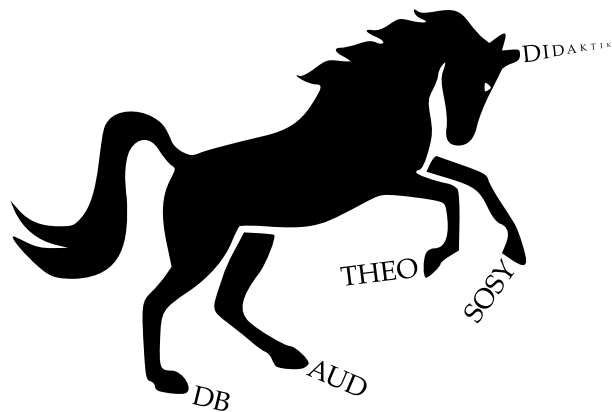


Die komplette Sammlung

Alle Aufgaben



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Inhaltsverzeichnis

I	Algorithmen und Datenstrukturen (AUD)	7
1	Rekursion	8
	Feld-Invertierer	8
	Fibonacci und Fakultät	10
	Potenz	11
	Rater	11
	Sortieren	12
	rekursives Backtracking	13
	Musikliste	16
	Aufgabe 2	20
	Rekursion	23
	Kellerspeicher: Türme von Hanoi	24
	46115 / 2014 / Frühjahr / Thema 2 / Aufgabe 4	28
	Aufgabe 4	28
	66115 / 2014 / Frühjahr / Thema 1 / Aufgabe 1	31
	66116 / 2014 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 5	35
	66116 / 2019 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 1	38
2	Suche	44
	66116 / 2019 / Herbst / Thema 2 / Aufgabe 1	44
	46115 / 2015 / Herbst / Thema 1 / Aufgabe 3	45
	Aufgabe 3	45
	46115 / 2016 / Frühjahr / Thema 2 / Aufgabe 3	50
	46115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 2	52
	46115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 3	55
	66115 / 2020 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 5	56
	66115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 2	59
	66116 / 2017 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 4	62
3	Sortieralgorithmen	67
	66116 / 2017 / Herbst / Thema 1 / Aufgabe 4	67
	66116 / 2017 / Herbst / Thema 1 / Aufgabe 4	69
	66116 / 2017 / Herbst / Thema 1 / Aufgabe 4	69
	66116 / 2017 / Herbst / Thema 1 / Aufgabe 4	73
	46115 / 2013 / Frühjahr / Thema 2 / Aufgabe 6	75
	Aufgabe 6	75
	46115 / 2016 / Frühjahr / Thema 1 / Aufgabe 8	77
	46115 / 2017 / Frühjahr / Thema 2 / Aufgabe 4	78
	Aufgabe 7	80
	Aufgabe zum Mergesort	81

46115 / 2019 / Herbst / Thema 1 / Aufgabe 4	82
46115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 1	86
46116 / 2017 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 4	87
66115 / 2006 / Herbst / Thema 1 / Aufgabe 4	89
66115 / 2014 / Herbst / Thema 2 / Aufgabe 6	91
66115 / 2015 / Herbst / Thema 2 / Aufgabe 2	94
66115 / 2016 / Frühjahr / Thema 1 / Aufgabe 6	95
66115 / 2016 / Herbst / Thema 2 / Aufgabe 7	97
66115 / 2017 / Frühjahr / Thema 1 / Aufgabe 2	100
66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 7	102
66115 / 2018 / Herbst / Thema 2 / Aufgabe 8	103
66115 / 2019 / Herbst / Thema 1 / Aufgabe 5	106
66115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 1	111
66116 / 2016 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 3	118
4 Algorithmische Komplexität (O-Notation)	122
66116 / 2016 / Herbst / Thema 1 / Aufgabe 3	122
66116 / 2016 / Herbst / Thema 1 / Aufgabe 3	123
66116 / 2016 / Herbst / Thema 1 / Aufgabe 3	124
66116 / 2016 / Herbst / Thema 1 / Aufgabe 3	124
66116 / 2016 / Herbst / Thema 1 / Aufgabe 3	125
46115 / 2016 / Herbst / Thema 2 / Aufgabe 2	126
46115 / 2020 / Frühjahr / Thema 2 / Aufgabe 6	127
46115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 1	128
66115 / 2011 / Frühjahr / Thema 1 / Aufgabe 1	129
66115 / 2012 / Herbst / Thema 2 / Aufgabe 6	133
66115 / 2014 / Herbst / Thema 2 / Aufgabe 5	134
66115 / 2015 / Frühjahr / Thema 2 / Aufgabe 5	141
66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 6	146
66115 / 2019 / Herbst / Thema 1 / Aufgabe 6	148
66115 / 2019 / Herbst / Thema 2 / Aufgabe 6	149
66115 / 2020 / Frühjahr / Thema 2 / Aufgabe 8	151
66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 4	153
5 Master-Theorem	156
6 Algorithmenmuster	157
66115 / 2020 / Herbst / Thema 1 / Aufgabe 4	157
66115 / 2020 / Herbst / Thema 1 / Aufgabe 4	158
Formalisierung	158
66115 / 2020 / Herbst / Thema 1 / Aufgabe 4	159
66115 / 2020 / Herbst / Thema 1 / Aufgabe 4	160
66115 / 2020 / Herbst / Thema 1 / Aufgabe 4	161
66115 / 2020 / Herbst / Thema 1 / Aufgabe 4	162
Realisierung des Programms	163
46114 / 2008 / Herbst / Thema 2 / Aufgabe 3	166

46115 / 2016 / Herbst / Thema 2 / Aufgabe 4	167
46115 / 2017 / Herbst / Thema 2 / Aufgabe 3	169
46115 / 2018 / Herbst / Thema 2 / Aufgabe 5	172
66115 / 2007 / Frühjahr / Thema 2 / Aufgabe 1	175
66115 / 2009 / Herbst / Thema 2 / Aufgabe 6	176
66115 / 2012 / Herbst / Thema 1 / Aufgabe 4	176
66115 / 2016 / Herbst / Thema 1 / Aufgabe 4	178
66115 / 2017 / Herbst / Thema 1 / Aufgabe 8	180
66115 / 2018 / Herbst / Thema 2 / Aufgabe 6	181
66115 / 2019 / Frühjahr / Thema 1 / Aufgabe 6	184
66115 / 2020 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 4	185

7 Listen 190

66115 / 2020 / Herbst / Thema 2 / Aufgabe 4	190
66115 / 2020 / Herbst / Thema 2 / Aufgabe 4	190
66115 / 2020 / Herbst / Thema 2 / Aufgabe 4	193
66115 / 2020 / Herbst / Thema 2 / Aufgabe 4	196
66115 / 2020 / Herbst / Thema 2 / Aufgabe 4	197
66115 / 2020 / Herbst / Thema 2 / Aufgabe 4	197
46115 / 2014 / Herbst / Thema 1 / Aufgabe 8	197
Aufgabe 8	197
46115 / 2019 / Herbst / Thema 1 / Aufgabe 6	199
46115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 2	202
46116 / 2010 / Frühjahr / Thema 1 / Aufgabe 1	204
46116 / 2011 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 1	207
Firmenstruktur	207
66115 / 2007 / Frühjahr / Thema 1 / Aufgabe 7	209
66115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 2	212
66116 / 2012 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 1	215
66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 5	221

8 Bäume 223

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	223
Klassendiagramm und Implementierung	223
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	229
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	229
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	231
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	232
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	233
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	237
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	238
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5	239
46115 / 2010 / Frühjahr / Thema 1 / Aufgabe 5	241
5. Datenstrukturen und Algorithmen: Binäre Suchbäume und AVL-Bäume	241
46115 / 2011 / Frühjahr / Thema 1 / Aufgabe 3	246
Aufgabe 2-3-4-B-Baum und AVL-Baum	246

46115 / 2012 / Frühjahr / Thema 1 / Aufgabe 6	252
Aufgabe 6	252
46115 / 2013 / Frühjahr / Thema 2 / Aufgabe 4	254
„Streuspeicherung“	254
46115 / 2014 / Frühjahr / Thema 1 / Aufgabe 7	255
46115 / 2014 / Frühjahr / Thema 2 / Aufgabe 3	258
Frühjahr 2014 (46115) - Thema 2 Aufgabe 3	258
46115 / 2015 / Herbst / Thema 2 / Aufgabe 1	264
46115 / 2017 / Herbst / Thema 2 / Aufgabe 6	267
46115 / 2019 / Frühjahr / Thema 2 / Aufgabe 3	267
46115 / 2019 / Herbst / Thema 2 / Aufgabe 7	269
46115 / 2020 / Frühjahr / Thema 2 / Aufgabe 7	271
46115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 4	271
66112 / 2003 / Herbst / Thema 2 / Aufgabe 8	272
66112 / 2005 / Frühjahr / Thema 2 / Aufgabe 8	278
66112 / 2005 / Herbst / Thema 2 / Aufgabe 6	279
66114 / 2016 / Herbst / Thema 2 / Aufgabe 5	281
66115 / 2010 / Herbst / Thema 2 / Aufgabe 3	283
66115 / 2012 / Herbst / Thema 2 / Aufgabe 7	283
66115 / 2012 / Herbst / Thema 2 / Aufgabe 8	288
66115 / 2013 / Frühjahr / Thema 1 / Aufgabe 6	290
66115 / 2013 / Herbst / Thema 2 / Aufgabe 7	292
66115 / 2013 / Herbst / Thema 2 / Aufgabe 8	295
66115 / 2014 / Frühjahr / Thema 1 / Aufgabe 2	297
66115 / 2014 / Frühjahr / Thema 1 / Aufgabe 3	299
66115 / 2016 / Frühjahr / Thema 2 / Aufgabe 4	303
66115 / 2016 / Frühjahr / Thema 2 / Aufgabe 7	305
66115 / 2017 / Herbst / Thema 2 / Aufgabe 8	306
66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 8	311
66115 / 2018 / Herbst / Thema 1 / Aufgabe 5	314
66115 / 2019 / Frühjahr / Thema 2 / Aufgabe 1	317
66115 / 2019 / Herbst / Thema 1 / Aufgabe 7	317
66115 / 2019 / Herbst / Thema 2 / Aufgabe 7	319
66115 / 2019 / Herbst / Thema 2 / Aufgabe 9	322
66115 / 2020 / Frühjahr / Thema 2 / Aufgabe 10	323
66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 2	325
66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 5	329
66115 / 2020 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 1	332
66115 / 2020 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 3	334
66115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 3	337
66115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 5	339
66116 / 2013 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 3	341
66116 / 2015 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 3	343
66116 / 2015 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 3	343
66116 / 2016 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 5	344
66116 / 2017 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 2	347

