprachen (Typ 3)
- b) Kontextfreie Sprachen (Typ 2)
- c) Kontextsensitive Sprachen (Typ 1)
- d) Rekursiv aufzählbare Sprachen (Typ 0)

Aufgabe 5

Mit Hilfe des Java-Programms DezHex werden dezimale Ganzzahlen in hexadezimaler Schreibweise (auch hexadekadisch bzw. sedezimal genannt) dargestellt. Die Eingabewerte werden dabei fortgesetzt ganzzahlig durch 16 dividiert, der jeweilige Rest (≥ 0) ist der Wert einer Ziffer der gesuchten Hexadezimalzahl. Ergänzen Sie die Klasse Hex durch eine rekursive Methode revout (int z), welche die gesuchte Hexadezimalzahl ausgibt. Die Hexadezimalzeichen sind dabei in der korrekten Reihenfolge auszugeben.

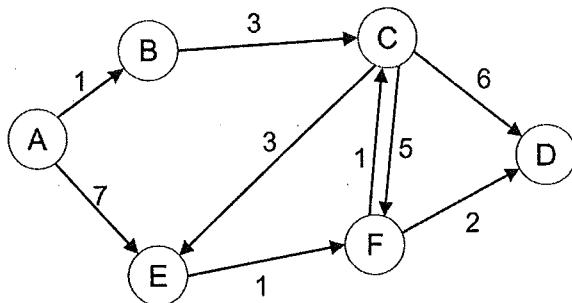
Quelltext DezHex

```
1  class Hex {  
2      char[] ziffern = {'0','1','2','3','4','5','6','7',  
3                      '8','9','A','B','C','D','E','F'};  
4      int h = 16;  
5  
6      // Methode revout()  
7      ...  
8  }  
9  
10 class Converter {  
11     public static void main(String[] args) throws IOException {  
12         int zahl;  
13         BufferedReader stdin =  
14             new BufferedReader(new InputStreamReader(System.in));  
15         String inData;  
16  
17         System.out.println("Dezimalzahl eingeben:");  
18         inData = stdin.readLine();  
19         zahl = Integer.parseInt(inData);  
20  
21         Hex hex = new Hex();  
22         hex.revout(zahl);  
23     }  
24 }
```

Fortsetzung nächste Seite!

Aufgabe 6

Gegeben sei der folgende gerichtete Graph $G = (V, E, d)$ mit den angegebenen Kantengewichten.



- Geben Sie eine formale Beschreibung des abgebildeten Graphen G durch Auflistung von V , E und d an.
- Erstellen Sie die Adjazenzmatrix A zum Graphen G.
- Berechnen Sie unter Verwendung des Algorithmus nach Dijkstra – vom Knoten A beginnend – den kürzesten Weg, um alle Knoten zu besuchen.
Die Restknoten werden in einer Halde (engl. Heap) gespeichert. Geben Sie zu jedem Arbeitsschritt den Inhalt dieser Halde an.

Thema Nr. 2

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 $x > y$ ist.

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 die Objekte der Datentypen leer.

In welcher Reihenfolge werden die Zahlen bei der nachfolgenden Sequenz

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

Gegeben sind die Funktionen

$f: \mathbb{N} \rightarrow \mathbb{N}$, $f(n) = 2 n \log n$ und

$g: \mathbb{N} \rightarrow \mathbb{N}$, $g(n) = n^3$.

Zeigen Sie, dass $f \in O(g(n))$.

Aufgabe 3: binäre Suchbäume

Fügen Sie nacheinander die Zahlen 10 15 22 8 6 20 7
in einen anfangs leeren binären Suchbaum ein und zeichnen Sie den Suchbaum!

Aufgabe 4: AVL-Bäume

- a) Fügen Sie nacheinander die Zahlen 10 15 22 8 6 20 7 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 wieder gelöscht werden. Was muss man tun?
Beschreiben Sie dies informell!

Aufgabe 5: reguläre Mengen

Gegeben sei die Menge L aller Worte $w \in \{a,b\}^*$, bei denen das erste und das zweite Zeichen gleich sind oder w das leere Wort (λ oder ϵ) ist.

- a) Geben Sie einen regulären Ausdruck für L an!
b) Geben Sie einen deterministischen endlichen Automaten für L an.

Aufgabe 6: Abschlusseigenschaften

Zeigen Sie durch die Konstruktion entsprechender Automaten, Grammatiken oder Ausdrücke, dass die regulären Mengen abgeschlossen sind

- (i) unter Vereinigung.
(ii) unter Komplementbildung.

Aufgabe 7: kontextfreie Sprachen

Zeigen Sie durch die Konstruktion einer kontextfreien Grammatik oder eines Kellerautomaten, dass die Sprache L kontextfrei ist.

$$L = \{w = a^p b^q c^r \mid p, q, r \geq 1, p=q \text{ oder } q=r\}$$

Fortsetzung nächste Seite!

Aufgabe 8: Turingmaschinen

Konstruieren Sie eine deterministische Turingmaschine M für die Sprache

$$L = \{w = a^p b^q c^r \mid p, q, r \geq 1, p=q \text{ oder } q=r\}.$$

Beschreiben Sie informell, wie Ihre Turingmaschine arbeitet. (M liest die Zeichen a und schreibt ...)

Aufgabe 9: P & NP

Das Hamilton-Kreis-Problem ist die Frage, ob es in einem ungerichteten Graph $G = (V, E)$ einen einfachen Kreis K gibt, der jeden Knoten genau einmal durchläuft (wobei der Start- und der Endknoten gleich sind).

Erläutern Sie, warum dieses Problem in NP ist.

(Es ist nicht gefordert zu zeigen, dass dieses Problem NP-schwer ist).

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2013

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

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

**Frühjahr
2013**

46115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen
— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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

Anzahl der Druckseiten dieser Vorlage: **7**

Bitte wenden!

Thema Nr. 1

Aufgabe 1:
Reguläre Sprachen

Sei L die Menge aller Worte der Länge ≥ 2 über dem Alphabet $\{a,b\}$, bei denen das zweite und das zweitletzte Zeichen identisch sind.

- Geben Sie alle Worte der Länge ≤ 4 von L an.
- Zeigen Sie, dass L regulär ist.

Aufgabe 2:
Pumping Lemma

- Formulieren Sie das Pumping Lemma für reguläre Sprachen.
- Zeigen Sie mit Hilfe des Pumping Lemmas, dass die Sprache $\{a^{i+j} b^i c^j \mid i, j \geq 0\}$ nicht regulär ist.

Aufgabe 3:
Kontextfreie Sprachen

Zeigen Sie, dass die Sprache $L = \{a^i b^p c^q d^j \mid i, j, p, q \geq 0 \text{ und } i=j \text{ und } p \geq q\}$ kontextfrei ist.

Wenn Sie eine passende Grammatik G / einen passenden Automaten M angegeben haben, brauchen Sie nicht zu beweisen, dass $L=L(G)$ bzw. $L=L(M)$ ist.

Aufgabe 4:
Turingmaschinen

Sei $L = \{u v \mid u \in \{a,b\}^*, v \in \{c,d\}^*, \#_a(u) = \#_c(v) \text{ und } \#_b(u) = \#_d(v)\}$
wobei $\#_a(u)$ die Anzahl der in u vorkommenden a 's ist.

- Geben Sie eine Turingmaschine M an, die L erkennt.
Beschreiben Sie in Worten, wie Ihre Turingmaschine arbeitet.
- Welche Laufzeit (Zeitkomplexität) hat Ihre Turingmaschine (in O-Notation).
Begründen Sie Ihre Angabe auf der Grundlage der Beschreibung.

Aufgabe 5:
O-Notation

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

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

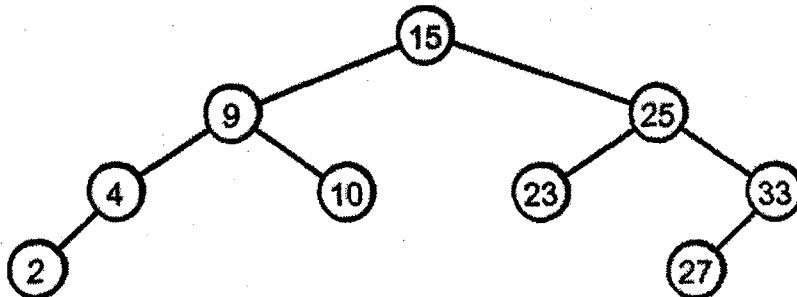
Fortsetzung nächste Seite!

Aufgabe 6:**Sortieren**

- Erläutern Sie, wie Bubblesort arbeitet.
- Sortieren Sie die folgende Liste von Zahlen mit Bubblesort und geben Sie die Zwischenergebnisse nach jeder Runde an.
2,10,3,8,1,9,5,4,6,7
- Für welche Eingaben ist die Laufzeit von Bubblesort maximal (der worst case)?

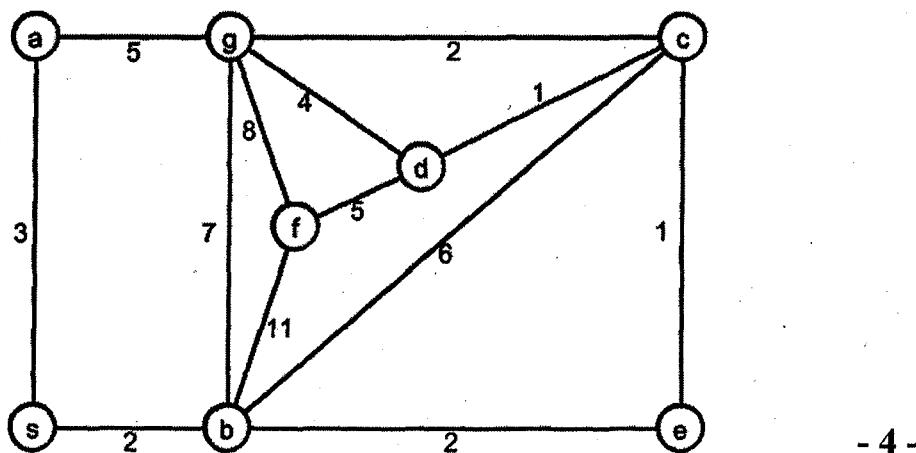
Aufgabe 7:**AVL-Bäume**

Gegeben sei der folgende AVL-Baum T. Führen Sie auf T folgende Operationen durch:
 Fügen Sie zunächst den Wert 1 in T ein. Fügen Sie anschließend den Wert 26 in T ein.
 Balancieren Sie nach jeder Operation falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.

**Aufgabe 8:****Minimaler Spannbaum**

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

Berechnen Sie mit einem Algorithmus Ihrer Wahl einen minimalen Spannbaum von G.
 Verwenden Sie den Knoten s als Startknoten.



Thema Nr. 2**Aufgabe 1:**

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

- a) Sei

$$L_1 = \{0^n\$1^{2n} \mid n \in \mathbb{N}_0\} \cup \{0^{3n}\$1^n \mid n \in \mathbb{N}_0\}.$$

Beispiele: $00\$1111 \in L_1$, $\$ \in L_1$ $000\$1 \in L_1$.

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

(a2) 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 Σ . Dann gibt es ...“

- c) Zeigen Sie, dass die Sprache L_1 die Eigenschaften des Pumping-Lemmas für kontextfreie Sprachen erfüllt.

Aufgabe 2:

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

$$L_2 = \{a^i \mid i \in \mathbb{N}, i \text{ ist durch } 3, \text{ aber nicht durch } 2 \text{ teilbar}\}$$

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

- b) Beweisen oder widerlegen Sie die folgende Behauptung:

Beh.: Ist die Sprache L regulär, dann ist auch jede echte Teilmenge von L regulär.

- c) Zu einem Wort $w = a_1a_2 \dots a_n$ sei

$$\text{spiegelung}(w) = \{a_n \dots a_1\}.$$

Außerdem sei $\text{spiegelung}(\varepsilon) = \{\varepsilon\}$.

Beweisen oder widerlegen Sie die folgende Behauptung:

Beh.: Ist die Sprache L regulär, dann ist auch

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

regulär.

Fortsetzung nächste Seite!

Aufgabe 3: „Binärer Suchbaum“

Ein *Suchbaum* ist ein binärer Baum mit der Eigenschaft, dass für jeden Knoten n alle Datenelemente im linken Teilbaum kleiner und alle Elemente im rechten Teilbaum größer sind als das Element des Knotens n.

- Fügen Sie in einen anfangs leeren binären Suchbaum der Reihe nach die Datenelemente 7, 0, 9, 11, 18, 4, 5, 3, 13, 24, 2 ein. Zeichnen Sie nur den Endzustand.
- Geben Sie eine Einfügereihenfolge (keinen Baum) für obige Zahlen an, bei der die Höhe *minimal* wird.
- Geben Sie eine weitere Einfügereihenfolge dieser Datenelemente an, bei der die Höhe des entstehenden Baums *maximal* wird.
- Fügen Sie die Zahlen aus Teilaufgabe a) in der gegebenen Reihenfolge in einen anfangs leeren AVL-Baum ein. Zeichnen Sie dazu nur den jeweils entstandenen Baum zu folgenden „Zeitpunkten“:
 - Unmittelbar vor dem Einfügen der Zahl 18
 - Nach dem Einfügen der Zahl 18 jeweils vor und nach der Rotation
 - Nach dem Einfügen aller Zahlen (also den vollständigen AVL-Baum)
- Fügen Sie nun Ihrem vollständigen AVL-Baum aus der letzten Teilaufgabe noch die Zahl 30 hinzu und geben Sie den schließlich entstandenen AVL-Baum an.

Aufgabe 4: „Streuspeicherung“

Die Werte 7, 0, 9, 11, 18, 4, 5, 3, 13, 24, 2 sollen in eine Hashtabelle der Größe 11 (Fächer 0 bis 10) eingetragen werden. Die zur Hashfunktion $h(x) = (7 \cdot x) \% 11$ gehörenden Schlüssel sind in der folgenden Tabelle bereits ausgerechnet:

x	7	0	9	11	18	4	5	3	13	24	2
$h(x)$	5	0	8	0	5	6	2	10	3	3	3

- Fügen Sie die oben genannten Schlüssel in der vorgegebenen Reihenfolge in einen Streuspeicher ein, welcher zur Kollisionsauflösung verkettete Listen verwendet, und stellen Sie die endgültige Streutabelle dar.
- Fügen Sie die gleichen Schlüssel mit linearem Sondieren bei Schrittweite +1 zur Kollisionsauflösung in eine neue Hash-Tabelle ein. Geben Sie für jeden Schlüssel an, auf welche Felder beim Einfügen zugegriffen wird und ob Kollisionen auftreten. Geben Sie die gefüllte Streutabelle an.
- Wie hoch ist der “Load”-Faktor (die Belegung) der Hashtabelle aus a) bzw. b) in Prozent? Können Sie weitere Schlüssel einfügen?
- Würden Sie sich bei dieser Zahlensequenz für das Hashing-Verfahren nach a) oder nach b) entscheiden? Begründen Sie kurz Ihre Entscheidung.

Fortsetzung nächste Seite!

Aufgabe 5: „Graphen“

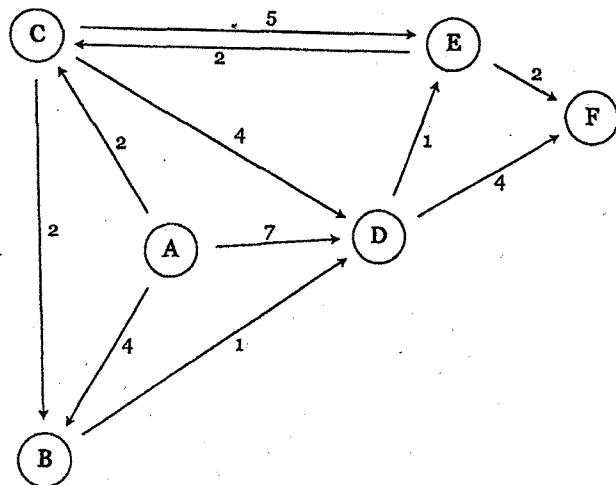
Gegeben seien folgende ungerichtete Graphen in textueller Notation, wobei die erste Menge die Menge der Knoten und die zweite Menge die Menge der Kanten ist:

$$G_1 = (\{1,2,3,4,5,6\}, \{\{1,2\}, \{1,4\}, \{2,3\}, \{3,4\}, \{4,5\}, \{4,6\}, \{5,6\}\})$$

$$G_2 = (\{1,2,3,4,5,6\}, \{\{1,2\}, \{1,3\}, \{2,3\}, \{4,5\}, \{4,6\}, \{5,6\}\})$$

$$G_3 = (\{1,2,3,4,5,6\}, \{\{1,2\}, \{1,6\}, \{2,3\}, \{2,5\}, \{3,4\}, \{4,5\}, \{5,6\}\})$$

- Zeichnen Sie die obigen Graphen.
- Erstellen Sie zu jedem Graphen die zugehörige Adjazenzmatrix mit X als Symbol für eine eingetragene Kante.
- Für welche(n) der drei Graphen existiert ein Eulerzyklus? Begründen Sie kurz, wieso es bei dem bzw. den anderen Graphen keinen Eulerzyklus gibt.
- Betrachten Sie nun folgenden gerichteten Graphen G_4 :



Bestimmen Sie die kürzeste Entfernung vom Knoten A zu jedem anderen Knoten des Graphen. Verwenden Sie dazu den Algorithmus von Dijkstra und tragen Sie Ihre einzelnen Rechenschritte in eine Tabelle folgender Form ein (schreiben Sie neben jede Zeile die Prioritätswarteschlange der noch zu bearbeitenden Knoten, priorisiert nach ihren Wegkosten):

A	B	C	D	E	F	Warteschlange
0	∞	∞	∞	∞	∞	A
...						

- Nennen Sie einen geeigneten Algorithmus, um einen minimalen Spannbaum für den obigen Graphen G_4 zu ermitteln und wenden Sie diesen schrittweise (z.B. tabellarisch) an.

Aufgabe 6: Sortierverfahren“

- a) Vervollständigen Sie die folgende Sortierung mit MergeSort (Sortieren durch Mischen) – beginnen Sie dabei Ihren „rekursiven Abstieg“ immer im linken Teilfeld:

D	40	5	89	95	85	84		14	25	20	52	7	71	
----------	----	---	----	----	----	----	--	----	----	----	----	---	----	--

Notation: Markieren Sie Zeilen mit **D(ivide)**, in denen das Array zerlegt wird, und mit **M(erge)**, in denen Teilarrays zusammengeführt werden. Beispiel:

D	82	89	44	
D	82	89	44	
M	82	44	89	
M	44	82	89	

- b) Sortieren Sie mittels HeapSort (Haldensortierung) die folgende Liste weiter:

I	99	63	91	4	36	81	76	99	
R	76	...							

Notation: Markieren Sie die Zeilen wie folgt:

I: Initiale Heap-Eigenschaft hergestellt (größtes Element am Anfang der Liste).

R: Erstes und letztes Element getauscht und letztes „gedanklich entfernt“.

S: Erstes Element nach unten „versickert“ (Heap-Eigenschaft wiederhergestellt).

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2014

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

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

**Frühjahr
2014**

46115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen
— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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

Anzahl der Druckseiten dieser Vorlage: **6**

Bitte wenden!

Thema Nr. 1

Annahmen:

Sie dürfen als bekannt und bewiesen voraussetzen:

Die Sprache $\{a^n b^n \mid n \geq 1\}$ ist nicht regulär

Die Sprache $\{a^n b^n c^n \mid n \geq 1\}$ ist nicht kontext-frei.

Um zu zeigen, dass eine Sprache L regulär (kontextfrei) ist, reicht die Angabe einer entsprechenden Beschreibung (Automat, Grammatik, Ausdruck).

Sie müssen nicht mehr zeigen, dass Ihre Beschreibung korrekt ist und genau die vorgegebene Sprache beschreibt.

Aufgabe 1: Reguläre Mengen

Sei L die Menge aller Wörter über dem Alphabet $\{a,b\}$, bei denen das erste, zweite und letzte Zeichen übereinstimmen.

- Geben Sie alle Worte bis zur (einschließlich) Länge drei an!
- Beschreiben Sie L
 - durch einen regulären Ausdruck
 - durch einen deterministischen endlichen Automaten A !

Aufgabe 2: Regulär oder nicht

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

Ist L regulär oder nicht?

Begründen Sie Ihre Antwort durch die Angabe einer passenden Beschreibung für L oder den Nachweis, dass L nicht regulär sein kann!

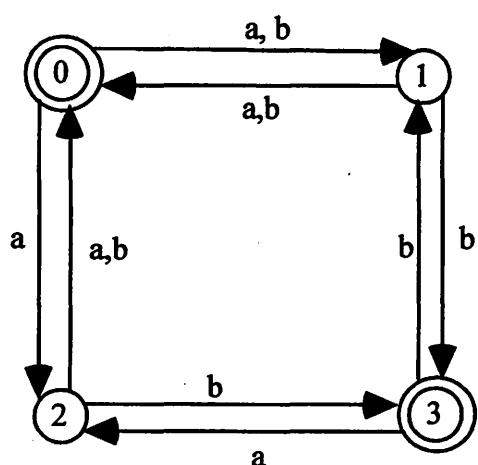
Aufgabe 3: NFA und DFA

Konstruieren Sie zum angegebenen nicht-deterministischen Automaten

$$A = (\{0,1,2\}, \{a,b\}, \delta, 0, \{0,3\})$$

einen äquivalenten deterministischen endlichen Automaten!

A hat die Zustände $0,1,2,3$; 0 ist der Startzustand und $0,3$ sind Endzustände.



Fortsetzung nächste Seite!

Aufgabe 4: Kontextfrei

Zeigen Sie, dass die Sprache L kontext-frei ist!

$$L = \{ a^n b^k c^k b^m c^m a^n \mid k, m, n \geq 1 \}$$

Aufgabe 5: Berechenbarkeit und Komplexität

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

- 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!

Aufgabe 6: O-Notation

Gegeben sind die Funktionen f und g (über den natürlichen Zahlen):

$$g(n) = 100(\sqrt{n} + 1)^2 \text{ und } f(n) = n^3 - 2n + 3$$

Zeigen Sie dass $g \in O(f)$ gilt!
(Es reicht nicht zu sagen, dass eine Funktion stärker steigt).

Aufgabe 7: Heap und binärer Suchbaum und AVL Baum

- a) Fügen Sie nacheinander die Zahlen 11, 1, 2, 13, 9, 10, 7, 5
- in einen leeren binären Suchbaum
und zeichnen Sie den Suchbaum jeweils nach dem Einfügen von „9“ und „5“
 - in einen leeren Min-Heap ein, der bzgl. „ \leq “ angeordnet ist
und geben Sie den Heap nach „9“ und nach „5“ an
 - in einen leeren AVL- Baum ein!
Geben Sie den AVL Baum nach „2“ und „5“ an und
beschreiben Sie die ggf. notwendigen Rotationen beim Einfügen dieser beiden Elemente!

Aufgabe 8: Minimaler Spannbaum

Bestimmen Sie einen minimalen Spannbaum für einen ungerichteten Graphen, der durch die nachfolgende Entfernungsmatrix gegeben ist!

Die Matrix ist symmetrisch und „ ∞ “ bedeutet, dass es keine Kante gibt.

Geben Sie die Spannbaumkanten ein!

Wie groß (Wert) ist der minimale Spannbaum?

	A	B	C	D	E	F	G	H
A	0	8	-1	∞	8	∞	7	∞
B	8	0	∞	2	∞	∞	∞	9
C	-1	∞	0	5	8	1	7	∞
D	∞	2	5	0	6	6	∞	∞
E	8	∞	8	6	0	6	3	∞
F	∞	∞	1	6	6	0	11	4
G	7	∞	7	∞	3	11	0	5
H	∞	9	∞	∞	∞	4	5	0

Thema Nr. 2**Aufgabe 1:**

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

a) Sei

$$L_1 = \{0^n\$1^{2n} \mid n \in \mathbb{N}_0\} \cup \{0^{2n}\$1^n \mid n \in \mathbb{N}_0\} .$$

Beispiele: $00\$1111 \in L_1$, $\$ \in L_1$, $00\$1 \in L_1$.

- a1) Zeigen Sie, dass L_1 kontextfrei ist, indem Sie eine kontextfreie Grammatik G angeben mit $L(G) = L_1$, und
 a2) begründen Sie, warum Ihre Grammatik *genau* die Sprache L_1 erzeugt.

