

Abiturprüfung 2019

INFORMATIK

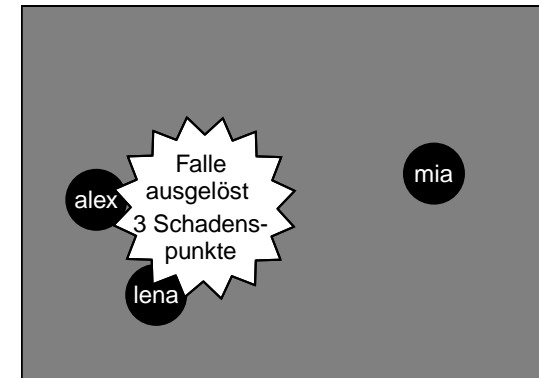
Arbeitszeit: 180 Minuten

Der Fachausschuss wählt je eine Aufgabe aus den Gebieten Inf1 und Inf2 zur Bearbeitung aus.

Der Fachausschuss ergänzt im folgenden Feld die erlaubten objektorientierten Programmiersprachen:

BE

Bei dem Onlinespiel „Achtung Falle“ können beliebig viele Spieler gleichzeitig eingeloggt sein, ihre kreisförmigen Figuren in einer zweidimensionalen, quadratischen Spielwelt bewegen und miteinander interagieren. Die Spielwelt ist 1000 mal 1000 Einheiten groß. Unter anderem geht es in dem Spiel darum, Fallen zu erkennen und zu entschärfen. Folgende Abbildung zeigt einen Ausschnitt aus einer möglichen Spielwelt:



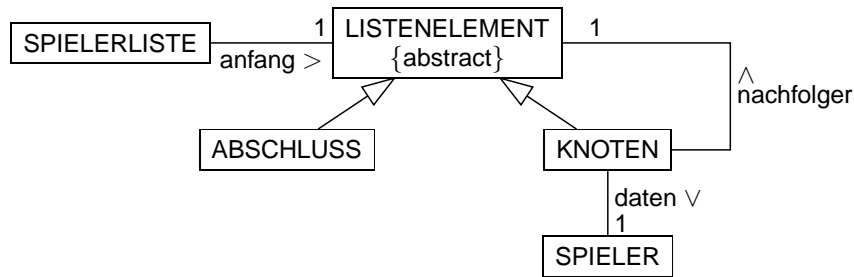
1. Das Spielsystem verwendet zur Verwaltung der momentan in der Spielwelt eingeloggt und somit aktiven Spieler eine einfach verkettete Liste. Wenn sich ein registrierter Nutzer unter Angabe von Benutzernamen und Passwort einloggt, wird ein Objekt der Klasse SPIELER angelegt und in die Liste aller aktiven Spieler (Klasse SPIELERLISTE) eingefügt.

Das SPIELER-Objekt verwaltet den Benutzernamen des Spielers, dessen momentanen Punktestand sowie die aktuelle ganzzahlige x- und y-Position des Spielers in der Spielwelt. Beim Anlegen des SPIELER-Objekts wird dem Konstruktor lediglich der Benutzername übergeben, der Punktestand ist zu Beginn immer 100, die x- und die y-Position werden anfangs jeweils im Bereich von 0 bis 999 zufällig gesetzt.

(Fortsetzung nächste Seite)

- a) Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine mögliche Implementierung der Klasse SPIELER. Beschränken Sie sich dabei auf die Attribute, den Konstruktor und die Methode zum Setzen des Punktestandes. Gehen Sie davon aus, dass eine Methode *zufallszahlErzeugen(min, max)* dieser Klasse bereits vollständig implementiert ist, die eine ganzzahlige Zufallszahl aus dem durch die beiden ganzzahligen Übergabeparameter festgelegten Bereich zurückgibt.

Die Liste der aktiven Spieler ist als einfach verkettete Liste unter Verwendung des Softwaremusters Kompositum gemäß dem abgebildeten Klassendiagramm realisiert.



- b) Zeichnen Sie ein Objektdiagramm für eine Spielerliste, die zwei Spieler enthält. Auf die Angabe von Attributen der SPIELER-Objekte kann verzichtet werden.

(Fortsetzung nächste Seite)

- c) Die Klasse SPIELERLISTE soll folgende Methoden besitzen:

- *punktestandAendern(benutzername, wert)* verändert den Punktestand des SPIELER-Objekts, dessen Benutzername die übergebene Zeichenkette ist, indem der angegebene ganzzahlige Wert addiert wird. Man geht dabei davon aus, dass ein SPIELER-Objekt mit dem angegebenen Benutzernamen höchstens einmal in der Liste enthalten ist.
- *maximum()* gibt den höchsten Punktestand zurück, der in der Liste vorkommt bzw. 0, falls die Liste leer ist. Vereinfachend dürfen Sie davon ausgehen, dass es keine negativen Punktestände gibt.

Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine mögliche Implementierung dieser Methoden und aller dazu nötigen Methoden in der Listenstruktur. Verwenden Sie dabei so weit wie möglich das Prinzip der Rekursion.

Hinweis: Eine Methode *istGleich(zk)* der Klasse ZEICHENKETTE, die genau dann *wahr* zurückgibt, wenn die übergebene Zeichenkette mit der des ausführenden Objekts übereinstimmt, darf ebenso als bereits implementiert vorausgesetzt werden wie sämtliche Standardmethoden der Klasse SPIELER zum Lesen und Setzen von Attributwerten.

Im Spiel „Achtung Falle“ sind Fallen versteckt, die von den Spielern entschärft werden können. In der Spielwelt befinden sich zu jeder Zeit 100 Fallen. Es ist von Falle zu Falle unterschiedlich, wie viel Schaden sie bei ihrer Auslösung verursacht (1 bis 10 Schadenspunkte) und wie schwierig das Entschärfen ist (Komplexitätsstufe 1 bis 10). Sobald eine Falle entschärft oder ausgelöst worden ist, verschwindet sie; gleichzeitig taucht an einer anderen Stelle der Spielwelt eine neue Falle auf, sodass die Anzahl der Fallen stets gleich bleibt. Die Fallen werden vom Spielsystem verwaltet.

