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.

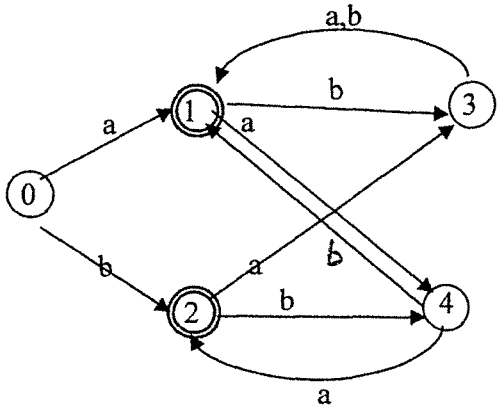
Aufgabe 2: reguläre Sprachen

Gegeben sei die Sprache $L = \{a\ w \mid n \geq 1,\ w \in \{b,c\}^*$ 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 $P = \{S \rightarrow A\ X,\ X \rightarrow S\ B,\ S \rightarrow ab,\ A \rightarrow a,\ B \rightarrow b\}$
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 Turingmachine M an, die L erkennt.
Die Eingabeworte werden durch $ (oder das Blanksymbol) abgeschlossen.

a) Beschreiben Sie in Worten, wie Ihre Turingmachine arbeitet.
b) 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 -> N und g:N -> N, wobei
$f(n) = 0.01\ n^3 + n - 1$ und
$g(n) = 100\ (n+2)^2 + 30$

Welche der folgenden Aussagen gilt? Beweisen Sie Ihre Angabe.
a) $f(n) \in O(g(n))$
b) $g(n) \in O(f(n))$

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.

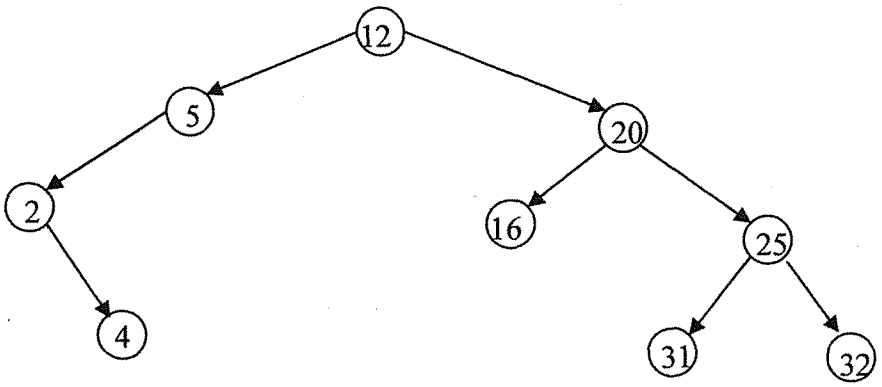
Beschreiben Sie den Ablauf von Quicksort durch Angabe der Zwischenergebnisse nach jedem Durchlauf.
Markieren Sie das jeweils gewählte Pivotelement.

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

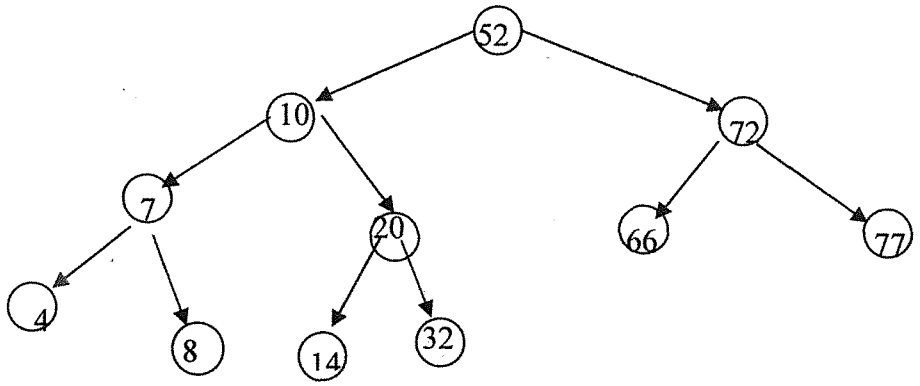
Fortsetzung nächste Seite!