66116 / 2020 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 6	348
---	-----

9 Graphen 349

66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6	349
Dijkstra-Algorithmus	349
66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6	350
Flugverbindung zwischen sieben Städten	350
66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6	352
Spannbaum	352
66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6	355
Algorithmus von Prim	355
66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6	355
Standardbeispiel TUM	355
66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6	356
Breiten- & Tiefensuche	356
Aufgabe 8 ¹	361
46114 / 2006 / Frühjahr / Thema 2 / Aufgabe 6	361
Aufgabe 6 (Graphrepräsentation)	361
46114 / 2008 / Herbst / Thema 1 / Aufgabe 2	362
Aufgabe 10: Graphen I	363
Frühjahr 2014 (46115) - Thema 1 Aufgabe 8	366
46115 / 2018 / Frühjahr / Thema 1 / Aufgabe 8	367
46115 / 2018 / Frühjahr / Thema 2 / Aufgabe 4	369
46115 / 2019 / Herbst / Thema 2 / Aufgabe 8	371
46115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 3	372
46115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 4	373
66112 / 2004 / Frühjahr / Thema 1 / Aufgabe 5	374
66115 / 2012 / Frühjahr / Thema 1 / Aufgabe 7	376
66115 / 2013 / Frühjahr / Thema 2 / Aufgabe 5	377
66115 / 2013 / Herbst / Thema 2 / Aufgabe 9	378
66115 / 2015 / Frühjahr / Thema 2 / Aufgabe 7	379
66115 / 2016 / Frühjahr / Thema 2 / Aufgabe 6	380
66115 / 2017 / Frühjahr / Thema 1 / Aufgabe 1	381
66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 10	384
66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 11	386
66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 9	387
66115 / 2019 / Herbst / Thema 2 / Aufgabe 8	391
66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 3	392
66115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 3	393
66115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 4	393
Sonstige	394
66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 1	394

¹<https://favtutor.com/blogs/depth-first-search-java>

Teil I

Algorithmen und Datenstrukturen
(AUD)

Rekursion

Feld-Invertierer

- (a) Erstellen Sie eine neue Klasse `ArrayInvertierer` mit einer rekursiven Methode, die den Inhalt eines ihr übergebenen 1D-Arrays gefüllt mit Strings invertiert. Auf diese Weise kann z. B. ein deutscher Satz im Array gespeichert und dann verkehrt herum ausgegeben werden.

Wichtig: Nicht das übergebene Array soll verändert werden, sondern ein Neues erstellt und von der Methode zurückgegeben werden.

Tipp: Sie dürfen dafür gerne auch rekursive Hilfsmethoden benutzen.

- (b) Implementieren Sie dann eine `main`-Methode, in der Sie zwei verschieden lange `String`-Arrays erzeugen und die Wortreihenfolge umkehren lassen. Das Ergebnis soll auf der Konsole ausgegeben werden und könnte z. B. wie folgt aussehen.

```
Den Satz
Ich find dich einfach klasse!
wuerde Meister Yoda so aussprechen:
klasse! einfach dich find Ich
```

```
Den Satz
Das war super einfach/schwer
wuerde Meister Yoda so aussprechen:
einfach/schwer super war Das
```

[optional] Wenn das ursprüngliche `String`-Array selbst verändert werden soll, braucht die rekursive Methode keine Rückgabe. Versuchen Sie, diese Aufgabe ohne das Nutzen einer Hilfsmethode zu lösen.

Lösungsvorschlag

```
public class ArrayInvertierer {

    /**
     * Invertiert das übergebene String Feld rekursiv durch Aufruf der Methode
     ↪ in
     * {@link invertiereRekursiv} mit dem jeweils aktuellen Arrayindex als
     * Startwert.
     *
     * @param quelle Das Feld, dessen Inhalt invertiert werden soll.
     * @param ziel    Hilfs-Feld.
     * @param index   aktueller Index
     */
    private static void invertiereRekursiv(String[] quelle, String[] ziel, int
    ↪ index) {
        if (index < quelle.length) {
```



```
        ziel[quelle.length - index - 1] = quelle[index];
        invertiereRekursiv(quelle, ziel, ++index);
    }
}

/**
 * Invertiert das übergebene String Feld rekursiv durch Aufruf der Methode
→ in
 * {@link invertiereRekursiv} mit dem Hilfsfeld und dem ersten Feldindex
→ als
 * Startwert.
 *
 * @param quelle Das Feld, dessen Inhalt invertiert werden soll.
 *
 * @return Ein neues Feld, das den umgekehrten Inhalt besitzt.
 */
private static String[] invertiereRekursiv(String[] quelle) {
    String[] ziel = new String[quelle.length];
    invertiereRekursiv(quelle, ziel, 0);
    return ziel;
}

/**
 * Die Lösung für die optionale Aufgaben. In situ bedeutet, dass kein neues
→ Feld
 * erzeugt wird.
 *
 * @param quelle Ein Feld mit Wörtern.
 * @param index Die Index-Nummer, die bearbeitet werden soll.
 */
private static void invertiereRekursivInSitu(String[] quelle, int index) {
    if (index < quelle.length / 2) {
        int gespiegelterIndex = quelle.length - 1 - index;
        String tmp = quelle[gespiegelterIndex];
        quelle[gespiegelterIndex] = quelle[index];
        quelle[index] = tmp;
        invertiereRekursivInSitu(quelle, ++index);
    }
}

/**
 * Hilfsmethode zur Ausgabe des String-Arrays in einem Satz.
 *
 * @param feld Ein Feld mit Wörtern.
 */
private static void gibFeldAus(String[] feld) {
    System.out.println(String.join(" ", feld));
}

/**
 * Lass Meister Yoda sprechen.
 *
 * @param satz Ein Feld mit Wörtern.
 * @param inSitu Bei wahr wird die Methode {@link invertiereRekursivInSitu}
```

```

*          verwendet. Achtung! Dadurch wird das Feld verändert.
*/
public static void lassYodaSprechen(String[] satz, boolean inSitu) {
    System.out.println("\nDen Satz");
    System.out.print(" ");
    gibFeldAus(satz);
    System.out.println("würde Meister Yoda so aussprechen:");
    System.out.print(" ");
    if (!inSitu) {
        gibFeldAus(invertiereRekursiv(satz));
    } else {
        invertiereRekursivInSitu(satz, 0);
        gibFeldAus(satz);
    }
}

public static void main(String[] args) {
    lassYodaSprechen(new String[] { "Ich", "find", "dich", "einfach",
        ↪ "klasse!" }, false);
    lassYodaSprechen(new String[] { "Das", "war", "super", "einfach/schwer"
        ↪ }, true);
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/ab_2/ArrayInvertierer.java](https://github.com/bschlangaul/aufgaben/aud/ab_2/ArrayInvertierer.java)

Fibonacci und Fakultät

```

public class Rekursion {

    public static int fak(int n) {
        if (n == 1) {
            return 1;
        }
        return n * fak(n - 1);
    }

    public static int fib(int n) {
        if (n <= 1) {
            return n;
        }
        return fib(n - 1) + fib(n - 2);
    }

    public static void main(String[] args) {
        System.out.println(fak(6));
        System.out.println(fib(6));
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/pu_1/Rekursion.java](https://github.com/bschlangaul/aufgaben/aud/pu_1/Rekursion.java)

Potenz

Gegeben ist folgende Methode.

```
public int function(int b, int e) {
    if (e == 1) {
        return b * 1;
    } else {
        e = e - 1;
        return b * function(b, e);
    }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/aufgaben/aud/ab_1/Rekursion.java

- (a) Beschreiben Sie kurz, woran man erkennt, dass es sich bei der gegebenen Methode um eine rekursive Methode handelt. Gehen Sie dabei auf wichtige Bestandteile der rekursiven Methode ein.

Lösungsvorschlag

Die Methode mit dem Namen `function` ruft sich in der letzten Code-Zeile selbst auf. Außerdem gibt es eine Abbruchbedingung (`if (e == 1) { return b * 1; }`), womit verhindert wird, dass die Rekursion unendlich weiter läuft.

- (b) Geben Sie die Rekursionsvorschrift für die Methode `function` an. Denken Sie dabei an die Angabe der Zahlenbereiche!

Lösungsvorschlag

$$\text{int function}(\text{int } b, \text{int } e) = \begin{cases} \text{return } b * 1, & \text{falls } e = 1. \\ \text{return } b * \text{function}(b, e-1), & \text{falls } e > 1. \end{cases}$$

- (c) Erklären Sie kurz, was die Methode `function` berechnet.

Lösungsvorschlag

Die Methode `function` berechnet die Potenz b^e .

Rater

```
import java.util.Scanner;

public class Rater {

    public static int rateRekursiv(int anfang, int ende) {
        int mitte;
        int antwort;
        int ergebnis;
        if (anfang == ende) {
            ergebnis = anfang; // Abbruchfall
        } else {
```

```

    mitte = (anfang + ende) / 2; // halbiere Intervall
    System.out.println("Ist Ihre Zahl groesser als " + mitte + "? 0: ja, 1: nein");
    Scanner scanner = new Scanner(System.in); // Antwort einlesen
    antwort = scanner.nextInt();
    scanner.close();
    if (antwort == 0) { // suche rechts
        ergebnis = rateRekursiv(mitte + 1, ende);
    } else {
        ergebnis = rateRekursiv(anfang, mitte); // suche links
    }
}
return ergebnis;
}

public static int rateIterativ(int anfang, int ende) {
    int mitte;
    int antwort;
    while (anfang != ende) {
        mitte = (anfang + ende) / 2; // halbiere das Intervall
        System.out.println("Ist Ihre Zahl groesser als " + mitte + "? 0:ja, 1: nein");
        Scanner scanner = new Scanner(System.in); // Antwort einlesen
        antwort = scanner.nextInt();
        scanner.close();
        if (antwort == 0) {
            anfang = mitte + 1; // suche rechts
        } else {
            ende = mitte; // suche links
        }
    }
    return anfang;
}

public static void main(String[] args) {
    System.out.println("Ihre Zahl ist " + rateIterativ(1, 500));
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/pu_1/Rater.java](https://github.com/bschlangaul/aufgaben/blob/master/aud/pu_1/Rater.java)

Sortieren

In dieser Aufgabe soll ein gegebenes Integer Array mit Hilfe von **Selection Sort** sortiert werden. Es soll eine iterative und eine rekursive Methode geschrieben werden. Verwenden Sie zur Implementierung jeweils die Methodenköpfe `selectionSortIterativ()` und `selectionSortRekursiv()`. Eine `swap`-Methode, die für ein gegebenes Array und zwei Indizes die Einträge an den jeweiligen Indizes des Arrays vertauscht, ist gegeben und muss nicht implementiert werden. Es müssen keine weiteren Methoden geschrieben werden!

