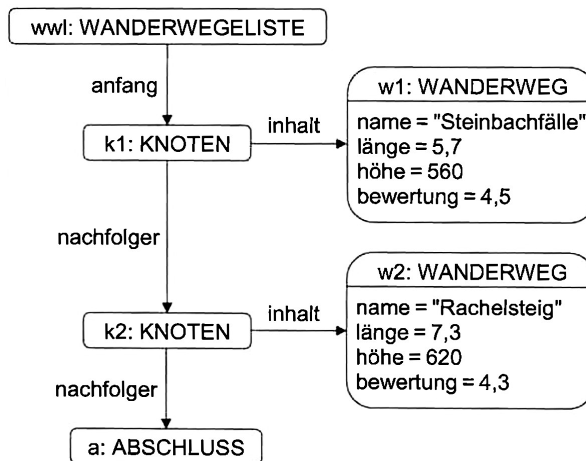
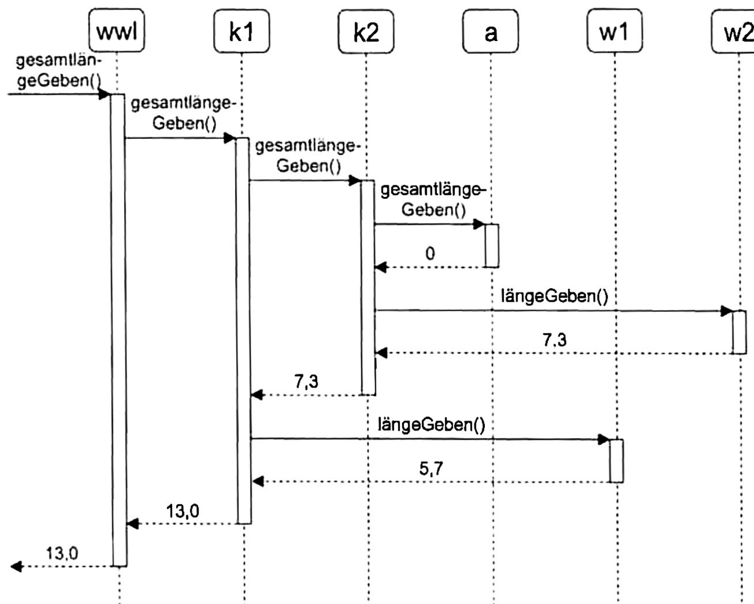
Informatik Abitur Bayern 2023 / II - Lösung

Autor:
Unsin (1)
Kufner (2, 3)

1a Das Objektdiagramm muss ein Objekt der Klasse WANDERWEGLISTE (wwl), zwei Objekte der Klasse KNOTEN (k1, k2), zwei Objekte der Klasse WANDERWEG (w1, w2) und ein Objekte der Klasse Abschluss (a) enthalten. 4



1b 10



Wanderwege können gemeinsame Streckenabschnitte enthalten. Diese werden im berechneten Summenwert mehrfach gezählt, in der Gesamtlänge des Wegnetzes jedoch nur einfach, deshalb kann der Summenwert größer sein als die Gesamtlänge des Wegnetzes.

1c Die Schwierigkeit muss zuerst für jeden Wanderweg berechnet werden, deshalb muss die Klasse WANDERWEG eine geeignete Methode enthalten, z.B. mit dem Namen *schwierigkeitGeben().* Diese Methode bildet den Quotienten der Attribute höhe und länge und gibt diesen zurück. 5
In der Klasse WANDERWEGLISTE berechnet eine weitere Methode den Durchschnitt der Schwierigkeiten. Hierfür wird ein Zähler (Datentyp int) und eine Summe (Datentyp float) mit Startwert 0 initialisiert. Für jedes Listenelement wird der Zähler um 1 erhöht und die Summe um den Wert von *schwierigkeitGeben().* Wenn das Listenende erreicht ist, wird der Quotient von Summe und Zähler gebildet, dieser ist die durchschnittliche Schwierigkeit.