- d) Modellieren Sie die Fallen objektorientiert. Beschränken Sie sich dabei auf die oben beschriebenen Aspekte. Beschreiben Sie zudem, mit welcher Datenstruktur Sie die Verwaltung der Fallen durch das Spielsystem implementieren würden, und begründen Sie kurz Ihre Entscheidung.

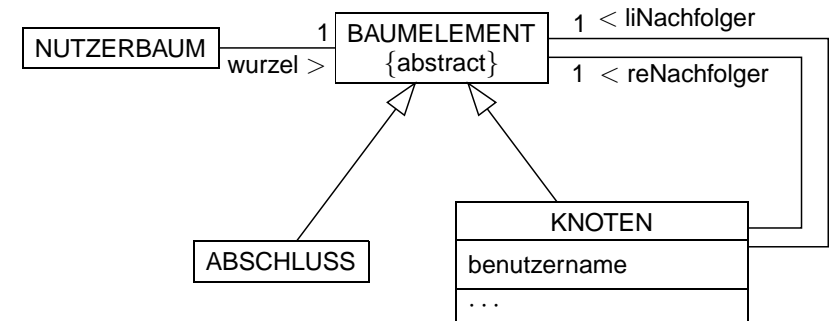
(Fortsetzung nächste Seite)

2. Die persönlichen Daten aller registrierten Nutzer werden in einer Datenbank gespeichert. Zur Neuanmeldung werden alle vergebenen Benutzernamen in einem lexikographisch geordneten Binärbaum in den Arbeitsspeicher geladen. Schon während der Eingabe des gewünschten Benutzernamens kann bei jedem Tastendruck überprüft werden, ob der im Textfeld stehende Name bereits vergeben ist.

- 3 a) Zeichnen Sie den Baum, der entsteht, wenn in einen leeren Baum die folgenden neun Benutzernamen in der angegebenen Reihenfolge eingefügt werden:
lena – benni – tom – olli – mia – alex – sammy – zoe – nora.
- 4 b) Erläutern Sie, warum in einem für Suchanfragen optimal aufgebauten geordneten Binärbaum nicht nur die Anzahl der Ebenen möglichst gering sein sollte, sondern zusätzlich in der untersten Ebene möglichst wenige Knoten sein sollten.
Geben Sie unter diesen Gesichtspunkten an, welche Einträge für die Wurzel eines Baums mit den Benutzernamen aus Teilaufgabe 2a geeignet sind.
- 5 c) Bei „Achtung Falle“ sind ungefähr 150 000 Nutzer registriert. Der Binärbaum zum Überprüfen von Benutzernamen hat somit ungefähr 150 000 Einträge. Gehen Sie davon aus, dass nach jeder Aufnahme eines neuen Nutzers dieser Baum gegebenenfalls umstrukturiert wird, sodass er für die Suche immer optimal aufgebaut ist.
Schätzen Sie ab, wie lange das Überprüfen eines eingegebenen Benutzernamens maximal dauert. Nehmen Sie an, dass im Rahmen der Überprüfung ein ganzer Rekursionsschritt mit Vergleich $5 \cdot 10^{-6}$ s benötigt. Beurteilen Sie zudem, ob es eine spürbare Verzögerung für den neuen Nutzer bedeutet, wenn während der Eingabe des gewünschten Benutzernamens nach jedem Tastendruck eine Überprüfung durchgeführt wird.

(Fortsetzung nächste Seite)

Der Binärbaum zur Verwaltung der Benutzernamen ist unter Verwendung des Softwaremusters Kompositum gemäß dem abgebildeten Klassendiagramm realisiert.



- 8 d) Die Klasse NUTZERBAUM soll die Methode *istFrei(benutzername)* besitzen, welche genau dann *falsch* zurückgibt, wenn das übergebene ZEICHENKETTE-Objekt bereits als Benutzername im Binärbaum vorkommt.
Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine mögliche Implementierung dieser Methode und aller dazu nötigen Methoden in der Baumstruktur. Verwenden Sie dabei so weit wie möglich das Prinzip der Rekursion und nutzen Sie die Vorteile des geordneten Binärbaums.
Hinweis: Als bereits implementiert vorausgesetzt werden darf eine Methode *vergleichenMit(zk)* der Klasse ZEICHENKETTE, die
- eine negative ganze Zahl zurückgibt, wenn das ausführende Objekt lexikographisch kleiner als die übergebene Zeichenkette *zk* ist,
 - eine positive ganze Zahl zurückgibt, wenn das ausführende Objekt lexikographisch größer als die übergebene Zeichenkette *zk* ist,
 - die Zahl 0 zurückgibt, wenn die Zeichenketten übereinstimmen.

(Fortsetzung nächste Seite)

- e) In der Klassenstruktur des Binärbaums werden die Methoden *m1* und *m2* implementiert. LISTE ist eine Klasse zur Verwaltung einfach verketteter Listen. Die Klasse LISTE verfügt über die Methode *hintenEinfuegen(zk)*, mit der eine übergebene Zeichenkette hinten eingefügt werden kann. Beschreiben Sie, was bei dem Methodenaufruf *m1('s')* zurückgegeben wird.