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

„Sei L eine reguläre Sprache über dem Alphabet Σ . Dann gibt es ...“

c) Zeigen Sie mittels Pumping-Lemma, für reguläre Sprachen, dass die Sprache L_1 nicht regulär ist.

Aufgabe 2:

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

$L_2 = \{a^i \mid i \in \mathbb{N}, i \text{ ist durch } 2, \text{ aber nicht durch } 3 \text{ teilbar}\}$
 akzeptiert. ($\mathbb{N} = \{1, 2, \dots\}$)

b) Beweisen oder widerlegen Sie die folgende Behauptung:

„Ist die Sprache L nicht regulär, dann ist auch jede echte Teilmenge von L nicht regulär.“

c) Für $n \geq 2$ sei zu einem Wort $w = a_1a_2\dots a_n$:

$$\text{kopflos}(w) = \{a_2 \dots a_n\} .$$

Außerdem sei $\text{kopflos}(a) = \{\varepsilon\}$ und $\text{kopflos}(\varepsilon) = \emptyset$.

Beweisen oder widerlegen Sie die folgende Behauptung:

„Ist die Sprache L regulär, dann ist auch

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

regulär.“

Fortsetzung nächste Seite!

Aufgabe 3:

- a) Fügen Sie die Zahlen 17, 7, 21, 3, 10, 13, 1, 5 nacheinander in der vorgegebenen Reihenfolge in einen binären Suchbaum ein und zeichnen Sie das Ergebnis!
- b) Implementieren Sie in einer objektorientierten Programmiersprache oder in entsprechendem Pseudocode eine rekursiv festgelegte Datenstruktur, deren Gestaltung sich an folgender Definition eines binären Baumes orientiert!

Ein binärer Baum ist entweder ein leerer Baum oder besteht aus einem Wurzelement, das einen binären Baum als linken und einen als rechten Teilbaum besitzt.

Bei dieser Teilaufgabe können Sie auf die Implementierung von Methoden (außer ggf. notwendigen Konstruktoren) verzichten!

- c) Beschreiben Sie durch Implementierung in einer gängigen objektorientierten Programmiersprache oder in entsprechendem Pseudocode, wie bei Verwendung der obigen Datenstruktur die Methode *loescheKnoten(w)* gestaltet sein muss, mit der der Knoten mit dem Eintrag *w* aus dem Baum entfernt werden kann, ohne die Suchbaumeigenschaft zu verletzen!

Aufgabe 4:

Für Binomialkoeffizienten $\binom{n}{k}$ gelten neben den grundlegenden Beziehungen $\binom{n}{0} = 1$ und $\binom{n}{n} = 1$ auch folgende Formeln:

$$A) \binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$$

$$B) \binom{n}{k} = \binom{n-1}{k-1} \cdot \frac{n}{k}$$

- a) Implementieren Sie unter Verwendung von Beziehung A) eine rekursive Methode *binRek(n, k)* zur Berechnung des Binomialkoeffizienten in einer objektorientierten Programmiersprache oder entsprechendem Pseudocode!
- b) Implementieren Sie unter Verwendung von Beziehung B) eine iterative Methode *binIT(n, k)* zur Berechnung des Binomialkoeffizienten in einer objektorientierten Programmiersprache oder entsprechendem Pseudocode!
- c) Geben Sie die Laufzeitkomplexität der Methoden *binRek(n, k)* und *binIT(n, k)* aus den vorhergehenden beiden Teilaufgaben in O-Notation an!

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Herbst 2014

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	Herbst 2014	46115
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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:

Sei L die Menge der nicht-leeren Wörter über dem Alphabet $\{a, b, c\}$, bei denen das erste und das letzte Zeichen verschieden sind.

- Geben Sie einen regulären Ausdruck an, der L beschreibt.
- Geben Sie einen deterministischen endlichen Automaten A an mit $L(A) = L$. Hier reicht z. B. das Zustandsdiagramm.

Aufgabe 2:

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

- Geben Sie eine textuelle Beschreibung für L der Form „ L besteht aus allen Wörtern, die...“.
- Zeigen oder widerlegen Sie, dass L regulär ist.
Begründen (Beweisen) Sie Ihre Antwort durch die Angabe einer entsprechenden Beschreibung für L oder zeigen Sie, dass L nicht regulär sein kann.

Aufgabe 3:

Zeigen Sie, dass es zu jedem deterministischen endlichen Automaten (DFA) A einen DFA B gibt mit $L(B) = \Sigma^* - L(A)$, d.h., B erkennt das Komplement von A .

Aufgabe 4:

Zeigen Sie, dass die Sprache $L = \{a^k b^k a^m b^m a^n b^n \mid k, n, m \geq 1\}$ kontextfrei ist.

Aufgabe 5:

Geben Sie eine Turingmaschine M an, die die Sprache $L = \{a^{n^2} \mid n \geq 1\}$ erkennt und beschreiben Sie in Worten, wie Ihre Turingmaschine arbeitet.

Tipp: n^2 ist die Summe der ungeraden Zahlen von 1 bis $2n - 1$.

Aufgabe 6:

Fügen Sie nacheinander die Zahlen 10, 4, 20, 12, 5, 7, 1, 2, 8, 9

- in einen leeren binären Suchbaum ein und zeichnen Sie den Suchbaum.
- in einen leeren AVL Baum ein.
Beschreiben (zeichnen) Sie den AVL-Baum nach dem Einfügen von 5 und am Ende.
Beschreiben Sie, wann ggf. rotiert werden muss, und geben Sie die Rotationen an.

Aufgabe 7:

In einen bzgl. \leq angeordneten leeren Min-Heap werden nacheinander

- a) die Zahlen 1, 18, 3, 20, 22, 7, 30, 26, 16 eingefügt.

Geben Sie den Min-Heap vor und nach dem Einfügen von 16 an.

- b) Anschließend wird mit `deletemin()` die 1 gelöscht.

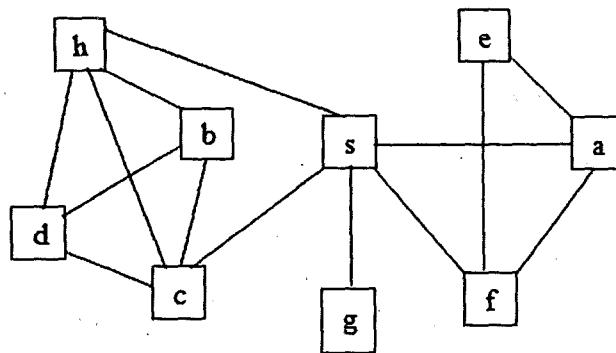
Beschreiben Sie, wie das realisiert wird, und geben Sie den Heap danach an.

Aufgabe 8:

Führen Sie auf dem folgenden ungerichteten Graphen G eine Tiefensuche ab dem Knoten s aus. Unbesuchte Nachbarn eines Knotens sollen dabei in alphabetischer Reihenfolge abgearbeitet werden.

Die Tiefensuche soll auf Basis eines Stacks implementiert werden.

Geben Sie die Reihenfolge der besuchten Knoten, also die dfs-number der Knoten, und den Inhalt des Stacks in jedem Schritt an.

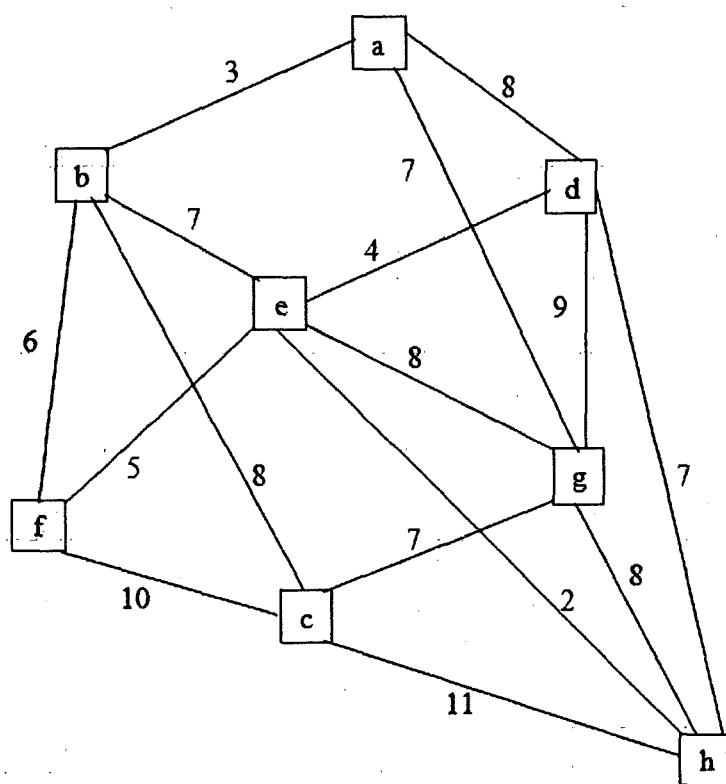


Aufgabe 9:

Berechnen Sie einen minimalen Spannbaum im Graphen G.

Markieren Sie die zum Spannbaum gehörenden Kanten.

Welchen Wert hat der Spannbaum?



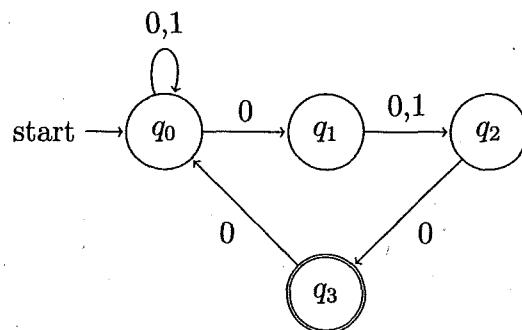
Thema Nr. 2
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

Reguläre Sprachen I

- a) Sei $\Sigma = \{a, b\}$. Für ein $w \in \Sigma^*$ und $x \in \Sigma$ bezeichnen wir mit $|w|_x$ die Anzahl der x in w . Sei $L = \{w \in \Sigma^* \mid |w|_a \text{ ungerade}\}$.
 - i) Geben Sie einen regulären Ausdruck α mit $L(\alpha) = L$ an.
 - ii) Eine Grammatik heißt *linkslinear*, wenn es für jede Produktion Nichtterminale A, B und ein Terminalsymbol x gibt, so dass die Produktion die Form $A \rightarrow Bx$ oder $A \rightarrow \varepsilon$ hat. Geben Sie eine linkslineare Grammatik für die Sprache L an.
- b) Geben Sie einen deterministischen endlichen Automaten an, der die gleiche Sprache erkennt, wie der unten aufgeführte nichtdeterministische Automat. Erläutern Sie, wie Ihre Konstruktion mit dem ursprünglichen Automaten zusammenhängt.



Aufgabe 2:

Reguläre Sprachen II

- a) Erläutern Sie kurz, warum reguläre Sprachen unter Schnitt abgeschlossen sind!

Sei nun $\Sigma = \{a, b, c\}$. Sie dürfen als gegeben annehmen, dass die Sprache $L_0 = \{a^i b^i \mid i \in \mathbb{N}\}$ nicht regulär ist.

- b) Ist $L_b = \{a^i b^j c^j \mid i, j \in \mathbb{N}_0\}$ regulär? Zeigen Sie, warum (nicht)!
- c) Ist $L_c = \{a^i b^j c^k \mid i, j, k \in \mathbb{N}_0\}$ regulär? Zeigen Sie, warum (nicht)!

Aufgabe 3:**Kontextfreie Sprachen**

- a) Geben Sie eine kontextfreie Grammatik für die folgende Sprache an:

$$L = \{(ba)^n b^m a^{2n} \mid n \geq 0, m \text{ gerade}\}$$

- b) Betrachten Sie die Grammatik $G_2 = (N_2, \Sigma_2, P_2, S)$ mit $N_2 = \{S, T, U, V, W, A, B, C, D\}$ und $\Sigma_2 = \{a, b, c\}$, wobei P_2 aus den folgenden Produktionen besteht:

$$\begin{array}{lllll} S \rightarrow SS \mid AV & T \rightarrow UU \mid CW \mid d & U \rightarrow UU \mid d & V \rightarrow TB & W \rightarrow TD \\ A \rightarrow a & B \rightarrow b & C \rightarrow c & D \rightarrow d & \end{array}$$

Verwenden Sie den CYK-Algorithmus, um zu zeigen, dass das Wort $w_2 = adbb$ nicht in $L(G_2)$ enthalten ist.

Aufgabe 4:**Komplexitätstheorie**

In einem ungerichteten Graphen $G = (V, E)$ ist eine *Clique* der Größe k eine Teilmenge $V' \subseteq V$ mit $|V'| = k$, sodass alle $u, v \in V'$ benachbart sind, d.h. $\{u, v\} \in E$.

Wir betrachten das 2-Cliquenproblem 2-CLIQUE, welches für einen ungerichteten Graphen G und natürliche Zahl k die Frage stellt, ob G zwei disjunkte Cliques der Größe mindestens k enthält.

Formal ist das Problem 2-CLIQUE wie folgt definiert:

Gegeben: Ungerichteter Graph $G = (V, E)$ und natürliche Zahl $k \in \mathbb{N}$.

Problem: Existieren Cliques $V_1, V_2 \subseteq V$ mit $|V_1| \geq k$, $|V_2| \geq k$ und $V_1 \cap V_2 = \emptyset$

- a) Begründen Sie: 2-CLIQUE liegt in NP.
- b) Geben Sie eine polynomielle Reduktion von CLIQUE auf 2-CLIQUE an. Das Problem CLIQUE ist die Frage, ob ein Graph eine Clique der Größe mindestens k besitzt.
- c) Zeigen Sie: 2-CLIQUE ist NP-vollständig.

Hinweis: Sie dürfen verwenden, dass CLIQUE NP-hart ist.

Aufgabe 5:**Berechen- und Entscheidbarkeit**

Sei $\Sigma = \{0, 1\}$. Wir nehmen an, dass jedes $w \in \Sigma^*$ eine Turingmaschine kodiert und bezeichnen diese Turingmaschine mit M_w . Die von M_w berechnete Funktion bezeichnen wir mit φ_w .

Welche der folgenden Mengen sind entscheidbar? Begründen Sie kurz.

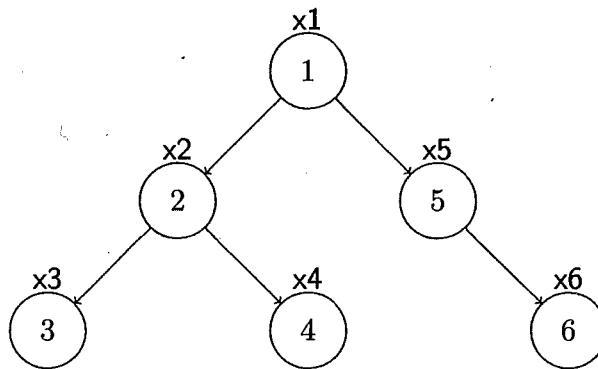
- a) $L_1 = \{w \in \Sigma^* \mid M_w \text{ besitzt mindestens } 13 \text{ Zustände}\}$
- b) $L_2 = \{w \in \Sigma^* \mid \text{es gibt } x, \text{ so dass } \varphi_w(x) = 01\}$
- c) $L_3 = \{w \in \Sigma^* \mid \exists n \in \mathbb{N} : w = 0^n 1^n 0^n\}$

Aufgabe 6:**Algorithmen und Datenstrukturen**

Für diese Aufgabe nehmen wir eine Datenstruktur für Knoten in Binäräbäumen an: Ein Knoten `node` besitzt Zeiger auf sein linkes bzw. rechtes Kind `node.left` bzw. `node.right` sowie einen Wert `node.value ∈ N`. Zeiger auf `nil` markieren Blätter.

Wir zeichnen Knoten als Kreise, in welche wir die Werte schreiben. Optional geben wir den Namen des Knoten über dem Kreis an. Pfeile symbolisieren Zeiger auf Kinder. Der Einfachheit halber ignorieren wir Blätter.

Im nachfolgend abgebildeten Baum gilt zum Beispiel `x5.left = nil`, `x5.right = x6` und `x5.value = 5`.



Die Methode `print(n)` gibt die natürliche Zahl `n` aus. Nachfolgend abgebildeter Pseudocode liefert mit Aufruf von `traverse(x1)` die Ausgabe 1, 2, 3, 4, 5, 6.

```

procedure traverse(node) {
    print node.value
    if (node.left ≠ nil)
        traverse (node.left)
    if (node.right ≠ nil)
        traverse (node.right)
}
    
```

a) Binäre Suchbäume:

- Modifizieren Sie die Werte im oben abgebildeten Baum so, dass er einen binären Suchbaum für die Werte $\{1, 2, 3, 4, 5, 6\}$ darstellt. (Zeichnen Sie diesen Baum!)
- Geben Sie eine Pseudocode-Implementierung für `procedure contains(x, n)` an, welche `true` zurückliefert, falls sich der Wert `n` in dem binären Suchbaum mit Wurzelknoten `x` befindet, und `false` andernfalls.
- Wie hängt die Laufzeit Ihrer Implementierung mit der Höhe des Suchbaums zusammen?
Hinweis: Die Höhe ist die Anzahl der Knoten auf einem längsten Pfad zu einem Blatt, im obigen Beispielbaum 3
- Wie viele Knoten kann ein binärer Suchbaum der Höhe n maximal enthalten? Wie viele minimal?

b) Baumtraversierung:

- i) Zeichnen Sie einen Baum, der sich strukturell vom oben abgebildeten Baum unterscheidet, für dessen Wurzel allerdings `traverse` (wie im oben gegebenen Beispiel) ebenfalls die Ausgabe 1, 2, 3, 4, 5, 6 liefert.
- ii) Modifizieren Sie `traverse` so zu `traverse_sorted`, dass die Werte in einem binären Suchbaum *sortiert* ausgegeben werden. Verwenden sie dabei *keinen* der Vergleichsoperatoren \leq und \geq !

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2015

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

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

**Frühjahr
2015**

46115

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

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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

Anzahl der Druckseiten dieser Vorlage: **10**

Bitte wenden!

Thema Nr. 1

Aufgabe 1: reguläre Sprachen

Gegeben sei die Sprache L.

L besteht aus der Menge aller Worte über dem Alphabet $\{a, b, c\}$,
die mit a beginnen und mit b enden
und die nie zwei aufeinander folgende c's enthalten.

- (i) Geben Sie einen regulären Ausdruck für L an.
- (ii) Geben Sie einen vollständigen deterministischen endlichen Automaten für L an.

Aufgabe 2: reguläre Sprachen

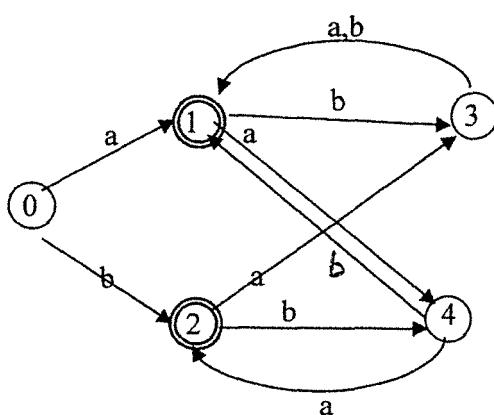
Gegeben sei die Sprache $L = \{a^n w \mid n \geq 1, w \in \{b,c\}^* \text{ und } |w| = n\}$.
Dabei bezeichnet $|w|$ die Länge des Wortes w.

- (i) Ist L regulär? Ja oder Nein?
- (ii) Beweisen Sie Ihre Antwort
 - bei JA, durch Angabe eines regulären Ausdrucks oder eines endlichen Automaten,
 - bei NEIN, durch das Pumping Lemma.

Aufgabe 3: Minimierung DFA

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

Geben Sie den minimierten DFA an.



Fortsetzung nächste Seite!

Aufgabe 4: kontextfreie Sprachen

Gegeben sei die Sprache $L = \{a^n b^n c b^m a^m \mid n, m \geq 1\}$
 Zeigen Sie, dass L kontextfrei ist.

Aufgabe 5: kontextfreie Sprachen

Gegeben sei die Grammatik $G = (\{S, X, A\}, \{a, b\}, S, P)$ mit den Produktionen $P = \{S \rightarrow AX, X \rightarrow SB, S \rightarrow ab, A \rightarrow a, B \rightarrow b\}$
 Zeigen Sie, dass das Wort aaa bbb von G erzeugt wird.

Aufgabe 6: Berechenbarkeit und Komplexität

Gegeben sei die Sprache $L = \{a^n c^{2n} \mid n \geq 1\}$

Geben Sie eine Turingmaschine M an, die L erkennt.
 Die Eingabeworte werden durch $\$$ (oder das Blanksymbol) abgeschlossen.

- Beschreiben Sie in Worten, wie Ihre Turingmaschine arbeitet.
- Konstruieren Sie M durch Angabe der Befehle und kommentieren Sie, was die Befehle in Bezug auf die Arbeitsweise von M unter a) leisten.

Aufgabe 7: O-Notation

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

$$f(n) = 0.01 n^3 + n - 1 \text{ und}$$

$$g(n) = 100(n+2)^2 + 30$$

Welche der folgenden Aussagen gilt? Beweisen Sie Ihre Angabe.

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

Aufgabe 8: Sortieren mit Quicksort

Sortieren Sie die folgende Sequenz S (im Array oder einer Liste gespeichert) von Zahlen mit Quicksort.

Als Pivotelement soll das jeweils erste Element gewählt werden.

Beschreiben Sie den Ablauf von Quicksort durch Angabe der Zwischenergebnisse nach jedem Durchlauf.

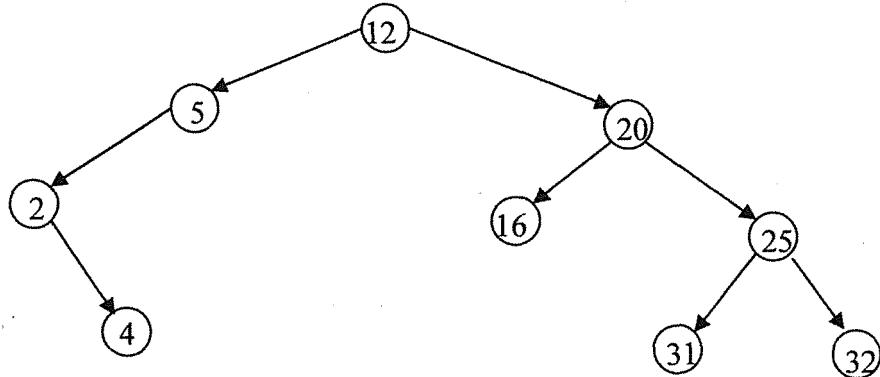
Markieren Sie das jeweils gewählte Pivotelement.

$$S = 6, 3, 17, 2, 1, 20, 4, 9, 12, 5$$

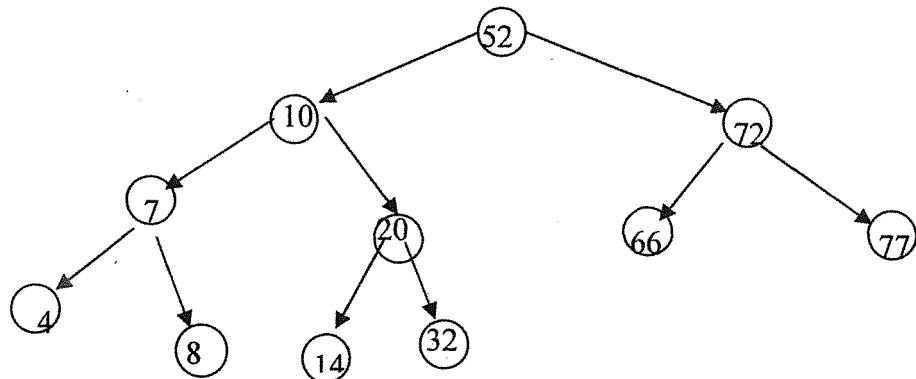
Fortsetzung nächste Seite!

Aufgabe 9: AVL-Bäume

- (a) Gegeben sei der folgende fehlerhafte AVL-Baum T.
Geben Sie die Fehler an.



- (b) Fügen Sie in den nachfolgenden AVL Baum die 2 ein und beschreiben Sie die dazu notwendigen Operationen (Rotationen).
Geben Sie den resultiererenden AVL Baum durch eine Zeichnung an.