1d In der Klasse WANDERWEGLISTE muss eine Methode ohne Rückgabewert ergänzt werden: 8

```
void wanderwegeLöschen(float grenzwert) {
```


Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

```
        anfang = anfang.wanderwegLöschen(grenzwert);
}
```

In der Klasse Listenelement wird die Methode deklariert, aber noch nicht initialisiert:

```
abstract Listenelement wanderwegLöschen(float grenzwert);
```

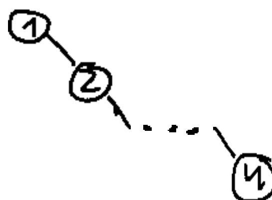
In der Klasse ABSCHLUSS muss nun die Methode mit Rumpf definiert werden:

```
Listenelement wanderwegLöschen(float grenzwert) {
        return this; // Listenende erreicht
}
```

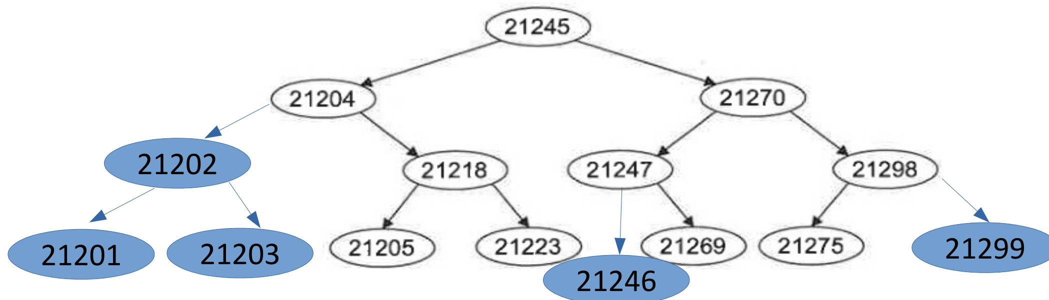
In den Klassen KNOTEN muss ebenfalls die Methode mit Rumpf definiert werden. Hier findet die Unterscheidung je nach Bewertung statt.

```
Listenelement wanderwegLöschen (float grenzwert) {
        nachfolger = nachfolger.wanderwegLöschen(grenzwert);
        if (inhalt.bewertungGeben() < grenzwert) {
                return nachfolger; // Element muss gelöscht werden
        else {
                return this;
        }
}
```

2a Wenn die Buchungs-IDs nicht zufällig vergeben werden, so entsteht beim Einfügen ein entarteter Baum. Wenn die IDs von 1 aufsteigend in den Baum einsortiert werden, so entsteht eine Listenstruktur und die schnelle Suche eines balancierten Binärbaums ist nicht mehr möglich. — 2



2b — 4



Beispielhafte Einfügereihenfolge: 21202 – 21201 – 21203 – 21246 – 21299

2c — 5

Suchvorgang	Verkettete Liste	Binärbaum
Suche nach Buchung anhand Buchungs-ID	Gesamte Liste wird durchsucht, also 25000 Vergleiche	Gesuchtes El. in max. Tiefe ➠ $\log_2(25000) = 14{,}61$ ➠ 15 Vergleiche
Suche nach Person	25000 Vergleiche, in jedem Element muss Name gesucht und verglichen werden	25000 Vergleiche, alle Knoten müssen besucht und der Name verglichen werden

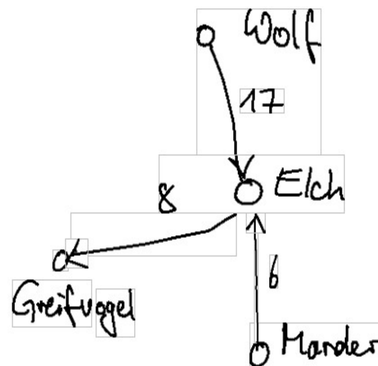
2d — 11

```
class BUCHUNGSVERZEICHNIS {
    private BAUMELEMENT wurzel;

    public int gesamtzahlÜbernachtungenGeben(int jahr) {
```


Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

```
            return wurzel.gesamtzahlGeben(jahr);
        }
    }

    abstract class BAUMELEMENT {
        abstract int gesamtzahlGeben(int jahr);
    }

    class KNOTEN extends BAUMELEMENT {
        private BUCHUNG daten;
        private BAUMELEMENT rechterNachfolger;
        private BAUMELEMENT linkerNachfolger;

        public int gesamtzahlGeben(jahr) {
            int ergebnis = 0;

            ergebnis = ergebnis + linkerNachfolger.gesamtzahlGeben(jahr);
            if (daten.gebeJahr() == jahr) {
                ergebnis = ergebnis + daten.gibAnzahlÜbernachtungen() * daten.gibAn-
zahlPersonen();
            }
            ergebnis = ergebnis + rechterNachfolger.gesamtGeben(jahr);

            return ergebnis;
        }

    }

    class ABSCHLUSS extends BAUMELEMENT {
        public int gesamtzahlGeben(int jahr) {
            return 0;
        }
    }
```