Lösungsvorschlag

iterativ

```

public static void selectionSortIterativ(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }
        swap(arr, i, min);
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/sortier/SelectionSort.java](https://github.com/bschlangaul/aufgaben/aud/sortier/SelectionSort.java)

rekursiv

```

public static void selectionSortRekursiv(int[] arr, int i) {
    if (i == arr.length - 1) {
        return;
    }
    int min = i;
    for (int j = i + 1; j < arr.length; j++) {
        if (arr[j] < arr[min]) {
            min = j;
        }
    }
    swap(arr, i, min);
    selectionSortRekursiv(arr, i + 1);
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/sortier/SelectionSort.java](https://github.com/bschlangaul/aufgaben/aud/sortier/SelectionSort.java)

rekursives Backtracking

Folgende Methode soll das Feld a (garantiert der Länge $2n$ und beim ersten Aufruf von außen mit 0 initialisiert) mittels rekursivem Backtracking so mit Zahlen $1 \leq x \leq n$ befüllen, dass jedes x genau zweimal in a vorkommt und der Abstand zwischen den Vorkommen genau x ist. Sie soll genau dann `true` zurückgeben, wenn es eine Lösung gibt.

Beispiele:

- `fill(2, [])` → `false`
- `fill(3, [])` → `[3; 1; 2; 1; 3; 2]`
- `fill(4, [])` → `[4; 1; 3; 1; 2; 4; 3; 2]`

```

boolean fill (int n , int[] a) {
    if (n <= 0) {
        return true;
    }
    // TODO
    return false;
}

```



```
public static boolean fill(int n, int[] a) {
    if (n <= 0) {
        return true;
    }
    for (int i = 0; i < a.length - n - 1; i++) {
        // Zwischen i und j müssen genau n andere Zahlen sein
        int j = i + n + 1;
        if (a[i] == 0 && a[j] == 0) {
            a[i] = a[j] = n;
            if (fill(n - 1, a)) {
                return true;
            }
            a[i] = a[j] = 0;
        }
    }
    return false;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/muster/backtracking/RekursivesBacktracking.java](https://github.com/bschlangaul/aufgaben/aud/muster/backtracking/RekursivesBacktracking.java)

```
fill(0, []):
fill(1, []): false
fill(2, []): false
fill(3, []): 3 1 2 1 3 2
fill(4, []): 4 1 3 1 2 4 3 2
fill(5, []): false
fill(6, []): false
fill(7, []): 7 3 6 2 5 3 2 4 7 6 5 1 4 1
fill(8, []): 8 3 7 2 6 3 2 4 5 8 7 6 4 1 5 1
fill(9, []): false
fill(10, []): false
fill(11, []): 11 6 10 2 9 3 2 8 6 3 7 5 11 10 9 4 8 5 7 1 4 1
```

Kompletter Code

```
public class RekursivesBacktracking {

    public static boolean fill(int n, int[] a) {
        if (n <= 0) {
            return true;
        }
        for (int i = 0; i < a.length - n - 1; i++) {
            // Zwischen i und j müssen genau n andere Zahlen sein
            int j = i + n + 1;
            if (a[i] == 0 && a[j] == 0) {
                a[i] = a[j] = n;
                if (fill(n - 1, a)) {
                    return true;
                }
                a[i] = a[j] = 0;
            }
        }
    }
}
```

```

    return false;
}

public static void executeFill(int n) {
    int[] a = new int[n * 2];
    boolean result = fill(n, a);
    System.out.print("fill(" + n + ", []): ");
    if (result) {
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
    } else {
        System.out.print("false");
    }

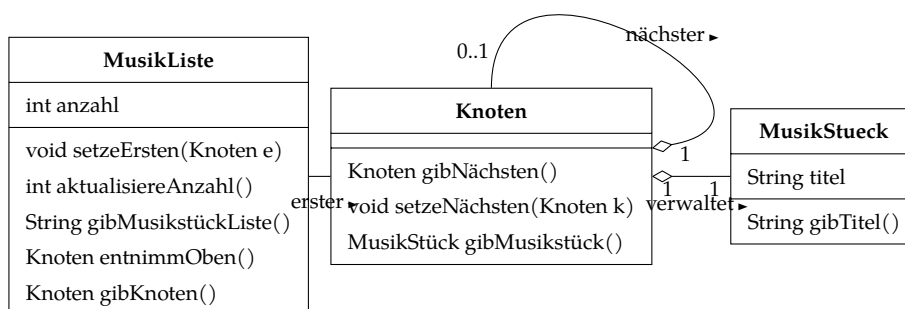
    System.out.println();
}

public static void main(String[] args) {
    executeFill(0);
    executeFill(1);
    executeFill(2);
    executeFill(3);
    executeFill(4);
    executeFill(5);
    executeFill(6);
    executeFill(7);
    executeFill(8);
    executeFill(9);
    executeFill(10);
    executeFill(11);
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/muster/backtracking/RekursivesBacktracking.java](https://github.com/src/main/java/org/bschlangaul/aufgaben/aud/muster/backtracking/RekursivesBacktracking.java)

Musikliste



Das Klassendiagramm zeigt den Aufbau einer Playlist.

(a) Implementieren Sie das Klassendiagramm.

Klasse „MusikListe“

```
public class MusikListe {
    private Knoten erster;
    private int anzahl;

    public MusikListe() {
        erster = null;
        anzahl = 0;
    }

    public void setzeErsten(Knoten knoten) {
        erster = knoten;
        aktualisiereAnzahl();
    }

    public int aktualisiereAnzahl() {
        if (erster == null) {
            anzahl = 0;
        } else {
            int zähler = 1;
            Knoten knoten = erster;
            while (!(knoten.gibNächsten() == null)) {
                knoten = knoten.gibNächsten();
                zähler = zähler + 1;
            }
            anzahl = zähler;
        }
        return anzahl;
    }

    public String gibMusikstückListe() {
        String ausgabe = " ";
        if (anzahl >= 1) {
            Knoten knoten = erster;
            ausgabe = knoten.gibMusikstück().gibTitel();
            for (int i = 1; i <= anzahl - 1; i++) {
                knoten = knoten.gibNächsten();
                ausgabe = ausgabe + " | " + knoten.gibMusikstück().gibTitel();
            }
        }
        return ausgabe;
    }

    public Knoten entnimmOben() {
        if (erster == null) {
            return erster;
        }
        Knoten alterKnoten = erster;
        erster = erster.gibNächsten();
        aktualisiereAnzahl();
        return alterKnoten;
    }
}
```

```
public Knoten gibKnoten(int position) {
    if ((position < 1) || (position > anzahl)) {
        System.out.println(" FEHLER ! ");
        return null;
    }
    Knoten knoten = erster;
    for (int i = 1; i <= position - 1; i++) {
        knoten = knoten.gibNächsten();
    }
    return knoten;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java)

Klasse „Knoten“

```
public class Knoten {
    private Knoten nächster;
    private Knoten vorheriger;
    private MusikStueck lied;

    public Knoten(MusikStueck lied) {
        nächster = null;
        this.lied = lied;
        vorheriger = null;
    }

    public Knoten gibNächsten() {
        return nächster;
    }

    public void setzeNächsten(Knoten nächsterKnoten) {
        nächster = nächsterKnoten;
    }

    public MusikStueck gibMusikstück() {
        return lied;
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/Knoten.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/Knoten.java)

Klasse „MusikStueck“

```
public class MusikStueck {
    private String titel;

    public MusikStueck(String titel) {
        this.titel = titel;
    }
}
```

```
public String gibTitel() {  
    return titel;  
}  
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikStueck.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/MusikStueck.java)

- (b) Schreiben Sie eine Testklasse, in der Sie eine Playlist mit mindestens vier Liedern erstellen.

Lösungsvorschlag

```
private MusikListe macheListe() {  
    MusikListe liste = new MusikListe();  
  
    MusikStueck stueck1 = new MusikStueck("Hangover");  
    MusikStueck stueck2 = new MusikStueck("Roar");  
    MusikStueck stueck3 = new MusikStueck("On the Floor");  
    MusikStueck stueck4 = new MusikStueck("Whistle");  
  
    Knoten platz1 = new Knoten(stueck1);  
    Knoten platz2 = new Knoten(stueck2);  
    Knoten platz3 = new Knoten(stueck3);  
    Knoten platz4 = new Knoten(stueck4);  
  
    liste.setzeErsten(platz1);  
    platz1.setzeNächsten(platz2);  
    platz2.setzenVorherigen(platz1);  
    platz2.setzeNächsten(platz3);  
    platz3.setzenVorherigen(platz2);  
    platz3.setzeNächsten(platz4);  
    platz4.setzenVorherigen(platz3);  
    liste.aktualisiereAnzahl();  
    return liste;  
}
```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListeTest.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/MusikListeTest.java)

Aufgabe 2

Die Playlist aus Aufgabe 1 soll nun erweitert werden. Aktualisieren Sie Ihren Code entsprechend!

- (a) Bisher wurde das erste Element der Musikliste in einer öffentlich sichtbaren Variable gespeichert, dies ist jedoch nicht sinnvoll. Erstellen Sie eine Methode `setzeErsten()`, mit der anstatt dessen die Liste der erstellten Musikstücke angesprochen werden kann.

Lösungsvorschlag

```
public void setzeErsten(Knoten knoten) {  
    erster = knoten;  
    aktualisiereAnzahl();  
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java](#)

- (b) Außerdem wird ein Attribut `anzahl` benötigt, dass mit Hilfe der Methode `aktualisiereAnzahl()` auf dem aktuellen Stand gehalten werden kann.