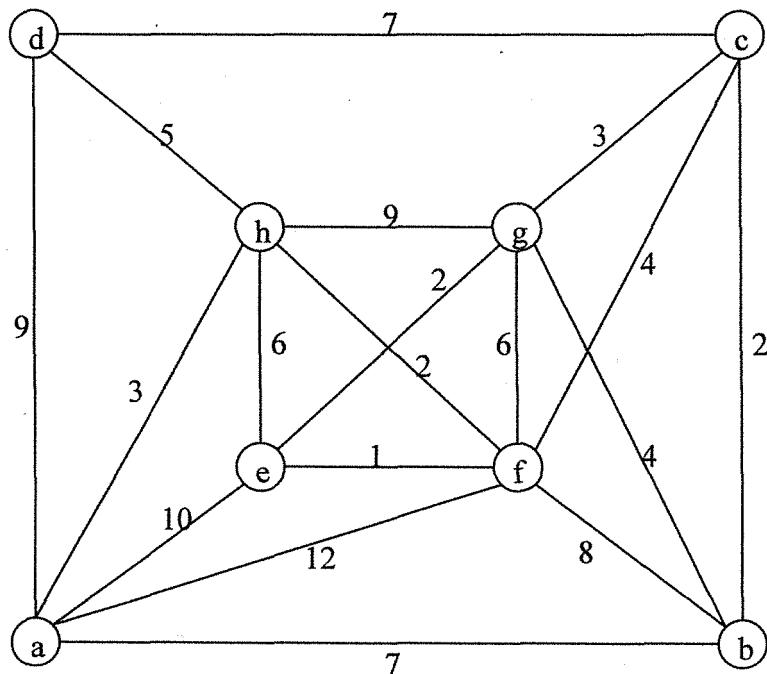


Fortsetzung nächste Seite!

Aufgabe 10: Das kürzeste Wege Problem (Dijkstra-Algorithmus)

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

Berechnen Sie die kürzesten Wege vom Startknoten a zu allen anderen Knoten mit Hilfe des Algorithmus von Dijkstra.



Thema Nr. 2

Teilaufgabe 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 2 ist. Zum Beispiel sind $00101010 \in L$, aber $11011000 \notin L$, denn nach dem Einlesen von 11011 beträgt der Unterschied 3.

1. Konstruieren Sie einen deterministischen endlichen Automaten für L .
2. Führen Sie auf Ihrem Automaten den Minimierungsalgorithmus durch. Möglicherweise ist Ihr Automat bereits minimal. In diesem Fall liefert der Algorithmus nichts Neues, muss aber trotzdem durchgeführt werden. Dokumentieren Sie Ihre Schritte geeignet.

Sei $A = (Q, \delta, q_0, E)$ ein nichtdeterministischer endlicher Automat für L . Es sei $w_1 = 11$, $w_2 = 1$, $w_3 = \varepsilon$, $w_4 = 0$, $w_5 = 00$. Machen Sie sich klar, dass der Automat jedes dieser Wörter verarbeiten können muss und dass keine zwei dieser Wörter vom Startzustand aus in denselben Zustand führen dürfen, da sonst auch falsche Wörter akzeptiert würden. Folgern Sie, dass der Automat mindestens fünf Zustände haben muss. Schreiben Sie Ihre Argumentation schlüssig und vollständig auf.

Teilaufgabe 2

Die Sprache $L = \{a^n b^n \mid n \geq 1\}$ über dem Alphabet $\Sigma = \{a, b\}$ ist bekanntlich kontextfrei. Im Allgemeinen ist das Komplement zweier kontextfreier Sprachen nicht wieder kontextfrei. In diesem speziellen Fall aber schon; wir wollen nunmehr eine Grammatik für dieses Komplement \bar{L} konstruieren.

1. Geben Sie eine kontextfreie Grammatik für L an.
2. Begründen Sie, dass $\bar{L} = L_1 \cup L_2 \cup L_3 \cup \{\in\}$, wobei $L_1 = \overline{a^* b^*}$ und $L_2 = \{a^m b^n \mid m > n\}$ und $L_3 = \{a^m b^n \mid m < n\}$.
3. Zeigen Sie nun, dass \bar{L} kontextfrei ist.

Fortsetzung nächste Seite!

Teilaufgabe 3: „Streuspeicherung“

Gegeben seien die folgenden Schlüssel s zusammen mit ihren Streuwerten h(s):

s	A	B	C	D	E
h(s)	1	0	1	0	1

- a) Fügen Sie A, B, C, D, E in dieser Reihenfolge mit der Streufunktion h(s) in eine Streutabelle der Größe 5 folgender Form ein! Verwenden Sie zur Kollisionsauflösung **verkettete Listen** (der zuletzt eingetragene Schlüssel steht unten):

Bucket >	(Beispiel)	0	1	2	3	4
Schlüssel <	X Y					

- b) Fügen Sie die Schlüssel A, B, C, D, E in dieser Reihenfolge mit der Streufunktion h(s) in eine neue Streutabelle gleicher Größe und Form ein. Verwenden Sie zur Kollisionsauflösung diesmal aber **lineares Sondieren** mit Schrittweite +2.

Bucket >	0	1	2	3	4
Schlüssel >					

- c) Ergänzen Sie die Methode **insert(s, hs)**, die einen Schlüssel s mit dem zugehörigen Streuwert hs in eine anfangs leere Streutabelle (intern hier: **buckets**) einfügt. Die Streutabelle realisiert die Kollisionsauflösung mit verketteten Listen. Beachten Sie bitte, dass jeder Schlüssel höchstens einmal in der Streutabelle vorkommen darf (d. h., ein erneutes Einfügen wird ignoriert).

```

class Streutabelle {
    LinkedList<String>[] buckets;

    Streutabelle(int size) {
        buckets = new LinkedList[size];
    }

    void insert(String s, int hs) {
        assert s != null && 0 <= hs && hs < buckets.length;
        ...
    }
}

```


Hinweis: Aus der Java-API der Klasse `LinkedList<E>` dürfen Sie verwenden:

`public LinkedList()`

Constructs an empty list.

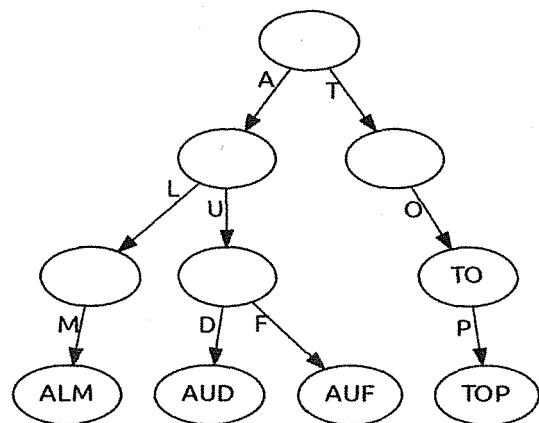
`public boolean contains(Object o)`

Returns true if this list contains the specified element.

`public void addLast(E e)`

Appends the specified element to the end of this list.

Teilaufgabe 4: “Spezielle Bäume”



Ein *Trie* ist eine Variante eines Suchbaums. Im Folgenden sei jeder Kante in einem *Trie* implizit ein Großbuchstabe zugeordnet. Ein Pfad durch den Baum stellt daher eine Zeichenkette dar. Um eine Zeichenkette einzufügen, wird sowohl die Zeichenkette als auch der *Trie* (vom Wurzelknoten aus) durchlaufen. In jedem Schritt wird die Ausgangskante des aktuellen Knotens verfolgt, die dem nächsten Zeichen im String entspricht. Falls es eine solche Kante noch nicht gibt, wird sie samt Zielknoten neu angelegt. Sind alle Zeichen abgearbeitet, wird die Zeichenkette im erreichten Knoten vermerkt. Der exemplarisch abgebildete *Trie* entstand durch das Einfügen der Wörter ALM, AUD, AUF und TO und TOP.

Gegeben ist folgender Ausschnitt der Klasse *Trie*, die schrittweise in einer beliebigen objekt-orientierten Programmiersprache Ihrer Wahl oder Pseudocode vervollständigt werden soll:

```

public class Trie {
    // bis zu 26 Kinder; null, wenn kein Kind mit zugehörigem Buchstaben:
    Trie[] children = new Trie['Z' - 'A' + 1];
    String myString; // null, wenn keine Zeichenkette zu speichern ist
  
```

- a) Implementieren Sie die iterative Methode `insert(String s)`, die den *Trie* um die nötigen Knoten erweitert und `s` im korrekten Knoten speichert:

```

void insert(String s) {
    assert s != null;
    for (char c : s.toCharArray())
        assert c >= 'A' && c <= 'Z';
    Trie curr = this;
    ...
  
```

Fortsetzung nächste Seite!

- b) Sei $n := s.length()$. Welche Laufzeitkomplexität (im O-Kalkül bzw. in Landau-Notation) hat **insert(s)**?
- c) Implementieren Sie eine **rekursive** Methode **printSorted()**. Sie soll alle im *Trie* enthaltenen Wörter in aufsteigender alphabetischer Reihenfolge ausgeben. Wörter, die Präfix eines anderen Wortes sind, sollen vor dem längeren Wort ausgegeben werden (z.B. TO vor TOP).
- d) Implementieren Sie eine **rekursive** Methode **contains(String s)**, die prüfen soll, ob s im *Trie* enthalten ist.

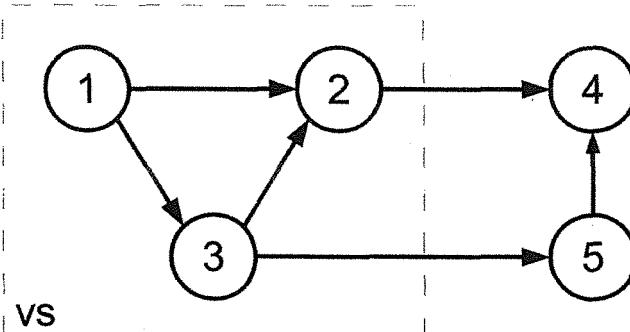
Teilaufgabe 5: „Funktionale Programmierung“

Gegeben seien gerichtete Graphen G folgender Form:

```
type V = Int // Knoten
type E = (V, V) // Kante: (Startknoten, Endknoten)
type G = List[E]

// Beispiel:
val graph = List((1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5))
```

- a) Definieren Sie in einer funktionalen Sprache Ihrer Wahl (hier exemplarisch Scala) eine Funktion **outEdges(vs, g)**, die **parallel** diejenigen Kanten in g filtert und zurückgibt, die direkt von Knoten der Knotenliste vs zu Knoten außerhalb von vs führen. Sie dürfen die Methode **contains(t: T): Boolean** von **List[T]**



verwenden.

Beispiel (siehe Bild):

```
outEdges(List(1, 2, 3), graph) == List((2, 4), (3, 5))
```

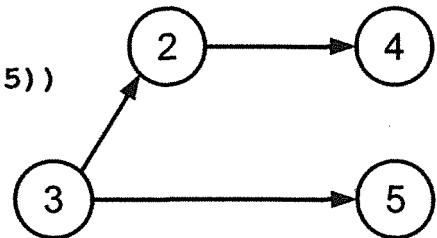
```
def outEdges: (List[V], G) => List[E] = (vs, g) => ...
```


- b) Ergänzen Sie die Funktion **tree (v, g)**, die einen beliebigen gerichteten Baum (als Kantenliste) mit Wurzel v bestimmt, welcher alle von v aus in g erreichbaren Knoten enthält. **helper(vs)** bearbeitet rekursiv (mittels **outEdges**) die bereits besuchten Knoten vs:

Beispiel (auch andere gerichtete Bäume möglich!):

```
tree(3, graph) == List((3,2), (2,4), (3,5))
```

```
def tree: (V, G) => G = (v, g) => {  
    def helper: List[V] => G = vs => ...
```



46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Herbst 2015

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	Herbst 2015	46115
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen
— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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:

Die Sprache L über dem Alphabet $\Sigma = \{a, b\}$ enthält alle Wörter, die das Wort baab oder das Wort abba oder beide enthalten. Z.B. ist $baababb \in L$, aber $babababa \notin L$.

1. Geben Sie einen regulären Ausdruck für L an.
2. Konstruieren Sie einen nichtdeterministischen endlichen Automaten mit höchstens 8 Zuständen für L .
3. Konstruieren Sie einen deterministischen Automaten für L mit der Potenzmengenkonstruktion. Sie dürfen sich wie üblich auf die erreichbaren Zustände beschränken.
4. Konstruieren Sie den Minimalautomaten für L . Sie können entweder das übliche Verfahren anwenden und Ihre Schritte geeignet dokumentieren, oder aber den Minimalautomaten direkt angeben und geeignet begründen, dass es sich um den richtigen Automaten handelt.

Aufgabe 2:

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 Σ 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 mit $i_k \leq n$ gibt, 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 $u_1 u_3 u_2 u_3 = b abb ba abb = bab bb aa bb = v_1 v_3 v_2 v_3$.

Es ist bekannt, dass das PCP ein unentscheidbares Problem ist.

1. Erläutern Sie, was es genau bedeutet, dass das PCP unentscheidbar ist.
2. Begründen Sie, dass das PCP rekursiv aufzählbar ist.
3. Seien jetzt zwei Alphabete Δ und Σ sowie Funktionen $f, g : \Delta \rightarrow \Sigma^*$ durch ihre Wertetabellen gegeben, z.B. $\Delta = \{c, d, e\}$, $\Sigma = \{a, b\}$ und $f(c) = b, f(d) = ba, f(e) = abb$. Man setzt diese Funktionen in offensichtlicher Weise ("homomorph") auf Wörter über Δ fort, z.B. $f(ccde) = bbaabb$.

Zeigen Sie durch Reduktion vom PCP, dass es unentscheidbar ist, festzustellen, ob ein Wort $w \in \Delta^+$ mit $f(w) = g(w)$ existiert.

Aufgabe 3:

Eine Folge von Zahlen a_1, \dots, a_n heiße *unimodal*, wenn sie bis zu einem bestimmten Punkt echt ansteigt und dann echt fällt. Zum Beispiel ist die Folge 1, 3, 5, 6, 5, 2, 1 unimodal, die Folgen 1, 3, 5, 4, 7, 2, 1 und 1, 2, 3, 3, 4, 3, 2, 1 aber nicht.

1. Entwerfen Sie einen Algorithmus, der zu (als Array) gegebener *unimodaler* Folge a_1, \dots, a_n in Zeit $O(\log n)$ das Maximum $\max a_i$ berechnet. Ist die Folge nicht unimodal, so kann Ihr Algorithmus ein beliebiges Ergebnis liefern. Größenvergleiche, arithmetische Operationen und Arrayzugriffe können wie üblich in konstanter Zeit ($O(1)$) getätigten werden. Hinweise: binäre Suche, divide-and-conquer.
2. Begründen Sie, dass Ihr Algorithmus tatsächlich in Zeit $O(\log n)$ läuft.
3. Schreiben Sie Ihren Algorithmus in Pseudocode oder in einer Programmiersprache Ihrer Wahl, z.B. Java, auf. Sie dürfen voraussetzen, dass die Eingabe in Form eines Arrays der Größe n vorliegt.
4. Beschreiben Sie in Worten ein Verfahren, welches in Zeit $O(n)$ feststellt, ob eine vorgelegte Folge unimodal ist oder nicht.
5. Begründen Sie, dass es kein solches Verfahren (Test auf Unimodalität) geben kann, welches in Zeit $O(\log n)$ läuft.

Thema Nr. 2
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

“Streuspeicherung“

Fügen Sie die folgenden Werte in der gegebenen Reihenfolge in eine Streutabelle der Größe 8 (mit den Indizes 0 – 7) und der Streufunktion $h(x) = x \bmod 8$ ein. Verwenden Sie die jeweils angegebene Hash-Variante bzw. Kollisionsauflösung:

15, 3, 9, 23, 1, 8, 17, 4

- a) Offenes Hashing: Zur Kollisionsauflösung wird Verkettung verwendet.

Beispiel: Für die beiden Werte 8 und 16 würde die Lösung wie folgt aussehen:

Bucket	Inhalt
0	8 - 16
1	
2	

- b) Geschlossenes Hashing: Zur Kollisionsauflösung wird lineares Sondieren (nur hochzählend) mit Schrittweite +5 verwendet. Treten beim Einfügen Kollisionen auf, dann notieren Sie die Anzahl der Versuche zum Ablegen des Wertes im Subskript (z.B. das Einfügen des Wertes 8 gelingt im 5. Versuch: „8₍₅₎“).

Beispiel: Für die beiden Werte 8 und 16 würde die Lösung wie folgt aussehen:

Bucket	Inhalt
0	8
1	
2	
3	
4	
5	16 ₍₂₎

- c) Welches Problem tritt auf, wenn zur Kollisionsauflösung lineares Sondieren mit Schrittweite 4 verwendet wird? Warum ist 5 eine bessere Wahl?
 d) Geben Sie für Ihre Lösungen zu a) und b) jeweils den Lastfaktor an.
 e) Gegeben sei eine Streutabelle, in der die gleichen Schlüssel mehrfach enthalten sein können. Welches der beiden Verfahren aus den Teilaufgaben a) und b) ist in der Worst-Case-Laufzeit performanter beim Einfügen eines neuen Eintrags? Welches Verfahren ist performanter beim Abrufen des Wertes zu einem Schlüssel? Begründen Sie kurz Ihre Antworten.

Aufgabe 2:“Abstrakte Datentypen (adt)“

Der **Abstrakte Datentyp (adt)** **AA** (`AssociativeArray`) entspricht einer Reihung vom Typ τ mit beliebiger Länge und hat folgende Eigenschaften: Der Konstruktor `create` instanziert eine neue leere Reihung. Mit den Methoden `get` und `set` kann man die Elemente nach ganzzahligen (nicht notwendigerweise positiven) Indizes auslesen bzw. mit einem Wert $\neq \text{null}$ aktualisieren (Aufrufe von `set` mit `null` sollen vom adt „ignoriert“ werden). `delete` löscht (falls vorhanden) die Belegung eines Indexes. Lesende Zugriffe auf unbelegte/gelöschte Indizes liefern `null`. Die Anzahl der tatsächlich (also mit Wert $\neq \text{null}$) belegten Indizes kann mit `size` abgefragt werden.

- a) Ergänzen Sie die fehlenden Axiome, um das vorangehend beschriebene Verhalten formal darzustellen:

```

adt AA
sorts AA, T, int
ops
  create:          → AA
  get:   AA × int    → T
  set:   AA × int × T → AA
  size:  AA           → int
  delete: AA × int    → AA
axs
  size(create) = 0

  get(create, i) = null

  delete(create, i) = create

end AA

```

- b) Ergänzen Sie die fehlenden Methoden so, dass die folgende Klasse genau das Verhalten des obigen **adt** wiedergibt. Sie dürfen dabei keine anderen Datenstrukturen verwenden, also insbesondere auch keine `HashMap` aus der Java-API.

```

public class AA<T> {
    private int key;
    private T value;
    private AA<T> tail;

    private AA(int key, T value, AA<T> tail) {
        this.key = key;
        this.value = value;
        this.tail = tail;
    }

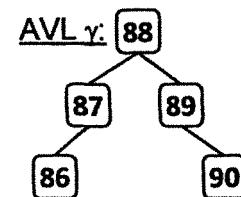
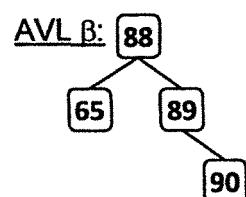
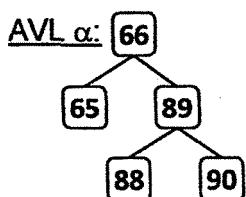
    public static <T> AA<T> create() {
        return new AA<T>(0, null, null);
    }

    // TODO: Code hier ergänzen
}

```

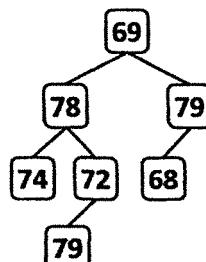
Aufgabe 3:“Bäume“

- a) Fügen Sie die Zahlen **83, 85, 67, 72, 69** in der gegebenen Reihenfolge
- in einen *linksvollständigen Binärbaum* bzw.
 - in einen *binären Suchbaum*
- ein. Stellen Sie jeweils nur das Endergebnis dar.
- b) Fügen Sie **77** in jeden der folgenden *AVL-Bäume* ein und führen Sie anschließend bei Bedarf die erforderliche(n) Rotation(en) aus. Zeichnen Sie den Baum zunächst unmittelbar nach dem Einfügen und anschließend im Endzustand nach den Rotationen.

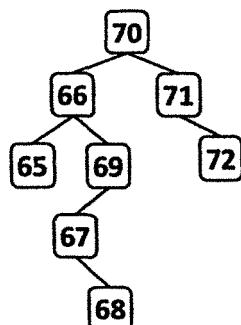


- c) Traversieren Sie den folgenden Baum in der jeweils angegebenen Art (Abstieg zuerst im linken Teilbaum) und geben Sie dabei die Knoten in der entsprechenden Reihenfolge aus:

- in-order
- post-order



- d) Löschen Sie den Knoten **70** aus dem folgenden *binären Suchbaum* und ersetzen Sie ihn dabei durch einen geeigneten Wert aus dem linken Teilbaum. Zeichnen Sie nur den Endzustand:



Aufgabe 4:"Formale Verifikation - Induktionsbeweis"

Gegeben sei die folgende Methode **function**:

```
double function(int n) {  
    if (n == 1)  
        return 0.5 * n;  
    else  
        return 1.0 / (n * (n + 1)) + function(n - 1);  
}
```

Beweisen Sie folgenden Zusammenhang mittels vollständiger Induktion:

$$\forall n \geq 1: \text{function}(n) = f(n) \text{ mit } f(n) := 1 - \frac{1}{n+1}$$

Hinweis: Eventuelle Rechenun genauigkeiten, wie z. B. in Java, bei der Behandlung von Fließkommazahlen (z. B. double) sollen beim Beweis nicht berücksichtigt werden – Sie dürfen also annehmen, Fließkommazahlen würden mathematische Genauigkeit aufweisen.

Aufgabe 5:

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

a) Sei

$$L_1 = \{w \in \Sigma^* \mid w \text{ enthält mehr } a's \text{ als } b's\} .$$

Beispiele: $aba \in L_1$, $bbbaabaabaa \in L_1$, $a \in L_1$.

Zeigen Sie, dass L_1 kontextfrei ist, indem Sie die Arbeitsweise eines Kellerautomaten beschreiben, der L_1 akzeptiert.

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

„Sei L eine reguläre Sprache über dem Alphabet Σ . Dann gibt es ...“

c) Zeigen Sie mit Hilfe des Pumping-Lemmas für reguläre Sprachen, dass die Sprache L_1 nicht regulär ist.

Aufgabe 6:

a) 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.

b) Gegeben sei das folgende Problem E :

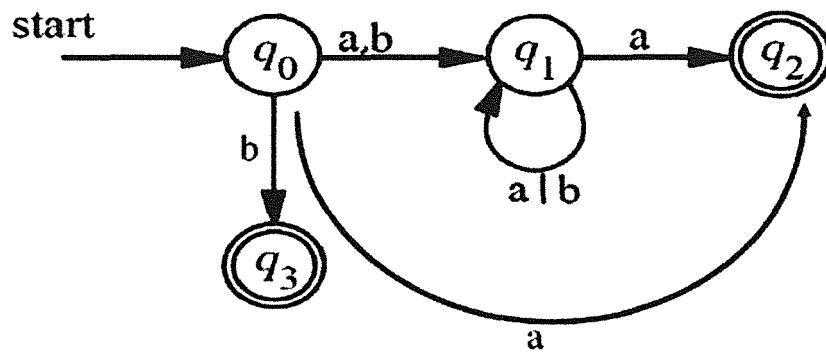
- Entscheide, ob es für die deterministische Turing-Maschine M mit der Gödelnummer $\langle M \rangle$ mindestens eine Eingabe $w \in \mathbb{N}_0$ gibt, so dass w eine Quadratzahl ist und die Maschine M gestartet mit w hält.

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 7:

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



ϵ bezeichnet das leere Wort. Konstruieren Sie zu N mit der Potenzmengen-Konstruktion einen äquivalenten deterministischen endlichen Automaten A . Zeichnen Sie nur die 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!

Hinweise:

- $\{q_0\}$ ist **nicht** der Startzustand des deterministischen endlichen Automaten.
 - In einem deterministischen endlichen Automaten darf es keine ϵ -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.: Es gibt reguläre Sprachen L , die mindestens eine echte Teilmenge U enthalten, so dass U nicht regulär ist.

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2016

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

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

**Frühjahr
2016**

46115

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

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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

Anzahl der Druckseiten dieser Vorlage: **6**

Bitte wenden!

Thema Nr. 1

Aufgabe 1: reguläre Mengen/Ausdrücke

Sei L die Sprache (Menge) aller Binärzahlen

- ohne führende Nullen,
- die durch 2 teilbar sind und
- die eine ungerade Anzahl an Nullen haben.

