Abiturprüfung 2021

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:

INF1. MODELLIERUNG UND PROGRAMMIERUNG

I.

BE

Im US-Bundesstaat Nevada findet jedes Jahr ein neuntägiges Kunstfestival mit ca. 70 000 Besuchern statt, dessen Höhepunkt das Verbrennen einer überdimensionalen Statue – des Burning Man – am sechsten Festivaltag ist.

Es wird angenommen, dass für das Festival eine zentrale Verwaltungssoftware eingesetzt wird. Mit dieser Software kommuniziert eine App, über die sich die Besucher des Festivals mit ihrem Namen, ihrem Geburtsdatum und ihrer E-Mail-Adresse registrieren müssen.

Die Besucher übernachten alle in Camps auf dem Festivalgelände. Nach dem Registrieren muss jeder Besucher über die App eines dieser Camps zum Übernachten auswählen. Mithilfe der App können auch die Veranstaltungen des Festivals angezeigt und gebucht werden. Für jede Veranstaltung ist eine maximale Besucherzahl festgelegt. Außerdem werden auf dem Festival viele Kunstwerke ausgestellt, die einen eindeutigen Namen haben und deren jeweiliger Künstler bekannt ist. Die Besucher können Kunstwerke in der App mit „Gefällt mir“ markieren.

In der eigens für das Festival konzipierten „Stadt“ Black Rock City gibt es mehrere festgelegte Orte.

1. Jedem Camp, jeder Veranstaltung und jedem Kunstwerk ist genau ein Ort zugeordnet. An jedem Ort befindet sich höchstens ein Camp. Dagegen kann es an einem Ort mehrere Veranstaltungen und Kunstwerke geben.

7 a) Entwickeln Sie für die beschriebene Situation ein Klassendiagramm, das ausschnittsweise als Planungsgrundlage für die Verwaltungssoftware des Festivals dienen kann. Verwenden Sie die Klassen BESUCHER, CAMP, VERANSTALTUNG, KUNSTWERK und ORT. Auf die Angabe von Methoden kann verzichtet werden.

Zur Verwaltung der Plätze eines Camps hat die Klasse CAMP als Attribut ein Feld *besucher*, in dem Referenzen auf BESUCHER-Objekte gespeichert werden können. Die Länge des Felds entspricht dabei genau der Anzahl der Plätze des Camps und der Feldindex der jeweiligen Platznummer.

2 b) Nennen Sie zwei Kriterien, die in diesem Kontext zur Entscheidung für die Datenstruktur Feld geführt haben könnten.

(Fortsetzung nächste Seite)

BE

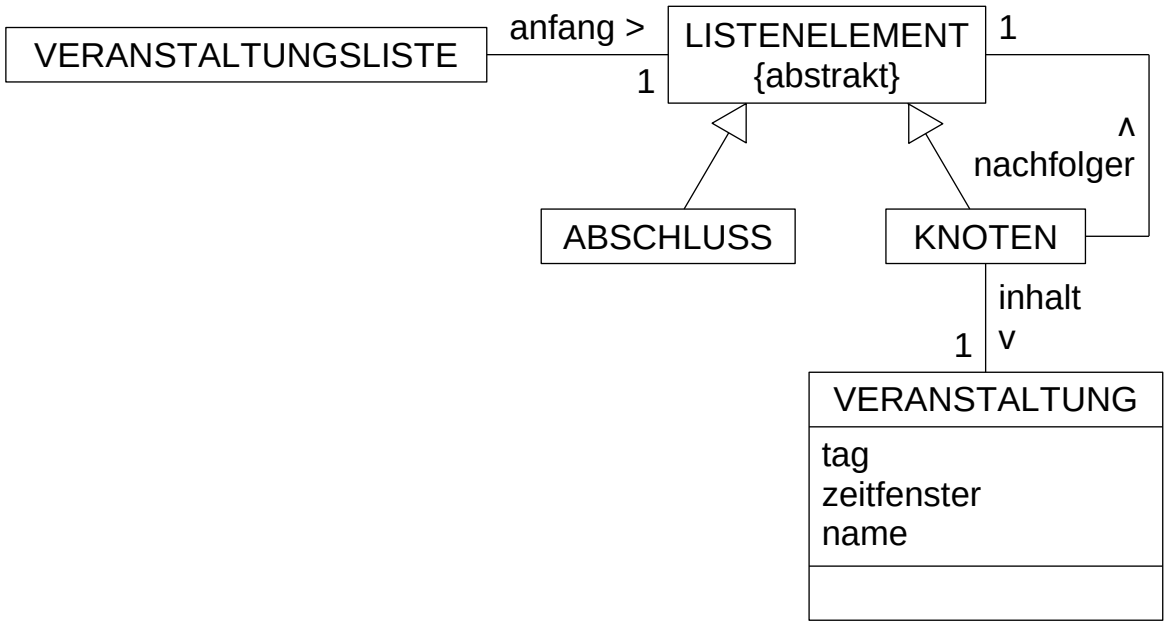
7

c) Die Klasse CAMP soll eine Methode *nachbarplaetzeVerfuegbar(n)* besitzen, die überprüft, ob noch mindestens *n* benachbarte Plätze frei sind, und den entsprechenden Wahrheitswert zurückgibt.

Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine Implementierung der Methode *nachbarplaetzeVerfuegbar*. Dabei kann vereinfachend davon ausgegangen werden, dass benachbarte Feldelemente im Feld *besucher* Plätzen entsprechen, die auch in der Realität benachbart sind.

