ulm university universität ulm

1. Klausur zur Veranstaltung

Einführung in die Informatik II - Vertiefung

und **Allgemeine Informatik 2**

im Sommersemester 2021

Prüfer: Dr. Jens Kohlmeyer — Fakultät Ingenieurwissenschaften, Informatik, Psychologie

31.07.2021, 9 Uhr — **Bearbeitungszeit: 90 min**

Nachname:	Vorname:	Matrikelnummer:
Studiengang und Abschluss:		Fachsemester:

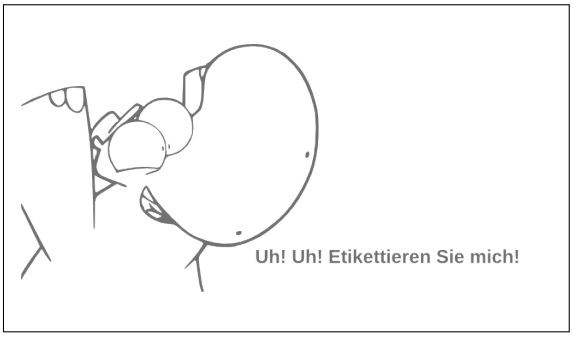
Hiermit erkläre ich, dass ich prüfungsfähig bin.
Sollte ich nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

_____

Datum, Unterschrift des Prüfungsteilnehmers

Zur allgemeinen Beachtung:

- Füllen Sie das Deckblatt vollständig und korrekt aus.
- Lesen Sie sich zunächst die Klausur sorgfältig durch (sie besteht aus 13 Seiten).
- Bearbeiten Sie die Aufgaben direkt auf den Aufgabenblättern.
- Als Hilfsmittel ist das vorgegebene Cheatsheet im DIN-A4-Format zugelassen. Beide Seiten dürfen handschriftlich beschrieben sein.
- Aufgaben, welche nicht mit einem dokumentenechten Stift in den Farben blau oder schwarz bearbeitet worden sind, werden nicht bewertet.



Zusätzlich benötigtes Papier wird Ihnen von der Aufsicht zur Verfügung gestellt.

Punkteverteilung						
1 von 12	2 von 12	3 von 12	4 von 12	5 von 12	$\sum$ von 60	**Note**
						Korrektur
Einsichtnahme ohne Nachkorrektur ◯				Einsichtnahme mit Nachkorrektur ◯		

Aufgabe 1 - Verständnisfragen

3 + 4 + 5 = 12 Punkte

a) Erläutern Sie welche **notwendige** Bedingung gelten muss, damit ein gerichteter Graph topologisch sortierbar ist und beschreiben Sie, warum die Nicht-Erfüllung den Graphen topologisch un-sortierbar macht.

b) Erläutern Sie warum die Funktionsweise des *Algorithmus von Dijkstra* garantiert, dass die kürzesten Pfade zu allen erreichbaren Knoten gefunden werden.

c) Erläutern Sie die Begriffe *Black-Box-Test* und *White-Box-Test*. Gehen Sie dabei auf die Unterschiede zwischen den Tests und die unterschiedlichen Abdeckungskriterien ein.

Diese Seite wurde für ein besseres Layout leer gelassen.

Aufgabe 2 - Suchen

3 + 2 + 7 = 12 Punkte

a) Welche der folgenden Aussagen zu den Algorithmen für Mustervergleiche treffen zu?
Hinweis: Mehrere Antworten können richtig sein.

- ◯ Der Brute-Force-Algorithmus (BF-Algorithmus) benötigt **immer** mehr Vergleiche als der Knuth-Morris-Pratt-Algorithmus (KMP-Algorithmus).
- ◯ Es gibt Fälle in denen der BF-Algorithmus weniger Vergleiche benötigt als der Boyer-Moore-Algorithmus (BM-Algorithmus).
- ◯ Es gibt Fälle in denen der BM-Algorithmus weniger Vergleiche benötigt als der KMP-Algorithmus.
- ◯ Der KMP-Algorithmus benötigt im *worst-case* O(N+M) Vergleiche.
- ◯ Der BM-Algorithmus benötigt im *average-case* O(N/M) Vergleiche.
- ◯ Die Randtabelle im KMP-Algorithmus für ein Wort der Länge 6 hat insgesamt 7 Einträge.

b) Erstellen Sie die delta1-Tabelle, welche für den BM-Algorithmus benötigt wird, für das Pattern *banana*.