In der Klasse NUTZERBAUM:

```
Methode LISTE m1(ZEICHEN z)
    liste = neues leeres LISTE-Objekt
    wurzel.m2(z, liste)
    gib liste zurück
endeMethode
```

In der Klasse KNOTEN:

```
Methode m2(ZEICHEN z, LISTE liste)
    ZEICHEN z1 = erstes Zeichen von benutzername
    wenn z alphabetisch vor z1 steht dann
        liNachfolger.m2(z, liste)
    sonst
        wenn z alphabetisch nach z1 steht dann
            reNachfolger.m2(z, liste)
        sonst
            liNachfolger.m2(z, liste)
            liste.hintenEinfuegen(benutzername)
            reNachfolger.m2(z, liste)
        endeWenn
    endeWenn
endeMethode
```

In der Klasse ABSCHLUSS:

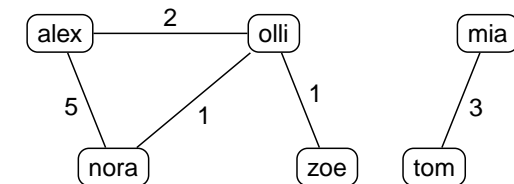
```
Methode m2(ZEICHEN z, LISTE liste)
endeMethode
```

(Fortsetzung nächste Seite)

3. Oft hat eine im Spiel „Achtung Falle“ versteckte Falle eine so hohe Komplexitätsstufe, dass es für einen Spieler allein nicht möglich ist, sie zu entschärfen. Deshalb schließen sich die Spieler zu Gruppen, sogenannten Clans, von fünf bis zehn im Spiel angemeldeten Spielern zusammen. Beliebige Spieler eines Clans können dann auch gemeinsam versuchen, eine Falle zu entschärfen. Für jeden Clan gibt es einen Graphen, der mithilfe einer Adjazenzmatrix implementiert wird.

In einem solchen Graphen wird dargestellt, wie oft ein Spieler in der Vergangenheit gemeinsam mit einem anderen Spieler eine Falle entschärft hat. Haben beispielsweise Spieler X und Spieler Y dreimal gemeinsam eine Falle entschärft, so besteht eine Kante mit dem Gewicht 3 zwischen X und Y. Ob die Entschärfung jeweils zu zweit oder gemeinsam mit noch weiteren Spielern erfolgte, spielt dabei keine Rolle.

- a) Geben Sie eine zu dem abgebildeten Graphen gehörende Adjazenzmatrix an und nennen Sie zwei charakteristische Eigenschaften des Graphen.



- b) Zeichnen Sie den Graphen, der aus dem oben stehenden Graphen entsteht, wenn zusätzlich folgende Fallen entschärft werden:

- eine Falle gemeinsam von zoe, olli und tom,
- eine Falle gemeinsam von olli und tom,
- eine Falle allein von nora sowie
- eine Falle gemeinsam von alex, nora, olli und zoe.

(Fortsetzung nächste Seite)

Die Klasse GRAPH speichert die Knoten eines solchen Graphen in einem Feld *nutzer* und die Adjazenzmatrix in einem zweidimensionalen Feld *matrix*. Im Attribut *anzahl* wird die Anzahl der Knoten, also die Anzahl der Clanmitglieder, gespeichert.

6

c) Die Klasse GRAPH soll die Methode *kantenzahlGeben(index)* erhalten, welche die Anzahl der an den Knoten mit dem Index *index* anschließenden Kanten zurückgibt.

Stellen Sie einen Algorithmus für die Methode *kantenzahlGeben* graphisch dar.

9

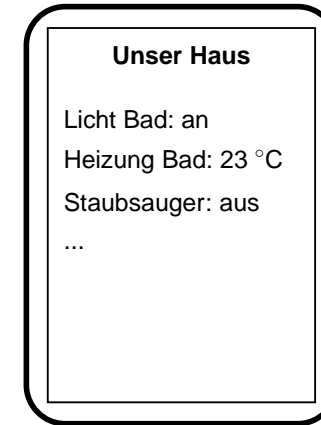
d) Es soll herausgefunden werden, welcher Spieler des Clans die meisten Kooperationspartner hat. Dazu soll die Klasse GRAPH um eine Methode *besterKooperatorGeben()* erweitert werden, die den Index des Knotens zurückgibt, an den sich die meisten Kanten anschließen. Falls mehrere Knoten die gleiche Anzahl anschließender Kanten besitzen, wird der mit dem kleinsten Index zurückgegeben.

Formulieren Sie unter Verwendung der Methode aus Teilaufgabe 3c einen Algorithmus für die Methode *besterKooperatorGeben*.

Geben Sie denjenigen Spieler aus dem Graphen von Teilaufgabe 3a an, den diese Methode als besten Kooperator ermittelt.

Nennen Sie einen Grund, warum ein anderer Spieler sich selbst als kooperativer ansehen könnte.

Ein Softwareentwicklungsteam wird beauftragt, eine Steuerungs-App für Smart-Home-Systeme zu entwickeln.



4

1. Nennen Sie mindestens vier typische Phasen bei der Durchführung eines Softwareprojekts und ordnen Sie folgende Vorgänge den entsprechenden Phasen zu:

- Erstellen des Pflichtenhefts
- Einsatz von Softwaremustern
- Aufwandsabschätzung
- Modellieren mithilfe von Diagrammen

(Fortsetzung nächste Seite)

In einer Vorstudie werden die zu steuernden Geräte durch Objekte simuliert.

2. Für jedes simulierte Gerät soll der Ist-Zustand – wie in der Abbildung auf Seite 10 beispielhaft dargestellt – mittels einer Methode *informationAusgeben()* angezeigt werden können.

Alle Geräte besitzen einen eindeutigen Namen und können ein- oder ausgeschaltet sein.

Bei einigen Geräten werden außerdem ein ganzzahliger Messwert und die zugehörige Einheit gespeichert, z. B. bei der Heizung aktuell 23 °C für die Temperatur. Falls ein solches Gerät eingeschaltet ist, soll der aktuell gemessene Wert mit Einheit, sonst „aus“ angezeigt werden.