- a) Beschreiben Sie L durch einen regulären Ausdruck.
- b) Konstruieren Sie einen deterministischen endlichen Automaten (DFA) A für L (durch Angabe von A oder eines Diagramms für A).

Aufgabe 2: kontextfrei

Zeigen Sie, dass die Sprache $L = \{a^p b^q c^q \mid p, q > 0\}$

- a) kontextfrei ist.
- b) nicht regulär ist.

Aufgabe 3: regulär

Die Aussage: „jede Teilmenge einer regulären Menge ist regulär“ ist falsch.
Begründen Sie dies.

Aufgabe 4: Turingmaschinen

Geben Sie eine Turingmaschine M an, die die Sprache $\{w d w \mid w \in \{a,b,c\}^*\}$ akzeptiert.
Beschreiben Sie in Worten, wie M arbeitet.

Welche Laufzeit hat M auf Inputs der Länge $2n+1$ (oder $n = |w|$)?

Aufgabe 5: NP

Beschreiben Sie, was es heißt, dass ein Problem (Sprache) NP-vollständig ist.

Geben Sie ein NP-vollständiges Problem Ihrer Wahl an und erläutern Sie, dass (bzw.) warum es in NP liegt.

Aufgabe 6: Datentypen

Betrachten Sie das Verhalten von Kellern (stack), Warte-Schlangen (queue) und Prioritätswarteschlangen (heap).

Für die Prioritäten sind die Werte maßgeblich mit x vor y genau dann, wenn $x > y$.

in(x) bedeutet, dass die Zahl x eingefügt wird. out() gibt eine Zahl aus.
Am Anfang (und am Ende) sind alle Datentypen leer.

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

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

in(7), in(3), in(8), in(5), in(6), out(), out(), out(), in(2), in(1), out(), out(), out(), in(4), out(), out()?

Aufgabe 7: O-Notation

Gegeben sind die Funktionen (über den natürlichen Zahlen):

$$g(n) = 100 n \log n + 5 n + 10 \quad \text{und} \quad f(n) = n^3.$$

Zeigen Sie, dass $g \in O(f)$ gilt.
(Es reicht nicht zu sagen, dass eine Funktion stärker steigt.)

Aufgabe 8: Sortieren

- a) Sortieren Sie das Array mit den Integer Zahlen
25, 1, 12, 27, 30, 9, 33, 34, 18, 16
 - i) mit BubbleSort
 - ii) mit Quicksort,
wenn als Pivotelement das jeweils erste Element gewählt wird.

Beschreiben Sie die Abläufe der Sortierverfahren

- i) bei BubbleSort durch eine Angabe der Zwischenergebnisse nach jedem Durchlauf
 - ii) bei Quicksort durch die Angabe der Zwischenergebnisse nach den rekursiven Aufrufen.
- b) Welche Laufzeit (asymptotisch, in O-Notation) hat BubbleSort
bei beliebig großen Arrays mit n Elementen.
Begründen Sie Ihre Antwort.

Aufgabe 9: AVL-Bäume

Fügen Sie nacheinander die Zahlen 10, 18, 7, 14, 12, 19, 20

in einen anfangs leeren AVL-Baum ein.

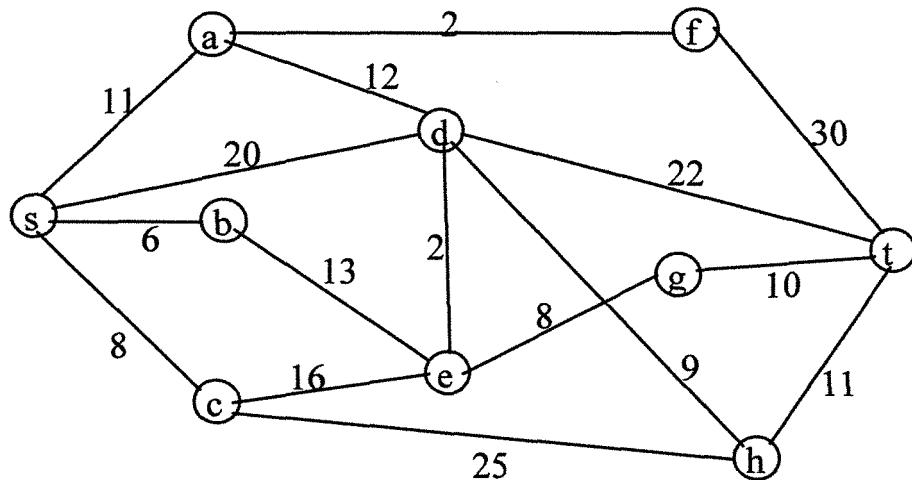
Repräsentieren (zeichnen) Sie alle AVL-Bäume jeweils vor einer notwendigen Rotation und beschreiben Sie die Rotationen (z.B. LL-Rotation für die Nummern).

Aufgabe 10: minimaler Spannbaum

Bestimmen Sie einen minimalen Spannbaum für den nachfolgend gezeichneten Graphen.

Zeichnen Sie die Spannbaumkanten ein.

Wie groß (Wert) ist der minimale Spannbaum?



Thema Nr. 2

Aufgabe 1:

Betrachten Sie die Sprache L aller Wörter über dem einelementigen Alphabet $\Sigma = \{a\}$, deren Länge eine Zweierpotenz ist, also $L = \{a, aa, aaaa, \dots\}$.

- Begründen Sie ausführlich, warum diese Sprache nicht kontextfrei ist.
- Betrachten Sie nun die folgende Grammatik G :

$$S \rightarrow a | SS$$

Begründen Sie $L \subseteq L(G)$, aber $L \neq L(G)$. Ordnen Sie $L(G)$ bestmöglich in die Chomsky-Hierarchie ein und begründen Sie Ihre Antwort geeignet.

- Bringen Sie folgende Grammatik in Chomsky-Normalform und führen Sie sodann auf der Eingabe $w = aabbb$ den Cocke-Younger-Kasami (CYK) Algorithmus durch.

$$\begin{aligned} S &\rightarrow XY \\ X &\rightarrow aXb \mid aXbb \mid ab \\ Y &\rightarrow bY \mid b \end{aligned}$$

Aufgabe 2:

Ein Raumausstattungsunternehmen steht immer wieder vor dem Problem, feststellen zu müssen, ob ein gegebener rechteckiger Fußboden mit rechteckigen Teppichresten ohne Verschnitt ausgelegt werden kann. Alle Längen sind hier ganzzahlige Meterbeträge. Haben sie beispielsweise zwei Reste der Größen 3×5 und einen Rest der Größe 2×5 , so kann ein Fußboden der Größe 8×5 ausgelegt werden.

Das Unternehmen beauftragt eine Softwarefirma mit der Entwicklung eines Programms, welches diese Frage für beliebige Größen von Fußboden und Teppichresten entscheiden soll. Bei der Abnahme weist die Softwarefirma darauf hin, dass das Programm im wesentlichen alle Möglichkeiten durchprobiert und daher für große Eingaben schnell ineffizient wird. Auf die Frage, ob man das nicht besser machen könne, lautet die Antwort, dass das vorgelegte Problem NP-vollständig sei und daher nach derzeitigem Kenntnisstand der theoretischen Informatik nicht mehr zu erwarten sei.

- Fixieren Sie ein geeignetes Format für Instanzen des Problems und geben Sie konkret an, wie die obige Beispielinstanz in diesem Format aussieht.
- Begründen Sie, dass das Problem in NP liegt.
- Begründen Sie, dass das Problem NP-schwer (=NP-hart) ist durch Reduktion vom NP-vollständigen Problem SUBSET-SUM

Aufgabe 3:

Sie sollen mithilfe von Falltests eine neue Serie von Smartphones auf Bruchsicherheit testen.

Dazu wird eine Leiter mit n Sprossen verwendet; die höchste Sprosse, von der ein Smartphone heruntergeworfen werden kann ohne zu zerbrechen, heiße "höchste sichere Sprosse". Das Ziel ist, die höchste sichere Sprosse zu ermitteln. Man kann davon ausgehen, dass die höchste sichere Sprosse nicht von der Art des Wurfs abhängt und dass alle verwendeten Smartphones sich gleich verhalten. Eine Möglichkeit, die höchste sichere Sprosse zu ermitteln, besteht darin, ein Gerät erst von Sprosse 1, dann von Sprosse 2, etc. abzuwerfen, bis es schließlich beim Wurf von Sprosse k beschädigt wird (oder Sie oben angelangt sind). Sprosse $k-1$ (bzw. n) ist dann die höchste sichere Sprosse. Bei diesem Verfahren wird maximal ein Smartphone zerstört, aber der Zeitaufwand ist ungünstig.

- a) Bestimmen Sie die Zahl der Würfe bei diesem Verfahren im schlechtesten Fall.
- b) Geben Sie nun ein Verfahren zur Ermittlung der höchsten sicheren Sprosse an, welches nur $O(\log n)$ Würfe benötigt, dafür aber möglicherweise mehr Smartphones verbraucht.
- c) Es gibt eine Strategie zur Ermittlung der höchsten sicheren Sprosse mit $O(\sqrt{n})$ Würfen, bei dessen Anwendung höchstens zwei Smartphones kaputtgehen. Finden Sie diese Strategie und begründen Sie Ihre Funktionsweise und Wurfzahl.

Tipp: der erste Testwurf erfolgt von Sprosse $\lceil \sqrt{n} \rceil$.

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Herbst 2016

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	Herbst	
Kennwort: _____	2016	46115
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 12

Bitte wenden!

Thema Nr. 1
 (Aufgabengruppe)

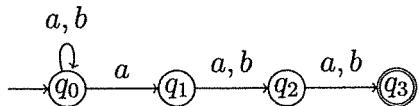
Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

Reguläre Sprachen

- a) Konstruieren Sie aus dem NEA \mathcal{A} mit der Potenzmengenkonstruktion einen (deterministischen) EA, der dieselbe Sprache akzeptiert.

\mathcal{A} :



- b) Beschreiben Sie möglichst einfach, welche Sprachen von den folgenden regulären Ausdrücken beschrieben werden:

1. $(a \mid b)^*a$
2. $(a \mid b)^*a(a \mid b)^*a(a \mid b)^*$
3. $(a \mid b)^*a(bb)^*a(a \mid b)^*$

Aufgabe 2:

Kontextfreie Grammatiken

Betrachten Sie die folgende kontextfreie Grammatik $G = (\{S, A, B, C\}, \{a, b, c\}, S, P)$ mit den Produktionen:

$$S \rightarrow AB \mid C$$

$$A \rightarrow a \mid \varepsilon$$

$$B \rightarrow BC$$

$$C \rightarrow c$$

- a) Beschreiben Sie in Worten möglichst genau die von G erzeugte Sprache $L(G)$.

- b) Geben Sie ohne Begründung an, welche Variablen in der Grammatik

- nützlich,
- erzeugend oder
- erreichbar

sind!

- c) Ist die Grammatik in Chomsky-Normalform? Begründen Sie Ihre Antwort.

Aufgabe 3:

Klassifizierung von Sprachen

Welche der folgenden Sprachen über dem Alphabet $\{a, b\}$ sind

- a) regulär?
- b) kontextfrei, aber nicht regulär?
- c) nicht kontextfrei?

1. $L_1 = \{w \in \{a\}^* \mid \text{die Länge von } w \text{ ist } 2^n \text{ für eine natürliche Zahl } n\}$
2. $L_2 = \{a^k b^\ell \mid k \leq \ell\} \cup \{a^k b^\ell \mid k \geq \ell\}$
3. $L_3 = \{a^k b^\ell \mid k < \ell\} \cup \{a^k b^\ell \mid k > \ell\}$

Geben Sie zu Ihrer Antwort jeweils einen Beweis an. Ein Beweis durch Angabe eines Automaten oder eines regulären Ausdruckes ist erlaubt. Wird der Beweis durch die Angabe einer Grammatik geführt, so ist die Bedeutung der Variablen zu erläutern.

Aufgabe 4:

Probleme und Komplexität

Betrachten Sie die beiden folgenden Probleme:

VERTEXCOVER

Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \{1, 2, 3, \dots\}$.

Frage: Gibt es eine Menge $C \subseteq V$ mit $|C| \leq k$, so dass für jede Kante (u, v) aus E mindestens einer der Knoten u und v in C ist?

VERTEXCOVER3

Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \{3, 4, 5, \dots\}$.

Frage: Gibt es eine Menge $C \subseteq V$ mit $|C| \leq k$, so dass für jede Kante (u, v) aus E mindestens einer der Knoten u und v in C ist?

Geben Sie eine polynomielle Reduktion von VERTEXCOVER auf VERTEXCOVER3 an und begründen Sie anschließend, dass Ihre Reduktion korrekt ist.

Aufgabe 5:

Hashing

1. Separate Chaining und Linear Probing am Beispiel

- a) Fügen Sie folgende Werte in der angegebenen Reihenfolge in eine anfangs leere Hashtabelle A der Größe 11 ein:

$$(35, -2, 107, -18, 46, 53, 20, 96, 81).$$

Verwenden Sie die Hashfunktionen

$$h_1(k) = |k| \mod 11 \quad \text{sowie} \quad h_2(k) = |3 \cdot k^2 - 2 \cdot k - 5| \mod 11$$

und verwenden Sie *separate chaining* (Hashing mit Verkettungen) und *linear probing* (lineares Sondieren) zur Kollisionsbehandlung (es sind also insgesamt vier Hashtabellen zu erzeugen).

- b) Löschen Sie anschließend den Wert 53 aus den beiden Hashtabellen, welche mittels linear probing erzeugt wurden.

2. Analyse von Hashing

- a) Seien $m, n \in \mathbb{N}$ beliebig. T sei eine Hashtabelle der Größe m , die eine Menge S mit $|S| = n$ speichert und zur Kollisionsauflösung Hashing mittels *separate chaining* verwendet. Wir betrachten für alle $0 \leq i < m$ die Urbildmengen $S_i := \{s \in S \mid h(s) = i\}$ und ihre Größen $n_i := |S_i|$.

Warum ist die asymptotische worst-case Laufzeit einer Suchoperation in T von der Ordnung $\Theta(1 + \max_i n_i)$?

(Sie dürfen annehmen, dass die Hashfunktion in konstanter Zeit ausgewertet werden kann.)

- b) Seien $m \in \mathbb{N}$ die Größe einer Hashtabelle und $n \in \mathbb{N}$ beliebig. Zeigen Sie, dass für jede endliche Menge \mathcal{U} mit $|\mathcal{U}| \geq (n-1)m+1$ und jede Hashfunktion $h : \mathcal{U} \rightarrow \{0, \dots, m-1\}$ eine Menge $S \subseteq \mathcal{U}$ mit $|S| = n$ existiert, so dass alle Elemente von S durch h auf denselben Eintrag der Hashtabelle abgebildet werden.

Aufgabe 6:

Suchbäume: AVL-Bäume

1. Fügen Sie nacheinander folgende Werte in einen zu Beginn leeren AVL-Baum ein:

32, 10, 8, 19, 25, 13, 37, 3.

Zeichnen Sie den resultierenden AVL-Baum nach jeder Einfügeoperation und geben Sie an, welche Operationen Sie verwendet haben.

2. Löschen Sie den Wert 37 aus dem sich ergebenden Baum aus Aufgabe 1. Zeichnen Sie den resultierenden AVL-Baum nach der Löschoperation und geben Sie an, welche Operationen Sie verwendet haben.

Thema Nr. 2
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

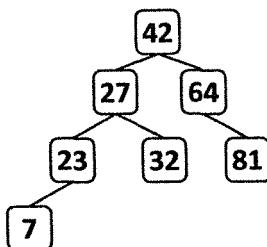
Aufgabe 1: „Bäume“

Ein Suchbaum ist ein binärer Baum, dessen Knoten (einschließlich Blätter) beliebige Werte so speichern, dass alle Werte im linken Teilbaum eines Knotens N kleiner sind als der Wert in N, während die Werte im rechten Teilbaum allesamt größer sind.

- a) Fügen Sie in einen leeren binären Suchbaum der Reihe nach die folgenden Schlüssel ein und geben Sie den fertigen Baum an:

27, 42, 23, 32, 7, 64, 81

- b) Geben Sie jeweils eine Einfügereihenfolge (nicht den Baum) für die obigen Schlüssel an, bei der die Höhe des entstehenden Baums *maximal* bzw. *minimal* wird.
 c) Wie teuer (Komplexitätsmaß in O-Notation) ist der Zugriff auf ein beliebiges Element in einem entarteten (Höhe maximal) bzw. perfekt balancierten (Höhe minimal) binären Suchbaum mit n Einträgen?
 d) Betrachten Sie den folgenden Baum wahlweise entweder als AVL-Baum oder als Rot-Schwarz-Baum. Fügen Sie in den teilweise gefüllten Baum zusätzlich die Zahl 99 ein. Zeichnen Sie zunächst den Baum direkt nach dem Einfügen des Elements und geben Sie
- im Falle des AVL-Baums die durch das Einfügen veränderten Tupel [H/B] aus Höhe H und Balancefaktor B der betroffenen Knoten an bzw.
 - im Falle des Rot-Schwarz-Baums die Farben der einzelnen Knoten an.



- e) Betrachten Sie Ihr Ergebnis aus Teilaufgabe d). An welchen Knoten ist eine Rotation erforderlich? Beschreiben Sie jeweils die Art der Rotation.
 f) Zeichnen Sie den Baum (Ihr Ergebnis aus Teilaufgabe d)) nach der Reorganisation (diesmal ohne Höhen/Balance bzw. Farben).

Aufgabe 2: „Algorithmenkomplexität“

Geben Sie jeweils die kleinste, gerade noch passende Laufzeitkomplexität folgender Java-Methoden im O-Kalkül (Landau-Notation) in Abhängigkeit von n und ggf. der Länge der Arrays an:

```
int matrixSumme(int n, int[][] feld) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            sum += feld[i][j];
        }
    }
    return sum;
}
```

```
int find(int key, int[][] keys) {
    int a = 0, o = keys.length;
    while (o - a > 1) {
        int m = (a + o) / 2;
        if (keys[m][0] > key)
            o = m;
        else
            a = m;
    }
    return a;
}
```

```
public static void T(int n) {
    if (n <= 0)
        return;
    for (int i = 0; i < 192; ++i) {
        T(n / 4);
    }
    for (int j = 0; j < n; j += 2) {
        foo(n); // in O(n*n*n)
    }
    for (int i = 0; i < 64; ++i) {
        T(n / 4);
    }
    bar(n); // in O(n)
}
```

```
int bitCount(int n) {
    int count = 0;
    while (n > 0) {
        if (n % 2 == 1) {
            count++;
        }
        n >>= 1; // bit-shift right
    }
    return count;
}
```

```
public void simulierte(long n, long[] baelle) {
    for (int i = 0; i < baelle.length; i++) {
        baelle[i] = 0;
    }
    while (--n >= 0) {
        int anzBaelle = baelle.length - 1;
        int delta = simLinRek(anzBaelle);
        // simLinRek in O(anzBaelle)
        if (baelle.length % 2 == 0) {
            delta--;
        }
        int pos = delta / 2 + baelle.length / 2;
        baelle[pos]++;
    }
}
```

Hinweis: Setzen Sie für $T(n)$ den Hauptsatz der Laufzeitfunktionen an:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Aufgabe 3: „Streutabellen (Hash)“

Gegeben sei folgende Hash-Funktion $h(k)$ für ganzzahlige Schlüssel k sowie folgende Schlüssel:

k	1	2	3	4	5	6	9	12
$h(k)$	3	6	1	4	7	2	3	4

- a) Fügen Sie die Schlüsselwerte in der Reihenfolge 1, 6, 9, 2, 3, 12, 4, 5 in eine Streutabelle mit acht Feldern (von 0 bis 7) ein. Verwenden Sie zur Kollisionsauflösung verkettete Listen wie im folgenden Beispiel:

Fach	Inhalt
0	42 → 666 → ⊥
...	

- b) Fügen Sie nun die Schlüsselwerte in der Reihenfolge 4, 5, 6, 12, 9, 3, 2, 1 in eine Streutabelle mit acht Feldern (von 0 bis 7) ein. Verwenden Sie diesmal zur Kollisionsauflösung jedoch lineares Sondieren mit konstanter Schrittweite +3 wie im folgenden Beispiel (die Streutabelle rechts genügt):

Schlüssel	Sondierte Fächer
42	3
666	3 (P), 6

⇒

Fach	Inhalt
0	
1	
2	
3	42
4	
5	
6	666
7	

- c) Begründen Sie kurz, warum bei der vorangehenden Streutabelle ein lineares Sondieren mit Schrittweite 3 besser ist als lineares Sondieren mit Schrittweite 4.

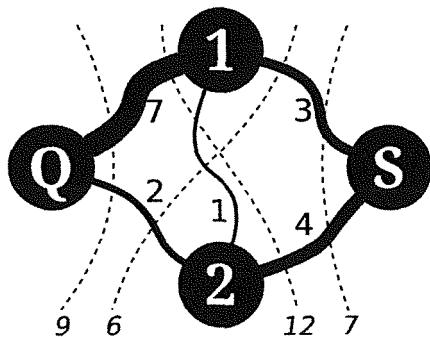
Aufgabe 4: „Rekursion und Dynamische Programmierung“

Mittels Dynamischer Programmierung (auch Memoization genannt) kann man insbesondere rekursive Lösungen auf Kosten des Speicherbedarfs beschleunigen, indem man Zwischenergebnisse „abspeichert“ und bei (wiederkehrendem) Bedarf „abruft“, ohne sie erneut berechnen zu müssen.

Gegeben sei folgende geschachtelt-rekursive Funktion für $n, m \geq 0$:

$$a(n, m) = \begin{cases} n + \left\lfloor \frac{n}{2} \right\rfloor & \text{falls } m = 0 \\ a(1, m - 1) & \text{falls } n = 0 \wedge m \neq 0 \\ a(n + \lfloor \sqrt{a(n - 1, m)} \rfloor, m - 1) & \text{sonst} \end{cases}$$

- Implementieren Sie die obige Funktion $a(n, m)$ zunächst ohne weitere Optimierungen als Prozedur/Methode in einer Programmiersprache Ihrer Wahl.
- Geben Sie nun eine DP-Implementierung der Funktion $a(n, m)$ an, die $a(n, m)$ für $0 \leq n \leq 100000$ und $0 \leq m \leq 25$ höchstens einmal gemäß obiger rekursiver Definition berechnet. Bitte beachten Sie, dass Ihre Prozedur trotzdem auch weiterhin mit $n > 100000$ oder $m > 25$ aufgerufen werden können soll.

Aufgabe 5: „Rekursion“

Die nebenstehende Skizze stellt die Wasserleitungen in einem Gebäude exemplarisch wie folgt dar: Q ist der Frischwasseranschluss der Stadtwerke, der beliebig viel Wasser liefern kann, S ist der Abwassergulli, der beliebig viel abführen kann. Neben den Verteilern Q und S hat das Netz im Beispiel zwei weitere Verteiler 1 und 2. Die Rohre zwischen den Verteilern sind unterschiedlich dick: z.B. gibt es zwischen Q und 1 eine Hauptleitung mit einer Kapazität von 7 l/s, aber das Abflussrohr von 1 zu S kann nur 3 l/s abführen. Den *maximalen Durchfluss* von Q nach S erhält man, indem man alle möglichen „Schnitte“ (gestrichelt) durchs Gebäude zieht, zu jedem Schnitt die Rohr-Durchfluss-Summe bestimmt, und von allen Schnitten den *minimalen Durchfluss* wählt - dieser Schnitt stellt den „begrenzenden Flaschenhals“ zwischen Q und S dar und erlaubt im Beispiel maximal 6 l/s.

nur 3 l/s abführen. Den *maximalen Durchfluss* von Q nach S erhält man, indem man alle möglichen „Schnitte“ (gestrichelt) durchs Gebäude zieht, zu jedem Schnitt die Rohr-Durchfluss-Summe bestimmt, und von allen Schnitten den *minimalen Durchfluss* wählt - dieser Schnitt stellt den „begrenzenden Flaschenhals“ zwischen Q und S dar und erlaubt im Beispiel maximal 6 l/s.