Lösungsvorschlag

```
private int anzahl;
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java](#)

- (c) Eine weitere Methode `gibMusikstückListe()` soll die Titel aller Lieder in der Liste als `String` zurückgeben.

Lösungsvorschlag

```
public String gibMusikstückListe() {
    String ausgabe = " ";
    if (anzahl >= 1) {
        Knoten knoten = erster;
        ausgabe = knoten.gibMusikstück().gibTitel();
        for (int i = 1; i <= anzahl - 1; i++) {
            knoten = knoten.gibNächsten();
            ausgabe = ausgabe + " | " + knoten.gibMusikstück().gibTitel();
        }
    }
    return ausgabe;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java](#)

- (d) Mit `entnimmOben()` soll der erste Titel aus der Liste entnommen werden können.

Lösungsvorschlag

```
public Knoten entnimmOben() {
    if (erster == null) {
        return erster;
    }
    Knoten alterKnoten = erster;
    erster = erster.gibNächsten();
    aktualisiereAnzahl();
    return alterKnoten;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java](#)

- (e) Es soll der Titel des Musikstücks ermittelt werden, das an einer bestimmten Position in der Musikliste abgespeichert ist. Implementieren Sie dazu die Methode `gibKnoten()`.

```

public Knoten gibKnoten(int position) {
    if ((position < 1) || (position > anzahl)) {
        System.out.println(" FEHLER ! ");
        return null;
    }
    Knoten knoten = erster;
    for (int i = 1; i <= position - 1; i++) {
        knoten = knoten.gibNächsten();
    }
    return knoten;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java)

- (f) Die Musikliste soll nun in eine doppelt verkettete Liste umgebaut werden. Fügen Sie entsprechende Attribute, getter- und setter-Methoden hinzu.

```

private Knoten vorheriger;

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/Knoten.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/Knoten.java)

```

public MusikStueck gibMusikstück() {
    return lied;
}

public Knoten gibVorherigen() {
    return vorheriger;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/Knoten.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/Knoten.java)

- (g) Testen Sie die Funktionalität der neuen Methoden in Ihrer Testklasse.

```

import static org.junit.Assert.*;
import org.junit.Test;

public class MusikListeTest {

    private MusikListe macheListe() {
        MusikListe liste = new MusikListe();

        MusikStueck stueck1 = new MusikStueck("Hangover");
        MusikStueck stueck2 = new MusikStueck("Roar");
        MusikStueck stueck3 = new MusikStueck("On the Floor");
        MusikStueck stueck4 = new MusikStueck("Whistle");

        Knoten platz1 = new Knoten(stueck1);
        Knoten platz2 = new Knoten(stueck2);
        Knoten platz3 = new Knoten(stueck3);
    }
}

```

```

    Knoten platz4 = new Knoten(stueck4);

    liste.setzeErsten(platz1);
    platz1.setzeNächsten(platz2);
    platz2.setzenVorherigen(platz1);
    platz2.setzeNächsten(platz3);
    platz3.setzenVorherigen(platz2);
    platz3.setzeNächsten(platz4);
    platz4.setzenVorherigen(platz3);
    liste.aktualisiereAnzahl();
    return liste;
}

@Test
public void methodeGibMusikstückListe() {
    MusikListe liste = macheListe();
    assertEquals("Hangover | Roar | On the Floor | Whistle",
        ↪ liste.gibMusikstückListe());
}

@Test
public void methodeEntnimmOben() {
    MusikListe liste = macheListe();
    assertEquals("Hangover", liste.entnimmOben().gibMusikstück().gibTitel());
    assertEquals("Roar | On the Floor | Whistle",
        ↪ liste.gibMusikstückListe());
}

@Test
public void methodeZähleEinträge() {
    MusikListe liste = macheListe();
    assertEquals(4, liste.zähleEinträge());
}
}

```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListeTest.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/MusikListeTest.java)

Rekursion

Die Anzahl der Titel in der Musikliste aus Aufgabe 1 kann auch unter Verwendung einer rekursiven Methode ermittelt werden. Implementieren Sie eine Methode `zähleEinträge()`, die analog zu `aktualisiereAnzahl()` angibt, wie viele Titel in der Musikliste gespeichert sind, dies aber rekursiv ermittelt! Testen Sie diese Methode in der Testklasse. Hinweis: Um für die gesamte Musikliste aufgerufen werden zu können, muss diese Methode in der Musikliste selbst und auch in der Klasse Knoten existieren!

Lösungsvorschlag

Klasse „MusikListe“

```

public int zähleEinträge() {
    if (erster == null) {

```

```

    return 0;
}
return erster.zähleEinträge();
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/MusikListe.java)

Klasse „Knoten“

```

public int ZähleEinträge() {
    if (this.gibNächsten() == null) {
        return 1;
    } else {
        return this.gibNächsten().ZähleEinträge() + 1;
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/musikliste/Knoten.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/Knoten.java)

Klasse „TestKlasse“

```

@Test
public void methodeZähleEinträge() {
    MusikListe liste = macheListe();
    assertEquals(4, liste.ZähleEinträge());
}
}

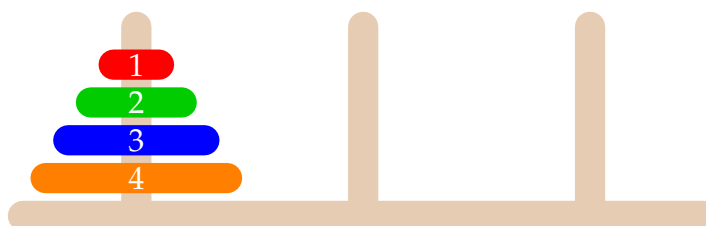
```

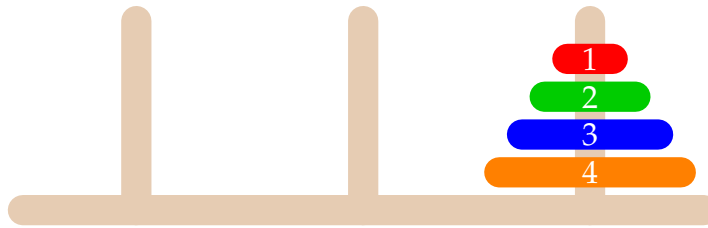
Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/aufgaben/aud/listen/musikliste/MusikListeTest.java](https://github.com/bschlangaul/aufgaben/aud/listen/musikliste/MusikListeTest.java)

Kellerspeicher: Türme von Hanoi

Betrachten wir das folgende Spiel (Türme von Hanoi), das aus drei Stäben 1, 2 und 3 besteht, die senkrecht im Boden befestigt sind. Weiter gibt es n kreisförmige Scheiben mit einem Loch im Mittelpunkt, so dass man sie auf die Stäbe stecken kann. Dabei haben die Scheiben verschiedene Radien, alle sind unterschiedlich groß. Zu Beginn stecken alle Scheiben auf dem Stab 1, wobei immer eine kleinere auf einer größeren liegt. Das Ziel des Spiels ist es nun, die Scheiben so umzuordnen, dass sie in der gleichen Reihenfolge auf dem Stab 3 liegen. Dabei darf immer nur eine Scheibe bewegt werden und es darf nie eine größere auf einer kleineren Scheibe liegen. Stab 2 darf dabei als Hilfsstab verwendet werden.

Ein Beispiel für 4 Scheiben finden Sie in folgendem Bild:





- (a) Ein **Element** hat immer einen Wert (Integer) und kennt das Nachfolgende **Element**, wobei immer nur das jeweilige **Element** auf seinen Wert und seinen Nachfolger zugreifen darf.

Lösungsvorschlag

```
public class Element {
    private int wert;
    private Element nächstes;

    public Element(int wert) {
        this.wert = wert;
        nächstes = null;
    }

    public int gibWert() {
        return wert;
    }

    public Element gibNächstes() {
        return nächstes;
    }

    public void setzeNächstes(Element next) {
        this.nächstes = next;
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/hanoi/Element.java](https://github.com/bschlangaul/aufgaben/aud/listen/hanoi/Element.java)

- (b) Ein **Turm** ist einem Stack (Kellerspeicher) nachempfunden und kennt somit nur das erste Element. Hinweis: Beachten Sie, dass nur kleinere Elemente auf den bisherigen Stack gelegt werden können

Lösungsvorschlag

```
public class Turm {
    private Element oben;

    public Turm() {
        oben = null;
    }

    public Element gibOben() {
        return oben;
    }

    public void legeDrauf(Element e) {
```

```

    if (oben == null) {
        oben = e;
    } else if (oben.gibWert() == e.gibWert()) {
        System.out.println("Fehler! schieben sind gleich gross");
    } else if (oben.gibWert() < e.gibWert()) {
        ↪ System.out.println("Fehler! Größere Scheiben dürfen nicht auf kleinere");
    } else {
        e.setzeNächstes(oben);
        oben = e;
    }
}

public Element nimmHerunter() {
    if (oben == null) {
        System.out.println("Turm ist bereits leer!");
        return null;
    } else {
        Element merker = new Element(oben.gibWert());
        oben = oben.gibNächstes();
        return merker;
    }
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/hanoi/Turm.java](https://github.com/bschlangaul/aufgaben/aud/listen/hanoi/Turm.java)

- (c) In der Klasse `Hanoi` müssen Sie nur die Methode `public void hanoi (int n, Turm quelle, Turm ziel, Turm hilfe)` implementieren. Die anderen Methoden sind zur Veranschaulichung des Spiels! Entwerfen Sie eine rekursive Methode die einen Turm der Höhe n vom Stab `quelle` auf den Stab `ziel` transportiert und den Stab `hilfe` als Hilfsstab verwendet.