Bei der Erzeugung eines Objekts für ein Gerät wird der Name übergeben, für Geräte mit Messwert zusätzlich ein Minimal- und ein Maximalwert sowie die Einheit.

Eine Methode *istImErlaubtenBereich()* soll bei Geräten ohne Messwert immer *wahr* zurückgeben, bei Geräten mit Messwert genau dann *wahr*, wenn dieser im Bereich von Minimalwert bis Maximalwert liegt, unabhängig davon, ob das Gerät ein- oder ausgeschaltet ist.

5

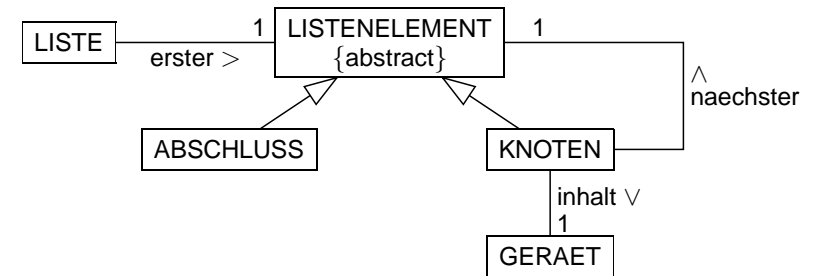
- a) Zeichnen Sie ein Klassendiagramm mit einer Klasse GERAET und mindestens einer weiteren Klasse für die oben beschriebene Situation.

13

- b) Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine mögliche Implementierung der Klassen aus Teilaufgabe 2a. Methoden zum Setzen oder Geben von Attributwerten müssen nicht implementiert werden. Die Ausgabe darf textuell ohne einheitliche Formatierung erfolgen.

(Fortsetzung nächste Seite)

Das Team entscheidet sich dafür, die Geräte in einer einfach verketteten Liste gemäß folgendem Klassendiagramm zu verwalten:



Die Klasse LISTE soll eine Methode *handlungsbedarfAnzeigen()* bereitstellen, die nur die Informationen derjenigen Geräte anzeigt, bei denen der aktuelle Messwert außerhalb des erlaubten Bereichs liegt, unabhängig davon, ob das Gerät ein- oder ausgeschaltet ist.

6

- c) Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine mögliche Implementierung der Methode *handlungsbedarfAnzeigen()* und aller dazu nötigen Methoden in der Listenstruktur. Verwenden Sie soweit wie möglich das Prinzip der Rekursion. Die Methoden aus Teilaufgabe 2b dürfen vorausgesetzt werden.

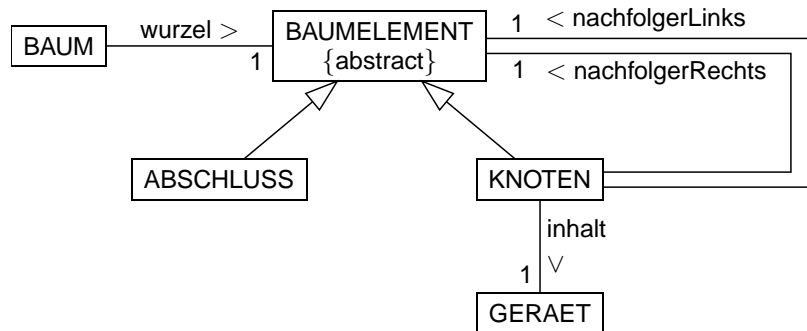
7

- d) Damit die Nutzer die Reihenfolge der angezeigten Geräte verändern können, soll die Klasse LISTE um eine Methode *vorruecken(geraet)* erweitert werden, die ein in der Liste gespeichertes Gerät, das nicht bereits an erster Stelle der Liste steht, mit seinem Vorgänger vertauscht.

Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache die dazu benötigte Methode *vorruecken* der Klasse KNOTEN, der als Parameter das entsprechende Gerät übergeben wird. Verwenden Sie das Prinzip der Rekursion. Methoden zum Setzen und Geben von Attributwerten dürfen als gegeben betrachtet werden. Die Methoden in den Klassen LISTE, LISTENELEMENT und ABSCHLUSS sind nicht gefordert.

(Fortsetzung nächste Seite)

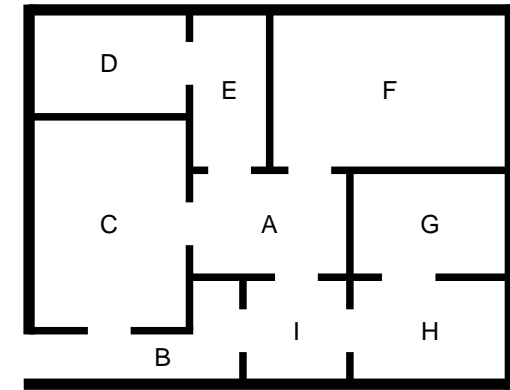
3. Um auch eine größere Anzahl von Smart-Home-Geräten verwalten zu können, zieht das Team – alternativ zur Liste – die Verwaltung der Geräte in einem lexikographisch geordneten Binärbaum gemäß abgebildetem Klassendiagramm in Betracht. Als Schlüssel für die Datenelemente soll eine Zeichenkette dienen, die die Art des Gerätes, seinen Standort und gegebenenfalls eine Nummer enthält.