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.

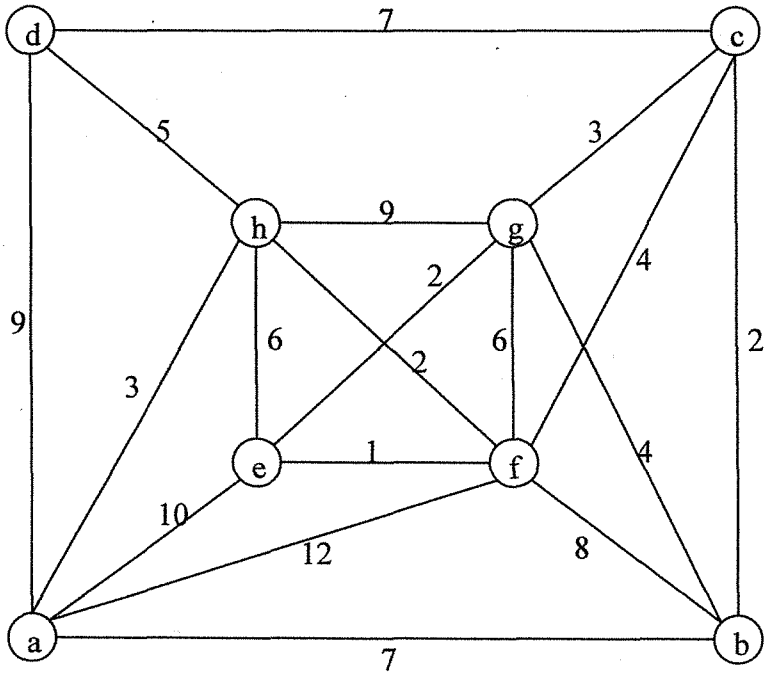


Fortsetzung nächste Seite!

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∈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 $\overline{L}$ konstruieren.

1. Geben Sie eine kontextfreie Grammatik für L an.

2. Begründen Sie, dass $\overline{L} = L_1 \cup L_2 \cup L_3 \cup \{\in\}$, 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 $\overline{L}$ 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;
            ...
```

Fortsetzung nächste Seite!

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

`public LinkedList()`

Constructs an empty list.

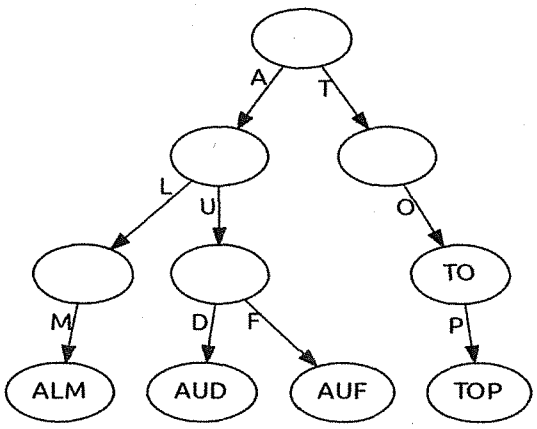
`public boolean contains(Object o)`

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, 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 zugehoerigem 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.

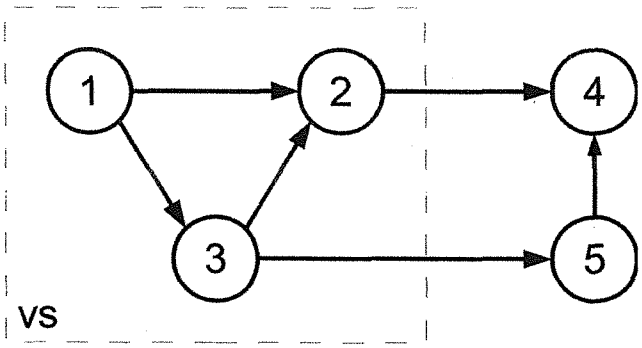
Teilaufgabe 5: „Funktionale Programmierung"

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, 4), (3, 4), (3, 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) => ...
```

Fortsetzung nächste Seite!

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 => ...
```