Lösungsvorschlag

```

public class Hanoi {
    private int anzahlScheiben = 3;
    private int[] turm0helper, turm1helper, turm2helper;
    Turm turm0 = new Turm();
    Turm turm1 = new Turm();
    Turm turm2 = new Turm();

    public Hanoi(int anzahlScheiben) {
        if (anzahlScheiben <= 0) {
            ↪ System.out.println("Zu wenige Scheiben. Defaultfall (anzahlScheiben = 3) wird angewendet");
        } else {
            this.anzahlScheiben = anzahlScheiben;
        }
        for (int i = this.anzahlScheiben; i > 0; i--) {
            turm0.legeDrauf(new Element(i));
        }
        zeigeTürme();
    }
}

```

```
/**
 * @param n      Höhe des Turms (Anzahl der Scheiben).
 * @param quelle Der Ausgangs-Turm.
 * @param ziel   Der Ziel-Turm.
 * @param hilfe  Der Hilfs-Turm in der Mitte.
 */
public void hanoi(int n, Turm quelle, Turm ziel, Turm hilfe) {
    if (n == 1) {
        verschiebeScheibe(quelle, ziel);
        System.out.println("Fertig!");
    } else {
        hanoi(n - 1, quelle, hilfe, ziel);
        verschiebeScheibe(quelle, ziel);
        hanoi(n - 1, hilfe, ziel, quelle);
    }
}

public void verschiebeScheibe(Turm quelle, Turm ziel) {
    // Bereinigt die Konsole
    System.out.print('\u000C');

    if (quelle == ziel) {
        System.out.println("Quelle ist gleich dem Ziel!");
        return;
    }
    ziel.legeDrauf(quelle.nimmHerunter());
    zeigeTürme();
}

/**
 * Zeige die drei Türme in der Konsole
 */
public void zeigeTürme() {
    Element merker = turm0.gibOben();
    int zeiger0 = 0;
    int zeiger1 = 0;
    int zeiger2 = 0;
    turm0helper = new int[this.anzahlScheiben];
    turm1helper = new int[this.anzahlScheiben];
    turm2helper = new int[this.anzahlScheiben];
    int i = 0;
    while (merker != null) {
        turm0helper[i++] = merker.gibWert();
        merker = merker.gibNächstes();
        zeiger0++;
    }
    merker = turm1.gibOben();
    i = 0;
    while (merker != null) {
        turm1helper[i++] = merker.gibWert();
        merker = merker.gibNächstes();
        zeiger1++;
    }
}
```

```

merker = turm2.gibOben();
i = 0;
while (merker != null) {
    turm2helper[i++] = merker.gibWert();
    merker = merker.gibNächstes();
    zeiger2++;
}

int help0 = zeiger0 % this.anzahlScheiben;
int help1 = zeiger1 % this.anzahlScheiben;
int help2 = zeiger2 % this.anzahlScheiben;

for (int j = 0; j < turm0helper.length; j++) {
    System.out.print(turm0helper[help0++] + " " + turm1helper[help1++] +
        ↪ " " + turm2helper[help2++]);
    System.out.println();
    help0 = help0 % this.anzahlScheiben;
    help1 = help1 % this.anzahlScheiben;
    help2 = help2 % this.anzahlScheiben;
}

try {
    Thread.sleep(500);
} catch (InterruptedException e) {
    // ignore
}

public static void main(String[] args) {
    Hanoi h = new Hanoi(5);
    h.hanoi(5, h.turm0, h.turm2, h.turm1);
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/hanoi/Hanoi.java](https://github.com/bschlangaul/aufgaben/aud/listen/hanoi/Hanoi.java)

46115 / 2014 / Frühjahr / Thema 2 / Aufgabe 4

Aufgabe 4

Für Binomialkoeffizienten $\binom{n}{k}$ gelten neben den grundlegenden Beziehungen $\binom{n}{0} = 1$ und $\binom{n}{n} = 1$ auch folgende Formeln:

Exkurs: Binomialkoeffizient

Der Binomialkoeffizient ist eine mathematische Funktion, mit der sich eine der Grundaufgaben der Kombinatorik lösen lässt. Er gibt an, auf wie viele verschiedene Arten man k bestimmte Objekte aus einer Menge von n verschiedenen Objekten auswählen kann (ohne Zurücklegen, ohne Beachtung der Reihenfolge). Der Binomialkoeffizient ist also die Anzahl der k -elementigen Teilmengen einer n -elementigen Menge.^a

^a<https://de.wikipedia.org/wiki/Binomialkoeffizient>

A $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$

B $\binom{n}{k} = \binom{n-1}{k-1} \cdot \frac{n}{k}$

- (a) Implementieren Sie unter Verwendung von Beziehung (A) eine rekursive Methode `binRek(n, k)` zur Berechnung des Binomialkoeffizienten in einer objektorientierten Programmiersprache oder entsprechendem Pseudocode!

Lösungsvorschlag

Zuerst verwandeln wir die Beziehung (A) geringfügig um, indem wir n durch $n - 1$ ersetzen:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

```
public static int binRek(int n, int k) {
    if (k == 0 || k == n) {
        return 1;
    } else {
        return binRek(n - 1, k - 1) + binRek(n - 1, k);
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java)

- (b) Implementieren Sie unter Verwendung von Beziehung (B) eine iterative Methode `binIt(n, k)` zur Berechnung des Binomialkoeffizienten in einer objektorientierten Programmiersprache oder entsprechendem Pseudocode!

Lösungsvorschlag

```
public static int binIt(int n, int k) {
    // Das Ergebnis wird als Kommazahl deklariert, da nicht alle
    // Zwischenergebnisse ganze Zahlen sind.
    double ergebnis = 1;
    while (k > 0) {
        ergebnis = ergebnis * n / k;
        n--;
        k--;
    }
    // Vor dem Zurückgeben kann das Ergebnis nun in eine ganze Zahl
    // umgewandelt werden.
    return (int) ergebnis;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java)

- (c) Geben Sie die Laufzeitkomplexität der Methoden `binRek(n, k)` und `binIt(n, k)` aus den vorhergehenden beiden Teilaufgaben in O-Notation an!

Komplette Java-Klasse

```
/**
 * <a href="https://www.studon.fau.de/file2889270_download.html">Angabe:
 * ↪ PUE_AUD_WH.pdf</a>
 * <a href="https://www.studon.fau.de/file3081306_download.html">Lösung:
 * ↪ PUE_AUD_WH_Lsg.pdf</a>
 */
public class Binomialkoeffizient {

    /**
     * Berechnet rekursiv den Binominalkoeffizienten „n über k“. Dabei muss gelten:
     * n &#x3E;= 0, k &#x3E;= 0 und n &#x3E;= k.
     *
     * @param n Ganzzahl n
     * @param k Ganzzahl k
     *
     * @return Eine Ganzzahl.
     */
    public static int binRek(int n, int k) {
        if (k == 0 || k == n) {
            return 1;
        } else {
            return binRek(n - 1, k - 1) + binRek(n - 1, k);
        }
    }

    /**
     * Berechnet iterativ den Binominalkoeffizienten „n über k“. Dabei muss gelten:
     * n &#x3E;= 0, k &#x3E;= 0 und n &#x3E;= k.
     *
     * @param n Ganzzahl n
     * @param k Ganzzahl k
     *
     * @return Eine Ganzzahl.
     */
    public static int binIt(int n, int k) {
        // Das Ergebnis wird als Kommazahl deklariert, da nicht alle
        // Zwischenergebnisse ganze Zahlen sind.
        double ergebnis = 1;
        while (k > 0) {
            ergebnis = ergebnis * n / k;
            n--;
            k--;
        }
        // Vor dem Zurückgeben kann das Ergebnis nun in eine ganze Zahl
        // umgewandelt werden.
        return (int) ergebnis;
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java)

Test

```
import static org.junit.Assert.assertEquals;
```

```
import org.junit.Test;

public class BinomialkoeffizientTest {

    public void testeRek(int n, int k, int ergebnis) {
        assertEquals(ergebnis, Binomialkoeffizient.binIt(n, k));
    }

    public void testeIt(int n, int k, int ergebnis) {
        assertEquals(ergebnis, Binomialkoeffizient.binIt(n, k));
    }

    public void teste(int n, int k, int ergebnis) {
        testeRek(n, k, ergebnis);
        testeIt(n, k, ergebnis);
    }

    @Test
    public void teste() {
        teste(0, 0, 1);

        teste(1, 0, 1);
        teste(1, 1, 1);

        teste(2, 0, 1);
        teste(2, 1, 2);
        teste(2, 2, 1);

        teste(3, 0, 1);
        teste(3, 1, 3);
        teste(3, 2, 3);
        teste(3, 3, 1);

        teste(4, 0, 1);
        teste(4, 1, 4);
        teste(4, 2, 6);
        teste(4, 3, 4);
        teste(4, 4, 1);
    }
}
```

Code-Beispiel auf Github ansehen: src/test/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/BinomialkoeffizientTest.java

66115 / 2014 / Frühjahr / Thema 1 / Aufgabe 1

- (a) Gegeben sei die Methode `BigInteger lfBig(int n)` zur Berechnung der eingeschränkten Linksfakultät:

$$!n := \begin{cases} n!(n-1) - (n-1)!(n-2) & \text{falls } 1 < n < 32767 \\ 1 & \text{falls } n = 1 \\ 0 & \text{sonst} \end{cases}$$

```
import java.math.BigInteger;
import static java.math.BigInteger.ZERO;
```

```
import static java.math.BigInteger.ONE;

public class LeftFactorial {

    BigInteger sub(BigInteger a, BigInteger b) {
        return a.subtract(b);
    }

    BigInteger mul(BigInteger a, BigInteger b) {
        return a.multiply(b);
    }

    BigInteger mul(int a, BigInteger b) {
        return mul(BigInteger.valueOf(a), b);
    }

    // returns the left factorial !n
    BigInteger lfBig(int n) {
        if (n <= 0 || n >= Short.MAX_VALUE) {
            return ZERO;
        } else if (n == 1) {
            return ONE;
        } else {
            return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
        }
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java)

Implementieren Sie unter Verwendung des Konzeptes der *dynamischen Programmierung* die Methode `BigInteger dp(int n)`, die jede n auch bei mehrfachem Aufrufen mit dem gleichen Parameter höchstens einmal rekursiv berechnet. Sie dürfen der Klasse `LeftFactorial` genau ein Attribut beliebigen Datentyps hinzufügen und die in `lfBig(int)` verwendeten Methoden und Konstanten ebenfalls nutzen.

Lösungsvorschlag

Wir führen ein Attribut mit dem Namen `store` ein und erzeugen ein Feld vom Typ `BigInteger` mit der Länge $n + 1$. Die Länge des Feld $n + 1$ hat den Vorteil, dass nicht ständig $n - 1$ verwendet werden muss, um den gewünschten Wert zu erhalten.

In der untenstehenden Implementation gibt es zwei Methoden mit dem Namen `dp`. Die untenstehende Methode ist nur eine Hüllmethode, mit der nach außen hin die Berechnung gestartet und das `store`-Feld neu gesetzt wird. So ist es möglich `dp()` mehrmals hintereinander mit verschiedenen Werten aufzurufen (siehe `main()`-Methode).