- 6 a) Zeichnen Sie – basierend auf obigem Klassendiagramm – ein Objektdiagramm eines Binärbaums zu insgesamt zwei verwalteten Geräten. Bei den Geräten soll nur das Schlüsselattribut aufgeführt werden.
- 4 b) In einem Studentenwohnheim werden 534 Geräte verwaltet. Beschreiben Sie die beiden Extremfälle hinsichtlich der Ebenenanzahl, die sich beim Aufbau des Baumes ergeben können, und bestimmen Sie die Anzahl der Ebenen für diese Fälle.
- 7 c) Für folgende Geräte ist der Schlüssel jeweils in Klammern angeführt. Sie sollen in einen zunächst leeren Baum eingefügt werden:
 Heizung (HWZ), Fernseher (FWZ) und Licht (LWZ) im Wohnzimmer; Licht im Bad (LB); Licht (LK), Kühlschrank (KK), Herd (HK), Spülmaschine (SK) und Dunstabzug (DK) in der Küche.
 Nennen Sie eine Reihenfolge zum Einfügen dieser Geräte, die zu einem für Suchanfragen besonders geeigneten Baum führt, und zeichnen Sie den Baum. Als Bezeichner für die Knoten sollen die Schlüssel der Datenelemente verwendet werden.
 Geben Sie außerdem an, in welcher Reihenfolge die Knoten beim Preorder- und Postorder-Durchlauf jeweils besucht werden.

(Fortsetzung nächste Seite)

4. Für einen smarten Saugroboter im Haus wird ein Graph für die zu saugenden Räume erstellt. Die Räume stellen die Knoten, die Türen die Kanten dar. Im dargestellten Grundriss befindet sich die Eingangstür links unten. Für sie muss keine Kante in den Graphen aufgenommen werden. Der Roboter startet seinen Arbeitszyklus bei der Ladestation im Flur A und saugt nacheinander alle Räume.



- 5 a) Zeichnen Sie den zugehörigen Graphen und geben Sie seine Adjazenzmatrix an, bei der die Knoten in alphabetischer Reihenfolge aufgeführt werden.
- 9 b) Formulieren Sie unter Verwendung der Adjazenzmatrix und mit Angabe der benötigten Attribute einen möglichen Algorithmus für den Durchlauf des Graphen. Geben Sie die damit erhaltene Bearbeitungsreihenfolge der Räume durch den Roboter an. Nennen Sie die Eigenschaft des Graphen, die es hier ermöglicht, dass alle Räume gesaugt werden.
- 7 c) In obigem Grundriss stellen die Räume D, F und G Sackgassen dar, da sie jeweils nur von einem einzigen anderen Raum aus zugänglich sind. Formulieren Sie unter Verwendung der Adjazenzmatrix einen Algorithmus, der alle Knoten eines beliebigen Graphen ausgibt, die genau einen Nachbarknoten haben.

(Fortsetzung nächste Seite)

4

Zur Anzeige und Einstellung der gewünschten Temperatur im Wohnzimmer soll sich für den Nutzer folgende Ansicht öffnen:


HEIZUNG WOHNZIMMER


TEMPERATUR 18 °C


3

b) Erläutern Sie allgemein die Funktionsweise des MVC am Beispiel des Anklickens des „+“-Buttons im Fenster. Die konkreten Methodenaufrufe sind nicht verlangt.

4

R1: <Takt> → 





R2: <3/4> → 

R3: <1/2> → 

3

3

Mit T_4 wird die formale Sprache bezeichnet, die genau alle Viervierteltakte enthält, die mit den Zeichen aus Σ gebildet werden können. Verwenden Sie für die weiteren Teilaufgaben anstelle der Notensymbole Buchstaben gemäß folgender Tabelle:

Notensymbol				
Buchstabe	g	p	h	v

c) Zeichnen Sie das Zustandsübergangsdiagramm eines erkennenden endlichen Automaten, der genau T_4 akzeptiert.

(Fortsetzung nächste Seite)

BE
7

d) Formulieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine Implementierung für den in Teilaufgabe 1c erstellten Automaten. Dabei soll es u. a. eine Methode *istViervierteltakt(eingabe)* geben, die überprüft, ob die übergebene Zeichenkette *eingabe* den Vorgaben für einen Viervierteltakt entspricht, und einen entsprechenden Wahrheitswert zurückgibt. Dazu ruft sie für jedes Zeichen der Eingabe jeweils die Methode *zustand-Wechseln* auf.

Hinsichtlich der Übergänge genügt es, wenn Sie beispielhaft die vom Startzustand ausgehenden Übergänge implementieren.

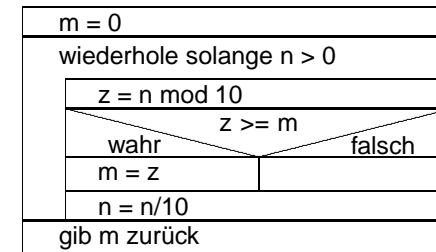
Hinweis: Sie dürfen dabei folgende Methoden der Klasse ZEICHENKETTE verwenden:

- *laenge()* gibt die Länge der Zeichenkette zurück,
- *zeichenAn(n)* gibt das *n*-te Zeichen der Zeichenkette zurück; die Zählung beginnt bei 0.

(Fortsetzung nächste Seite)

BE
2.
4
6

2. Auf eine gegebene natürliche Zahl *n* wird folgender Algorithmus angewendet:



Hinweis: Der Operator „/“ liefert das Ergebnis der ganzzahligen Division ohne Rest, der Operator „mod“ den Rest der ganzzahligen Division. Beispielsweise gilt: 317/10 = 31 und 317 mod 10 = 7.

a) Stellen Sie für *n* = 243 tabellarisch dar, welche Werte die Variablen *n*, *z* und *m* im Laufe der Abarbeitung des Algorithmus annehmen, und beschreiben Sie allgemein die Bedeutung des Rückgabewerts.

b) Schreiben Sie ein Programm für die Registermaschine mit nebenstehendem Befehlssatz, das den gegebenen Algorithmus umsetzt. Verwenden Sie für die Variablen *n*, *z* und *m* die Speicherzellen 100, 101 und 102. Gehen Sie insbesondere davon aus, dass der Wert für *n* bereits in Speicherzelle 100 gespeichert ist.