Implementieren Sie die geforderten Methoden in einer geeigneten Programmiersprache Ihrer Wahl oder Pseudocode. Folgender Java-Code und der Auszug aus der Java-API für `List<E>` seien zu Ihrer Unterstützung vorgegeben:

```
class Verteiler {
    final List<Rohr> rohre = new ArrayList<>(); // "Adjazenzliste mit Kanten"
}
```

```
class Rohr {
    final double df; final Verteiler e, a;

    Rohr(Verteiler ende, double durchfluss, Verteiler anderesEnde) {
        this.e = ende; this.df = durchfluss; this.a = anderesEnde;
        assert df > 0 && e != null && a != null;
        ende.rohre.add(this);
        anderesEnde.rohre.add(this);
    }
}
```

boolean add(E e): Appends the specified element to the end of this list.

boolean contains(Object o): Returns true if this list contains the specified element.

boolean remove(Object o): Removes the first occurrence of the specified element from this list, if it is present.

- a) Finden Sie ausgehend von der Quelle Q zunächst alle über Rohre erreichbaren Verteiler. Implementieren Sie dafür folgende Methode rekursiv:

```
class MaxFlowMinCut {
    // Fügt <v> und alle von <v> aus erreichbaren Verteiler zu <ev> hinzu.
    void erreichbareVerteiler(Verteiler v, List<Verteiler> ev) {
        ...
    }
}
```

- b) Ein „Schnitt“ durch das Rohrsystem wird mit zwei Listen dargestellt: Eine enthält alle Verteiler auf der Seite des Schnitts, die die Quelle Q (inklusive) enthält, und die andere die Verteiler auf der Seite der Senke S (inklusive). Ergänzen Sie folgende Methode, die den Durchfluss an einem „Schnitt“ bestimmt:

```
// Bestimmt den Durchfluss am Schnitt zwischen <quelleSeite> und <senkeSeite>.
double schnittDurchfluss(List<Verteiler> quelleSeite, List<Verteiler> senkeSeite) {
    ...
}
```

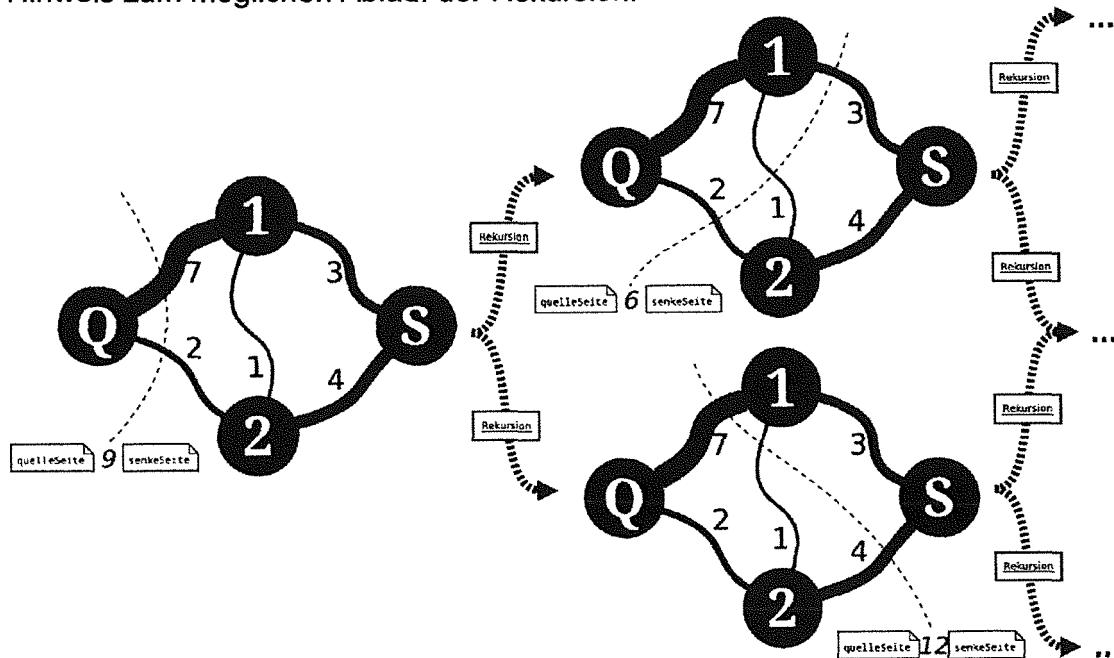
c) Die Berechnung des *maximalen Durchflusses* beginnt mit folgender Methode:

```
// Bestimmt den maximal möglichen Durchfluss von <quelle> zu <senke>.
public double maximalerDurchfluss(Verteiler quelle, Verteiler senke) {
    List<Verteiler> quelleSeite = new ArrayList<>();
    quelleSeite.add(quelle);
    List<Verteiler> senkeSeite = new ArrayList<>();
    erreichbareVerteiler(quelle, senkeSeite);
    senkeSeite.remove(quelle);
    return durchfluss(quelleSeite, senkeSeite, senke);
}
```

Implementieren Sie die rekursive Hilfsmethode **durchfluss**. Sie muss *nicht* optimal sein, d.h. sie darf bereits betrachtete Konfigurationen auch mehrfach untersuchen.

```
// Bestimmt den maximal möglichen Durchfluss von <quelleSeite> zu <senke>.
double durchfluss( List<Verteiler> quelleSeite,
                    List<Verteiler> senkeSeite, Verteiler senke) {
    Verteiler[] ssa = senkeSeite.toArray(new Verteiler[senkeSeite.size()]);
    double df = schnittDurchfluss(quelleSeite, senkeSeite);
    ...
}
```

Hinweis zum möglichen Ablauf der Rekursion:



Aufgabe 6

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

(a) Sei

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

Beispiele: $001011 \in L_1$, $1100 \in L_1$, $0101 \in L_1$.

(a1) Zeigen Sie, dass L_1 kontextfrei ist, indem Sie eine kontextfreie Grammatik G angeben mit $L(G) = L_1$, und (a2) begründen Sie, warum Ihre Grammatik *genau* die Sprache L_1 erzeugt.

(b) Formulieren Sie das Pumping-Lemma für reguläre Sprachen:

„Sei L eine reguläre Sprache über dem Alphabet Σ . Dann gibt es ...“

- (c) Sei $L_2 = \{0^n 1^m 0^k \mid n, m, k \in \mathbb{N}_0, n \geq 1, m \geq 1, k \geq 1\}$. Zeigen Sie, dass L_2 die Eigenschaft des Pumping-Lemmas für reguläre Sprachen (vgl. (b)) besitzt.
- (d) Zeigen Sie mit Hilfe des Pumping-Lemmas für reguläre Sprachen, dass die Sprache L_1 aus (a) nicht regulär ist.

Aufgabe 7

Im Nachfolgenden bezeichne M immer eine deterministische Turingmaschine, die als Eingabe eine natürliche Zahl bekommt, und $\langle M \rangle$ sei Gödelnummer von M .

a) Zeigen Sie, dass die Menge

$$L_1 = \{\langle M \rangle \mid \text{es gibt mindestens eine Zahl } n, \text{ so dass } M \text{ gestartet mit } n \text{ hält}\}$$

rekursiv aufzählbar (= partiell-entscheidbar) ist.

Geben Sie dazu einen Algorithmus A an, der als Eingabe eine Gödelnummer $\langle M \rangle$ bekommt und damit gestartet genau dann hält, wenn $\langle M \rangle \in L_1$ ist.

b) Sei $\text{Co-}H_0 = \{\langle M \rangle \mid M \text{ gestartet mit } 0 \text{ hält nicht}\}$. Es ist bekannt, dass $\text{Co-}H_0$ nicht rekursiv aufzählbar (= partiell-entscheidbar) ist.

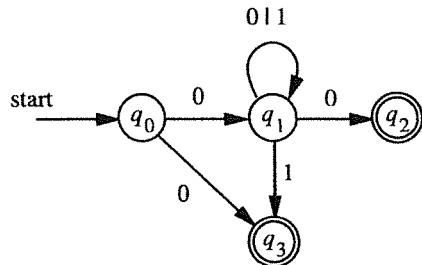
Zeigen Sie durch Reduktion von $\text{Co-}H_0$, dass die Menge

$$L_2 = \{\langle M \rangle \mid \text{es gibt genau eine Zahl } n, \text{ so dass } M \text{ gestartet mit } n \text{ hält}\}$$

nicht rekursiv aufzählbar (= partiell-entscheidbar) ist. Zeigen Sie also konkret: $\text{Co-}H_0 \leq L_2$.

Aufgabe 8

- (a) Gegeben sei der folgende nichtdeterministische endliche Automat N :



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

Hinweis: In einem deterministischen endlichen Automaten muss es an jedem Zustand für jedes Zeichen einen Übergang geben.

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

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2017

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

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

**Frühjahr
2017**

46115

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

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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!

Die Fragen in dieser Klausur sollen in erster Linie das grundlegende Verständnis der Konzepte und Zusammenhänge prüfen. Man kann sie oft in sehr unterschiedlicher Tiefe und Detaillierungsgrad beantworten. Die Anzahl der Punkte für die Teilfragen gibt einen ersten Hinweis darauf, welcher Detailgrad der Antwort erwartet wird. Um einen weiteren Hinweis zu geben, wird bei jeder Frage die Länge der Antwort in der Musterlösung, in der Anzahl getippter Zeilen, angegeben.

87 Punkte

Aufgabe 1: (Verständnis Automaten) (17 Pkt.)

- a) Beschreiben Sie den strukturellen Unterschied zwischen einem nicht-deterministischen (NFA) und einem deterministischen (DFA) endlichen Automaten. (2 Zeilen) (1 Pkt.)
- b) Kann ein NFA ebenfalls zum Lösen des Wortproblems benutzt werden? Wenn ja, was muss dann beim Erkennen eines Wortes beachtet werden? Wenn nein, wieso nicht? (1 Zeile)(2 Pkt.)
- c) Wenn man einen NFA mit n Knoten zu einem DFA macht, wieviel Knoten kann der DFA dann im schlimmsten Fall haben? (1 Zeile)(1 Pkt.)
- d) Sei A ein DFA, welcher die Sprache L_A akzeptiert. Sei B derjenige DFA, der aus A entsteht, wenn Endzustände mit Nicht-Endzuständen „vertauscht“ werden. Drücken Sie die von B akzeptierte Sprache L_B in Abhängigkeit von L_A aus. (1 Zeile)(1 Pkt.)
- e) Welche Eigenschaft muss ein DFA haben, wenn er eine Sprache mit unendlich vielen Wörtern akzeptiert? (1 Zeile)(1 Pkt.)
- f) NFAs und DFAs akzeptieren die gleiche Menge von Sprachen. Gilt das auch für nicht-deterministische und deterministische Kellerautomaten? Falls ja, wieso, falls nein, geben Sie ein Gegenbeispiel an. (1 Zeile)(2 Pkt.)
- g) Auf welche Weise benutzt man Kellerautomaten zum Erkennen von Worten einer Typ-2-Sprache (nach „Chomsky“), deren Grammatik gegeben ist? Erklären Sie die Top-down-Methode. (5 Zeilen)(4 Pkt.)
- h) Auf welche Weise benutzt man Turingmaschinen zum Erkennen von Wörtern einer Typ-1- oder Typ-0-Sprache (nach „Chomsky“), deren Grammatik gegeben ist? Erklären Sie die Bottom-up-Methode. (3 Zeilen)(3 Pkt.)
- i) Wie äußert sich bei der Bottom-up-Methode in der Turingmaschine der Unterschied zwischen Typ-1- und Typ-0-Sprachen? (3 Zeilen)(2 Pkt.)

Fortsetzung nächste Seite!

Aufgabe 2: (Verständnis Formale Sprachen) (14 Pkt.)

a) Was ist das Wortproblem einer formalen Sprache? (1 Zeile)(1 Pkt.)

b) Was ist der Unterschied zwischen dem Typ einer Sprache und dem Typ einer Grammatik? (1 Zeile)(1 Pkt.)

c) Wieso ist das Wortproblem für Sprachen vom Typ 1, 2 und 3 (nach „Chomsky“) entscheidbar? Skizzieren Sie ein einfaches Entscheidungsverfahren und begründen Sie, weshalb es immer terminiert. Warum funktioniert es nicht bei Typ-0-Sprachen (nach „Chomsky“)? (6 Zeilen) (4 Pkt.)

- d) Wenn das Wortproblem einer Sprache L entscheidbar ist, ist dann auch das Wortproblem des Komplements von L entscheidbar? Begründen Sie Ihre Antwort. (1 Zeile)(1 Pkt.)
- e) Wie kann man eine Abschätzung für die Pumping-Zahl bei Typ-3-Sprachen (nach „Chomsky“) bekommen? (1 Zeile)(1 Pkt.)
- f) Wie kann man eine Abschätzung für die Pumping-Zahl bei Typ-2-Sprachen (nach „Chomsky“) bekommen? (1 Zeile)(1 Pkt.)
- g) Wieso ist die Komplexität des Wortproblems bei Typ-2-Sprachen (nach „Chomsky“) höchstens $O(n^3)$? (3 Zeilen)(2 Pkt.)
- h) Geben Sie ein Beispiel für eine Sprache, die nicht vom Typ 0 (und auch nicht vom Typ 1,2,3) nach „Chomsky“ ist, und skizzieren Sie einen Beweis. (3 Zeilen)(3 Pkt.)

Aufgabe 3: (Verständnis Berechenbarkeitstheorie) (22 Pkt.)

- a) Was müsste man tun, um die Church-Turing-These zu widerlegen? (2 Zeilen)(2 Pkt.)
- b) Die Berechenbarkeitstheorie betrachtet nur Funktionen, die natürliche Zahlen auf natürliche Zahlen abbildet. Geben Sie zwei Methoden an, um Probleme, die für Zeichenketten definiert sind, auf Probleme von natürlichen Zahlen abzubilden. (6 Zeilen)(4 Pkt.)
- c) Haben Turingmaschinen mit parallelen Bändern mehr Berechnungskraft als solche mit nur einem Band? Begründen Sie Ihre Antwort kurz. (2 Zeilen)(2 Pkt.)
- d) Haben Turingmaschinen, die mit beliebigen Symbolen arbeiten mehr Berechnungskraft als solche, die nur mit 0 und 1 arbeiten? Begründen Sie Ihre Antwort kurz. (1 Zeile)(2 Pkt.)
- e) Warum ist die Loop-Programmiersprache schwächer als die While-Programmiersprache? (3 Zeilen)(2 Pkt.)
- f) Kann man aus einem Semientscheidungsverfahren für eine Menge M und einem Semient-scheidungsverfahren für das Komplement von M ein Entscheidungsverfahren für M machen? Wenn ja, wie, wenn nein, warum nicht? (2 Zeilen)(2 Pkt.)

- g) Unter welcher Bedingung kann man aus einem Semientscheidungsverfahren für eine Menge M ein Aufzählungsverfahren für M machen? Skizzieren Sie das Verfahren. (4 Zeilen)(3 Pkt.)
- h) Ist die Menge der terminierenden Turingmaschinen mit leerer Eingabe rekursiv aufzählbar? Wenn ja, wie, wenn nein, warum nicht? (3 Zeilen)(3 Pkt.)
- i) Braucht man für die Bearbeitung von n^*n -Matrizen immer geschachtelte Schleifen? Begründen Sie Ihre Antwort. (1 Zeile)(2 Pkt.)

- Aufgabe 4:** (Verständnis Komplexitätstheorie) (13 Pkt.)
- a) Warum betrachtet man $O(n)$ und z.B. $O(2n)$ als gleichwertig? (2 Zeilen)(1 Pkt.)
- b) Die Komplexität von binärer Suche im sortierten Array ist $O(\log(n))$. Finden Sie ein intuitives Argument, welches auch Schülerinnen und Schüler verstehen können, warum es kein schnelleres Verfahren gibt. (3 Zeilen)(2 Pkt.)
- c) Wie ist das Verhältnis von Platz-Komplexität und Zeit-Komplexität? Begründen Sie Ihre Antwort. (2 Zeilen)(2 Pkt.)
- d) Ist das SAT-Problem Loop-berechenbar? Wenn ja, warum, wenn nein, warum nicht? (4 Zeilen)(3 Pkt.)
- e) Das Konvexe-Hüllen Problem in 2 Dimensionen, ist das Problem, für eine 2D-Punktmenge das kleinste umfassende konvexe Polygon (Folge von 2D-Punkten) zu finden. Es gibt einen Algorithmus, der dieses Problem in $O(n \log(n))$ Zeit löst. Wie müsste man vorgehen, um zu beweisen, dass es keinen schnelleren Algorithmus geben kann? (3 Zeilen)(3 Pkt.)
- f) Könnte man einen polynomiellen Algorithmus zur Lösung des Graph-Färbeproblems auch zur Lösung des Travelling-Salesman Problems benutzen? Begründen Sie Ihre Antwort. (2 Zeilen)(2 Pkt.)

Fortsetzung nächste Seite!

Aufgabe 5: (Verständnis Algorithmen und Datenstrukturen) (21 Pkt.)

- a) Betrachte eine Hashtabelle der Größe $m = 10$.

Welche der folgenden Hashfunktionen für Hashing mit verketteten Listen ist dafür besser geeignet? Begründen Sie Ihre Wahl.

1) $h_1(x) = (4x+4) \bmod m$

2) $h_2(x) = (5x+5) \bmod m$.

(2 Zeilen 3Pkt.)

- b) Betrachte wieder eine Hashtabelle der Größe $m = 10$.

Welche der folgenden Hashfunktionen für Hashing mit *offener Adressierung* ist dafür am besten geeignet? Begründen Sie Ihre Wahl.

1) $h_1(x,i) = (3x+(i+1)m) \bmod m$

2) $h_2(x,i) = (3x+(i+1)(m-1)) \bmod m$.

(2 Zeilen 3Pkt.)

- c) Angenommen Sie haben für die Implementierung einer beliebigen, aber sortierten Menge von Ganzzahlen die Wahl zwischen einem Array, welches die Zahlen in sortierter Reihenfolge enthält, und verketteten Listen. Falls „insert“ die bei weitem häufigste Operation ist, welche Version würden Sie wählen? Begründen Sie Ihre Wahl. (4 Zeilen 4Pkt.)

- d) Warum könnte man auch mit AVL-Bäumen Priority-Queues implementieren?

(1 Zeilen 2Pkt.)

- e) Der A*-Algorithmus ist ein Verfahren um in einem Graphen mit gewichteten Kanten den kürzesten Weg zwischen zwei Knoten zu finden. Er benutzt dazu eine Gewichtungsfunktion $f(k) = g(k) + h(k)$, wobei $g(k)$ die tatsächliche Entfernung zwischen Startknoten und Knoten k angibt, und $h(k)$ die geschätzte Entfernung von Knoten k zum Ziel. A* führt in einer OpenList die aktuelle Menge der zu untersuchenden Knoten. Am Anfang enthält sie den Startknoten. Danach wählt er einen Knoten aus der OpenList mit dem kleinsten f -Wert und ersetzt ihn durch seine Nachfolgeknoten.

Frage: Soll der A*-Algorithmus terminieren, sobald er zum ersten mal den Zielknoten erreicht hat? Wenn ja, warum, wenn nein, geben Sie ein Gegenbeispiel. (5 Zeilen(3 Pkt.)

- f) Könnte man eine Implementierung des A*-Algorithmus auch so steuern, dass er sich wie der Dijkstra-Algorithmus zum Finden kürzester Pfade verhält? Wenn ja, wie müsste man es tun, wenn nein, warum nicht? (1Zeile)(2 Pkt.)

- g) Angenommen, Sie wollen mit Ihrer Klasse 5 Barockkirchen in Bayern besuchen, und versuchen, den kürzesten Weg durch alle 5 Kirchen zu planen. Skizzieren Sie für diesen Fall Vor- und Nachteile des A*-Algorithmus gegenüber dem Dijkstra-Algorithmus.

Thema Nr. 2
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

Für $w \in \{a, b\}^*$ bezeichne w^R das „gespiegelte“ Wort zu dem Wort w (d.h. das Wort, das entsteht, wenn die Buchstaben von w in umgekehrter Reihenfolge geschrieben werden, z.B. $aaaba^R = abaaa$).

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

$$L_1 = \{vwv^R \mid v, w \in \{a, b\}^* \text{ und } |w| \text{ ist ungerade}\}$$

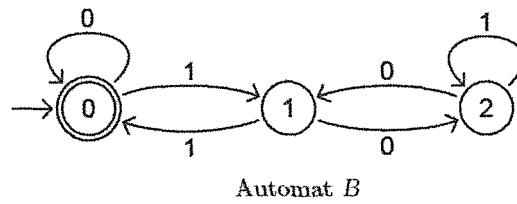
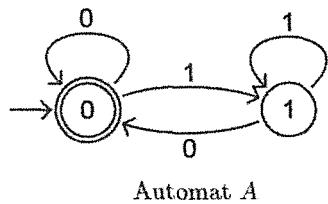
Aufgabe 2:

- a) Für $w \in \{a, b\}^*$ bezeichne $|w|_a$ die Anzahl der Buchstaben a in w und $|w|_b$ die Anzahl der Buchstaben b in w . Ein Wort $u \in \{a, b\}^*$ heißt Präfix eines Wortes $w \in \{a, b\}^*$, falls es ein $v \in \{a, b\}^*$ mit der Eigenschaft $w = uv$ gibt.

Geben Sie einen deterministischen, endlichen Automaten an, der folgende Sprache akzeptiert!

$$L_2 = \{w \in \{a, b\}^* \mid |w|_a = |w|_b \text{ und für jedes Präfix } u \text{ von } w \text{ gilt } -2 \leq |u|_a - |u|_b \leq 2\}$$

- b) Gegeben seien die folgenden deterministischen, endlichen Automaten A und B . Die von ihnen akzeptierten Sprachen bezeichnen wir mit L_A und L_B .



Konstruieren Sie einen nichtdeterministischen, endlichen Automaten, der folgende Sprache akzeptiert!

$$L_3 = \overline{L_A \cap L_B}$$

Aufgabe 3:

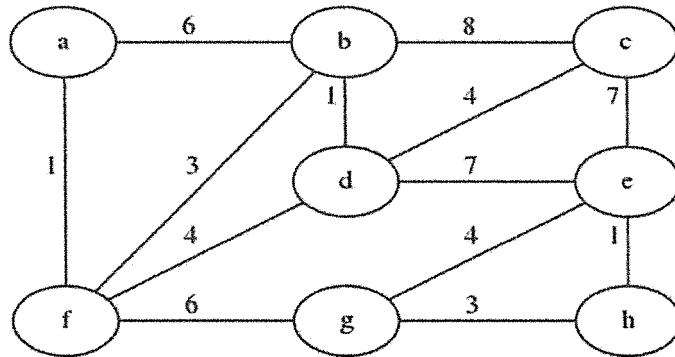
Beantworten Sie folgende Fragen und beweisen Sie Ihre Antworten!

- a) Existiert eine berechenbare Funktion $f: D \rightarrow \mathbb{N}$, $D \subseteq \mathbb{N}$, mit endlichem Definitionsbereich und unentscheidbarem Wertebereich?
- b) Existiert eine berechenbare Funktion $g: D \rightarrow \mathbb{N}$, $D \subseteq \mathbb{N}$, mit unentscheidbarem Definitionsbereich und entscheidbarem Wertebereich?
- c) Existiert eine berechenbare Funktion $h: \mathbb{N} \rightarrow \mathbb{N}$ mit entscheidbarem Definitionsbereich und unentscheidbarem Wertebereich?

Fortsetzung nächste Seite!

Aufgabe 4:

Gegeben sei der folgende ungerichtete Graph $G = (V, E)$ mit Kantenlänge $l(e)$ für jede Kante $e \in E$.



- Berechnen Sie mit einem Algorithmus Ihrer Wahl einen minimalen Spannbaum von G und geben Sie diesen an.
- Nutzen Sie den Algorithmus von Dijkstra, um die kürzesten Wege vom Startknoten a zu allen anderen Knoten zu finden. Die Startdistanz jedes Knotens betrage $d(v) = \infty$ für $v \in V \setminus \{a\}$. Geben Sie für jeden Schritt den gewählten Knoten und ggf. die neue Distanz der bereits besuchten Knoten an.
- Existiert ein Eulerpfad in G ? Falls ja, geben Sie ein Beispiel an. Falls nein, begründen Sie!
- Existiert ein Eulerkreis in G ? Falls ja, geben Sie ein Beispiel an. Falls nein, begründen Sie!

Aufgabe 5:

Bei Bubblesort wird eine unsortierte Folge von Elementen a_1, a_2, \dots, a_n von links nach rechts durchlaufen, wobei zwei benachbarte Elemente a_i und a_{i+1} getauscht werden, falls sie nicht in der richtigen Reihenfolge stehen. Dies wird so lange wiederholt, bis die Folge sortiert ist.