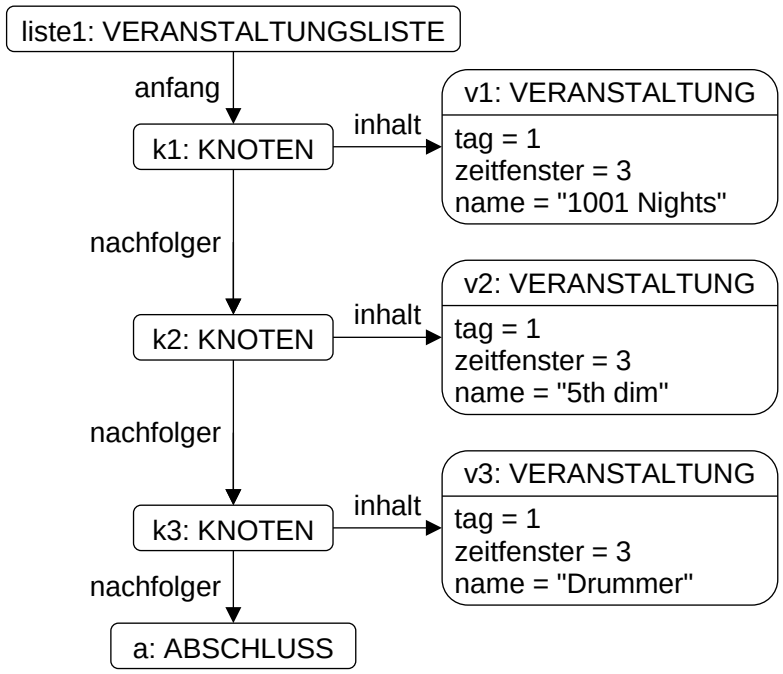
2. Um die Veranstaltungen des Festivals zeitlich gut aufeinander abstimmen zu können, sind pro Tag vier feste, sich nicht überlappende Zeitfenster vorgegeben, in denen Veranstaltungen stattfinden. Die Tage des Festivals sind von 1 bis 9 durchnummeriert, die Zeitfenster von 1 bis 4.

Besucher des Festivals können Veranstaltungen online buchen. Die von einem Besucher gebuchten Veranstaltungen werden in einer sortierten Liste gespeichert, wobei die Sortierung zunächst aufsteigend nach Tag und dann aufsteigend nach Zeitfenster erfolgt. Es ist nicht ausgeschlossen, dass mehrere Veranstaltungen gebucht werden, die gleichzeitig stattfinden. Die Liste basiert auf folgendem Klassendiagramm:



Die Klasse VERANSTALTUNG kann einschließlich der Standardmethoden zum Lesen und Setzen von Attributwerten als implementiert vorausgesetzt werden.

(Fortsetzung nächste Seite)

BE	
18	a) Die Klasse VERANSTALTUNGSLISTE soll folgende Methoden besitzen: • *sortiertEinfuegen(veranstaltungNeu)*: fügt eine Veranstaltung in die Liste gemäß obiger Sortierung ein, • *anzahlGeben(tag, zeitfenster)*: gibt die Anzahl der Veranstaltungen zurück, die der Besucher an diesem Tag im angegebenen Zeitfenster gebucht hat, um ihn so gegebenenfalls darüber zu informieren, dass er Veranstaltungen besuchen will, die gleichzeitig stattfinden. Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine Implementierung der beiden angegebenen sowie der dafür nötigen Methoden aller Klassen der Listenstruktur. Wenden Sie dabei jeweils das Prinzip der Rekursion an.
9	b) Folgendes Objektdiagramm zeigt die Veranstaltungsliste eines Besuchers, der mehrere Veranstaltungen gebucht hat, die gleichzeitig stattfinden. 

(Fortsetzung nächste Seite)

BE

Gegeben sind folgende Methoden:

In der Klasse VERANSTALTUNGSLISTE:

```
Methode m()
    anfang = anfang.m(0, 0)
endeMethode
```

In der abstrakten Klasse LISTENELEMENT:

```
abstrakte Methode m(tag, zeitfenster)
```

In der Klasse KNOTEN:

```
Methode m(tag, zeitfenster)
    meinTag = inhalt.tagGeben()
    meinZeitfenster = inhalt.zeitfensterGeben()
    nachfolger = nachfolger.m(meinTag, meinZeitfenster)
    wenn tag ist gleich meinTag und
            zeitfenster ist gleich meinZeitfenster dann
        gib nachfolger zurück
    sonst
        gib Referenz auf ausführendes Objekt zurück
    endeWenn
endeMethode
```

In der Klasse ABSCHLUSS:

```
Methode m(tag, zeitfenster)
    gib Referenz auf ausführendes Objekt zurück
endeMethode
```

Stellen Sie in einem Sequenzdiagramm den Ablauf dar, der durch einen Aufruf der Methode *m* für das Objekt *liste1* ausgelöst wird. Beschränken Sie sich auf die Objekte der Klassen VERANSTALTUNGSLISTE, KNOTEN und ABSCHLUSS.

Beschreiben Sie zudem, welche Funktionalität die Methode *m* allgemein hat.

(Fortsetzung nächste Seite)

BE

3. Zur Speicherung aller auf dem Festival ausgestellten Kunstwerke wird ein geordneter Binärbaum verwendet, der lexikographisch nach dem eindeutigen Namen des Kunstwerks geordnet ist. Bei der Implementierung sollen das Konzept der Trennung von Struktur und Daten sowie das Entwurfsmuster Kompositum berücksichtigt werden.

3 a) Der geordnete Binärbaum wird unter anderem zur Suche nach Kunstwerken mit einem bestimmten Namen bzw. nach Kunstwerken eines bestimmten Künstlers verwendet.

Entscheiden Sie jeweils begründet, ob sich dabei ein Vorteil gegenüber einer ebenso nach dem Namen des Kunstwerks geordneten Liste ergeben kann.

8 b) Der Ort eines Kunstwerks muss z. B. wegen schlechten Wetters kurzfristig innerhalb des Festivalgeländes geändert werden können.

Die Klasse KUNSTWERKBAUM soll zu diesem Zweck eine Methode *ortAendern(name, ortNeu)* besitzen, die dem Kunstwerk mit dem Namen *name* den Ort *ortNeu* zuweist.

Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine Implementierung dieser sowie aller dafür in den Klassen der Baumstruktur notwendigen Methoden. Wenden Sie das Prinzip der Rekursion an.

Hinweise: Die Klasse KUNSTWERK darf einschließlich ihrer Standardmethoden zum Lesen und Setzen von Attributwerten als implementiert vorausgesetzt werden, ebenso 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)

BE

Zur Datensicherung wird der geordnete Binärbaum mittels Preorder-Traversierung durchlaufen. Die ausgelesenen Daten werden dabei der Reihe nach in eine Datei gespeichert.