```
BigInteger[] store;

BigInteger dp(int n, BigInteger[] store) {
    if (n > 1 && store[n] != null) {
        return store[n];
    }
}
```



```

    }
    if (n <= 0 || n >= Short.MAX_VALUE) {
        return ZERO;
    } else if (n == 1) {
        return ONE;
    } else {
        BigInteger result = sub(mul(n, dp(n - 1, store)), mul(n - 1, dp(n - 2,
        ↪ store)));
        store[n] = result;
        return result;
    }
}

BigInteger dp(int n) {
    store = new BigInteger[n + 1];
    return dp(n, store);
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java)

- (b) Betrachten Sie nun die Methode `lfLong(int)` zur Berechnung der vorangehend definierten Linksfakultät ohne obere Schranke. Nehmen Sie im Folgenden an, dass der Datentyp `long` unbeschränkt ist und daher kein Überlauf auftritt.

```

long lfLong(int n) {
    if (n <= 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java)

Beweisen Sie *formal* mittels *vollständiger Induktion*:

$$\forall n \geq 0 : \text{lfLong}(n) \equiv \sum_{k=0}^{n-1} k!$$

Lösungsvorschlag

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. _____

$$n = 1 \Rightarrow \text{lfLong}(1) = 1 = \sum_{k=0}^{n-1} k! = 0! = 1$$

$$\begin{aligned}n = 2 &\Rightarrow \text{lfLong}(2) \\&= (n + 1) \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! \\&= 2 * \text{lfLong}(1) - 1 * \text{lfLong}(0) \\&= 2 \\&= \sum_{k=0}^1 k! \\&= 1! + 0! \\&= 1 + 1 \\&= 2\end{aligned}$$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. _____

$$\text{lfLong}(n) = \sum_{k=0}^{n-1} k!$$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss.

$$\begin{aligned}
A(n+1) &= \text{lfLong}(n+1) \\
&= (n+1) * \text{lfLong}(n) - n * \text{lfLong}(n-1) \\
&= (n+1) \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{(n-1)-1} k! && \text{Formel eingesetzt} \\
&= (n+1) \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{subtrahiert} \\
&= n \sum_{k=0}^{n-1} k! + \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{ausmultipliziert mit } (n+1) \\
&= \sum_{k=0}^{n-1} k! + n \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{Reihenfolge der Terme geändert} \\
&= \sum_{k=0}^{n-1} k! + n \left((n-1)! + \sum_{k=0}^{n-2} k! \right) - n \sum_{k=0}^{n-2} k! && (n-1)! \text{ aus Summenzeichen entfernt} \\
&= \sum_{k=0}^{n-1} k! + n \left((n-1)! + \sum_{k=0}^{n-2} k! - \sum_{k=0}^{n-2} k! \right) && \text{Distributivgesetz } ac - bc = (a-b)c \\
&= \sum_{k=0}^{n-1} k! + n(n-1)! && +\Sigma - \Sigma = 0 \\
&= \sum_{k=0}^{n-1} k! + n! && \text{Fakultät erhöht} \\
&= \sum_{k=0}^n k! && \text{Element zum Summenzeichen hinzugefügt} \\
&= \sum_{k=0}^{(n+1)-1} k! && \text{mit } (n+1) \text{ an der Stelle von } n
\end{aligned}$$

66116 / 2014 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 5

Lösen Sie folgende Aufgabe in einer objektorientierten Programmiersprache Ihrer Wahl. Ein Regal habe 5 Fächer, in die je 30 Disks passen. Das Regal beinhaltet DVDs, CDs und BDs der Genres *Musik*, *Action*, *Komödie*, *Thriller* und *Fantasy*. Jede Disk ist mit 1 bis 10 bewertet, wobei 10 für sehr gut steht.

- Deklarieren Sie einen Aufzählungsdatentyp für den **Typ** der Disk. Deklarieren Sie einen weiteren Aufzählungsdatentyp für das **Genre** der Disk.

Lösungsvorschlag

```
public enum Typ {
```

```
DVD, CD, BD
}
```

Feld (Array)

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Typ.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Typ.java)

```
public enum Genre {
    MUSIK, ACTION, KOMOEDIE, THRILLER, FANTASY
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Genre.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Genre.java)

- Deklarieren Sie eine Klasse **Disk**, die den Typ, das Genre, die Bewertung, sowie den Titel der Disk speichert.

Lösungsvorschlag

```
public class Disk {
    private Typ typ;
    private Genre genre;
    private int bewertung;
    private String titel;

    /**
     * Erzeuge eine neue Disk. Die Bewertung wird automatisch mit Hilfe des
     * → Genres
     * berechnet.
     *
     * @param typ    Typ der Disk
     * @param genre  Genre der Disk
     * @param titel  Titel der Disk
     */
    public Disk(Typ typ, Genre genre, String titel) {
        this.typ = typ;
        this.genre = genre;
        this.titel = titel;
        this.bewertung = (int) erstelleStdBewertung(this.genre);
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Disk.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Disk.java)

- Deklarieren Sie ein Array **Regal**, mit den Abmessungen des oben genannten Regals, das Disks enthält. Initialisieren Sie das Array als leeres Regal.

Lösungsvorschlag

```
public class Regal {
    private Disk[] [] regal = new Disk[5][30];

    /**
     * Erzeuge eine neues (leeres) Regal.
     */
    public Regal() {
        for (int i = 0; i < regal.length; i++) {
            for (int j = 0; j < regal[i].length; j++) {
                regal[i][j] = null;
            }
        }
    }
}
```

```

    }
  }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Regal.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Regal.java)

- Initialisieren Sie die Bewertung einer Disk. Schreiben sie dazu eine rekursive Methode `erstelleStdBewertung`, der ein Genre einer Disk übergeben wird und die die Bewertungen für alle Disks nach folgenden Regeln bezüglich des Genres vergibt:

- Eine Disk mit Genre Musik wird mit einer 3 bewertet.
- Komödien werden mit 2 bewertet.
- Thriller werden mit dem zweifachen einer Komödie bewertet
- Ein Fantasy-Film wird mit dem 1,5 fachen eines Thrillers bewertet.
- Ein Actionfilm wird wie ein Thriller bewertet.

Lösungsvorschlag

```

public double erstelleStdBewertung(Genre genre) {
    if (genre.equals(Genre.MUSIK)) {
        return 3;
    } else if (genre.equals(Genre.KOMOEDIE)) {
        return 2;
    } else if (genre.equals(Genre.THRILLER)) {
        return 2 * erstelleStdBewertung(Genre.KOMOEDIE);
    } else if (genre.equals(Genre.FANTASY)) {
        return 1.5 * erstelleStdBewertung(Genre.THRILLER);
    } else if (genre.equals(Genre.ACTION)) {
        return erstelleStdBewertung(Genre.THRILLER);
    } else {
        return 0;
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Disk.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Disk.java)

- Schreiben Sie eine Methode `mittlereBewertung`, die die mittlere Bewertung der Disks im Regal berechnet.

Lösungsvorschlag

```

public double mittlereBewertung() {
    int anzahl = 0;
    int bewertungspunkteGesamt = 0;
    for (int i = 0; i < regal.length; i++) {
        for (int j = 0; j < regal[i].length; j++) {
            if (regal[i][j] != null) {
                anzahl++;
                bewertungspunkteGesamt += regal[i][j].gibBewertung();
            }
        }
    }
}

```

```

    if (anzahl == 0) {
        // Falls das Regal komplett leer ist.
        return 0;
    }
    return bewertungspunkteGesamt / anzahl;
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Regal.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2014/herbst/regal/Regal.java)

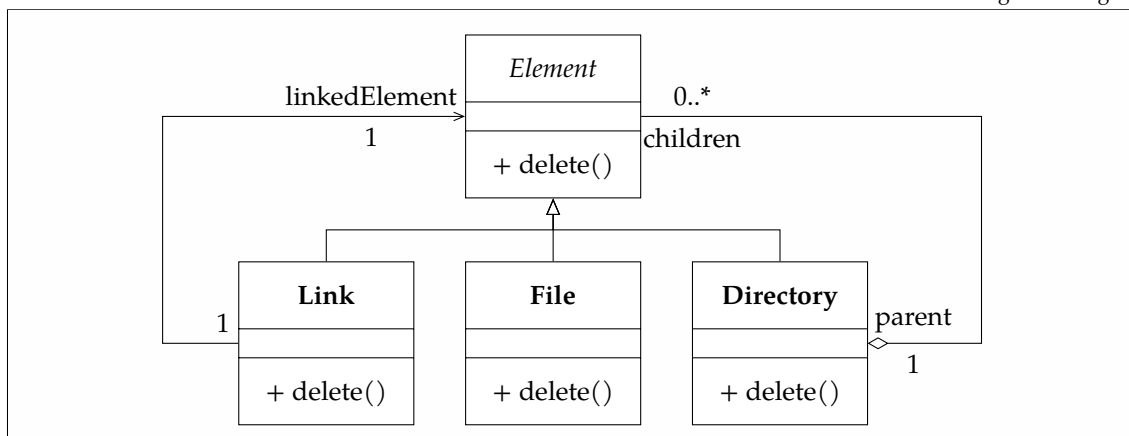
Entwurfsmuster
Kompositum (Composite)
Klassendiagramm
Implementierung in Java

66116 / 2019 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 1

Hierarchische Dateisysteme bestehen aus den FileSystemElements Ordner, Dateien und Verweise. Ein Ordner kann seinerseits Ordner, Dateien und Verweise beinhalten; jedem Ordner ist bekannt, welche Elemente (children) er enthält. Mit Ausnahme des Root-Ordners auf der obersten Hierarchieebene ist jeder Ordner, jede Datei und jeder Verweis Element eines Elternordners. Jedem Element ist bekannt, was sein Elternordner ist (parent). Ein Verweis verweist auf einen Verweis, eine Datei oder einen Ordner (link). Wenn ein Ordner gelöscht wird, werden alle seine Bestandteile ggf. rekursiv ebenfalls gelöscht. Sie dürfen die Lösungen für Aufgabenteil b) und c) in einem gemeinsamen Code kombinieren.

- (a) Modellieren Sie diesen Sachverhalt mit einem UML-Klassendiagramm. Benennen Sie die Rollen von Assoziationen und geben Sie alle Kardinalitäten an. Ihre Lösung soll mindestens eine sinnvolle Spezialisierungsbeziehung enthalten.