(Fortsetzung nächste Seite)

BE

load x kopiert den Wert aus der Speicherzelle x in den Akkumulator
 loadi n lädt die ganze Zahl n in den Akkumulator
 store x kopiert den Wert aus dem Akkumulator in die Speicherzelle x
 add x addiert den Wert aus der Speicherzelle x zum Wert im Akkumulator
 addi n addiert die ganze Zahl n zum Wert im Akkumulator
 sub x subtrahiert den Wert aus der Speicherzelle x vom Wert im Akkumulator
 subi n subtrahiert die ganze Zahl n vom Wert im Akkumulator
 mul x multipliziert den Wert im Akkumulator mit dem Wert aus der Speicherzelle x
 muli n multipliziert den Wert im Akkumulator mit der ganzen Zahl n
 div x dividiert den Wert im Akkumulator durch den Wert aus der Speicherzelle x (ganzzahlige Division)
 divi n dividiert den Wert im Akkumulator durch die ganze Zahl n (ganzzahlige Division)
 mod x speichert den Rest bei der Ganzzahldivision des Akkumulatorwerts durch den Wert aus der Speicherzelle x in den Akkumulator
 modi n speichert den Rest bei der Ganzzahldivision des Akkumulatorwerts durch die ganze Zahl n in den Akkumulator
 jmp x springt zum Befehl in Speicherzelle x
 jeq x springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator Null ist
 jgt x springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator positiv ist
 jlt x springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator negativ ist
 jge x springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator positiv oder Null ist
 jle x springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator negativ oder Null ist
 jne x springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator nicht Null ist
 end beendet die Abarbeitung des Programms

(Fortsetzung nächste Seite)

BE
2

3. Ein Theater setzt eine Platzbuchungssoftware ein, die auf einem Server läuft. Für jeden Nutzer, der sich online am Buchungssystem anmeldet, wird auf dem Server ein Objekt der Klasse CLIENTPROZESS erzeugt, das als eigenständiger Thread (Prozess) agiert und die Kommunikation mit dem Nutzer abwickelt. Nachdem der Nutzer eine Veranstaltung ausgewählt hat, greift das Objekt der Klasse CLIENTPROZESS auf ein Feld zu, in dem Objekte der Klasse SITZPLATZ für alle Sitzplätze der gewählten Veranstaltung referenziert sind. Die Klasse SITZPLATZ besitzt ein Attribut *gebucht*, das angibt, ob der betreffende Sitzplatz bereits gebucht ist oder nicht, sowie eine Methode *buchen*, die *falsch* zurückgibt, falls der Sitzplatz bereits gebucht ist, ansonsten den Platz als gebucht markiert und *wahr* zurückgibt.
- a) Beschreiben Sie ein Konzept, das Sie bei einer Implementierung der Methode *buchen* nutzen können, um Mehrfachbuchungen eines Sitzplatzes zu verhindern.

(Fortsetzung nächste Seite)

- b) Um Theaterbesucherinnen und -besuchern auch die Buchung von zwei benachbarten Plätzen zu ermöglichen, wird die Klasse SITZPLATZ noch um eine Methode *buchenMit* erweitert, die als Parameter *nachbar* eine Referenz auf einen benachbarten Sitzplatz bekommt und prüft, ob eine gemeinsame Buchung beider Plätze möglich ist. Sie arbeitet nach folgendem Algorithmus:

```

methode buchenMit(nachbar)
  wenn gebucht ist gleich wahr // Platz schon gebucht?
    gib falsch zurück // Buchung nicht erfolgreich
  sonst
    gebucht = wahr // Platz buchen
    erfolg = nachbar.buchen() // Nachbarplatz buchen
    wenn erfolg ist gleich wahr // Nachbarplatz erfolgreich gebucht?
      gib wahr zurück // gemeinsame Buchung erfolgreich
    sonst
      gebucht = falsch // Platz wieder freigeben
      gib falsch zurück // gemeinsame Buchung nicht erfolgreich
  endeWenn
endeWenn
endeMethode

```

Hinweis: „=“ bedeutet Wertzuweisung.

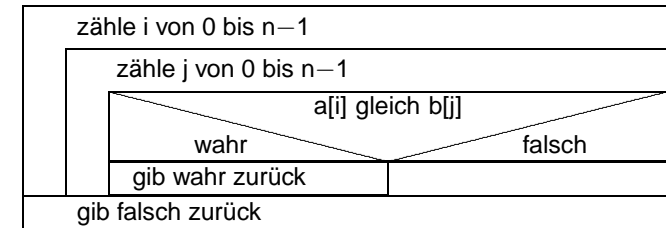
Entscheiden Sie begründet, ob es unter gewissen Bedingungen bei der Ausführung dieser Methode zu einer Verklemmung kommen kann.

(Fortsetzung nächste Seite)

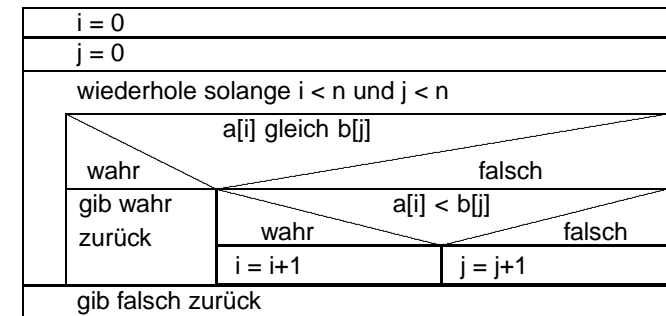
4. Gegeben sind zwei Felder *a* und *b* jeweils der Länge $n \geq 1$, deren Feldelemente Ganzzahlen enthalten. Die Felder sind jeweils aufsteigend nach der Größe der Ganzzahlen sortiert.