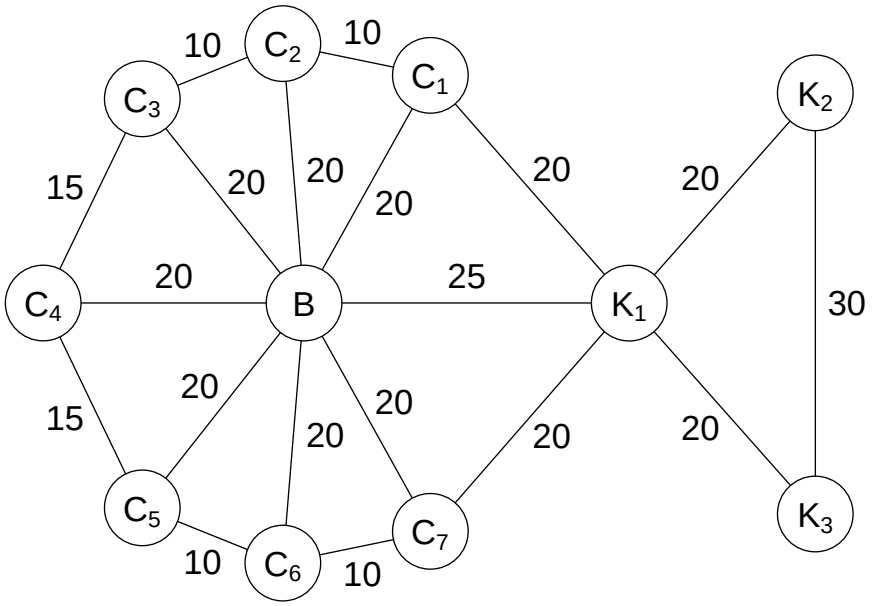
3 c) Folgende Kunstwerke wurden nacheinander ausgelesen:

„DesertWave“, „Carousel“, „Bee“, „Chakra“, „GiantPinball“, „Fascination“, „Paraluna“

Zeichnen Sie den zugrunde liegenden Baum.

2 d) Begründen Sie kurz, weshalb sich die Preorder-Traversierung zur Sicherung von Daten, die in einem geordneten Binärbaum gespeichert sind, als vorteilhaft gegenüber anderen Traversierungen erweist.

4. Aufgrund der großen Distanzen zwischen den Orten des Festivals nutzen viele Besucher Fahrräder als Transportmittel. Der nachfolgende Graph G enthält als Knoten die Orte mit den Camps C_1 bis C_7, mit dem Burning Man B sowie mit den besonders bedeutenden Kunstwerken K_1 bis K_3. Die Kantengewichte geben die durchschnittliche Fahrtdauer in Minuten an.



Im Folgenden werden Teilgraphen dieses Graphen betrachtet. Ein Teilgraph von G ist ein Graph, dessen sämtliche Knoten und Kanten auch in G enthalten sind.

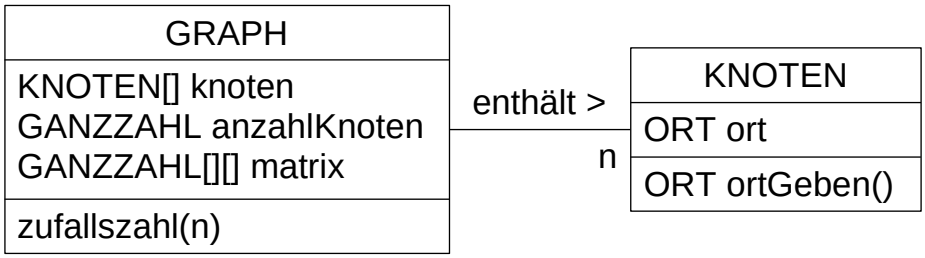
(Fortsetzung nächste Seite)

BE	
7	a) Die Besucher wünschen sich, dass jeder der gegebenen elf Orte von jedem anderen Ort über einen direkten Radweg erreichbar ist. Den dazugehörigen ungerichteten Graphen bezeichnet man als vollständig. Zeichnen Sie einen Teilgraphen von G mit drei Knoten, der vollständig ist. Zeichnen Sie außerdem einen Teilgraphen von G mit fünf Knoten, der mit möglichst wenig weiteren Kanten zu einem vollständigen Graphen ergänzt werden kann. Geben Sie an, wie viele Kanten dafür hinzugefügt werden müssen. Bestimmen Sie darüber hinaus, wie viele Radwege zusätzlich zu den 19 vorhandenen noch gebaut werden müssen, um den Wunsch der Besucher zu erfüllen.
6	b) Alle Orte sollen über Radschnellwege erreichbar sein. Um zu entscheiden, welche der bestehenden Verbindungsstrecken dazu sinnvollerweise ausgebaut werden, werden sogenannte minimale Spannbäume des Graphen betrachtet. Ein Spannbaum von G ist ein Baum, der sämtliche Knoten und eine Teilmenge der Kanten von G enthält. Ein solcher Spannbaum heißt minimal, wenn kein anderer Spannbaum für G existiert, bei dem die Summe der Kantengewichte kleiner ist. Geben Sie einen minimalen Spannbaum von G an.

(Fortsetzung nächste Seite)

BE
8
80

c) Der Verwaltung des Graphen G liegt folgendes Klassendiagramm zugrunde:



Die Methode *zufallszahl(n)* liefert eine zufällige Ganzzahl im Bereich von 0 bis n−1.

Unentschlossene Besucher sollen sich mithilfe der App einen Vorschlag anzeigen lassen können, welchen Ort sie als nächstes besuchen könnten. Der vorgeschlagene Ort soll über eine direkte Radwegverbindung vom aktuellen Aufenthaltsort des Besuchers erreichbar sein.

Die Klasse GRAPH soll zu diesem Zweck eine Methode *ortVorschlagen(index)* besitzen, die für den Knoten mit Index *index* einen seiner direkten Nachbarknoten zufällig auswählt und dessen Ort zurückgibt.

Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine Implementierung der Methode *ortVorschlagen*.

INF1. MODELLIERUNG UND PROGRAMMIERUNG

II.

BE

Für den Tischtennisverein „TTV BounceOut“ soll eine App entwickelt werden.