c) Suchen Sie mit dem Algorithmus von Boyer-Moore (BM) im unteren Text nach dem **ersten Vorkommen** des Patterns *banana* aus Teilaufgabe **b)**. Füllen Sie für jedes Verschieben des Musters eine Zeile der nachstehenden Tabelle aus. Schreiben Sie in jede Zeile das gesamte Muster und **umkringeln** Sie alle Zeichen des Musters, die mit dem Text verglichen werden. Geben Sie für jeden Shift an, wie Sie diesen bestimmt haben. Notieren Sie dies **hinter** dem Muster auf dem Sie den Shift berechnet haben.
Hinweis: Sie benötigen nicht alle Zeilen der Tabelle.

b	a	n	b	a	n	a	n	a	s	e	l

Diese Seite wurde für ein besseres Layout leer gelassen.

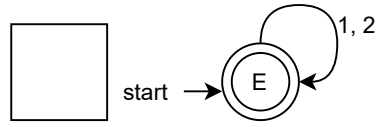
Aufgabe 3 - Formale Sprachen

4 + 4 + 3 + 1 = 12 Punkte

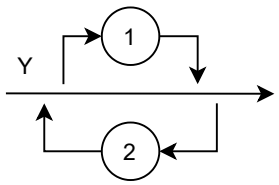
a) Im Folgenden sehen Sie einen regulären Ausdruck, einen endlichen Automaten, ein Syntaxdiagramm und eine EBNF. Markieren Sie, welche davon jeweils die gleiche Sprache erzeugen bzw. beschreiben, indem Sie in die Box eine Zahl eintragen. *Wären Sie zum Beispiel der Meinung, dass alle unterschiedliche Sprachen beschreiben bzw. erzeugen, dann tragen Sie in jede Box eine andere Zahl ein.*

☐ $\alpha = (1?2^*)^*$

☐



☐



☐ $A \rightarrow 1[A] \mid 2\{A\}$

b) Definieren Sie einen **deterministischen** endlichen Automaten der alle durch 3 teilbaren Zahlen akzeptiert.
Hinweis: Eine Zahl ist genau dann durch 3 teilbar, wenn ihre Quersumme durch 3 teilbar ist. 0 ist durch 3 teilbar!

c) Geben Sie eine Grammatik an, welche eine beliebige Menge von *öffnenden* Klammern, gefolgt von der **doppelten** Menge an *schließenden* Klammern produziert.

d) Ist die Sprache die von der Grammatik aus Aufgabe **c)** generiert wird vom Typ 3? Begründen Sie.

Aufgabe 4 - Java - Datenstrukturen

4 + 3 + 5 = 12 Punkte

Betrachten Sie folgende Klassen:

```java
public class Person {
    String name;
    Sex sex;
    Person parentA;
    Person parentB;

    public Person(String name, Sex sex) {
        this.name = name;
        this.sex = sex;
    }

    public static Person aFamily() {
        Person a = new Person("A", Sex.FEMALE);
        Person b = new Person("B", Sex.MALE);
        Person c = new Person("C", Sex.OTHER);
        Person d = new Person("D", Sex.OTHER);
        Person e = new Person("E", Sex.FEMALE);
        Person f = new Person("F", Sex.MALE);
        Person g = new Person("G", Sex.OTHER);
        a.parentA = b;
        b.parentA = d;
        f.parentA = g;
        a.parentB = c;
        b.parentB = e;
        c.parentB = f;
        System.out.println(a.noMissingParents());
        return a;
    }
}
```

```java
public enum Sex {
    MALE, FEMALE, OTHER;
}
```

a) Zeichnen Sie das Objekt, welches von der Methode `aFamily()` zurückgeliefert wird, so wie alle Objekte die mit diesem direkt oder indirekt verbunden sind, in UML Objektdiagramm Notation. Schreiben Sie hierbei auch den Attributnamen einer Referenz auf den zugehörigen gerichteten Pfeil zwischen Objekten.

b) Implementieren Sie die Methode `noMissingParents()` der Klasse `Person` welche überprüft, ob in der gesamten Familie der Person entweder beide Elternteile existieren oder beide Elternteile unbekannt (`null`) sind und in diesem Fall `true` zurück gibt. Sonst soll `false` zurück gegeben werden.

```
public boolean noMissingParents() {
```