Es soll festgestellt werden, ob es eine Ganzzahl gibt, die sowohl in Feld *a* als auch in Feld *b* enthalten ist. Zur Lösung dieses Problems werden zwei unterschiedliche Algorithmen vorgeschlagen.

Algorithmus I:



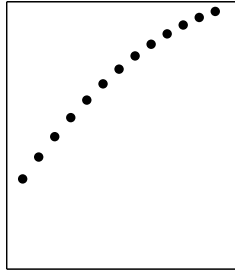
Algorithmus II:



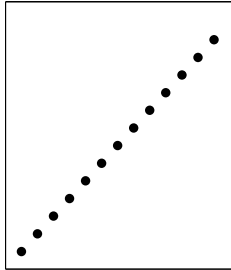
Hinweis: „gib ... zurück“ beendet die Abarbeitung des Algorithmus.

(Fortsetzung nächste Seite)

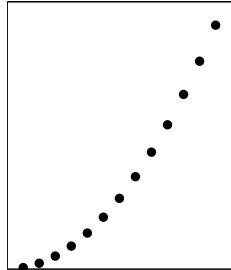
Geben Sie für jeden der beiden Algorithmen zunächst für $n = 4$, dann allgemein in Abhängigkeit von n an, wie oft der Vergleich „ $a[i]$ gleich $b[j]$ “ im ungünstigsten Fall ausgeführt werden muss, und begründen Sie damit, welches der drei folgenden Diagramme das ungünstigste Laufzeitverhalten des jeweiligen Algorithmus in Abhängigkeit von n darstellt:



A



B



C

Hinweis: Die Diagramme sind linear skaliert mit n als Rechtswert und der ungünstigsten Laufzeit als Hochwert.

1. Zur Speicherung von Tabellen in einer Textdatei ist ein spezielles Dateiformat nützlich, das durch folgende Regeln beschrieben wird:

- Jede Tabellenzeile entspricht genau einer Zeile in der Textdatei.
- Die Inhalte der einzelnen Zellen einer Tabellenzeile werden in der Textdatei durch Semikolons (;) getrennt.
- Jeder Zelleninhalt kann in Anführungszeichen (") eingeschlossen sein. Er muss in Anführungszeichen eingeschlossen sein, falls der Zelleninhalt ein Semikolon enthält.
- Kein Zelleninhalt darf ein Anführungszeichen enthalten.

Beispiel:

Textdatei

Name;Geschwister

Wladimir;"Vitali"

Florian;

Johanna;"Rosli; Eva; Hannelore"

Tabelle

Name	Geschwister
Wladimir	Vitali
Florian	
Johanna	Rosli; Eva; Hannelore

Die Menge aller Zeichenketten, die als Zeile in einer solchen Textdatei stehen können, wird als formale Sprache L bezeichnet. Der zugrunde liegende Zeichenvorrat Σ enthält alle druckbaren Zeichen, insbesondere das Semikolon und das Anführungszeichen.

5

a) Stellen Sie die Regeln einer Grammatik zur Sprache L mithilfe von Syntaxdiagrammen dar.

Zur Vereinfachung dürfen Sie davon ausgehen, dass bereits eine Produktionsregel existiert, mit der vom Nichtterminal $\langle \text{Zeichen} \rangle$ ausgehend sämtliche Zeichen aus Σ außer dem Semikolon und dem Anführungszeichen abgeleitet werden können.

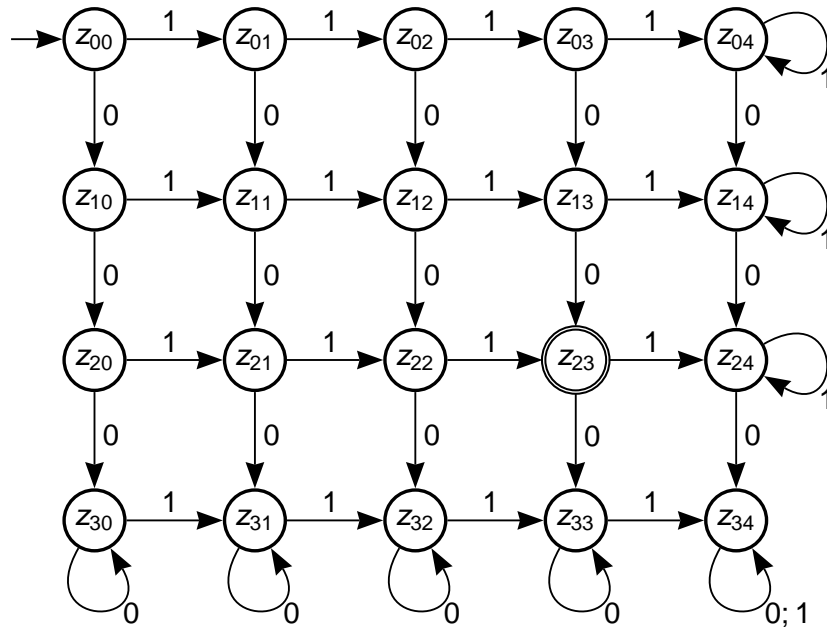
6

b) Zeichnen Sie das Zustandsübergangsdiagramm eines erkennenden endlichen Automaten, der genau die Wörter der Sprache L akzeptiert.

Hinweis: Verwenden Sie vereinfachend das Symbol Z für alle Zeichen aus Σ außer dem Semikolon und dem Anführungszeichen.

(Fortsetzung nächste Seite)

2. Gegeben ist das Zustandsübergangsdiagramm eines erkennenden endlichen Automaten:



Die von dem Automaten akzeptierte Sprache L enthält nur endlich viele Wörter.

4

- a) Bestimmen Sie die Anzahl der Elemente von L und charakterisieren Sie die vom Automaten akzeptierten Wörter.

2

- b) Beschreiben Sie allgemein die charakteristische Eigenschaft eines endlichen Automaten, der nur endlich viele Wörter akzeptiert.