1. Die einzelnen Mannschaften des Vereins treffen sich zu Trainings- und Wettkampfterminen. Eine Mannschaft umfasst stets genau acht Spielerinnen und Spieler. Diesen sind die festen Spieler-IDs 0 bis 7 zugeordnet. Bei jedem Wettkampftermin kommen genau sechs der acht Spielerinnen und Spieler einer Mannschaft zum Einsatz. Es ist nicht vorgesehen, dass sich während einer Saison die Zusammensetzung der Mannschaft ändert.

   Mit der App soll die Anwesenheit der Mitglieder einer Mannschaft bei den einzelnen Terminen, wie in nachfolgender Tabelle beispielhaft dargestellt, erfasst werden können.

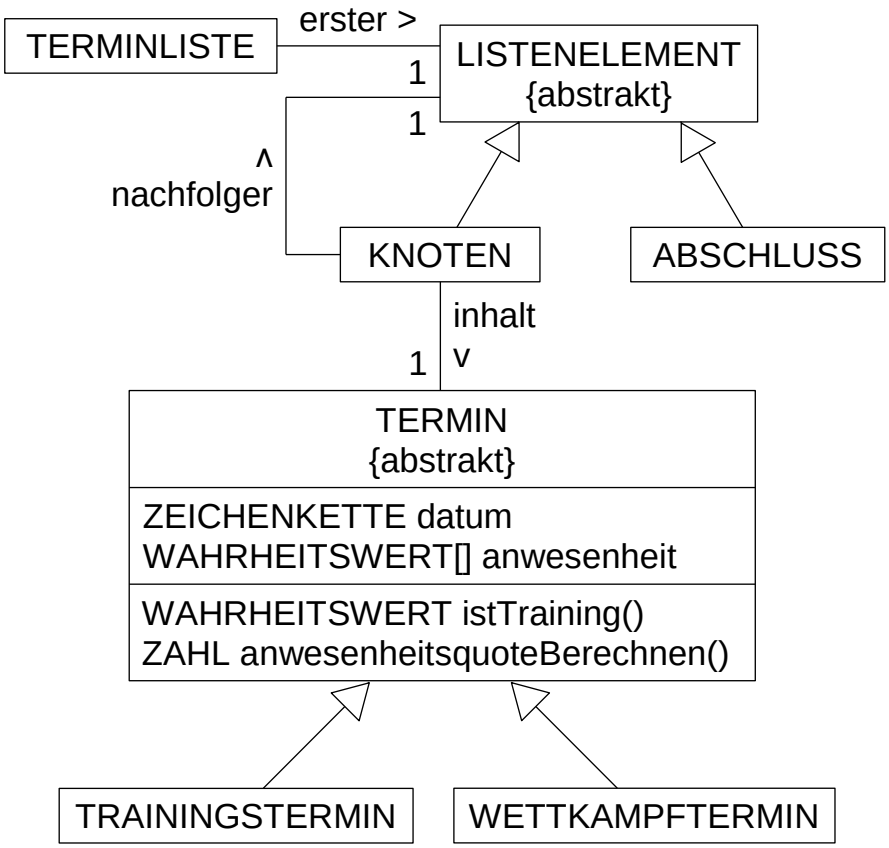
	Anna	Ben	Caro	Daniela	Emma	Fred	Monika	Tobias
05.01.21 (Training)	nein	ja	nein	ja	ja	ja	ja	nein
07.01.21 (Wettkampf)	ja	nein	ja	ja	nein	ja	ja	ja
12.01.21 (Training)	ja	nein	ja	ja	ja	ja	nein	ja

   Trainingstermine werden gelegentlich auch spontan festgelegt, daher sollen neue Termine auch kurzfristig in der App angelegt werden können.

(Fortsetzung nächste Seite)

BE

Die Entwickler der App haben sich entschieden, die Termine einer Saison in einer Liste zu speichern, die auf folgendem Klassendiagramm basiert:



Bei den Attributen der Klasse TERMIN handelt es sich um eine Zeichenkette *datum*, die das Datum des Termins repräsentiert, sowie um ein Feld *anwesenheit* von Wahrheitswerten, in dem der Feldindex genau der Spieler-ID entspricht. Der Wert des Feldelements ist genau dann *wahr*, wenn der entsprechende Spieler bzw. die entsprechende Spielerin seine bzw. ihre Teilnahme am jeweiligen Termin zugesagt hat. Für beide Attribute existieren entsprechende Methoden zum Lesen und Setzen der Attributwerte.

Die Methode *istTraining* der Klasse TERMIN liefert genau dann *wahr*, wenn es sich bei dem Termin um einen Trainingstermin handelt. Die Methode *anwesenheitsquoteBerechnen* gibt den Anteil der Mannschaftsmitglieder zurück, die den Termin zum Zeitpunkt der Methodenausführung zugesagt haben.

2

a) Begründen Sie, warum sich zur Verwaltung der Termine obige Listenstruktur besser eignet als ein Feld. Beziehen Sie sich dabei auf die oben beschriebenen Anforderungen an die App.

(Fortsetzung nächste Seite)

BE	
3	b) Beschreiben Sie eine Möglichkeit, die Methode *istTraining* umzusetzen.
5	c) Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine Implementierung der Methode *anwesenheitsquoteBerechnen* der Klasse TERMIN.

13	d) Die Klasse TERMINLISTE soll folgende zwei Methoden besitzen:
	• *anzahlTrainingsGeben(spielerID)*: gibt für den Spieler bzw. für die Spielerin mit der ID *spielerID* zurück, für wie viele Trainingstermine er bzw. sie zugesagt hat;
	• *trainingsbeteiligungAusgeben()*: gibt für jeden Trainingstermin die Anwesenheitsquote (in Prozent) in folgender Form auf der Konsole aus:
	`Training 05.01.21 62.5%`
	`Training 12.01.21 75%`
	Notieren Sie in einer auf dem Deckblatt angegebenen Programmiersprache eine Implementierung dieser beiden sowie aller dafür in den Klassen der Listenstruktur benötigten Methoden. Wenden Sie dabei jeweils das Prinzip der Rekursion an.
	Die Klasse TERMIN kann als vollständig implementiert vorausgesetzt werden. Berücksichtigen Sie, dass im Feld *anwesenheit* der Feldindex genau der Spieler-ID entspricht.

