ulm university universität ulm

2. Klausur zur Veranstaltung

Einführung in die Informatik II - Vertiefung

und **Allgemeine Informatik 2**

im Sommersemester 2021

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

25.09.2021, 10 Uhr — **Bearbeitungszeit: 90 min**

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

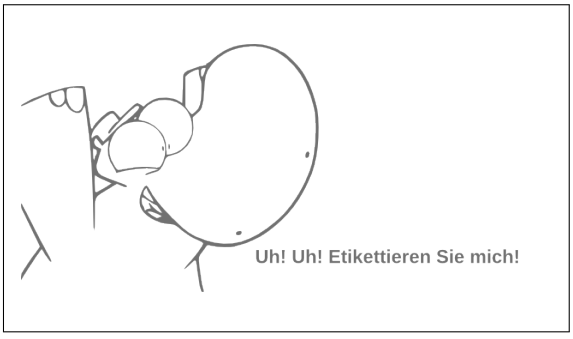
Hiermit erkläre ich, dass ich prüfungsfähig bin.
Sollte ich nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

_____

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 10 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) Erklären Sie, was der *Rand* eines Wortes und dessen *Dicke* sind, und geben Sie die Ränder der Worte *erdbeere* und *fffff* an.

b) Erläutern Sie die Vorgehensweise des *Boyer-Moore Algorithmus* zur Suche in Texten unter Verwendung der *Bad-Character-Heuristik*. Gehen Sie dabei auch auf die Unterschiede zum *Knuth-Morris-Pratt Algorithmus* ein.

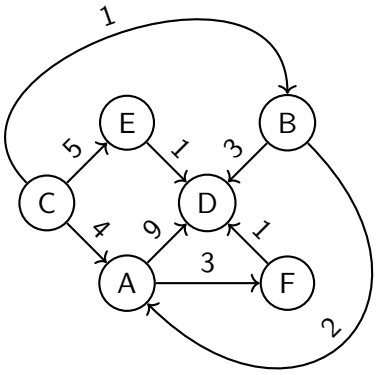
c) Nennen Sie die **5** Phasen der Softwareentwicklung. Beschreiben Sie für jede Phase kurz deren Funktion für die Softwareentwicklung.

Diese Seite wurde für ein besseres Layout leer gelassen.

Aufgabe 2 - Graphen

2 + 3.5 + 3.5 + 3 = 12 Punkte

Betrachten Sie folgenden gerichteten, gewichteten Graphen G:

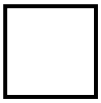


a) Erklären Sie den Begriff *Weg* im Kontext von Graphen. Geben Sie ein Beispiel aus dem Graph G an.

b) Wenden Sie den Dijkstra-Algorithmus mit Startknoten C auf G an.

c) Geben Sie die Adjazenzmatrix für G an.

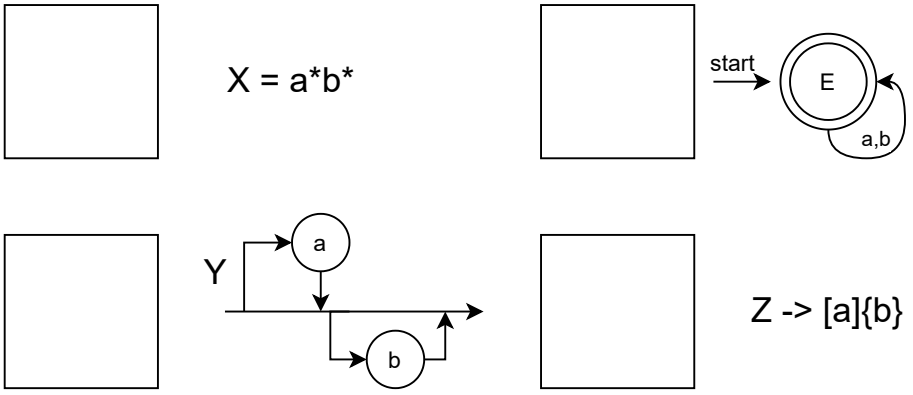
d) Ist G topologisch sortierbar? Begründen Sie Ihre Antwort. Im Falle, dass G sortierbar ist, geben Sie eine valide topologische Sortierung an.