- Sortieren Sie die folgende Zahlenfolge mit Bubblesort. Geben Sie die neue Zahlenfolge nach jedem (Tausch-)Schritt an: 3, 2, 4, 1
- Geben Sie den Bubblesort-Algorithmus für ein Array von natürlichen Zahlen in einer Programmiersprache Ihrer Wahl an. Die Funktion `swap(index1, index2)` kann verwendet werden, um zwei Elemente des Arrays zu vertauschen.
- Geben Sie eine obere Schranke für die Laufzeit an. Beschreiben Sie mögliche Eingabedaten, mit denen diese Schranke erreicht wird.

Fortsetzung nächste Seite!

Aufgabe 6:

Es wird das Verhalten von Warteschlange (Queue), Keller (Stack) sowie Vorratswarteschlange (Priority Queue) untersucht.

Alle genannten Datentypen implementieren die Methoden `push(x)` und `pop()`. Dabei legt `push(x)` ein Element in der jeweiligen Datenstruktur nach deren Regeln ab, `pop()` hingegen entfernt ein Element. Für die Priorität der Vorratswarteschlange gilt:

Prioritätsbedingung 1: „x vor y“ genau dann, wenn $x < y$.

Es werden nun auf allen drei Datenstrukturen die folgenden Operationen ausgeführt:

`push(3), push(1), push(7), push(8), pop(), pop(), push(6), push(5),
pop(), push(2), pop(), pop(), pop()`

- a) Geben Sie den internen Speicherzustand der jeweiligen Datenstruktur nach jeder Operation in einer geeigneten Schreibweise an.
- b) Skizzieren Sie, wie man die jeweilige Datenstruktur implementieren könnte. Was wären die resultierenden Laufzeiten für `push(x)` und `pop()`?

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Herbst 2017

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	Herbst 2017	46115
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen
— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Th. Informatik, Algorith./Datenstr.**

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

Anzahl der Druckseiten dieser Vorlage: **12**

Bitte wenden!

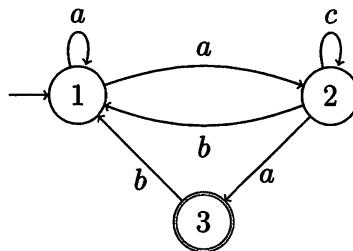
Thema Nr. 1
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

Endliche Automaten

Betrachten Sie den NEA \mathcal{A} :



- Konstruieren Sie mit Hilfe der Potenzmengenkonstruktion einen deterministischen endlichen Automaten, der dieselbe Sprache wie \mathcal{A} erkennt.
- Geben Sie einen deterministischen endlichen Automaten an, der das Komplement der Sprache $L(\mathcal{A})$ erkennt.

Aufgabe 2:

Berechenbarkeit

Zeigen Sie, dass die Sprache

$$\{(w_1; w_2) \mid M_{w_1}(\varepsilon) \neq M_{w_2}(\varepsilon)\}$$

nicht entscheidbar ist. Hier bezeichnet M_{w_i} die von w_i kodierte Turingmaschine und $M_{w_i}(\varepsilon)$ das Ergebnis der Berechnung von M_{w_i} bei Eingabe ε (für $i \in \{1, 2\}$). Sie dürfen davon ausgehen, dass die Maschinen immer entweder die Eingabe akzeptieren, ablehnen oder nicht anhalten. Wir definieren dann

$$M_{w_i}(\varepsilon) = \begin{cases} \text{ja} & \text{falls } M_{w_i} \text{ die Eingabe } \varepsilon \text{ akzeptiert} \\ \text{nein} & \text{falls } M_{w_i} \text{ die Eingabe } \varepsilon \text{ ablehnt} \\ \text{loop} & \text{falls } M_{w_i} \text{ bei Eingabe } \varepsilon \text{ nicht anhält} \end{cases}$$

Falls beide Maschinen nicht anhalten, dann ist

$$M_{w_1}(\varepsilon) = M_{w_2}(\varepsilon) = \text{loop}.$$

Aufgabe 3:

Kontextfreie Sprachen

- Geben Sie eine kontextfreie Grammatik für $L_1 = L(a^*(bc)^*d)$ an. Erklären Sie Ihre Grammatik (z.B. die Aufgaben der einzelnen Nichtterminale).

- b) Geben Sie eine kontextfreie Grammatik für $L_2 = \{a^nbc^m \mid n \neq m\}$ an. Erklären Sie Ihre Grammatik (z.B. die Aufgaben der einzelnen Nichtterminale).
- c) Gegeben sei die folgende kontextfreie Grammatik G mit Startsymbol S in Chomsky-Normalform:

$$S \rightarrow AA \mid AC$$

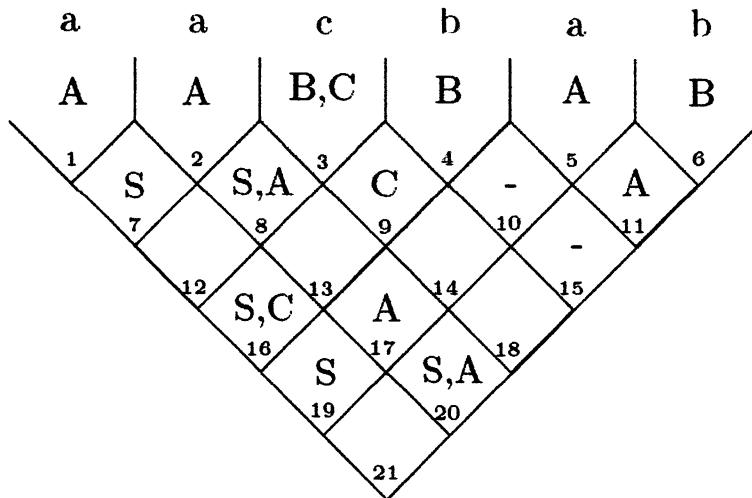
$$A \rightarrow AB \mid a$$

$$B \rightarrow b \mid c$$

$$C \rightarrow SC \mid BB \mid c$$

Betrachten Sie außerdem das nachfolgende Tableau, das durch Anwendung des CYK-Algorithmus mit der Grammatik G und dem Wort $w = aacbab$ entstanden ist, aber bei dem der Inhalt einiger Zellen gelöscht wurde. Beispielsweise bedeutet das C in Zelle 9, dass aus der Variable C das Teilwort cb hergeleitet werden kann.

- i) Geben Sie an, welche Variablen jeweils in den Zellen 12, 13, 14, 18 und 21 enthalten sein sollten und erklären Sie auch warum.
- ii) Liegt das Wort $w = aacbab$ in der Sprache $L(G)$? Begründen Sie Ihre Antwort mit Hilfe von (i).



Aufgabe 4:
Komplexität

Betrachten Sie die folgenden Probleme:

3SAT

Gegeben:

Eine aussagenlogische Formel in konjunktiver Normalform φ mit je drei Literalen pro Klausel.

Frage:

Ist φ erfüllbar?

4SAT

Gegeben:

Eine aussagenlogische Formel in konjunktiver Normalform φ mit je vier Literalen pro Klausel.

Frage:

Ist φ erfüllbar?

- Zeigen Sie, dass sich 3SAT in polynomieller Zeit auf 4SAT reduzieren lässt, d. h. $3SAT \leq_p 4SAT$.
- Was können Sie aus a) folgern, wenn Sie wissen, dass 3SAT NP-vollständig ist?
- Was können Sie aus a) folgern, wenn Sie wissen, dass 4SAT NP-vollständig ist?

Aufgabe 5:
Die Begründung zählt

Beantworten Sie die folgenden Fragen und geben Sie eine kurze Begründung (in ein bis drei Sätzen) an. Nur Ihre Begründung wird bewertet.

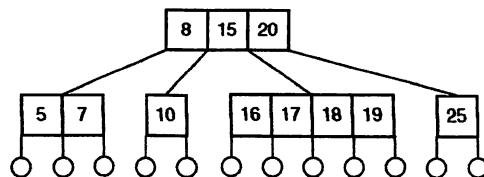
- Gibt es reguläre Sprachen L , so dass L^* eine endliche Sprache ist?
- Gibt es unendlich viele reguläre Ausdrücke α , so dass $L(\alpha^*) \cap L(\alpha) = L(\alpha)$?
- Gibt es WHILE-Programme, die NP-vollständige Probleme lösen?
- Falls L_1 und L_2 kontextfreie Sprachen sind, ist $L_1 \cap L_2$ dann immer eine entscheidbare Sprache?
- Wir nennen eine Sprache L LOOP-berechenbar, falls ihre charakteristische Funktion

$$\chi(w) := \begin{cases} 0 & \text{falls } w \notin L \\ 1 & \text{falls } w \in L \end{cases}$$

LOOP-berechenbar ist. Sind LOOP-berechenbare Sprachen unter Komplement abgeschlossen?

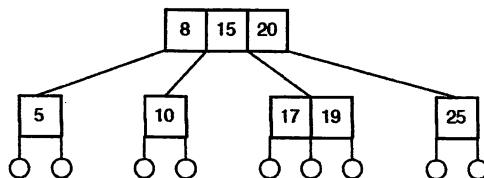
Aufgabe 6:
(2,4)-Bäume

- a) Gegeben sei der folgende Mehrwegsuchbaum T_1 :



Ist T_1 ein (2,4)-Baum? Begründen Sie Ihre Antwort!

- b) Gegeben sei der folgende (2,4)-Baum T :



Fügen Sie in T nacheinander die Schlüssel 7, 18 und 16 ein, so dass nach jedem Einfügeschritt wieder ein (2,4)-Baum entsteht. Zeichnen Sie dabei jeweils die so entstandenen (2,4)-Bäume T_2 bis T_4 nach jeder Einfügeoperation und geben Sie an, welche Operationen Sie verwendet haben.

- c) Löschen Sie nun aus T_4 nacheinander die Schlüssel 15, 10 und 8, so dass nach jedem Löschschnitt wieder ein (2,4)-Baum entsteht. Zeichnen Sie dabei jeweils die so entstandenen (2,4)-Bäume T_5 bis T_7 nach jeder Löschoperation und geben Sie an, welche Operationen Sie verwendet haben.
- d) Wie ist die Höhe eines (2,4)-Baums definiert? Welche Höhe hat T_7 ?
- e) Geben Sie an, welche Höhe ein (2,4)-Baum mit n Einträgen minimal und maximal haben kann und begründen Sie Ihre Antwort.

Aufgabe 7:**Durchmustern von Graphen**

Wir betrachten den Algorithmus zur Tiefensuche von einem Knoten v in einem ungerichteten Graphen G :

DFSREC(G, v)

Input : Graph $G = (V, E)$ und ein Knoten $v \in V$

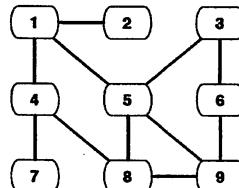
```

1 begin
2   setVertexLabel( $v, \underline{B}$ );
3   forall the  $e \in G.incidentEdges(v)$  do
4     if getEdgeLabel( $e$ ) =  $\underline{U}$  then
5        $w \leftarrow G.oppositeVertex(v, e)$ ;
6       if getVertexLabel( $w$ ) =  $\underline{U}$  then
7         setEdgeLabel( $e, \underline{D}$ );
8         DFSREC( $G, w$ );
9       else
10      setEdgeLabel( $e, \underline{B}$ );

```

Wir nehmen an, dass vor dem ersten Aufruf von DFSREC alle Knoten und Kanten von G mit \underline{U} markiert wurden.

- a) Geben Sie für den Graphen G in folgender Abbildung die DFS-Traversierung der Knoten an, **beginnend bei Knoten 1**. Die Reihenfolge, in der die Nachbarn eines Knotens in der Adjazenzliste gespeichert sind, entspricht ihrer aufsteigenden Nummerierung (z.B. sind die Nachbarn des Knotens 1 in der Reihenfolge 2, 4, 5 gespeichert).



- Zeichnen Sie dazu einen Richtungspfeil an jede besuchte Kante um anzugeben, von welchem Knoten sie besucht wurde.
- Nummerieren Sie die Kanten in der Reihenfolge in der sie besucht wurden, und
- geben Sie für jede Kante die Markierung an (sofern sie sich von \underline{U} unterscheidet).

Hinweise zu dieser Aufgabe:

- Achten Sie darauf, die zeitliche Abfolge Ihrer Zeichnungen zu dokumentieren!
- Die Markierung \underline{U} brauchen Sie nicht anzugeben.

b) Zeichnen Sie den Rekursionsbaum für den Aufruf $\text{DFSREC}(G, 1)$. Geben Sie für jeden Knoten im Rekursionsbaum an

- wie der Graph G zum Zeitpunkt des Aufrufs markiert ist,
- und wie der Graph markiert ist, wenn der Aufruf beendet ist.

Nummerieren Sie die Knoten des Rekursionsbaums gemäß der zeitlichen Reihenfolge der entsprechenden Aufrufe.

Thema Nr. 2
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

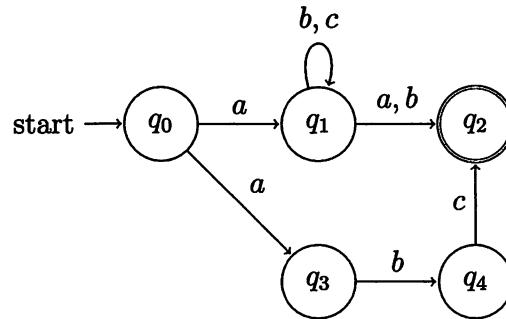
Aufgabe 1:

Reguläre Sprachen

- a) Sei $\Sigma = \{a, b, c\}$ und L_1 die Sprache aller Wörter über Σ , die mit ab beginnen, mit cb enden und ansonsten kein weiteres Teilwort cb enthalten.

Geben Sie einen endlichen Automaten an, der L_1 akzeptiert.

- b) Gegeben sei der unten aufgeführte endliche Automat A_2 , welcher die Sprache $L_2 = L(A_2)$ akzeptiert.



- i) Geben Sie einen regulären Ausdruck für L_2 an.
- ii) Geben Sie einen *deterministischen* endlichen Automaten an, der die Sprache L_2 akzeptiert.
- c) Zeigen Sie, dass es für jede nicht-reguläre Sprache L_3 eine nicht-reguläre Sprache L'_3 gibt, für die $L_3 \cap L'_3$ regulär ist.
- d) Zeigen Sie mit dem Pumping-Lemma für reguläre Sprachen, dass die Sprache $L_4 = \{a^i b^j \mid i, j \in \mathbb{N} \wedge i < j\}$ nicht regulär ist.

Aufgabe 2:
Kontextfreie Sprachen

- a) Sei $\Sigma = \{a, b, c\}$ und $L_1 \subseteq \Sigma^*$ mit $L_1 = \{a^i b^j c^i \mid i, j \in \mathbb{N}\}$.
- Geben Sie eine kontextfreie Grammatik für die Sprache L_1 an.
 - Geben Sie eine kontextfreie Sprache L'_1 an, sodass $L_1 \cap L'_1$ nicht kontextfrei ist.
- b) Gegeben sei die kontextfreie Grammatik $G_2 = (V, \Sigma, P, S)$ mit Sprache $L_2 = L(G_2)$, wobei $V = \{A, B, C, S, T, U\}$ und $\Sigma = \{a, b, c\}$. P bestehe aus den folgenden Produktionen:

$$\begin{array}{l} S \rightarrow BS \mid UC \\ A \rightarrow a \end{array}$$

$$\begin{array}{l} T \rightarrow UB \mid AC \\ B \rightarrow b \end{array}$$

$$\begin{array}{l} U \rightarrow AT \\ C \rightarrow c \end{array}$$

- Zeigen Sie $baacc \in L_2$.
- Folgende Tabelle entsteht durch Anwendung des CYK-Algorithmus für das Wort $aacb$. Übernehmen Sie die Tabelle auf Ihre Bearbeitung und vervollständigen Sie die fehlenden Einträge $(1,3), (2,4)$ und $(1,4)$!

		(1,4) :	
(1,3) :		(2,4) :	
(1,2) : \emptyset	(2,3) : T	(3,4) : \emptyset	
(1,1) : A	(2,2) : A	(3,3) : C	(4,4) : B

$a \qquad a \qquad c \qquad b$

- Wie entnehmen Sie der Tabelle, dass $aacb \notin L_2$ gilt?
- Für ein $X \in V$ sei G_X die Grammatik (V, Σ, P, X) . Wie entnehmen Sie obiger Tabelle ein $X \in V$, so dass $aacb \in L(G_X)$ gilt?

Aufgabe 3:**Dynamische Programmierung**

Die Methode pKR berechnet die n -te Primzahl ($n \geq 1$) kaskadenartig rekursiv und äußerst ineffizient:

```
long pKR(int n) {
    long p = 2;
    if (n >= 2) {
        p = pKR(n - 1); // beginne die Suche bei der vorhergehenden Primzahl
        int i = 0;
        do {
            p++; // pruefe, ob die jeweils naechste Zahl prim ist, d.h. ...
            for (i = 1; i < n && p % pKR(i) != 0; i++) {
                } // pruefe, ob unter den kleineren Primzahlen ein Teiler ist
            } while (i != n); // ... bis nur noch 1 und p Teiler von p sind
    }
    return p;
}
```

Überführen Sie pKR mittels *dynamischer Programmierung* (hier also *Memoization*) und mit möglichst wenigen Änderungen so in die linear rekursive Methode pLR, dass pLR(n, new long[n + 1]) ebenfalls die n -te Primzahl ermittelt:

```
private long pLR(int n, long[] ps) {
    ps[1] = 2;
    ...
}
```

Aufgabe 4:**Streuspeicherung - Hashing**

Gegeben seien die folgenden Schlüssel k zusammen mit ihren Streuwerten $h(k)$:

k	A	B	C	D	E	F	G
$h(k)$	1	3	3	3	2	3	3

Fügen Sie die Schlüssel k in alphabetischer Reihenfolge mit der Streufunktion $h(k)$ in eine Streutabelle nach folgendem Muster ein. Verwenden Sie *geschlossenes Hashing* mit Behältergröße $b=2$ und lösen Sie Kollisionen durch **lineares Sondieren** mit Schrittweite $c=+2$ auf.

Bucket ►	0	1	2	3	4
erster Schlüssel: ↗					
zweiter Schlüssel: ↘					

Aufgabe 5:**Binäre Suche**

Im Folgenden sollen Sie einen Schlüssel t in einem Feld (Array) ts mittels *binärer Suche* lokalisieren. Für die Schlüssel vom Typ T gibt es zwei Vergleicher $c1$ bzw. $c2$ und das Feld ts ist so sortiert, dass:

$$\forall i < j : ts[i] \prec_{c1} ts[j] \vee (ts[i] =_{c1} ts[j] \wedge ts[i] \preceq_{c2} ts[j])$$

Beispiel: Tupel $T := (\text{String}, \text{ Integer})$, $c1$ vergleicht Strings, $c2$ vergleicht Integers:

(AuD, 42)	(AuD, 666)	(AuD, 666)	(PFP, 41)	(PFP, 666)	(PFP, 4711)
-----------	------------	------------	-----------	------------	-------------

Hinweis zur API der Methode `int compare(T o1, T o2)` im Interface `Comparator<T>`:
Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

Ergänzen Sie die iterative Methode `suche` so, dass sie den Index von t in ts mit einer Laufzeit von $\mathcal{O}(\log(ts.length))$ zurückgibt, falls t in ts vorkommt, andernfalls sei ihr Ergebnis -1 :

```
<T> int suche(T[] ts, T t, Comparator<T> c1, Comparator<T> c2) {
    int a = 0, m, z = ts.length - 1; // Anfang, Mitte, Ende
    ...
}
```

Aufgabe 6:**Halden - Heaps**

Gegeben sei folgende Feld-Einbettung (Array-Darstellung) einer Min-Halde:

0	1	2	3	4	5	6	7	8	9	10	11
0	2	3	7	6	5	4	8	9	10	11	12

- Stellen Sie die Halde graphisch als (links-vollständigen) Baum dar.
- Entfernen Sie das kleinste Element (die Wurzel 0) aus der *obigen initialen* Halde, stellen Sie die Haldeneigenschaft wieder her und geben Sie nur das Endergebnis in Felddarstellung an.
- Fügen Sie nun den Wert 1 in die *obige initiale* Halde ein, stellen Sie die Haldeneigenschaft wieder her und geben Sie nur das Endergebnis in Felddarstellung an.

Aufgabe 7:**Sortieren**

- Führen Sie „*Sortieren durch Einfügen*“ lexikographisch *aufsteigend* und *in-situ (in-place)* so in einem Schreibtischlauf auf folgendem Feld (Array) aus, dass gleiche Elemente ihre relative Abfolge jederzeit beibehalten (also dass z.B. "A₁" stets vor "A₂" im Feld steht). Jede Zeile stellt den Zustand des Feldes dar, nachdem das *jeweils nächste* Element in die Endposition verschoben wurde. Der bereits sortierte Teilbereich steht vor |||. Gleiche Elemente tragen zwecks Unterscheidung ihre „*Objektidentität*“ als Index (z.B. "A₁".equals("A₂") aber "A₁" != "A₂").

L	A ₁	B ₁	F	A ₂	B ₂

...

- Ergänzen Sie die folgende Methode so, dass sie die Zeichenketten im Feld a lexikographisch aufsteigend *durch Einfügen sortiert*. Sie muss zum vorangehenden Ablauf passen, d.h. sie muss *iterativ* sowie *in-situ (in-place)* arbeiten und die relative Reihenfolge gleicher Elemente jederzeit beibehalten. Sie dürfen davon ausgehen, dass kein Eintrag im Feld null ist.

```
void sortierenDurchEinfuegen(String[] a) {
    // Hilfsvariable:
    String tmp;
    ...
}
```

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2018

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

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

**Frühjahr
2018**

46115

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

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Theoretische Informatik/Algorithmen/Datenstrukturen**

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: Reguläre Ausdrücke

Gegeben sei die Sprache L. L besteht aus der Menge aller Worte über dem Alphabet $\Sigma = \{0, 1, 2\}$, die mit 0 beginnen und bei denen das vorletzte und das letzte Zeichen übereinstimmen.

- a) Geben Sie alle Worte bis zur Länge vier von L an.
- b) Zeigen Sie durch Angabe eines regulären Ausdrucks, dass die Sprache L regulär ist.
- c) Konstruieren Sie (graphisch) einen nichtdeterministischen endlichen Automaten ohne Spontanübergänge (= ε -Kanten), der L akzeptiert.

Aufgabe 2: Komplexitätsklassen

- a) Definieren Sie die Komplexitätsklassen P und NP.
- b) Warum gilt $P \subseteq NP$?

Aufgabe 3: Turingmaschinen

Gegeben sei die Sprache $L = \{a^n b^{2n} | n \geq 1\}$.

- a) Geben Sie eine Turingmaschine M an, die L erkennt. Die Eingabeworte sind durch die Sonderzeichen \in vorne und $\$$ hinten begrenzt.
- b) Beschreiben Sie in Worten, wie Ihre Turingmaschine arbeitet.
- c) Konstruieren Sie M durch Angabe der Befehle und kommentieren Sie, was die Befehle in Bezug auf die Arbeitsweise von M unter a) leisten.

Aufgabe 4: Pumping-Lemma

Das Pumping-Lemma für reguläre Sprachen ist bekanntlich folgendermaßen definiert:

Satz (Pumping-Lemma)

- $L \in \text{REG} \Rightarrow (\exists k \in \mathbb{N}_0 \forall w \in L : |w| \geq k \Rightarrow \exists \text{ Zerlegung } w = xyz \text{ mit}$
 1. $|y| \geq 1$
 2. $|xy| \leq k$
 3. $\forall i \in \mathbb{N}_0 : xy^i z \in L$

Zeigen Sie damit, dass die Sprache $L = \{a^n b^{n+3} | n \geq 2\}$ nicht regulär ist.

Aufgabe 5: Mastertheorem

Der Hauptsatz der Laufzeitfunktionen ist bekanntlich folgendermaßen definiert:

Satz (Mastertheorem)

- Sei $T(n) = \begin{cases} d \in \Theta(1), & \text{falls } n \leq k \\ a \cdot T\left(\frac{n}{b}\right) + g(n), & \text{sonst} \end{cases} \quad \text{mit } k \in \mathbb{N}, a \geq 1 \text{ und } b > 1$
- 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 $0 < c < 1$
 $\Rightarrow T(n) \in \Theta(g(n))$

Sei nun folgende Funktion für die Anzahl an Schritten eines Algorithmus auf der Eingabegröße n gegeben:

$$T(n) = 4 T(n/2) + 2n \log_5(n) + 7n.$$

Ordnen Sie $T(n)$ einem der drei Fälle des Mastertheorems zu. Beweisen Sie, dass $T(n)$ die Voraussetzungen des Falles erfüllt und geben Sie die resultierende Laufzeitkomplexität von $T(n)$ an.