(Fortsetzung nächste Seite)

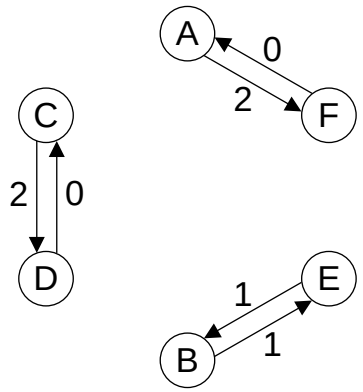
BE

2. Die App soll auch eine Unterstützung bei der Durchführung von Freizeitturnieren bieten, bei denen jedes Spiel nach einer vorgegebenen Dauer abgebrochen wird.

Für die Durchführung eignet sich das Schweizer System. Dabei hat jeder Spieler zu Beginn des Turniers 0 Punkte. Die Spielpaarungen werden in jeder Runde folgendermaßen bestimmt: Der Spieler mit der aktuell höchsten Punktzahl spielt gegen den punktbesten derjenigen Spieler, gegen die er noch nicht gespielt hat. Bei Punktgleichheit entscheidet der Zufall. Aus den übrigen Spielern werden anschließend die restlichen Paarungen genauso gebildet, wobei vereinfachend davon ausgegangen wird, dass dies stets möglich ist. Der Sieger einer Paarung erhält 2 Punkte, der Verlierer 0 Punkte. Steht es bei Spielabbruch unentschieden, gibt es für beide einen Punkt.

Die Software soll die Spielpaarungen und deren Ergebnisse in einem Graphen speichern.

Die Vereinsmitglieder Anna (A), Ben (B), Caro (C), Daniela (D), Emma (E) und Fred (F) führen ein solches Turnier durch. In der ersten Runde gewinnt Anna gegen Fred und Caro gegen Daniela; das Spiel zwischen Ben und Emma endet unentschieden. Diese Spielergebnisse werden mithilfe des folgenden Graphen repräsentiert:



8 a) In der zweiten Runde gewinnt Anna gegen Caro und Ben gegen Daniela; Emma und Fred trennen sich unentschieden.

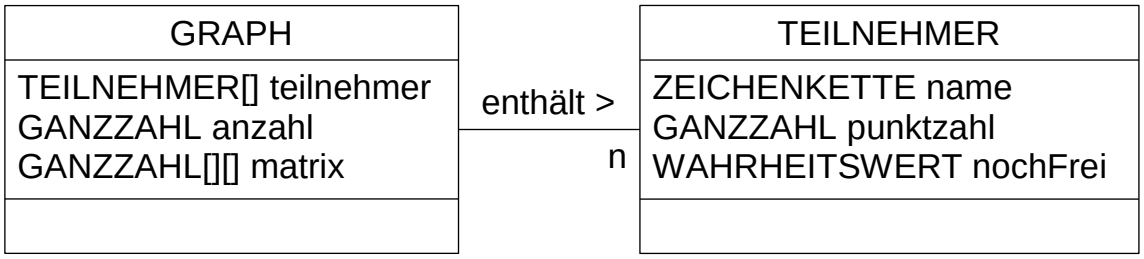
Geben Sie den aus den beiden Runden resultierenden Graphen sowie die zugehörige Adjazenzmatrix an.

Nennen Sie zudem die Spielpaarungen, die sich daraus für die dritte Runde ergeben.

(Fortsetzung nächste Seite)

BE

Um ein solches Turnier zu verwalten, wird folgendes Klassendiagramm zugrunde gelegt:



In der Klasse GRAPH steht das Attribut *anzahl* für die gerade Anzahl der Teilnehmer des Turniers. Die Einträge der Adjazenzmatrix *matrix* haben zu Beginn alle den Wert −1. In der Klasse TEILNEHMER gibt das Attribut *nochFrei* an, ob ein Teilnehmer noch für eine Paarung zur Verfügung steht oder bereits zugeordnet ist.

5

b) Eine Methode *punktzahlenAktualisieren()* der Klasse GRAPH sorgt nach jeder Runde dafür, dass für jeden Teilnehmer die Gesamtpunktzahl mithilfe der Adjazenzmatrix berechnet und im Attribut *punktzahl* gespeichert wird. Methoden zum Lesen und Setzen der Werte des Attributs *punktzahl* können als gegeben vorausgesetzt werden. Notieren Sie eine Implementierung dieser Methode in einer auf dem Deckblatt angegebenen Programmiersprache.

8

c) Gegeben ist folgende Methode *suchen* der Klasse GRAPH.

```
Methode suchen()
    index = −1
    punktzahl = −1
    zähle i von 0 bis anzahl−1
        wenn   (teilnehmer[i].nochFreiGeben() ist wahr) und
               (teilnehmer[i].punktzahlGeben() > punktzahl) dann
            index = i
            punktzahl = teilnehmer[i].punktzahlGeben()
        endeWenn
    endeZähle
    wenn index>−1 dann
        teilnehmer[index].nochFreiSetzen(false)
    endeWenn
    gib index zurück
endeMethode
```

(Fortsetzung nächste Seite)

BE

Beschreiben Sie, was die Methode bewirkt, und interpretieren Sie den Rückgabewert.

Beschreiben Sie zudem, wie diese Methode zum Finden der ersten Paarung einer Runde verwendet werden kann.

10 d) Damit bei Punktegleichheit nicht immer der Zufall über den höheren Rang entscheidet, wird als Zusatzpunktzahl die Anzahl aller Knoten ermittelt, die von einem Knoten ausschließlich über Kanten mit Gewicht 2 direkt oder indirekt erreichbar sind. Beispielsweise hat Anna drei Zusatzpunkte, nachdem sie in der zweiten Runde gegen Caro gewonnen hat.

Formulieren Sie einen rekursiven Algorithmus (z. B. in Pseudocode) für eine Methode *zusatzpunkteGeben(index)* der Klasse GRAPH, welche die Zusatzpunkte eines Spielers bzw. einer Spielerin wie oben beschrieben berechnet und zurückgibt. Dabei ist *index* der Index des betreffenden Spielers bzw. der betreffenden Spielerin im Feld *teilnehmer* der Klasse GRAPH.