Aufgabe 3 - Formale Sprachen

4 + 4.5 + 3.5 = 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.*



b) Serbische KFZ-Kennzeichen unterscheiden sich maßgeblich von deutschen. Der "Landkreis" in dem das Fahrzeug zugelassen wurde wird immer über 2 Buchstaben identifiziert. Darauf folgt eine immer 3-stellige Nummer auf welche dann wieder 2 Buchstaben folgen. Die einzige Ausnahme für diese Art der Kennzeichen bilden die Nummernschilder der Polizei. Diese haben statt des Zulassungskreises den kyrillischen Buchstaben П für Polizei.
Geben Sie einen **deterministischen** endlichen Automaten an, der die beschriebene Menge an serbischen Kennzeichen erkennt.

c) Ein Gebirge sei wie folgt kodiert:

- / stellt einen Aufstieg dar
- \ stellt einen Abstieg dar
- _ repräsentiert ebenes Gelände

Ferner gilt, dass es zu jedem Aufstieg einen Abstieg geben muss (und umgekehrt) und dass ein Gebirge stets mit ebenem Gelände beginnt und endet.
Geben Sie eine Grammatik in BNF-Notation an, welche derartige Gebirge als Wörter zulässt.
Beispiel: Folgendes Gebirge ist ein valides Gebirge: _//_//\\\\/_

Aufgabe 4 - Java - Datenstrukturen **4 + 4 + 4 = 12 Punkte**

Betrachten Sie folgende Klassen:

```java
public class Library {
    //Maps Genretype to Collection of books of that genre
    private Map<String, Collection<Book>> books;
}
```

```java
public class Book {
    public String title;
    protected Stack<Page> pages;
}
```

```java
public class Page {
    private String content;
}
```

a) Implementieren Sie die Methode `findBookByTitle(...)` der Klasse `Library`, welche alle Bücher der Bibliothek durchsucht und das Buch mit dem übergebenen Titel als `Optional` zurückgibt.
Hinweis: Sie können davon ausgehen, dass Büchernamen eindeutig sind.
Hinweis: `Optionals` von einem Objekt, das möglicherweise `null` ist, können mittels `Optional.ofNullable(object)` erzeugt werden.

```java
public Optional<Book> findBookByTitle(String title) {
```

b) Implementieren Sie die Methode `insertBook(...)` der Klasse `Library`, welche ein Buch mit übergebenem Genre in die Menge der Bücher zu diesem Genre hinzufügt. Achten Sie dabei darauf, dass Buchtitel pro Genre eindeutig sein müssen. Sollte ein Buch mit selbem Namen bereits existieren soll dieses mit dem übergebenen Buch ersetzt werden.

```
public void insertBook(Book book, String genre) {
```

c) Die Implementierung von Büchern mittels eines `Stacks` von `Pages` hat einen entscheidenden Nachteil: um auf Folgeseiten zugreifen zu können, müssen vorherige Seiten verworfen werden. Lösen Sie dieses Problem mit Hilfe eines weiteren Stack-Attributs der Klasse `Book`. Geben Sie hierfür die **Signatur** (Sichtbarkeit, Typ, Name) des Attributs an und **implementieren** Sie die Methode `turnPage(...)`, welche im Buch vor- bzw. zurückblättern kann abhängig davon ob `true` oder `false` übergeben wird. Im Falle, dass nicht mehr geblättert werden kann, soll keine Aktion durchgeführt werden.

```
public void turnPage(boolean forward) {
```

Aufgabe 5 - Java - Graphen **6 + 6 = 12 Punkte**

Betrachten Sie folgende Klasse, die einen Knoten eines gerichteten Graphen implementiert:

```
1 public class Node {
2     int value;
3     HashSet<Node> connected = new HashSet<>();
4 }
```

a) Implementieren Sie die Methode `insertValue(...)` der Klasse `Node`, welche einen neuen `Node`, mit dem `value` `toInsert`, in die `connected` Menge des Knotens einfügt, welcher den Wert `at` in seinem `value` stehen hat. Die Methode soll `true` zurückliefern wenn der Wert eingefügt werden konnte und `false` wenn nicht.
Hinweis: Sie können davon ausgehen, dass der Graph zusammenhängend ist und keine Zyklen enthält.
Hinweis: Rekursion erleichtert Ihnen hier die Implementierung.

```
public boolean insertValue(int at, int toInsert) {
```

b) Implementieren Sie die Methode `uniqueValues()` der Klasse `Node` welche überprüft, ob in den vom aktuellen Knoten aus erreichbaren Knoten kein `value` mehrfach vorkommt. Falls kein Wert mehrfach vorkommt, soll true zurückgegeben werden, ansonsten `false`.
Hinweis: Sie können davon ausgehen, dass der Graph keine Zyklen enthält.
Hinweis: Eine Hilfsmethode kann hier nützlich sein!

```
public boolean uniqueValues() {
```

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.