Aufgabe 6: Sortieren

- a) Gegeben ist das folgende Array von Zahlen:

[13, 4, 7, 32, 27, 11, 6, 17, 2]

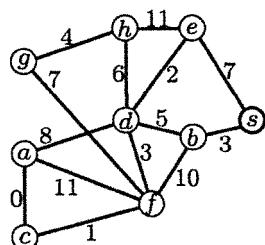
Sortieren Sie das Array mittels Mergesort aufsteigend von links nach rechts. Das Aufteilen einer Liste soll in der Mitte erfolgen und, falls notwendig, die zweite Liste ein Element länger sein als die erste. Listen der Länge zwei dürfen durch direkten Vergleich sortiert werden. Geben Sie die Eingabe und das Ergebnis jedes (rekursiven) Aufrufs an. Geben Sie abschließend die sortierte Liste an.

- b) Beantworten Sie folgende Fragen jeweils ohne Begründung oder Beweis.

- (i) Welche Worst-Case-Laufzeit (O-Notation) hat Mergesort für n Elemente?
- (ii) Welche Laufzeit hat Mergesort für n Elemente im Best-Case?
- (iii) Kann basierend auf paarweisen Vergleichen von Werten schneller (Laufzeitkomplexität) als Mergesort sortiert werden?

Aufgabe 7: Kürzeste Wege

- a) Berechnen Sie mithilfe des Algorithmus von Dijkstra die kürzesten Wege vom Knoten s zu allen anderen Knoten im folgenden aufgezeichneten Graphen G.



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

- b) Nehmen Sie jetzt an, dass die Kantengewichte auch negativ sein können. Geben Sie ein kleines Beispiel (max. 3 Knoten) an und machen Sie daran deutlich, dass der Dijkstra-Algorithmus dann falsche Ergebnisse liefern kann.

Aufgabe 8: Spannbaum

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

	s	a	b	c	d	e	f	g	h
s	-	-	3	-	-	7	-	-	-
a	-	-	-	0	8	-	11	-	-
b	3	-	-	-	5	-	10	-	-
c	-	0	-	-	-	-	1	-	-
d	-	8	5	-	-	2	3	-	6
e	7	-	-	-	2	-	-	-	11
f	-	11	10	1	3	-	-	7	-
g	-	-	-	-	-	-	7	-	4
h	-	-	-	-	6	11	-	4	-

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

- b) Welche Worst-Case-Laufzeitkomplexität hat der Algorithmus von Prim, wenn die grauen Knoten in einem Heap nach Distanz verwaltet werden? Sei dabei n die Anzahl an Knoten und m die Anzahl an Kanten des Graphen. Eine Begründung ist nicht erforderlich.
- c) Beschreiben Sie kurz die Idee des alternativen Ansatzes zur Berechnung eines minimalen Spannbaumes von Kruskal.

Thema Nr. 2
 (Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:

Gegeben sei der nichtdeterministische endliche Automat $N = (Q, \{0, 1\}, \delta, q_0, F)$ mit $Q = \{A, B, C\}$, $q_0 = A$, $F = \{C\}$ und

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

- a) Wenden Sie die Potenzmengenkonstruktion an, um einen deterministischen endlichen Automaten für die Sprache $L(N)$ zu erhalten.
- b) Beweisen oder widerlegen Sie: für reguläre Ausdrücke α, β gilt:

$$L((\alpha|\beta)^*) = L(\alpha^*|\beta^*)$$

- c) Für eine gegebene Sprache $L \subseteq \{a, b, c\}^*$ sei $\text{replace}(L)$ die Sprache bestehend aus genau den Wörtern, die entstehen, wenn in jedem Wort aus L jedes Vorkommen von c durch ab ersetzt wird.

Beispiel: $\text{replace}(\{a, c, ba, ab, abc, cab\}) = \{a, ab, ba, abab\}$

- i) Zeigen Sie: Wenn L regulär ist, ist auch $\text{replace}(L)$ regulär.
- ii) Widerlegen Sie: Wenn $\text{replace}(L)$ regulär ist, ist auch L regulär.

Aufgabe 2:

- a) Zeigen Sie: Die Sprache $\{(ba)^n b(ab)^n \mid n \in \mathbb{N}\}$ ist regulär.
- b) Zeigen Sie, dass $\{a^i c^j b^{2i} c^k \mid i, j, k \in \mathbb{N}\}$ kontextfrei ist, indem Sie eine geeignete kontextfreie Grammatik angeben.
- c) Gegeben sei die kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit Sprache $L(G)$, wobei $V = \{S, X, Y\}$ und $\Sigma = \{a, b, c, d, e\}$. P bestehe aus den folgenden Produktionen:

$$S \rightarrow X \mid SbY \quad X \rightarrow dSe \mid a \quad Y \rightarrow X \mid YcX$$

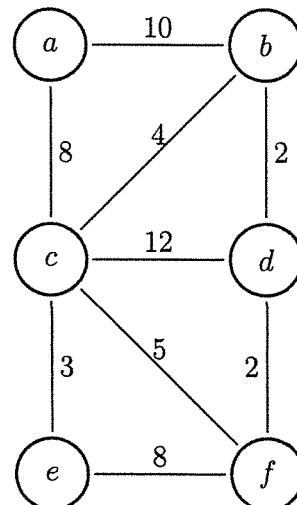
Bringen Sie G in Chomsky-Normalform.

Aufgabe 3:

- a) Ein AVL-Baum ist stets ein binärer Suchbaum. Geben Sie informell die zentrale Eigenschaft von AVL-Bäumen an, die an jedem Knoten erfüllt ist.
- b) Fügen Sie nacheinander die folgenden Zahlen in einen anfangs leeren AVL-Baum ein: 5, 7, 9, 12, 15.
Zeichnen Sie den Baum nach jedem Einfügen und nach jeder Rotation.
- c) In welcher Reihenfolge können die Zahlen 1, 2, 3, 4, 5, 6, 7, 8, 9 in einen anfangs leeren AVL-Baum eingefügt werden, sodass keine Rotationen stattfinden? Geben Sie eine mögliche Reihenfolge an und zeichnen Sie den daraus resultierenden AVL-Baum.

Aufgabe 4:

Sei G der folgende Graph.



- a) Bestimmen Sie mithilfe des Dijkstra-Algorithmus den Kürzeste-Wege-Baum im ungerichteten Graph G , ausgehend vom Knoten a. Geben Sie die Einzelschritte Ihrer Berechnung, inklusive der aktuellen Warteschlange, tabellarisch an. Zeichnen Sie den resultierenden Baum.

Ihre Tabelle sollte wie folgt beginnen:

a	b	c	d	e	f	Warteschlange
0	∞	∞	∞	∞	∞	a

- b) Der Algorithmus von Prim ist ein Algorithmus zur Bestimmung des minimalen Spannbaums in einem Graphen. Geben Sie einen anderen Algorithmus zur Bestimmung des minimalen Spannbaums an.
- c) Führen Sie den Algorithmus von Prim schrittweise auf G aus. Ausgangsknoten soll der Knoten a sein.

Ihre Tabelle sollte wie folgt beginnen:

a	b	c	d	e	f	Warteschlange
0	∞	∞	∞	∞	∞	a

Die Einträge der Tabelle geben an, wie weit der angegebene Knoten vom aktuellen Baum entfernt ist.

- d) Erklären Sie, warum der Kürzeste-Wege-Baum und der minimale Spannbaum nicht notwendigerweise identisch sind.

Aufgabe 5:

Eine S-Bahnstrecke mit n Stationen wird von einer Linie von Anfang bis Ende befahren. Wegen der vielen Haltepunkte ist die Fahrzeit sehr lang. Daher soll eine Expresslinie eingerichtet werden. Diese hält nicht an allen Stationen und kann dadurch die Fahrzeit erheblich verkürzen.

Einige Haltestellen werden als Expresshaltestellen ausgewählt und die Expresslinie hält nur an diesen Punkten. Durch eine geeignete Kombination an normalen Zügen und Expresszügen kann ein Nutzer seine Fahrzeit minimieren (die Fahrzeit ist hier die Gesamtzahl der Stopps des Nutzers – insbesondere vernachlässigen wir hierbei die Zeit für das eventuelle Wechseln eines Zuges).

Ziel ist es, dass jeder Nutzer eine Fahrzeit von höchstens $\mathcal{O}(\sqrt{n})$ hat. Finden Sie eine geeignete Verteilung der Expresshaltestellen. Beschreiben Sie, wie ein Nutzer effizient von Haltestelle i zu Haltestelle j fährt. Begründen Sie die daraus resultierende (asymptotische) Worst-Case-Fahrzeit.

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Herbst 2018

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____		
Kennwort: _____	Herbst 2018	46115
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Theoretische Informatik/Algorithmen/Datenstrukturen**

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

Anzahl der Druckseiten dieser Vorlage: **13**

Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1 (Reguläre Sprachen)

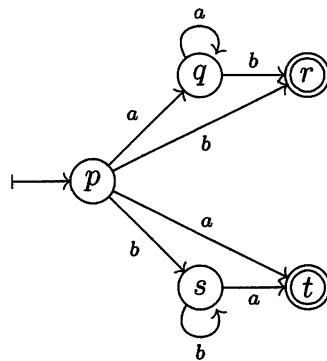
[24 PUNKTE]

- (a) [4 PUNKTE] Betrachten Sie die formale Sprache $L \subseteq \{a, b\}^*$ aller Wörter, die mit aa beginnen oder mit bb enden.

Geben Sie einen regulären Ausdruck für die Sprache L an.

- (b) [10 PUNKTE] Entwerfen Sie einen nichtdeterministischen endlichen Automaten, der die Sprache L aus Teilaufgabe (a) akzeptiert.

- (c) [10 PUNKTE] Wandeln Sie den folgenden nichtdeterministischen Automaten mit Hilfe der Potenzmengenkonstruktion in einen äquivalenten deterministischen Automaten um. (Berechnen Sie dabei nur erreichbare Zustände.)

**Aufgabe 2 (Kontextfreie Sprachen)**

[32 PUNKTE]

- (a) [4 PUNKTE] Betrachten Sie die formale Sprache L über dem Alphabet $\Sigma = \{0, 1, 2\}$ aller Wörter wv mit $w \in \{0, 1\}^*$ und $v \in \{1, 2\}^*$ so dass die Anzahl der 0en in w genau doppelt so hoch ist wie die der 2en in v .

Geben Sie zwei Wörter über Σ an, die in L enthalten sind und zwei Wörter über Σ , die nicht in L enthalten sind. Die Wörter sollen mindestens Länge 5 haben.

- (b) [10 PUNKTE] Geben Sie eine kontextfreie Grammatik für die Sprache L an.

Erklären Sie den Zweck der einzelnen Nichtterminale (Variablen) und der Grammatikregeln Ihrer Grammatik.

- (c) [2 PUNKTE] Geben Sie für eines Ihrer beiden Wörter aus Teilaufgabe (a), das in L enthalten ist, einen Ableitungsbaum des Wortes mit Ihrer Grammatik aus Teilaufgabe (b) an.
- (d) [10 PUNKTE] Beweisen Sie, dass die Sprache L aus Teilaufgabe (a) nicht regulär ist.
- (e) [6 PUNKTE] Ist die folgende Aussage richtig oder falsch? Begründen Sie Ihre Antwort:
„Jede Teilmenge einer kontextfreien Sprache liegt in der Klasse \mathcal{P} .“

Aufgabe 3 (Entscheidbarkeit)

[34 PUNKTE]

- (a) [6 PUNKTE] Betrachten Sie das folgende Entscheidungsproblem:

Eingabe: eine (geeignet codierte) Turingmaschine M , ein Eingabewort w und eine natürliche Zahl n

Aufgabe: entscheiden, ob die Turingmaschine M auf das Eingabewort w nach höchstens n Schritten hält

Ist dieses Problem entscheidbar? Begründen Sie Ihre Antwort.

- (b) [20 PUNKTE] Beweisen Sie mit Hilfe eines Reduktionsbeweises, dass das folgende Problem nicht entscheidbar ist:

Eingabe: eine (geeignet codierte) Turingmaschine M und sowie ein Eingabewort w

Aufgabe: entscheiden, ob M auf Eingabewort w gestartet hält und das Wort w nicht akzeptiert

- (c) [8 PUNKTE] Beweisen Sie, dass das Problem aus (b) semi-entscheidbar (= rekursiv aufzählbar) ist.

Aufgabe 4 (Hashing)

[71 PUNKTE]

Ein Wörterbuch, das die Operationen SUCHEN, EINFÜGEN und LÖSCHEN zur Verfügung stellt, kann mit einer Hashtabelle implementiert werden. Im Folgenden seien $m, n \in \mathbb{N}$ beliebig, $T[0, \dots, m - 1]$ eine Hashtabelle der Größe m und $S \subset \mathcal{U}$ eine Menge aus dem Universum \mathcal{U} mit $|S| = n$ die in T mittels der Hashfunktion $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$ gespeichert werden soll.

Hashing mit Verkettung

4.1 [2 Punkte] Beschreiben Sie *kurz*, wie eine Hashtabelle aufgebaut ist, die *Verkettung* zur Kollisionsbehandlung verwendet.

4.2 [6 Punkte] Beschreiben Sie *kurz*, wie die Wörterbuchoperationen SUCHEN, EINFÜGEN und LÖSCHEN in einer Hashtabelle wie in Aufgabe 4.1 beschrieben umgesetzt werden.

4.3 [6 Punkte] Zeigen Sie, dass für jede endliche Menge \mathcal{U} mit $|\mathcal{U}| \geq (n - 1)m + 1$ und jede Hashfunktion $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$ eine Menge $S \subseteq \mathcal{U}$ mit $|S| = n$ existiert, so dass alle Elemente von S durch h auf denselben Eintrag der Hashtabelle abgebildet werden.

Hinweis: Betrachten Sie die *größte* der Urbildmengen $U_i := \{s \in \mathcal{U} \mid h(s) = i\}$ (für $0 \leq i < m$) von $[0, \dots, m - 1]$ unter h .

4.4 [6 Punkte] Wir betrachten eine Hashtabelle, die *Verkettung* zur Kollisionsbehandlung verwendet und nehmen an, dass die Hashfunktion h in *konstanter* Zeit ausgewertet werden kann.

Für $0 \leq i < m$ sei $S_i := \{s \in S \mid h(s) = i\} \subset S$ die Menge der Urbilder von i in S . Begründen Sie, warum die asymptotische worst-case-Laufzeit einer Suchoperation $\Theta(1 + \max_i |S_i|)$ ist!

4.5 [3 Punkte] Was ist die asymptotische worst-case Laufzeit einer Einfüge- bzw. einer Löschoperation in einer Hashtabelle, die *Verkettung* zur Kollisionsbehandlung verwendet? Begründen Sie Ihre Antworten!

Hashing durch lineares Sondieren

4.6 [2 Punkte] Um die Position von $x \in S$ in einer Hashtabelle T , die *lineares Sondieren* zur Kollisionsbehandlung verwendet, zu bestimmen, wird mittels $h(x)$ sukzessive eine Indexfolge (i_0, \dots, i_{m-1}) berechnet (wobei $0 \leq i_j < m$ für alle $0 \leq j < m$). Geben Sie an, nach welcher Vorschrift die i_j berechnet werden.

- 4.7 [4 Punkte] Beschreiben Sie *kurz*, was in einem Eintrag $T[i]$ einer Hashtabelle T , die *lineares Sondieren* zur Kollisionsbehandlung verwendet, gespeichert sein kann. Geben Sie an, wie die Tabelle unmittelbar nach der Initialisierung aussieht.

Hinweis: Es gibt drei verschiedene Arten von Einträgen in T !

- 4.8 [2 Punkte] Geben Sie an, welcher Fehler auftreten kann, wenn ein neues Element in eine Hashtabelle eingefügt werden soll, bei der *lineares Sondieren* zur Kollisionsbehandlung verwendet wird.

- 4.9 [12 Punkte] Beschreiben Sie *kurz*, wie die Wörterbuchoperationen SUCHEN, EINFÜGEN und LÖSCHEN in einer Hashtabelle, die *lineares Sondieren* zur Kollisionsbehandlung verwendet, umgesetzt werden.

Hashing am Beispiel

Folgende Werte sollen in eine Hashtabelle eingefügt werden:

$$S = (5, 14, 17, 18, 16, 3, 20, 6, 7).$$

Wir betrachten dazu die beiden Hashfunktionen

$$h_1(k) = k \mod 11 \quad \text{sowie} \quad h_2(k) = (k \cdot (2 \cdot (k + 1) + 3)) \mod 11.$$

- 4.10 [6 Punkte] Erstellen Sie die Wertetabellen der beiden Hashfunktionen h_1 und h_2 für die Schlüssel in S .

- 4.11 [18 Punkte] Fügen Sie die Werte aus S in der angegebenen Reihenfolge in eine anfangs leere Hashtabelle der Größe 11 ein. Verwenden Sie dazu die Hashfunktionen h_1 und h_2 und verwenden Sie jeweils einmal *Verkettung* und einmal *lineares Sondieren* zur Kollisionsbehandlung (es sind also insgesamt vier Hashtabellen zu erzeugen). Geben Sie für die beiden Hashtabellen mit linearem Sondieren für jeden Schlüssel die Folge der untersuchten Indizes an.

- 4.12 [4 Punkte] Löschen Sie jetzt den Wert 20 aus den vier Hashtabellen und geben Sie die Hashtabellen erneut an.

Aufgabe 5 (Korrektheit von Algorithmen)

[32 PUNKTE]

Wir betrachten den nachstehenden Algorithmus:

MIN SUFFIX AVERAGE($A[1, \dots, n]$)

Eingabe : Ein Feld $A[1, \dots, n]$ von $n \geq 1$ Zahlen
Ausgabe : $\min_{1 \leq i \leq n} \frac{1}{(n-i+1)} \sum_{k=i}^n A[k]$

- 1 $b[0] \leftarrow \infty;$
- 2 $a[0] \leftarrow 0;$
- 3 für $s \leftarrow n$ runter bis 1 tue
 - 4 $l \leftarrow n - s + 1;$
 - 5 $a[l] \leftarrow \frac{(n-s) \cdot a[l-1] + A[s]}{(n-s+1)};$
 - 6 $b[l] \leftarrow \min(b[l-1], a[l]);$
- 7 zurück $b[n];$

5.1 [2 Punkte] Begründen Sie, dass der Wert der Variablen l der Nummer des aktuellen Schleifendurchlaufs entspricht (wenn wir mit dem Zählen der Schleifendurchläufe bei 1 beginnen).

Wir wollen beweisen, dass der Algorithmus korrekt ist, d.h., dass er für die Eingabe $A[1, \dots, n]$ den Wert

$$\min_{1 \leq i \leq n} \frac{1}{(n-i+1)} \sum_{k=i}^n A[k]$$

zurückgibt. Für $1 \leq l \leq n$ seien dazu folgende Werte definiert:

$$a_l := \frac{1}{l} \sum_{k=n-(l-1)}^n A[k] \quad \text{und} \quad b_l := \min_{n-(l-1) \leq i \leq n} \frac{1}{(n-i+1)} \sum_{k=i}^n A[k]$$

Wir wollen durch Induktion zeigen, dass für alle $1 \leq l \leq n$ die Aussage

$$\mathcal{A}_l : \text{Nach dem } l\text{-ten Durchlauf der Schleife ist } a[l] = a_l$$

gilt.

5.2 [4 Punkte] Führen Sie den Induktionsanfang $l = 1$ aus.

5.3 [8 Punkte] Führen Sie den Induktionsschritt $l - 1 \rightarrow l$ (für $1 < l \leq n$) aus.

Wir wollen als nächstes durch Induktion zeigen, dass für alle $1 \leq l \leq n$ die Aussage

\mathcal{B}_l : Nach dem l -ten Durchlauf der Schleife ist $b[l] = b_l$

gilt.

5.4 [4 Punkte] Führen Sie den Induktionsanfang $l = 1$ aus.

5.5 [10 Punkte] Führen Sie den Induktionsschritt $l - 1 \rightarrow l$ (für $1 < l \leq n$) aus.

Hinweis: Verwenden Sie das Resultat aus der vorigen Aufgabe: $a[l] = a_l$ für alle $1 \leq l \leq n$.

5.6 [4 Punkte] Folgern Sie, dass der Algorithmus MIN SUFFIX AVERAGE korrekt ist.

Thema Nr. 2
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1 (Reguläre Sprachen)**[16 PUNKTE]**

- (a) [2 PUNKTE] Geben Sie einen regulären Ausdruck für die Sprache L_1 an, die alle Wörter über dem Alphabet $\{a, b\}$ enthält, die mindestens zwei a 's haben.
- (b) [4 PUNKTE] Geben Sie einen möglichst einfachen und eleganten DEA für die Sprache $L((aa)^* + a^*b(a + b)^*)$ an. (Sie müssen dafür kein Konstruktionsverfahren nutzen.)
- (c) [2 PUNKTE] Betrachten Sie die folgenden Sprachen über dem Alphabet $\Sigma = \{a, b, c\}$:

$$L_3 = L((aa)^* + ab^*c)$$

$$L_4 = L((cc)^*(c + \varepsilon))$$

Konstruieren Sie einen Homomorphismus h mit $h(L_3) = L_4$. Begründen Sie Ihre Antwort.

Anmerkung: ein Homomorphismus ist hier eine Funktion $h : \Sigma \rightarrow \Sigma$, die buchstabenweise auf die Wörter einer Sprache angewendet wird. Z. B. falls wir den Homomorphismus h definieren als $h(a) := a$, $h(b) := a$ und $h(c) := c$, dann ist $h(abc) = aac$ und $h(\{abc, aa, ab, ac\}) = \{aac, aa, ac\}$.

- (d) [8 PUNKTE] Betrachten Sie die folgenden Sprachen über dem Alphabet $\Sigma = \{a, b\}$:

$$L_5 = \{w_1 abbb w_2 \mid w_1, w_2 \in \{a, b\}^*, \text{ die Länge von } w_2 \text{ ist teilbar durch } 4\}$$

$$L_6 = \{a^{n^2} \mid n \in \mathbb{N}, n \geq 1\}$$

Geben Sie für die Sprachen jeweils an, ob sie regulär sind oder nicht. Geben Sie einen Beweis für Ihre Antwort. Um Regularität zu zeigen, reicht die Angabe eines Automaten oder eines regulären Ausdrucks, der die Sprache akzeptiert.

Aufgabe 2 (Kontextfreie Sprachen)**[14 PUNKTE]**

Für den Beweis von Kontextfreiheit in dieser Frage reicht die Angabe eines Automaten oder einer Grammatik. (Beschreiben Sie dann die Konstruktionsidee des Automaten oder der Grammatik.) Für den Beweis von Nicht-Kontextfreiheit verwenden Sie eine der üblichen Methoden.

- (a) [3 Punkte] Seien L_1 und L_2 nicht-kontextfrei. Ist es dann immer so, dass auch $L_1 \cup L_2$ nicht-kontextfrei ist? Beweisen Sie Ihre Antwort.

- (b) [3 Punkte] Sei L_3 nicht-kontextfrei und L_4 regulär. Kann $L_3 \cup L_4$ kontextfrei sein? Kann $L_3 \cup L_4$ nicht-kontextfrei sein? Beweisen Sie Ihre Antworten.
- (c) [3+5 Punkte] Wir bezeichnen mit $\#\sigma(w)$ die Anzahl der Vorkommen des Zeichens σ im Wort w . Zum Beispiel ist $\#_a(baaca) = 3$ und $\#_b(baaca) = 1$. Betrachten Sie die folgenden Sprachen über dem Alphabet $\Sigma = \{a, b, c, d\}$:

$$L_5 = \{w \in L(a^*b^*c^*d^*) \mid \#_a(w) = \#_b(w) \text{ und } \#_c(w) = \#_d(w)\}$$

$$L_6 = \{w \in L((a + b + c + d)^*) \mid \#_a(w) = \#_b(w) \text{ und } \#_c(w) = \#_d(w)\}$$

Sind L_5 und L_6 jeweils kontextfrei oder nicht? Beweisen Sie Ihre Antwort.