3. Jede Mannschaft eines im Deutschen Tischtennis-Bund (DTTB) registrierten Vereins spielt in einer Liga mit einem bestimmten Spielsystem, hat einen Trainer, einen Namen sowie noch weitere mannschaftsspezifische Eigenschaften.

Zur Organisation des Spielbetriebs soll eine entsprechende Software zum Einsatz kommen. Die Software verwaltet unter anderem die Spieltermine einer Liga sowie die Mannschaften eines Vereins. Außerdem werden Ergebnisse von Wettkämpfen von der Software erfasst.

7 a) Modellieren Sie zunächst für das oben beschriebene Szenario den Ausschnitt des Systems mit den Klassen VEREIN, MANNSCHAFT und LIGA als Klassendiagramm. Geben Sie zu jeder Klasse wenigstens zwei sinnvolle Attribute an.

Nennen und beschreiben Sie außerdem noch zwei weitere Klassen, die für die Modellierung des Spielbetriebs einer Liga von Bedeutung sein könnten.

4 b) Erläutern Sie, wie Sie Ihr Klassendiagramm aus Teilaufgabe 3a an geeigneter Stelle nach dem Prinzip von Generalisierung und Spezialisierung sinnvoll erweitern können.

(Fortsetzung nächste Seite)

BE

4. In der Datenbank des DTTB sind zurzeit die Daten von etwa 10 000 Vereinen und 540 000 Spielerinnen und Spieler sowie fast 50 Millionen Spielergebnisse gespeichert. Jede Spielerin bzw. jeder Spieler erhält eine Platzziffer (D-Rang), die anhand der jeweiligen Spielergebnisse laufend aktualisiert wird. Vereinfachend wird angenommen, dass jede Platzziffer eindeutig ist.

   Neben dem D-Rang wird zu jedem Spieler bzw. zu jeder Spielerin der Name und das Geschlecht, die eindeutige Spieler-ID, der Jahrgang sowie der eindeutige Name des zugehörigen Vereins in der Tabelle *spieler* gespeichert:

   spieler:

SpielerID	DRang	Name	Geschlecht	Jahrgang	Verein
...	...	...	...	...	...

3 a) Erstellen Sie eine Datenbankabfrage (z. B. in SQL), die als Ergebnistabelle die Vereinsrangliste des Vereins „TTV BounceOut“ in folgender Form zurückgibt:

DRang	Name	Geschlecht
11142	Daniela	w
17584	Fred	m
20847	Anna	w
22135	Monika	w
24092	Ben	m
26020	Emma	w
26078	Tobias	m
27251	Caro	w

   Hinweis: Nutzen Sie zur Sortierung nach einer Spalte *Sortierspalte* den Zusatz „ORDER BY Sortierspalte“.

(Fortsetzung nächste Seite)

BE	
	Zur Erstellung einer Vereinsrangliste ist auch ein Vorgehen denkbar, bei dem alle Spielerinnen und Spieler des Vereins in einen nach dem D-Rang geordneten Binärbaum eingefügt und anschließend mithilfe eines geeigneten Baumdurchlaufs ausgegeben werden.
7	b) Die in Teilaufgabe 4a gegebenen Spielerdaten werden in alphabetischer Reihenfolge in einen solchen, zunächst leeren Binärbaum eingefügt. Zeichnen Sie den Baum, der hierdurch entsteht. Dabei genügt es, die Knoten nur mit dem Anfangsbuchstaben des Namens und dem D-Rang zu bezeichnen. Geben Sie den geeigneten Baumdurchlauf für die Ausgabe der Vereinsrangliste an. Geben Sie außerdem an, in welcher Reihenfolge die Knoten beim Preorder-Durchlauf bearbeitet werden.
2	c) Die Vereinsrangliste eines bestimmten Vereins wird wie folgt ermittelt: • Verwendung eines nach dem D-Rang geordneten Baumes, der alle 540 000 Spielerinnen und Spieler enthält • Geeigneter Baumdurchlauf mit Ausgabe nur der zum jeweiligen Verein gehörenden Spielerinnen und Spieler Nehmen Sie begründet Stellung, ob bei dieser Strategie der Vorteil des geordneten Binärbaums zum Tragen kommt.
3	d) Intern wird auch die Tabelle *spieler* als Binärbaum gespeichert. Ermitteln Sie die minimal und die maximal mögliche Höhe des Binärbaums.
80	

INF2. THEORETISCHE UND TECHNISCHE INFORMATIK

III.

BE

1. Formeln in Tabellenkalkulationssystemen enthalten häufig Zellbezüge, die durch Angabe einer Spalte (A, B, ..., Z, AA, AB, ..., AZ, BA, BB, ..., BZ, ..., ZZ, AAA, AAB, ...) gefolgt von der Angabe einer Zeile (1, 2, 3, ...) eine Zelle eindeutig bestimmen. Zusätzlich kann in einem Zellbezug sowohl vor dem Spaltenbezeichner als auch vor dem Zeilenbezeichner jeweils ein Dollarzeichen stehen. Beispielsweise sind GP$7 und C22 korrekte Zellbezüge.

   Die Sprache L enthält alle Zellbezüge, bei denen nur Spalten von A bis ZZ (in Großbuchstaben) und Zeilen von 1 bis 999 (ohne führende Nullen) einschließlich möglicher Dollarzeichen zur Verfügung stehen.

6 a) Geben Sie ein Zustandsübergangsdiagramm eines endlichen Automaten an, der genau die Wörter der Sprache L akzeptiert.

   Weisen Sie anhand Ihres Zustandsübergangsdiagramms durch Angabe der durchlaufenen Zustände nach, dass der Zellbezug R$818 syntaktisch korrekt ist.

4 b) Notieren Sie die Produktionsregeln einer Grammatik, die L erzeugt, in einer formalen Textnotation (z. B. in EBNF).

5 c) Manche Tabellenkalkulationssysteme stellen standardmäßig $2^{10} = 1024$ Spalten zur Verfügung. In diesem Fall enden die Spaltenbezeichner bei AMJ.

   Geben Sie ein Syntaxdiagramm an, das solche Spaltenbezeichner korrekt erzeugt.