(Fortsetzung nächste Seite)

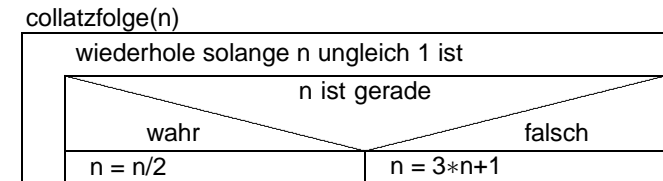
5

3. Für einen Onlinewettbewerb wird die Anmeldeseite aus organisatorischen Gründen 24 Stunden vor Beginn des Wettbewerbs freigeschaltet. Die Zugangscodes werden den Teilnehmerinnen und Teilnehmern jedoch erst zu Beginn des Wettbewerbs mitgeteilt. Es ist aber bekannt, dass diese aus einer zehnstelligen Kombination von Kleinbuchstaben des Alphabets bestehen.

Im Leitungsteam des Wettbewerbs wird das Szenario diskutiert, dass sich eine Teilnehmerin oder ein Teilnehmer durch einen früheren Zugang zu ihrem bzw. seinem Benutzerkonto einen unfairen Vorteil verschaffen will. Man geht bei einem Brute-Force-Angriff davon aus, dass der Angreifer pro Sekunde vier Milliarden Zugangscodes ausprobieren kann.

Eine Mitarbeiterin weist nach, dass die Zugangscodes hinsichtlich eines solchen Angriffs nicht sicher sind. Sie gibt zudem einen Änderungsvorschlag an, der die Zugangscodes vor dem Angriff besser schützen würde. Geben Sie eine mögliche Argumentationskette an und belegen Sie diese durch entsprechende Berechnungen.

4. Das Collatz-Problem ist ein immer noch ungelöstes Problem der Mathematik. Dabei geht es um Zahlenfolgen, die nach folgendem Algorithmus gebildet werden, wobei der Eingabewert n eine natürliche Zahl größer 0 ist:



Obwohl der Algorithmus sehr einfach ist, ist bis heute ungeklärt, ob er tatsächlich bei jedem beliebigen Startwert von n nach endlich vielen Durchläufen der Wiederholung terminiert.

2

- a) Geben Sie die Zahlenfolge an, die man mit dem Startwert 7 erhält, wenn n nach jedem Durchlauf der Wiederholung ausgegeben wird.

2

- b) Beschreiben Sie, wie man mithilfe der ganzzahligen Division ohne Rest prüfen kann, ob eine Zahl a durch eine andere Zahl b teilbar ist.

(Fortsetzung nächste Seite)

BE
7

c) Gegeben ist nun eine Registermaschine mit dem folgenden Befehlssatz:

load x	kopiert den Wert aus der Speicherzelle x in den Akkumulator
loadi n	lädt die ganze Zahl n in den Akkumulator
store x	kopiert den Wert aus dem Akkumulator in die Speicherzelle x
add x	addiert den Wert aus der Speicherzelle x zum Wert im Akkumulator
sub x	subtrahiert den Wert aus der Speicherzelle x vom Wert im Akkumulator
mul x	multipliziert den Wert im Akkumulator mit dem Wert aus der Speicherzelle x
div x	dividiert den Wert im Akkumulator durch den Wert aus der Speicherzelle x (ganzzahlige Division)
addi n	addiert die ganze Zahl n zum Wert im Akkumulator
subi n	subtrahiert die ganze Zahl n vom Wert im Akkumulator
muli n	multipliziert den Wert im Akkumulator mit der ganzen Zahl n
divi n	dividiert den Wert im Akkumulator durch die ganze Zahl n (ganzzahlige Division)
jmp x	springt zum Befehl in Speicherzelle x
jeq x	springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator Null ist
jgt x	springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator positiv ist
jlt x	springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator negativ ist
hold	beendet die Abarbeitung des Programms

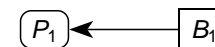
Geben Sie ein Programm für die Registermaschine an, das den gegebenen Algorithmus *collatzfolge(n)* umsetzt, wobei zusätzlich die Anzahl der Durchläufe der Wiederholung bestimmt werden soll.

Der Startwert für n steht am Anfang bereits in Speicherzelle 100.

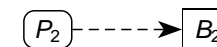
(Fortsetzung nächste Seite)

BE
40

5. Um Anforderungen und Zuteilungen von Betriebsmitteln graphisch darzustellen, kann jeder Prozess durch einen runden Knoten und jedes Betriebsmittel durch einen rechteckigen Knoten repräsentiert werden; die Beziehungen zwischen Prozessen und Betriebsmitteln können durch Pfeile wie folgt gekennzeichnet werden:



Betriebsmittel B_1 wird von Prozess P_1 belegt.



Prozess P_2 fordert Betriebsmittel B_2 an.

Eine derartige Darstellung nennt man Betriebsmittelzuteilungsgraph.

In einem Kindergarten spielen die drei Kinder Ada, Bill und Charles (A, B und C) zusammen im Sandkasten, in dem sich zwei Eimer (E_1 und E_2) und zwei Schaufeln (S_1 und S_2) befinden. Jedes Kind möchte eine eigene Sandburg bauen und benötigt dazu einen Eimer und eine Schaufel.

- 4 a) Beschreiben Sie im Kontext der im Sandkasten spielenden Kinder eine Situation, in der es zu einer Verklemmung kommt. Stellen Sie diese zudem durch einen geeigneten Betriebsmittelzuteilungsgraphen dar.
- 2 b) Beschreiben Sie allgemein, wie man in einem Betriebsmittelzuteilungsgraphen eine Verklemmung erkennen kann.
- 1 c) Geben Sie eine Möglichkeit an, wie die in Teilaufgabe 5a beschriebene Verklemmung aufgelöst werden kann.