Aufgabe 3 (Entscheidbarkeit und Komplexität)

[10 PUNKTE]

Sei $G = (V, E)$ ein ungerichteter Graph. Wir sagen, dass ein Knoten u von einem Knoten v gesehen wird, falls $u = v$ oder $(u, v) \in E$.

Wir nennen $A \subseteq V$ eine *alles sehende Menge (ASM)*, falls jeder Knoten in V von mindestens einem Knoten in A gesehen wird.

Wir nennen A *teilbar*, falls A partitionierbar in nicht-leere Teilmengen A_1 und A_2 ist, so dass jeder Knoten in V entweder von mindestens einem Knoten in A_1 oder von mindestens einem Knoten in A_2 gesehen wird (aber nicht von sowohl einem Knoten in A_1 als auch von einem Knoten in A_2).

Wir betrachten die folgenden Probleme:

ASM

Gegeben: Ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$

Gefragt: Hat G eine ASM A mit $|A| \leq k$?

Teilbare ASM

Gegeben: Ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$

Gefragt: Hat G eine teilbare ASM A mit $|A| \leq k$?

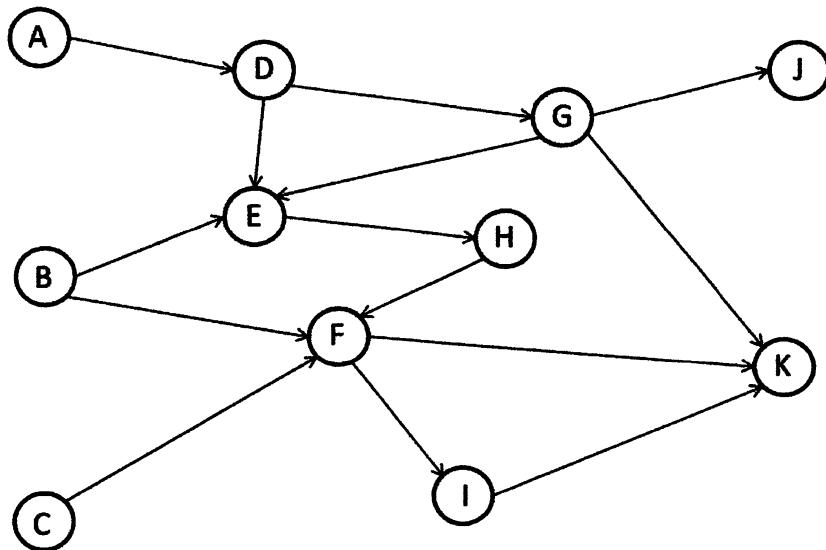
(a) [8 PUNKTE] Beweisen Sie, dass ASM in Polynomialzeit reduzierbar ist auf Teilbare ASM.

(b) [2 PUNKTE] Was können Sie aus a) folgern bezüglich der NP-Vollständigkeit der Probleme?

Aufgabe 4 (Längste Pfade)

[45 PUNKTE]

Gegeben sei der folgende azyklische, gerichtete Graph.



Der *kritische Pfad* eines solchen Graphen ist definiert als derjenige Pfad von einem beliebigen Knoten mit Eingangsgrad 0 zu einem beliebigen Knoten mit Ausgangsgrad 0 mit der maximalen Pfadlänge.

Analog ist der *kritische Pfad zu einem beliebigen Knoten* der Pfad mit maximaler Pfadlänge von einem Knoten mit Eingangsgrad 0 zu diesem Knoten. Im gegebenen Graphen ist beispielsweise der kritische Pfad zu Knoten E der von A über D und G nach E mit der Pfadlänge 3. Der Pfad von B aus ist kürzer (Pfadlänge 1), weil Knoten E nur direkt von Knoten B aus erreichbar ist. Von Knoten C aus ist Knoten E nicht erreichbar.

- (a) [5 PUNKTE] Bestimmen Sie den kritischen Pfad des gegebenen Graphen. Geben Sie die Knotenfolge des kritischen Pfades und seine Pfadlänge an.
- (b) [10 PUNKTE] Da der gegebene Graph azyklisch ist, können seine Knoten topologisch sortiert werden. Formal ist die *topologische Sortierung eines Graphen* $G = (V, E)$ eine injektive Abbildung

$$\text{ord} : V \rightarrow \{1, \dots, |V|\}$$

sodass gilt:

$$\forall (v, w) \in E : \text{ord}(v) < \text{ord}(w)$$

Dabei bezeichnet V die Knotenmenge und E die Kantenmenge des Graphen.

Führen Sie eine topologische Sortierung des Graphen durch. Geben Sie eine mögliche topologische Sortierung der Knoten des Graphen an.

- (c) [5 PUNKTE] Angenommen, Sie haben im gegebenen Graphen die kritischen Pfade zu allen Vorgängerknoten eines Knotens v bestimmt. Wie berechnet sich dann der kritische Pfad zum Knoten v ?

(d) [25 PUNKTE] Formulieren Sie anhand der Kenntnisse, die Sie durch die Aufgabenteile b und c gewonnen haben, einen Algorithmus, der die Länge des kritischen Pfades eines gerichteten, azyklischen Graphen berechnet und geben Sie die asymptotische Laufzeit Ihres Algorithmus in der O-Notation an.

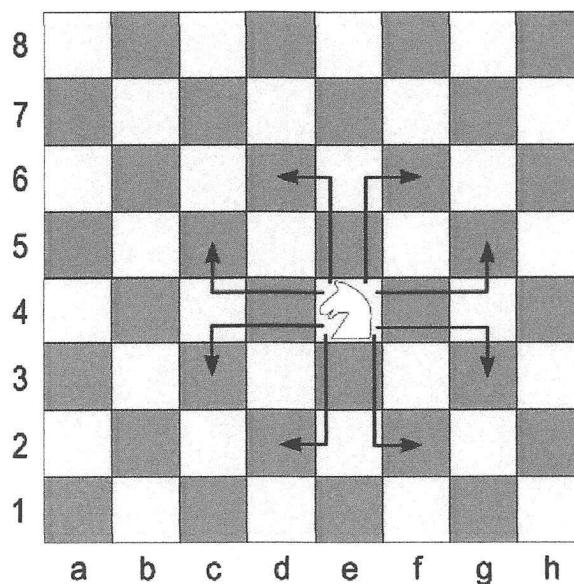
Hinweis: Sie dürfen in Ihrem Algorithmus eine Funktion $\text{topsort}(G)$ verwenden. Diese liefert eine topologische Sortierung der Knoten des Graphen G als Liste in einer asymptotischen Laufzeit in $O(|V| + |E|)$ zurück.

Aufgabe 5 (Backtracking)

[25 PUNKTE]

Das *Springerproblem* ist ein kombinatorisches Problem, das darin besteht, für einen Springer auf einem leeren Schachbrett eine Route von einem gegebenen Startfeld aus zu finden, auf der dieser jedes Feld des Schachbretts genau einmal besucht.

Ein Schachbrett besteht aus 8×8 Feldern. Ein Springer kann bei einem Zug von einem Ausgangsfeld aus eines von maximal 8 Folgefelder betreten, wie dies in der folgenden Abbildung dargestellt ist. Der Springer darf selbstverständlich nicht über den Rand des Schachbretts hinausspringen.



Eine Lösung des Springerproblems mit Startfeld $h1$ sieht wie folgt aus. Die Felder sind in ihrer Besuchsreihenfolge durchnummeriert. Der Springer bewegt sich also von $h1$ nach $f2$, dann von $f2$ nach $h3$ usw.

41	10	29	26	49	12	31	16
28	25	40	11	30	15	50	13
9	42	27	56	61	48	17	32
24	39	58	47	64	53	14	51
43	8	55	62	57	60	33	18
38	23	46	59	54	63	52	3
7	44	21	36	5	2	19	34
22	37	6	45	20	35	4	1

Formulieren Sie einen *rekursiven* Algorithmus zur Lösung des Springerproblems von einem vorgegebenen Startfeld aus. Es sollen dabei alle möglichen Lösungen des Springerproblems gefunden werden. Die Lösungen sollen durch *Backtracking* gefunden werden. Hierbei werden alle möglichen Teilrouten systematisch durchprobiert, und Teilrouten, die nicht zu einer Lösung des Springerproblems führen können, werden nicht weiterverfolgt. Dies ist durch rekursiven Aufruf einer Lösungsfunktion $huepf(x, y, z)$ zu realisieren, wobei

- x und y die Koordinaten des als nächstes anzuspringenden Feldes sind, und
- z die aktuelle Rekursionstiefe enthält. Wenn die Rekursionstiefe 64 erreicht und das betreffende Feld noch unbesucht ist, ist eine Lösung des Springerproblems gefunden.

Der initiale Aufruf Ihres Algorithmus kann beispielsweise über den Aufruf

$huepf(1, 8, 1)$

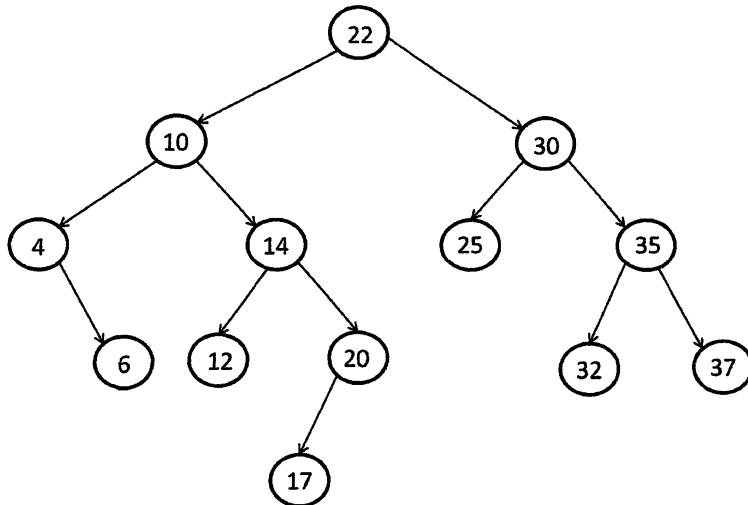
erfolgen.

Wählen Sie geeignete Datenstrukturen zur Verwaltung der unbesuchten Felder und zum Speichern gefundener (Teil)Lösungen. Der Algorithmus soll eine gefundene Lösung in der oben angegebenen Form ausdrucken, also als Matrix mit der Besuchsreihenfolge pro Feld.

Aufgabe 6 (Balancierte Bäume)

[20 PUNKTE]

Es sei der folgende AVL-Baum mit Schlüsseln aus \mathbb{N} gegeben:



- (a) [10 Punkte] Fügen Sie in den gegebenen Baum den Schlüssel 15 ein!

Rebalancieren Sie anschließend den Baum so, dass die AVL-Eigenschaft wieder erreicht wird. Zeichnen Sie den Baum nach jeder Einfach- und Doppelrotation und benennen Sie die Art der Rotation (Links-, Rechts-, Links-Rechts-, oder Rechts-Links-Rotation). Argumentieren Sie jeweils über die Höhenbalancen der Teilbäume. Zeichnen Sie nach jedem Schritt die Höhenbalancen in den Baum ein.

- (b) [10 Punkte] Löschen Sie den Schlüssel 25 aus dem gegebenen Baum! Rebalancieren Sie anschließend den Baum so, dass die AVL-Eigenschaft wieder erreicht wird. Zeichnen Sie den Baum nach jeder Einfach- und Doppelrotation und benennen Sie die Art der Rotation (Links-, Rechts-, Links-Rechts-, oder Rechts-Links-Rotation). Argumentieren Sie jeweils über die Höhenbalancen der Teilbäume.

46115

**Theoretische Informatik / Algorithmen / Datenstrukturen
(nicht vertieft)**

Frühjahr 2019

Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: _____	Frühjahr	
Kennwort: _____	2019	46115
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen
— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Theoretische Informatik/Algorithmen/Datenstrukturen**

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!

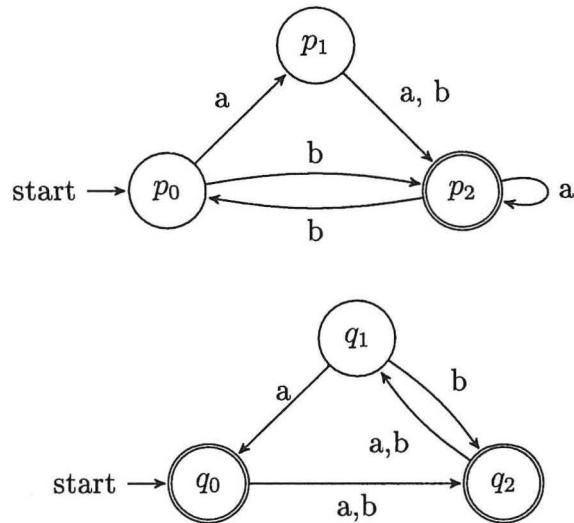
Aufgabe 1

Antworten Sie mit „Stimmt“ oder „Stimmt nicht“. Begründen Sie Ihr Urteil.

- (a) Sei $\Sigma = \{a, b\}$. Wenn $L_1 \leq \Sigma^*$ regulär ist, dann ist auch das Komplement $\Sigma^* \setminus L_1$ regulär.
- (b) Für reguläre Ausdrücke α, β gilt $L((\alpha + \beta)^*) = L(\alpha^* + \beta^*)$.
- (c) Kontextfreie Sprachen sind unter der Kleene-Stern-Operation abgeschlossen, d. h. für kontextfreies $L \subseteq \Sigma^*$ ist auch L^* kontextfrei.
- (d) Wenn es einen deterministischen Algorithmus gibt, der in polynomieller Zeit entscheidet, ob eine vorgelegte aussagenlogische Formel eine erfüllende Belegung hat, dann ist $\mathcal{P} = \mathcal{NP}$.
- (e) \mathcal{NP} -vollständige Probleme sind unentscheidbar.
- (f) Die Menge der (Kodierungen von) Turingmaschinen, die bei leerer Eingabe nach maximal 1000 Schritten halten, ist entscheidbar.

Aufgabe 2

- (a) Konstruieren Sie einen endlichen Automaten, dessen Sprache die Schnittmenge der Sprachen der folgenden zwei deterministischen endlichen Automaten ist.



- (b) Sei $L = \{a^i b^j c^{2i} \mid i, j \in \mathbb{N}\}$. Stellen Sie sich vor, dass ein Schüler/eine Schülerin S behauptet, einen deterministischen endlichen Automaten A konstruiert zu haben, der genau die Sprache L akzeptiert. Sie argumentieren mit dem Pumping-Lemma, dass L nicht regulär ist und deshalb die Sprache von A nicht L sein kann. S hält Ihre Argumentation für zu abstrakt und ist immer noch von der Korrektheit von A überzeugt.

Sie können S überzeugen, wenn Sie ein Wort w angeben, das Element von L ist, aber nicht von A akzeptiert wird oder von A akzeptiert wird und nicht Element von L ist.

Beschreiben Sie ein Verfahren, wie Sie in Abhängigkeit von A ein solches Wort w finden.

Aufgabe 3

Gegeben sei die kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit Sprache $L(G)$, wobei $V = \{S\}$ und $\Sigma = \{a, b\}$. P bestehe aus den folgenden Produktionen:

$$S \rightarrow SS \mid aSb \mid \varepsilon$$

- (a) Geben Sie alle neun Wörter $w \in L(G)$ mit Länge $|w| \leq 6$ an.
- (b) Bringen Sie G in Chomsky-Normalform und erklären Sie Ihre Vorgehensweise.

Aufgabe 4

Gegeben sei die Turingmaschine $M = (Q, q_0, F, \Sigma, \Gamma, \square, \delta)$ mit Zustandsmenge $Q = \{q_0, q_1, q_2, q_f\}$, wobei q_0 der Startzustand ist. Die Turingmaschine hält, wenn sie den Zustand q_f annimmt ($F = \{q_f\}$). 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 $\delta(q_0, a) = (q_1, b, L)$):

	a	b	\square
q_0	(q_1, b, L)	(q_0, b, R)	(q_1, \square, L)
q_1	(q_1, a, L)	(q_1, b, L)	(q_2, \square, R)
q_2	(q_2, a, L)	(q_2, b, R)	(q_f, \square, N)

- (a) Beschreiben Sie möglichst exakt, welche Schritte (Zustände, Schreibvorgänge, Bewegung des SL-Kopfes) die Turingmaschine bei der Abarbeitung des Wortes bab ausführt.
- (b) Beschreiben Sie möglichst exakt, welche Schritte (Zustände, Schreibvorgänge, Bewegung des SL-Kopfes) die Turingmaschine bei der Abarbeitung des Wortes $baba$ ausführt.
- (c) Charakterisieren Sie, bei welchen Eingaben die Turingmaschine hält und bei welchen nicht. Begründen Sie, ob Sie damit das Halteproblem gelöst haben.

Aufgabe 5 (Algorithmen und Datenstrukturen)

Gegeben sind eine endliche Menge von Wörtern $T = \{s_0, \dots, s_{m-1}\}$ und ein Zielwort w der Länge n . Gefragt ist, ob das Zielwort w als Konkatenation von Wörtern aus T zusammengesetzt werden kann, wobei die Wörter in T beliebig oft verwendet werden dürfen.

Beispiel:

$$\begin{aligned} w &= \text{apfelmuffinsundschokomuffins} \\ T &= \{f, lmu, e, ap, fin, ins, sund, kom, u, n, \\ &\quad su, dscho, unds, nd, uff, hok, felm\} \\ w &= \text{ap felm uff ins u n dscho kom uff ins} \end{aligned}$$

Mit $w(i, j)$ bezeichnen wir das Segment vom i -ten (Start bei 0) bis zum (einschl.) j -ten Buchstaben von w . Im Beispiel: $w(3, 10) = \text{elmuffin}$

Der folgende Algorithmus löst das Problem in exponentieller Zeit.

```
write(i, j): /* kann w(i, j) dargestellt werden? */
    if (i > j) return true;
    for (k=0..m-1)
        if (w(i, j) == s_k) return true;
    for (l=i..j-1)
        if (write(i, l) && write(l+1, j)) return true;
    return false;

main:
write (0, n-1);
```

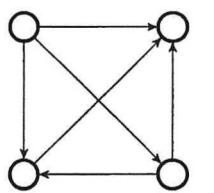
- (a) Entwerfen Sie mithilfe von dynamischer Programmierung einen Algorithmus, der das Problem in polynomieller Zeit löst.
- (b) Geben Sie die Laufzeit Ihres Algorithmus in O -Notation an, wobei Sie zur Vereinfachung annehmen dürfen, dass m , die Zahl der Muster, konstant ist.
- (c) Nennen Sie zwei weitere Beispiele für die Anwendungen dynamischer Programmierung.

Thema Nr. 2
 (Aufgabengruppe)

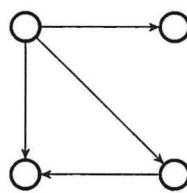
Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1 (Turniergraph)

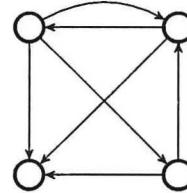
Ein *Turniergraph* ist ein gerichteter Graph auf n Knoten, in dem zwischen je zwei verschiedenen Knoten u, v genau eine Kante existiert - entweder (u, v) oder (v, u) , aber niemals beide oder keine, siehe Abbildung 1.



(a) Turniergraph



(b) Kein Turniergraph



(c) Kein Turniergraph

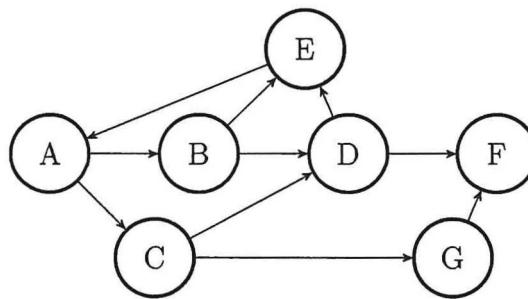
Abbildung 1: Beispiele zur Veranschaulichung der Definition des Turniergraphs

- (a) Zeigen Sie, dass in jedem Turniergraph **mindestens** ein Knoten existiert, von dem **mindestens** $(n - 1)/2$ Kanten ausgehen.
- (b) Eine Knotenmenge S *dominiert*, falls für jeden Knoten $x \notin S$ ein Knoten $s \in S$ existiert, so dass (s, x) eine Kante des Graphen ist.
 Geben Sie einen Algorithmus an, der in jedem Turniergraphen eine dominierende Knotenmenge der Größe $\mathcal{O}(\log n)$ findet. Argumentieren Sie, dass Ihr Algorithmus korrekt ist. Sie dürfen hierzu die Aussage von Teilaufgabe (a) nutzen.
- (c) Angenommen, es ist bekannt, dass die kleinste dominierende Knotenmenge $\mathcal{O}(\log n)$ Knoten enthält. Geben Sie einen Algorithmus an, der diese Knotenmenge in n^s Schritten berechnet, wobei $s \in \mathcal{O}(\log n)$ gilt.

Aufgabe 2 (Tiefensuche)

- (a) Führen Sie auf dem angegebenen Graphen eine Tiefensuche, ausgehend vom Knoten A aus. Dokumentieren Sie tabellarisch, in welcher Reihenfolge die Kanten betrachtet werden und ob die jeweils betrachtete Kante dem Tiefensuchbaum hinzugefügt wird. Falls mehrere Knoten zur Auswahl stehen, wählen Sie stets den Knoten mit der lexikographisch niedrigeren Bezeichnung.

Zeichnen Sie den resultierenden Baum.



- (b) Gegeben sei nun ein gerichteter Graph $G = (V, E)$, dessen Knoten mit natürlichen Zahlen beschriftet sind. Für jeden Knoten v soll der Knoten mit geringster Beschriftung bestimmt werden, der von v aus erreichbar ist. Geben Sie einen Algorithmus mit Laufzeit $\mathcal{O}(|V|(|V| + |E|))$ an, der dieses Problem löst.
- (c) Für das Problem aus Teil (b) existiert auch ein Algorithmus mit Laufzeit $\mathcal{O}(|V| + |E|)$. Geben Sie den verbesserten Algorithmus an. Argumentieren Sie, dass Ihr Algorithmus die geforderte Laufzeit erreicht.

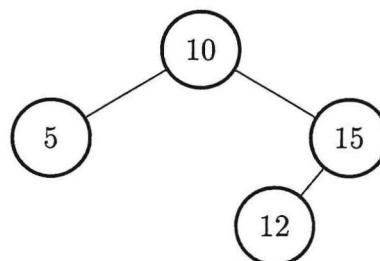
Aufgabe 3 (AVL-Bäume)

- (a) Zeigen oder widerlegen Sie die folgende Aussage: Wird ein Element in einen AVL-Baum eingefügt und unmittelbar danach wieder gelöscht, so befindet sich der AVL-Baum wieder in seinem Ursprungszustand.

- (b) Fügen Sie in den gegebenen Baum den Schlüssel 11 ein.

Rebalancieren Sie anschließend den Baum so, dass die AVL-Eigenschaft wieder erreicht wird. Zeichnen Sie den Baum nach jeder Einfach- und Doppelrotation und benennen Sie die Art der Rotation (Links-, Rechts-, Links-Rechts-, oder Rechts-Links-Rotation). Argumentieren Sie jeweils über die Höhenbalanzen der Teilbäume.

Tipp: Zeichnen Sie nach jedem Schritt die Höhenbalanzen in den Baum ein.



Aufgabe 4 (Formale Sprachen und Komplexität)

Das Shuffle-Produkt $L_1 \star L_2$ zweier Sprachen L_1, L_2 ist bekanntlich definiert durch

$$L_1 \star L_2 = \{u_1 v_1 u_2 v_2 \dots u_n v_n \mid u_1 u_2 \dots u_n \in L_1 \text{ und } v_1 v_2 \dots v_n \in L_2\}$$

wobei $u_1, \dots, u_n, v_1, \dots, v_n$ Wörter sind (möglicherweise leer). 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) Es sei $L = \{a^m b^n \mid m, n \geq 0\}$, also die Sprache des regulären Ausdrucks $a^* b^*$. Zeigen Sie, dass gilt $L \star L = \{a, b\}^*$.
- (b) Das Shuffle-Produkt regulärer Sprachen ist wiederum regulär. Begründen Sie dies durch eine geeignete Automatenkonstruktion. Mit anderen Worten: Beschreiben Sie, wie man aus zwei endlichen Automaten $A_1 = (\Sigma, Q, \delta, q_I, F)$ und $A_2 = (\Sigma, Q', \delta', q'_I, F')$ über demselben Alphabet einen Automaten für $L(A_1) \star L(A_2)$ konstruiert.
- (c) Begründen Sie, ob die folgende Aussage wahr ist:
“Jedes NP-vollständige Problem ist entscheidbar.”