Lösungsvorschlag



- (b) Implementieren Sie das Klassendiagramm als Java- oder C++-Programm. Jedes Element des Dateisystems soll mindestens über ein Attribut `name` verfügen. Übergeben Sie den Elternordner jedes Elements als Parameter an den Konstruktor; der Elternordner des Root-Ordners kann dabei als `null` implementiert werden. Dokumentieren Sie Ihren Code.

a

```
public abstract class Element {
```

```
protected String name;

protected Element parent;

protected Element(String name, Element parent) {
    this.name = name;
    this.parent = parent;
}

public String getName() {
    return name;
}

public abstract void delete();

public abstract boolean isDirectory();

public abstract void addChild(Element child);
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Element.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Element.java)

```
import java.util.ArrayList;
import java.util.List;

public class Directory extends Element {
    private List<Element> children;

    public Directory(String name, Element parent) {
        super(name, parent);
        children = new ArrayList<Element>();
        if (parent != null)
            parent.addChild(this);
    }

    public void delete() {
        System.out.println("The directory " + name +
            ↪ " was deleted and it's children were also deleted.");
        for (int i = 0; i < children.size(); i++) {
            Element child = children.get(i);
            child.delete();
        }
    }

    public void addChild(Element child) {
        children.add(child);
    }

    public boolean isDirectory() {
        return true;
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Directory.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Directory.java)


```
public class File extends Element {

    public File(String name, Element parent) {
        super(name, parent);
        parent.addChild(this);
    }

    public void delete() {
        System.out.println("The File " + name + " was deleted.");
    }

    public boolean isDirectory() {
        return false;
    }

    /**
     * Eine Datei kann keine Kinder haben. Deshalb eine Methode mit leerem
     * Methodenrumpf.
     */
    public void addChild(Element child) {
    }

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/File.java

```
public class Link extends Element {

    private Element linkedElement;

    public Link(String name, Element parent, Element linkedElement) {
        super(name, parent);
        this.linkedElement = linkedElement;
        parent.addChild(this);
    }

    public void delete() {
        System.out.println("The Symbolic Link " + name + " was deleted.");
        linkedElement.delete();
        System.out.println("The linked element " + name + " was deleted too.");
    }

    public void addChild(Element child) {
        if (linkedElement.isDirectory())
            linkedElement.addChild(child);
    }

    public boolean isDirectory() {
        return linkedElement.isDirectory();
    }

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Link.java

^aTU Darmstadt: Dr. Michael Eichberg - Case Study Using the Composite and Proxy Design Patterns

- (c) Ordnen Sie eine Methode `delete`, die Dateien, Ordner und Verweise rekursiv löscht, einer oder mehreren geeigneten Klassen zu und implementieren Sie sie. Zeigen Sie die Löschung jedes Elements durch eine Textausgabe von `name` an. `toString()` müssen Sie dabei nicht implementieren. Gehen Sie davon aus, dass Verweis- und Ordnerstrukturen azyklisch sind und dass jedes Element des Dateisystems höchstens einmal existiert. Wenn ein Verweis gelöscht wird, wird sowohl der Verweis als auch das verwiesene Element bzw. transitiv die Kette der verwiesenen Elemente gelöscht. Bedenken Sie, dass die Löschung eines Elements immer auch Konsequenzen für den dieses Element beinhaltenden Ordner hat. Es gibt keinen Punktabzug, wenn Sie die Löschung des Root-Ordners nicht zulassen.

Lösungsvorschlag

Siehe Antwort zu Aufgabe b)

- (d) Was kann im Fall von `delete` passieren, wenn die Linkstruktur zyklisch ist oder die Ordnerstruktur zyklisch ist? Kann es zu diesen Problemen auch dann führen, wenn weder die Linkstruktur zyklisch ist, noch die Ordnerstruktur zyklisch ist? Wie kann man im Programm das Problem lösen, falls man Zyklizitäten zulassen möchte?

Lösungsvorschlag

Falls die Link- oder Ordnerstruktur zyklisch ist, kann es aufgrund der Rekursion zu einer Endlosschleife kommen. Diese Problem tritt bei azyklischen Strukturen nicht auf, weil der rekursive Löschvorgang beim letzten Element abgebrochen wird (Abbruchbedingung).

Das Problem kann zum Beispiel durch ein neues Attribut `gelöscht` in der Klasse `Link` oder `Directory` gelöst werden. Dieses Attribut wird auf `true` gesetzt, bevor es in die Rekursion einsteigt. Rufen sich die Klassen wegen der Zyklizität selbst wieder auf, kommt es durch entsprechende IF-Bedingungen zum Abbruch.

Außerdem ist ein Zähler denkbar, der sich bei jeder Rekursion hochsetzt und ab einem gewissen Grenzwert zum Abbruch führt.

- (e) Was ist ein Design Pattern? Nennen Sie drei Beispiele und erläutern Sie sie kurz. Welches Design Pattern bietet sich für die Behandlung von hierarchischen Teil-Ganzes-Beziehungen an, wie sie im Beispiel des Dateisystems vorliegen?

Lösungsvorschlag

Design Pattern sind wiederkehrende, geprüfte, bewährte Lösungsschablonen für typische Probleme.

Drei Beispiele

Einzelstück (Singleton) Stellt sicher, dass nur *genau eine Instanz einer Klasse* erzeugt wird.

Beobachter (Observer) Das Observer-Muster ermöglicht einem oder mehreren Objekten, automatisch auf die *Zustandsänderung* eines bestimmten Objekts zu *reagieren*, um den eigenen Zustand anzupassen.

Stellvertreter (Proxy) Ein Proxy stellt einen Platzhalter für eine andere Komponente (Objekt) dar und kontrolliert den Zugang zum echten Objekt.

Für hierarchischen Teil-Ganzes-Beziehungen eignet sich das Kompositum (Composite). Es ermöglicht die Gleichbehandlung von Einzelementen und Elementgruppierungen in einer verschachtelten Struktur (z. B. Baum), sodass aus Sicht des Clients keine explizite Unterscheidung notwendig ist.

Suche

66116 / 2019 / Herbst / Thema 2 / Aufgabe 1

Für diese Aufgabe wird die Vorlage Suchalgorithmen benötigt, die auf dem Beiblatt genauer erklärt wird.

Vervollständigen Sie die Methode `sucheBinaer()`. Die fertige Methode soll in der Lage sein, beliebige Werte in beliebigen sortierten Arrays zu suchen. Im gelben Textfeld des Eingabefensters soll dabei ausführlich und nachvollziehbar angezeigt werden, wie die Methode vorgeht. Beispielsweise so:

```
Führe die Methode sucheSequenziell() aus:
Suche in diesem Feld: 3, 5, 7, 17, 42, 23
Überprüfen des Werts an der Position 0
Überprüfen des Werts an der Position 1
Überprüfen des Werts an der Position 2
Überprüfen des Werts an der Position 3
Fertig :-)
```

```
Führe die Methode sucheBinaer() aus:
Suche in diesem Feld: 3, 5, 7, 17, 42, 23
Suchbereich: 0 bis 5
Mitte: 2, also neuer Bereich: 3 bis 5
Mitte: 4, also neuer Bereich: 3 bis 3
Mitte: 3, Treffer : -)
```

(a) Sequenzielle Suche

Lösungsvorschlag

```
}

/**
 * Sequenzielle Suche: Durchsucht das Array nach dem Wert und gibt dessen
 * Position als Ergebnis zurück.
 *
 * @param array Ein Feld mit Zahlen.
 * @param wert Die Zahl, die gesucht werden soll.
 *
 * @return Die Indexnummer der gesuchten Zahl.
 */
public int sucheSequenziell(int[] array, int wert) {
    fenster.schreibeZeile("\nFühre die Methode sucheSequenziell() aus:");
    fenster.schreibe("Suche in diesem Feld: ");
    fenster.schreibeArray(array);
    fenster.schreibeZeile("");
    // Wiederhole für alle Elemente des Arrays:
    for (int i = 0; i < array.length; i++) {
        fenster.schreibeZeile("Überprüfen des Werts an der Position " + i);
        // Wenn das Element an der Stelle i der gesuchte Wert ist:
        if (array[i] == wert) {
            fenster.schreibeZeile("Fertig :-)");
            // Gib die Position i als Ergebnis zurück:

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/ab_2/Suchalgorithmen.java](https://github.com/bschlangaul/aufgaben/aud/ab_2/Suchalgorithmen.java)

(b) Binäre Suche

Lösungsvorschlag

```
}
}
```

```

    fenster.schreibeZeile("Nichts gefunden :-(");
    // Gib den Sonderfalls -1 (nichts gefunden) als Ergebnis zurück:
    return -1;
}

/**
 * Binäre Suche: Durchsucht das Array nach dem Wert und gibt dessen
 → Position als
 * Ergebnis zurück.
 *
 * @param array Ein Feld mit Zahlen.
 * @param wert Die Zahl, die gesucht werden soll.
 *
 * @return Die Indexnummer der gesuchten Zahl.
 */
public int sucheBinaer(int[] array, int wert) {
    fenster.schreibeZeile("\nFühre die Methode sucheBinaer() aus:");
    fenster.schreibe("Suche in diesem Feld: ");
    fenster.schreibeArray(array);
    fenster.schreibeZeile("");
    int u = 0;
    int o = array.length - 1;
    fenster.schreibeZeile("Suchbereich: " + u + " bis " + o);
    while (u <= o) {
        int m = (u + o) / 2;
        if (array[m] == wert) {
            fenster.schreibe("Mitte: " + m);
            fenster.schreibeZeile(", Treffer : -) ");
            return m;
        } else if (array[m] > wert) {

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/ab_2/Suchalgorithmen.java](https://github.com/bschlangaul/aufgaben/aud/ab_2/Suchalgorithmen.java)

46115 / 2015 / Herbst / Thema 1 / Aufgabe 3

Aufgabe 3

Eine Folge von Zahlen a_1, \dots, a_n heie unimodal, wenn sie bis zu einem bestimmten Punkt echt ansteigt und dann echt fllt. Zum Beispiel ist die Folge 1, 3, 5, 6, 5, 2, 1 unimodal, die Folgen 1, 3, 5, 4, 7, 2, 1 und 1, 2, 3, 3, 4, 3, 2, 1 aber nicht.

Exkurs: Unimodale Abbildung

Eine unimodale Abbildung oder unimodale Funktion ist in der Mathematik eine Funktion mit einem eindeutigen (lokalen und globalen) Maximum wie zum Beispiel $f(x) = -x^2$.^a

^ahttps://de.wikipedia.org/wiki/Unimodale_Abbildung

- (a) Entwerfen Sie einen Algorithmus, der zu (als Array) gegebener unimodaler Folge a_1, \dots, a_n in Zeit $\mathcal{O}(\log n)$ das Maximum $\max a_i$ berechnet. Ist die Folge nicht

unimodal, so kann Ihr Algorithmus ein beliebiges Ergebnis liefern. Größenvergleiche, arithmetische Operationen und Arrayzugriffe können wie üblich in konstanter Zeit ($\mathcal{O}(1)$) getätigt werden. Hinweise: binäre Suche, divide-and-conquer.