(Fortsetzung nächste Seite)

BE

2. Zwei Geheimdienste aus benachbarten, nicht englischsprachigen Ländern übermitteln Nachrichten nach folgendem Verfahren:

Wenn ein Agent eine Nachricht für einen Agenten des Nachbarlandes verfasst hat, wird sie zunächst von einem Übersetzer ins Englische übersetzt und von einem Codierer verschlüsselt. Ein Kurier bringt dann die Nachricht an die Landesgrenze und übergibt sie dort einem Kurier des Nachbarlandes. Dieser bringt sie in die Geheimdienstzentrale, wo sie entschlüsselt und in die Landessprache übersetzt wird, bevor der Adressat die Nachricht erhält.

4 a) Stellen Sie den beschriebenen Vorgang in einem Schichtenmodell dar. Benennen Sie die Schichten passend.

4 b) Erläutern Sie allgemein, welche Rolle Protokolle im Schichtenmodell spielen.

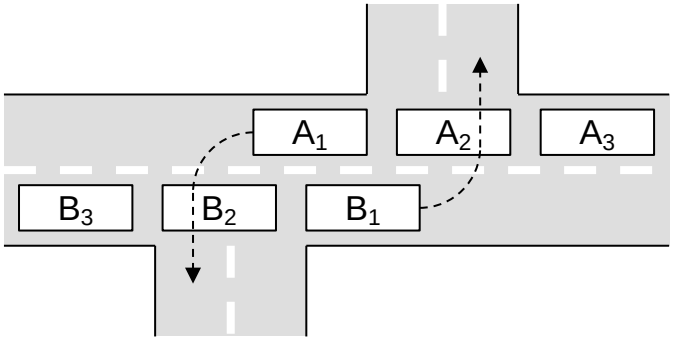
Verdeutlichen Sie zudem anhand einer Schicht Ihres Modells aus Teilaufgabe 2a, dass sich Änderungen am Protokoll nur auf die jeweilige Schicht auswirken.

4 c) Um die Nachricht zu decodieren, ist ein Schlüssel notwendig, der aus einer Zeichenkette gebildet wird. Dafür stehen 100 verschiedene Zeichen (Groß- und Kleinbuchstaben, Ziffern, Sonderzeichen) zur Verfügung, die beliebig miteinander kombiniert werden können. Ein Computer kann mit der Brute-Force-Methode fünf Millionen Schlüssel pro Sekunde auf einen codierten Text anwenden. Berechnen Sie, wie lang die Zeichenkette mindestens sein muss, damit dieser Computer an einem Tag weniger als 0,1 % aller möglichen Schlüssel dieser festen Länge ausprobieren kann.

(Fortsetzung nächste Seite)

BE

3. An einer versetzten Kreuzung sind die Autos A_1 und B_1 Linksabbieger (siehe Abbildung).



3 a) Erläutern Sie allgemein, was man in der Informatik unter einer Verklemmung versteht, und begründen Sie, warum die abgebildete Situation eine Verklemmung darstellt.

3 b) Geben Sie eine Verkehrsregel an, die eine Verklemmung wie die dargestellte generell verhindern würde, ohne dass die Möglichkeit des Linksabbiegens genommen wird.

Beschreiben Sie eine Strategie, mit der in der Informatik Verklemmungen vermieden werden.

(Fortsetzung nächste Seite)

BE

4. Gegeben ist eine Registermaschine mit folgendem Befehlssatz:

load x	kopiert den Wert aus der Speicherzelle x in den Akkumulator
load (x)	kopiert den Wert aus derjenigen Speicherzelle in den Akkumulator, deren Adresse in der Speicherzelle x steht
loadi n	lädt die ganze Zahl n in den Akkumulator
store x	kopiert den Wert aus dem Akkumulator in die Speicherzelle x
store (x)	kopiert den Wert aus dem Akkumulator in diejenige Speicherzelle, deren Adresse in der Speicherzelle x steht
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
jmp x	springt zum Befehl in Speicherzelle x
jeq x	springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator gleich null ist
jne x	springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator ungleich null ist
hold	beendet die Abarbeitung des Programms

Hinweis: Durch die Angabe einer Adresse in Klammern kann bei den Befehlen load und store indirekt adressiert werden. Enthält beispielsweise die Speicherzelle 50 den Wert 51 und die Speicherzelle 51 den Wert 3, so interpretiert der Befehl „load (50)“ den Wert 51 als Adresse einer Speicherzelle und lädt den Wert der Speicherzelle 51, also 3, in den Akkumulator.

BE

3

a) Gegeben ist folgendes Programm:

```
            loadi 101
            store 100
start:      load (100)
            jeq ende
            load 100
            addi 1
            store 100
            jmp start
ende:       hold
```

Die folgende Tabelle zeigt den Anfangszustand der Registermaschine und den Zustand nach jedem ausgeführten Befehl, der eine Änderung des Zustands bewirkt. Der Zustand wird auf den Akkumulator (AK) sowie die Speicherzellen 100, 101 und 102 beschränkt.

Führen Sie die Tabelle für den Ablauf des gesamten Programms fort.

Befehl	AK	100	101	102
(Anfangszustand)			7	0
loadi 101	101			
store 100		101		
...				

4

b) In den Speicherzellen ab 101 ist eine endliche Folge aus positiven ganzen Zahlen gespeichert. Die Speicherzelle direkt nach der Zahlenfolge ist mit 0 belegt.

Es soll geprüft werden, ob eine in Speicherzelle 99 gespeicherte Zahl in der gegebenen Zahlenfolge vorkommt oder nicht.

Modifizieren Sie das in Teilaufgabe 4a gegebene Programm entsprechend, sodass nach seinem Durchlauf in Speicherzelle 98 eine 1 steht, falls die gegebene Zahl in der Folge vorkommt, ansonsten eine 0.

40

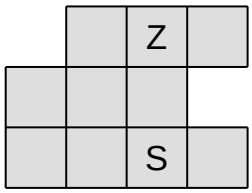
IV.

BE

1. Bei einem einfachen Computerspiel kann man die Spielfigur mit den Pfeiltasten bewegen, die im Folgenden mit l, r, o, u (für links, rechts, oben, unten) bezeichnet werden. Eine erfolgreiche Eingabesequenz steuert die Figur innerhalb eines Spielfelds vom Startfeld (S) zum Zielfeld (Z), wobei jedes Feld höchstens einmal betreten werden darf.

