Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Herbst 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:	12

Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

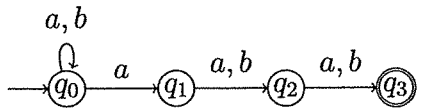
Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

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:

$$S \to AB \mid C$$

$$A \to a \mid \varepsilon$$

$$B \to BC$$

$$C \to c$$

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.

Fortsetzung nächste Seite!

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. $L_1 = \{w \in \{a\}^* \mid$ die Länge von w ist 2^n 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 $k \in \{1, 2, 3, \ldots\}$.
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 $k \in \{3, 4, 5, \ldots\}$.
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

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

Fortsetzung nächste Seite!

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 $h : \mathcal{U} \to \{0, \ldots, m-1\}$ 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.

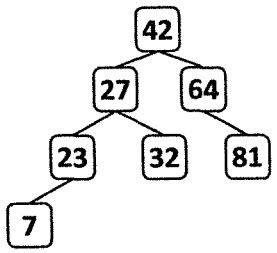
a) Fügen Sie in einen leeren binären Suchbaum der Reihe nach die folgenden Schlüssel ein und geben Sie den fertigen Baum an:

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
   a. im Falle des AVL-Baums die durch das Einfügen veränderten Tupel [H/B] aus Höhe H und Balancefaktor B der betroffenen Knoten an bzw.
   b. 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).

Fortsetzung nächste Seite!

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

```
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 void simuliere(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]++;
	}
}
```

```
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)
}
```

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

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

Fortsetzung nächste Seite!

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.

Fortsetzung nächste Seite!

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.

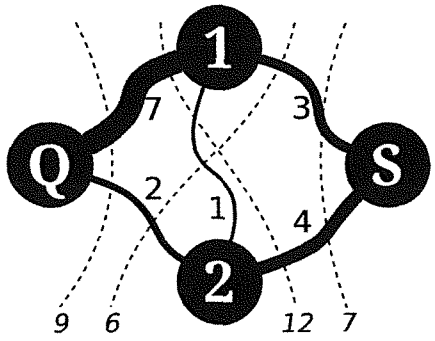
Gegeben sei folgende geschachtelt-rekursive Funktion für $n, m \geq 0$:

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

a) Implementieren Sie die obige Funktion $a(n,m)$ zunächst ohne weitere Optimierungen als Prozedur/Methode in einer Programmiersprache Ihrer Wahl.
b) 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.

Fortsetzung nächste Seite!

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.

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 {
      // Fuegt <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) {
      ...
```

Fortsetzung nächste Seite!

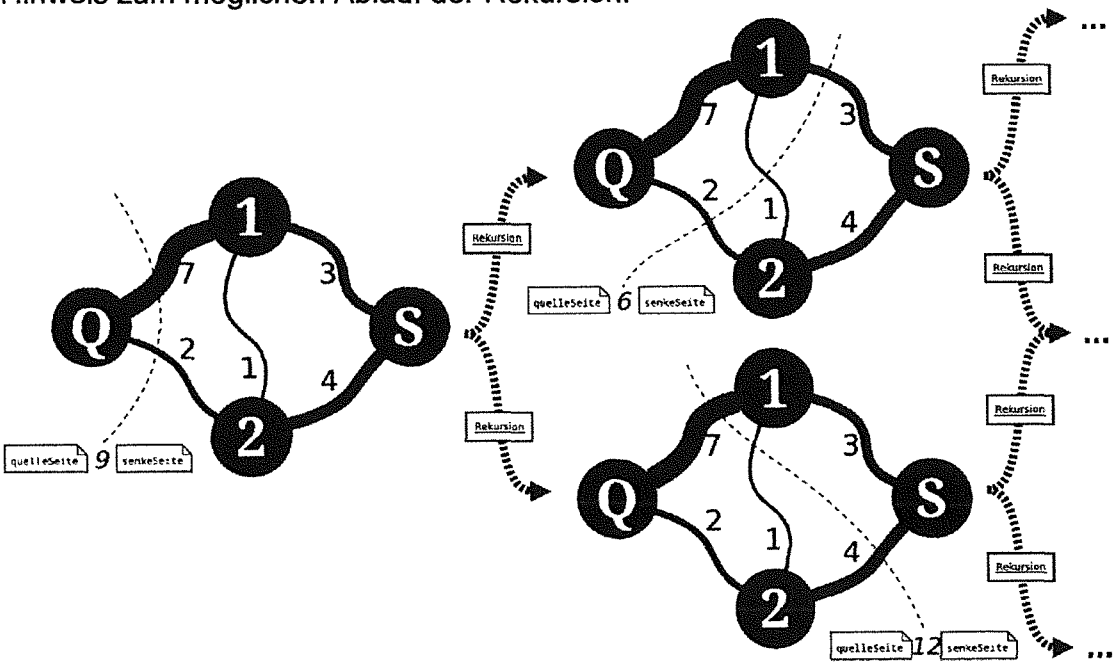
c) Die Berechnung des *maximalen Durchflusses* beginnt mit folgender Methode:

```
// Bestimmt den maximal moeglichen 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 moeglichen 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 $\mathbb{N}_0 = \{0,1,2,\ldots\}$.

(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

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

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

Zeigen Sie durch Reduktion von Co-H_0, dass die Menge

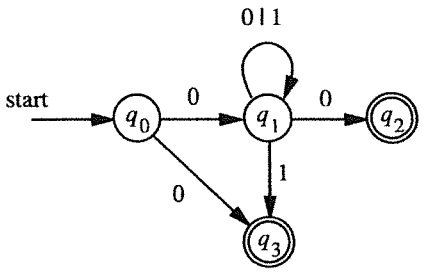
$$L_2 = \{\langle M \rangle \mid \text{es gibt } \textbf{genau} \text{ eine Zahl } n, \text{ so dass } M \text{ gestartet mit } n \text{ hält}\}$$

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

Fortsetzung nächste Seite!

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 zustande gekommen sind. Führen Sie *keine* „Vereinfachungen" durch!

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

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