3a Die Eigenschaft der **Gerichtetheit** ändert sich – der Graph ist jetzt gerichtet und war zuvor ungerichtet. 6



<u>Adjazenzmatrix:</u> (gelb markiert: Unterschiede zur Adjazenzmatrix des ungerichteten Graphen)

	G	M	E	W
G	0			
M		0	6	
E	8		0	
W			17	0

<u>Allgemeine Änderung:</u> Die Adjazenzmatrix eines ungerichteten Graphen ist symmetrisch an der Diagonalen. Durch die neue Gerichtetheit des Graphen ist dies nicht mehr der Fall und die „gegenüberliegenden“ Felder der vorhandenen Kanten (durch 1 markiert) sind nicht mehr mit 1 markiert, sondern mit „keine Kante“ (z. B. -1).

3b Ein Rundweg ist nicht möglich, da es eine Kante gibt (zwischen Elch und Marder), welche zwei Teile des Parks verbindet. Um alle Gehege zu besuchen, müsste diese Kante zweimal 5

Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

beschritten werden, was nicht möglich ist. Wenn eine Kante zwischen Wisent und Auerhahn existieren würde, wäre folgender Rundweg möglich:
Biber – Marder – Elch – Greifvogel – Braunbär – Kauz – Rothirsch – Otter – Wildschwein – Wildkatze – Wolf – Uhu – Luchs – Wisent – Auerhahn – Biber

3c Man muss nur alle Kanten, die von einem Knoten ausgehen, so ausrichten, dass diese vom Knoten wegzeigen, z.B. die Kante Otter-Wildschwein und Otter-Rothirsch vom Otter weggehend wählen. Dann ist das Gehege Otter nicht mehr erreichbar. — 3
Da ein solches Gehege keine eingehenden Kanten hat, muss die Adjazenzmatrix nach einer Zeile bzw. Spalte durchsucht werden, welche nur 0 bzw. „keine Kante“ enthält. Da die obige Adjazenzmatrix Kanten „von Zeile zu Spalte“ codiert, müsste hier nach einer Spalte mit solchen Werten gesucht werden.

3d Gesucht ist ein Verfahren, welches einen Pfad je zwei beliebige Knoten im Graphen findet. Eine Möglichkeit besteht darin, auf jedem Knoten die Tiefen- oder Breitensuche aufzurufen und alle Knoten zu markieren, die besucht werden. Wenn für jeden Startknoten alle anderen Knoten erreicht bzw. markiert werden, so ist die Umsetzung erfolgreich und alle Gehege sind für jedes Startgehege erreichbar. Hierbei wird angenommen, dass der Graph zusammenhängend ist. Wäre das nicht der Fall, so müssten die Knoten in jedem Teilgraph bekannt sein und das Verfahren auf jedem Teilgraphen durchgeführt werden. — 5

3e — 7

```
public boolean wegEinfuegen(String name1, String name2, int gehzeit, boolean ungerichtet) {
    // Indizes der Knoten in Variablen speichern
    int knotenIndex1 = indexErmitteln(name1);
    int knotenIndex2 = indexErmitteln(name2);

    // Prüfen, ob beide Knotennamen existieren, ansonsten Abbruch
    if (knotenIndex == -1 || knotenIndex2 == -1) {
        return false;
    }

    // Auf jeden Fall Kante von name1 zu name2 mit gegebenem Gewicht setzen
    matrix[knotenIndex1][knotenIndex2] = gehzeit;

    // Falls ungerichtet, so auch Kante in Gegenrichtung einfügen
    if (ungerichtet) {
        matrix[knotenIndex2][knotenIndex1] = gehzeit;
    }

    return true;
}
```

3f — 5

```
public String alleNachbargehegeGeben(String name) {
    // Lokale Variable mit Index des aktuellen Geheges, Initialisierung der Ergebniszeichen-
kette
    int aktuellerGehegeIndex = indexErmitteln(name);
    String ergebnis = "";

    //Adjazenzmatrix in der Zeile des aktuellen Geheges durchlaufen
    for (int i = 0; i < knoten.length; i++) {
        // Falls Eintrag nicht 0, so ex. eine Kante, also wird Zielknoten mit Gehdauer er-
mittelt und an String angehängt
        if (matrix[aktuellerGehegeIndex][i] != 0) {
            ergebnis = ergebnis + knoten[i].gehegenameGeben() + " ist " + matrix[aktueller-
GehegeIndex][i] + " Gehminuten entfernt!\n";
        }
    }

    // Nach Prüfen aller Knoten: Rückgabe des Strings
    return ergebnis;
}
```

80

Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.