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

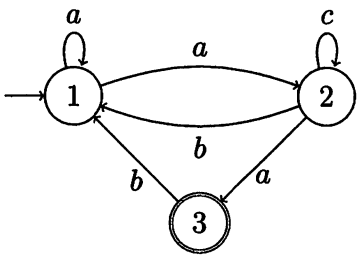
Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1:
Endliche Automaten

Betrachten Sie den NEA $\mathcal{A}$:



a) Konstruieren Sie mit Hilfe der Potenzmengenkonstruktion einen deterministischen endlichen Automaten, der dieselbe Sprache wie $\mathcal{A}$ erkennt.

b) 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

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

Fortsetzung nächste Seite!

b) Geben Sie eine kontextfreie Grammatik für $L_2 = \{a^n bc^m \mid n \neq m\}$ 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:

$$S \to AA \mid AC$$

$$A \to AB \mid a$$

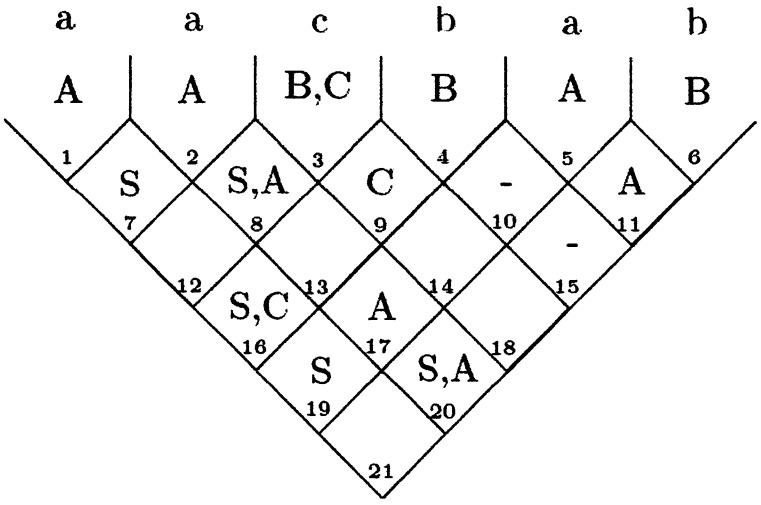
$$B \to b \mid c$$

$$C \to SC \mid BB \mid c$$

Betrachten Sie außerdem das nachfolgende Tableau, das durch Anwendung des CYK-Algorithmus mit der Grammatik G und dem Wort $w = aacbab$ enstanden 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).



Fortsetzung nächste Seite!

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?

a) Zeigen Sie, dass sich 3SAT in polynomieller Zeit auf 4SAT reduzieren lässt, d. h. $3SAT \leq_p 4SAT$.

b) Was können Sie aus a) folgern, wenn Sie wissen, dass 3SAT NP-vollständig ist?

c) Was können Sie aus a) folgern, wenn Sie wissen, dass 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.

a) Gibt es reguläre Sprachen L, so dass L^* eine endliche Sprache ist?

b) Gibt es unendlich viele reguläre Ausdrücke α, so dass $L(\alpha^*) \cap L(\alpha) = L(\alpha)$?

c) Gibt es WHILE-Programme, die NP-vollständige Probleme lösen?

d) Falls L_1 und L_2 kontextfreie Sprachen sind, ist $L_1 \cap L_2$ dann immer eine entscheidbare Sprache?

e) Wir nennen eine Sprache L LOOP-berechenbar, falls ihre charakteristische Funktion

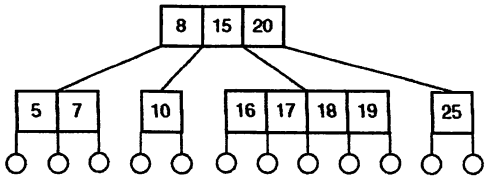
$$\chi(w) := \begin{cases} 0 & \text{falls } w \notin L \\ 1 & \text{falls } w \in L \end{cases}$$

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

Fortsetzung nächste Seite!

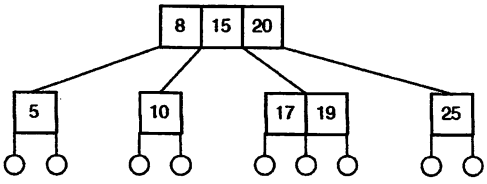
Aufgabe 6:
(2,4)-Bäume

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 Löschschritt 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.

Fortsetzung nächste Seite!

Aufgabe 7:

Durchmustern von Graphen

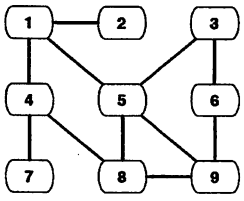
Wir betrachten den Algorithmus zur Tiefensuche von einem Knoten v in einem ungerichteten Graphen G:

DFSREC(G, v)

```
   Input : Graph G = (V, E) und ein Knoten v ∈ V
 1 begin
 2 |  setVertexLabel(v, U);
 3 |  forall the e ∈ G.incidentEdges(v) do
 4 |  |  if getEdgeLabel(e) = U then
 5 |  |  |  w ← G.oppositeVertex(v, e);
 6 |  |  |  if getVertexLabel(w) = U then
 7 |  |  |  |  setEdgeLabel(e, D);
 8 |  |  |  |  DFSREC(G, w);
 9 |  |  |  else
10 |  |  |  |  setEdgeLabel(e, 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 anzuzeigen, 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.

Fortsetzung nächste Seite!

b) Zeichnen Sie den Rekursionsbaum für den Aufruf 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.

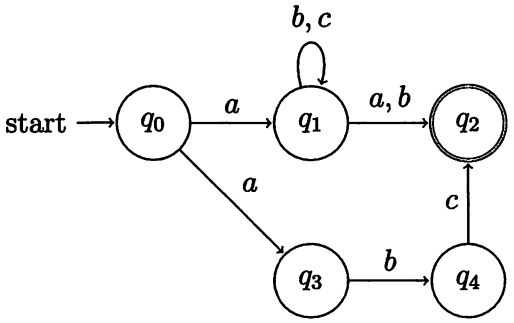
Thema Nr. 2
(Aufgabengruppe)

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.

Fortsetzung nächste Seite!

Aufgabe 2:
Kontextfreie Sprachen

a) Sei $\Sigma = \{a, b, c\}$ und $L_1 \subseteq \Sigma^*$ mit $L_1 = \{a^i b^j c^i \mid i, j \in \mathbb{N}\}$.

i) Geben Sie eine kontextfreie Grammatik für die Sprache L_1 an.

ii) 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:

$$S \to BS \mid UC \qquad T \to UB \mid AC \qquad U \to AT$$
$$A \to a \qquad B \to b \qquad C \to c$$

i) Zeigen Sie $baacc \in L_2$.

ii) 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	a	c	b

iii) Wie entnehmen Sie der Tabelle, dass $aacb \notin L_2$ gilt?

iv) 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?

Fortsetzung nächste Seite!

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;
  ...
```

Fortsetzung nächste Seite!

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
▼ k	*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 := (String, 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
    ...
```

Fortsetzung nächste Seite!

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

a) Stellen Sie die Halde graphisch als (links-vollständigen) Baum dar.

b) 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.

c) 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

a) 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. "A_1" stets vor "A_2" 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. "A_1".equals("A_2") aber "A_1" != "A_2").

L	A_1	B_1	F	A_2	B_2

...

b) 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;
     ...
```