   Für ein bestimmtes Level k des Computerspiels wird die Menge aller erfolgreichen Eingabesequenzen als Sprache L_k aufgefasst.

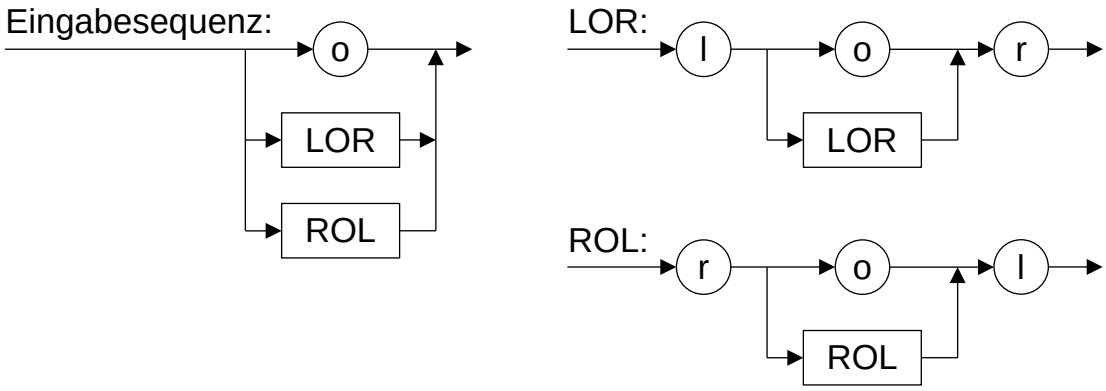
6 a) Die folgende Skizze zeigt das Spielfeld für Level 1:



Die Eingabesequenz „oo“ ist hier beispielsweise erfolgreich, die Sequenz „ollorr“ jedoch nicht, da dabei die Außenbegrenzung übertreten wird.

Zeichnen Sie das Zustandsübergangsdiagramm eines endlichen Automaten, der genau die Eingabesequenzen aus L_1 akzeptiert.

Das folgende Syntaxdiagramm beschreibt die Produktionsregeln einer Grammatik für die Sprache L_2 mit der Startvariablen <Eingabesequenz>:



3 b) Geben Sie die durch das Diagramm festgelegten Produktionsregeln in einer formalen Textnotation (z. B. EBNF) an.

(Fortsetzung nächste Seite)

BE	
4	c) Beschreiben Sie anhand einer Skizze ein mögliches Spielfeld von Level 2 inklusive Start- und Zielfeld. Begründen Sie, warum kein endlicher Automat existiert, der genau die Eingabesequenzen aus L_2 akzeptiert.

2. Bei der Deutschen Bahn können Fahrkarten an einem Fahrkartenautomaten oder am eigenen Endgerät (Computer, Smartphone etc.) gekauft werden. Zusätzlich können sie am Schalter oder teilweise auch im Zug gelöst werden, wobei auch hier ein Computer verwendet wird, um den Kaufvorgang abzuwickeln. Alle Endgeräte nutzen bei der Fahrkartenbuchung das Internet zur Kommunikation mit einem zentralen Fahrkartenserver.

5	a) Diskutieren Sie, inwieweit dieses Szenario mit einer Stern-Topologie realisiert werden kann. Geben Sie zudem zwei Vorteile und einen Nachteil der Stern-Topologie an.

Im Folgenden wird bei einer Fahrkartenbuchung immer davon ausgegangen, dass auch ein Sitzplatz reserviert wird. Dazu läuft auf dem Server für jeden verbundenen Client ein Prozess, der die Kommunikation mit diesem regelt und auf das zentrale Platzreservierungssystem als gemeinsam genutzte Ressource zugreift.

4	b) Erläutern Sie an einem Beispiel, warum es dabei zu einer fehlerhaften Buchung kommen kann. Nennen Sie ein Konzept, mit dem dieses Problem verhindert werden kann.
4	c) Will man ein Fahrrad mitnehmen, so muss zusätzlich ein Fahrradstellplatz im Zug reserviert werden. Bei der Buchung werden vom Clientprozess stets zuerst der Sitzplatz und dann der Fahrradstellplatz gebucht. Entscheiden Sie, ob es in der gegebenen Situation unter Anwendung des Konzepts aus Teilaufgabe 2b zu einer Verklemmung kommen kann, und begründen Sie Ihre Entscheidung.

(Fortsetzung nächste Seite)

BE

3. Gegeben ist folgende rekursive Methode *m*. Der Parameter *z* steht für eine beliebige nichtnegative ganze Zahl.

```
Methode m(z)
    wenn z<10 dann
        gib z zurück
    sonst
        gib (z%10)+m(z/10) zurück
    endeWenn
endeMethode
```

Hinweis: Der Operator / liefert den ganzzahligen Anteil der ganzzahligen Division, der Operator % deren Rest.

4 a) Geben Sie jeweils die Aufrufsequenz der Methodenaufrufe m(3) und m(752) an und beschreiben Sie allgemein die Bedeutung des Rückgabewerts von *m*.

3 b) Beschreibe Sie, wie sich eine Vergrößerung von *z* auf die Anzahl der rekursiven Methodenaufrufe auswirkt, und treffen Sie eine Aussage über das Laufzeitverhalten von *m* in Abhängigkeit von *z*.

(Fortsetzung nächste Seite)

BE

Gegeben ist eine Registermaschine mit folgendem 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 in Speicherzelle x zum Wert im Akkumulator
addi n	addiert die ganze Zahl n zum Wert im Akkumulator
sub x	subtrahiert den Wert in Speicherzelle x vom Wert im Akkumulator
subi n	subtrahiert die ganze Zahl n vom Wert im Akkumulator
div x	dividiert den Wert im Akkumulator durch den Wert in 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 Akkumulators durch den Wert in Speicherzelle x in den Akkumulator
modi n	speichert den Rest bei der Ganzzahldivision des Akkumulators 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 gleich null ist
jne x	springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator ungleich null ist
jge x	springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator gleich null oder positiv ist
jle x	springt zum Befehl in Speicherzelle x, falls der Wert im Akkumulator gleich null oder negativ 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

7

c) Setzen Sie die Methode *m* in ein Programm für die gegebene Registermaschine um.

40