c) Implementieren Sie die Methode `getAncestors(int maxGenerations)` der Klasse `Person`. Die Methode soll eine unsortierte Liste aller Vorfahren der Person zurückliefern. Dabei soll maximal `maxGenerations` weit zurückgeblickt werden. *Ein Wert von 1 für `maxGenerations` würde z.B bedeuten, dass nur die Eltern berücksichtigt werden, während ein Wert von 3 alle Vorfahren bis einschließlich der Ur-Großeltern zurückliefern würde.* Beachten Sie außerdem, dass die Person selbst **nicht** in der Liste ihrer Vorfahren enthalten sein darf!
Hinweis: Sie dürfen die Methode `addAll(List<T> list)` des Interfaces `List` verwenden. Diese fügt alle Elemente der übergebenen Liste an die Liste an, auf der die Methode aufgerufen wurde.
Hinweis: Direkte Vorfahren heißt hier, dass nur die Elternrelationen zwischen Objekten berücksichtigt werden sollen.

```
public List<Person> getAncestors(int maxGenerations) {
```

Diese Seite wurde für ein besseres Layout leer gelassen.

Aufgabe 5 - Java - Queues

1.5 + 2.5 + 3 + 5 = 12 Punkte

Betrachten Sie folgendes Interface und folgende Klassen:

```java
public interface CapacityLimitedQueue<T> {
    boolean enqueue(T element);
    T dequeue();
    int getCapacity();
}
```

```java
public class CapacityLimitedArrayQueue<T>
  implements CapacityLimitedQueue<T> {
    Object[] queue;
    int currentSize = 0;

    public CapacityLimitedArrayQueue(int size) {
        queue = new Object[size];
    }
    //...
    @Override
    public int getCapacity() {
        return queue.length;
    }
}
```

```java
public class QueueUtil {
    //...
}
```

a) Implementieren Sie die Methode `enqueue(T element)` der Klasse `CapacityLimitedArrayQueue` welche das übergebene Element der Queue anfügt insofern diese noch freie Plätze hat und in diesem Fall `true` zurückliefert. Sollte die Kapazität der Queue bereits gefüllt sein, soll das Element verworfen und `false` zurückgeliefert werden.
Beispiele:
Queues haben hier in den Beispielen eine Capacity von 3.
[1 ← 2], 3 ⇒ true; [1 ← 2 ← 3]
[1 ← 2 ← 3], 9 ⇒ false; [1 ← 2 ← 3]

```java
public boolean enqueue(T element) {
```

b) Implementieren Sie die Methode `dequeue()` der Klasse `CapacityLimitedArrayQueue` welche den Wert des vordersten Elements zurückgibt und das Element aus der Queue löscht.
Beispiele:
$[1 \leftarrow 2 \leftarrow 3] \Rightarrow 1; [2 \leftarrow 3]$
$[2 \leftarrow 3 \leftarrow 1] \Rightarrow 2; [3 \leftarrow 1]$

```
public T dequeue() {
```

c) Implementieren Sie die Methode `reverse` der Klasse `QueueUtil` welche die Reihenfolge der Elemente der übergebenen Queue umdreht.
Beispiele:
$[1 \leftarrow 2 \leftarrow 3] \Rightarrow [3 \leftarrow 2 \leftarrow 1]$
$[1 \leftarrow 2 \leftarrow 3 \leftarrow 1] \Rightarrow [1 \leftarrow 3 \leftarrow 2 \leftarrow 1]$

```
public static <T> void reverse(CapacityLimitedQueue<T> queue) {
```

d) Implementieren Sie die Methode `reprioritize` der Klasse `QueueUtil` welche die Reihenfolge der Elemente in der übergebenen Queue den Prioritäten aus der übergebenen `HashMap` anpasst. Beachten Sie dabei, dass die vergebenen Prioritäten im Intervall [0,127] liegen und niemals doppelt vergeben werden. *Die Methode ist somit nur für Queues mit einer Kapazität von <129 anwendbar.*
Beispiele:
queue: $[1 \leftarrow 2 \leftarrow 3]$, *priority*: $[(1 \rightarrow 10), (2 \rightarrow 127), (3 \rightarrow 100)] \Rightarrow [2 \leftarrow 3 \leftarrow 1]$
queue: $[1 \leftarrow 9]$, *priority*: $[(1 \rightarrow 10), (9 \rightarrow 22)] \Rightarrow [9 \leftarrow 1]$

```
public static <T> void reprioritize(CapacityLimitedQueue<T> queue,
HashMap<T,Integer> priority) {
```

Falls Sie noch Platz benötigen, so können Sie dieses Blatt benutzen. Weitere Blätter erhalten Sie von der Aufsicht. Machen Sie **eindeutig kenntlich**, welche Aufgaben hier bearbeitet bzw. fortgesetzt werden und **streichen** Sie den Lösungsversuch eindeutig durch, den sie nicht bewertet haben wollen.