Lösungsvorschlag

Wir wählen einen Wert in der Mitte der Folge aus. Ist der direkte linke und der direkte rechte Nachbar dieses Wertes kleiner, dann ist das Maximum gefunden. Ist nur linke Nachbar größer, setzen wir die Suche wie oben beschrieben in der linken Hälfte, sonst in der rechten Hälfte fort.

- (b) Begründen Sie, dass Ihr Algorithmus tatsächlich in Zeit $\mathcal{O}(\log n)$ läuft.

Lösungsvorschlag

Da der beschriebene Algorithmus nach jedem Bearbeitungsschritt nur auf der Hälfte der Feld-Elemente zu arbeiten hat, muss im schlechtesten Fall nicht die gesamte Folge durchsucht werden. Nach dem ersten Teilen der Folge bleiben nur noch $\frac{n}{2}$ Elemente, nach dem zweiten Schritt $\frac{n}{4}$, nach dem dritten $\frac{n}{8}$ usw. Allgemein bedeutet dies, dass im i -ten Durchlauf maximal $\frac{n}{2^i}$ Elemente zu durchsuchen sind. Entsprechend werden $\log_2 n$ Schritte benötigt. Somit hat der Algorithmus zum Finden des Maximums in einer unimodalen Folge in der Landau-Notation ausgedrückt die Zeitkomplexität $\mathcal{O}(\log n)$.

- (c) Schreiben Sie Ihren Algorithmus in Pseudocode oder in einer Programmiersprache Ihrer Wahl, z. B. Java, auf. Sie dürfen voraussetzen, dass die Eingabe in Form eines Arrays der Größe n vorliegt.

Lösungsvorschlag

Rekursiver Ansatz

```
public static int findeMaxRekursiv(int feld[], int links, int rechts) {
    if (links == rechts - 1) {
        return feld[links];
    }
    // bedeutet aufrunden
    // https://stackoverflow.com/a/17149572
    int mitte = (int) Math.ceil((double) (links + rechts) / 2);
    if (feld[mitte - 1] < feld[mitte]) {
        return findeMaxRekursiv(feld, mitte, rechts);
    } else {
        return findeMaxRekursiv(feld, links, mitte);
    }
}

public static int findeMaxRekursiv(int feld[]) {
    return findeMaxRekursiv(feld, 0, feld.length - 1);
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinder.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinder.java)

Iterativer Ansatz

```

public static int findeMaxIterativ(int[] feld) {
    int links = 0;
    int rechts = feld.length - 1;
    int mitte;

    while (links < rechts) {
        mitte = links + (rechts - links) / 2;
        if (feld[mitte] > feld[mitte - 1] && feld[mitte] > feld[mitte + 1]) {
            return feld[mitte];
        } else if (feld[mitte] > feld[mitte - 1]) {
            links = mitte + 1;
        } else {
            rechts = mitte - 1;
        }
    }
    return KEIN_MAX;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinder.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinder.java)

- (d) Beschreiben Sie in Worten ein Verfahren, welches in Zeit $\mathcal{O}(n)$ feststellt, ob eine vorgelegte Folge unimodal ist oder nicht.

Lösungsvorschlag

```

public static boolean testeUnimodalität(int[] feld) {
    if (feld.length < 2) {
        // Die Reihe braucht mindestens 3 Einträge
        return false;
    }

    if (feld[0] > feld[1]) {
        // Die Reihe muss zuerst ansteigen
        return false;
    }

    boolean maxErreicht = false;
    for (int i = 0; i < feld.length - 1; i++) {
        if (feld[i] > feld[i + 1] && !maxErreicht) {
            maxErreicht = true;
        }

        if (maxErreicht && feld[i] < feld[i + 1]) {
            // Das Maximum wurde bereits erreicht und die nächste Zahl ist
            // → größer
            return false;
        }
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinder.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinder.java)

- (e) Begründen Sie, dass es kein solches Verfahren (Test auf Unimodalität) geben kann, welches in Zeit $\mathcal{O}(\log n)$ läuft.

Da die Unimodalität nur durch einen Werte an einer beliebigen Stelle der Folge verletzt werden kann, müssen alle Elemente durchsucht und überprüft werden.

Komplette Klasse

```
public static int findeMaxRekursiv(int feld[], int links, int rechts) {
    if (links == rechts - 1) {
        return feld[links];
    }
    //      bedeutet aufrunden
    // https://stackoverflow.com/a/17149572
    int mitte = (int) Math.ceil((double) (links + rechts) / 2);
    if (feld[mitte - 1] < feld[mitte]) {
        return findeMaxRekursiv(feld, mitte, rechts);
    } else {
        return findeMaxRekursiv(feld, links, mitte);
    }
}

public static int findeMaxRekursiv(int feld[]) {
    return findeMaxRekursiv(feld, 0, feld.length - 1);
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinder.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinder.java)

Test

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class UnimodalFinderTest {

    private void testeMaxIterativ(int[] feld, int max) {
        assertEquals(max, UnimodalFinder.findeMaxIterativ(feld));
    }

    private void testeMaxRekursiv(int[] feld, int max) {
        assertEquals(max, UnimodalFinder.findeMaxRekursiv(feld));
    }

    private void testeMax(int[] feld, int max) {
        testeMaxIterativ(feld, max);
        testeMaxRekursiv(feld, max);
    }

    @Test
    public void findeMax() {
        testeMax(new int[] { 1, 2, 3, 1 }, 3);
    }
}
```

```

    }

    @Test
    public void findeMaxLaengeresFeld() {
        testeMax(new int[] { 1, 3, 4, 6, 7, 8, 9, 11, 6, 5, 4, 3, 2 }, 11);
    }

    @Test
    public void keinMaxAufsteigend() {
        testeMaxIterativ(new int[] { 1, 2, 3 }, UnimodalFinder.KEIN_MAX);
    }

    @Test
    public void keinMaxAbsteigend() {
        testeMaxIterativ(new int[] { 3, 2, 1 }, UnimodalFinder.KEIN_MAX);
    }

    @Test
    public void maxNegativeZahlen() {
        testeMax(new int[] { -2, -1, 3, 1 }, 3);
    }

    private void testeUnimodalität(int[] feld, boolean wahr) {
        assertEquals(wahr, UnimodalFinder.testeUnimodalität(feld));
    }

    @Test
    public void unimodalität() {
        testeUnimodalität(new int[] { 1, 2, 3, 1 }, true);
    }

    @Test
    public void unimodalitätFalsch() {
        testeUnimodalität(new int[] { 1, -2, 3, 1, 2 }, false);
        testeUnimodalität(new int[] { 1, 2, 3, 1, 2 }, false);
        testeUnimodalität(new int[] { 3, 2, 1 }, false);
    }
}

```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinderTest.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2015/herbst/UnimodalFinderTest.java)

46115 / 2016 / Frühjahr / Thema 2 / Aufgabe 3

Sie sollen mithilfe von Falltests eine neue Serie von Smartphones auf Bruchsicherheit testen.

Dazu wird eine Leiter mit n Sprossen verwendet; die höchste Sprosse, von der ein Smartphone heruntergeworfen werden kann ohne zu zerbrechen, heie „*höchste sichere Sprosse*“. Das Ziel ist, die höchste sichere Sprosse zu ermitteln. Man kann davon ausgehen, dass die höchste sichere Sprosse nicht von der Art des Wurfs abhängt und dass alle verwendeten Smartphones sich gleich verhalten. Eine Möglichkeit, die höchste sichere Sprosse zu ermitteln, besteht darin, ein Gerät erst von Sprosse 1, dann von Sprosse 2, etc. abzuwerfen, bis es schließlich beim Wurf von Sprosse k beschädigt wird (oder

Sie oben angelangt sind). Sprosse $k - 1$ (bzw. n) ist dann die höchste sichere Sprosse. Bei diesem Verfahren wird maximal ein Smartphone zerstört, aber der Zeitaufwand ist ungünstig.

- (a) Bestimmen Sie die Zahl der Würfe bei diesem Verfahren im schlechtesten Fall.

Lösungsvorschlag

Die Zahl der Würfe im schlechtesten Fall ist $\mathcal{O}(k)$, wobei k die Anzahl der Sprossen ist. Geht das Smartphone erst bei der höchsten Sprosse kaputt, muss es k mal heruntergeworfen werden. Die Komplexität entspricht der der linearen Suche.

- (b) Geben Sie nun ein Verfahren zur Ermittlung der höchsten sicheren Sprosse an, welches nur $\mathcal{O}(\log n)$ Würfe benötigt, dafür aber möglicherweise mehr Smartphones verbraucht.

Lösungsvorschlag

Man startet bei Sprosse $\frac{n}{2}$. Wenn das Smartphone kaputt geht, macht man weiter mit der Sprosse in der Mitte der unteren Hälfte, ansonsten mit der Sprosse in der Mitte der oberen Hälfte. Das Ganze rekursiv.

- (c) Es gibt eine Strategie zur Ermittlung der höchsten sicheren Sprosse mit $\mathcal{O}(\sqrt{n})$ Würfeln, bei dessen Anwendung höchstens zwei Smartphones kaputtgehen. Finden Sie diese Strategie und begründen Sie Ihre Funktionsweise und Wurfzahl.

Tipp: der erste Testwurf erfolgt von Sprosse $\lceil \sqrt{n} \rceil$.

Exkurs: Interpolationssuche

Die Interpolationssuche, auch Intervallsuche genannt, ist ein von der binären Suche abgeleitetes Suchverfahren, das auf Listen und Feldern zum Einsatz kommt.

Während der Algorithmus der binären Suche stets das mittlere Element des Suchraums überprüft, versucht der Algorithmus der Interpolationssuche im Suchraum einen günstigeren Teilungspunkt als die Mitte zu erraten. Die Arbeitsweise ist mit der eines Menschen vergleichbar, der ein Wort in einem Wörterbuch sucht: Die Suche nach Zylinder wird üblicherweise am Ende des Wörterbuches begonnen, während die Suche nach Aal im vorderen Bereich begonnen werden dürfte.^a

^ahttps://de.wikipedia.org/wiki/Quadratische_Binärsuche

Exkurs: Quadratische Binärsuche

Quadratische Binärsuche ist ein Suchalgorithmus ähnlich der Binärsuche oder Interpolationssuche. Es versucht durch Reduzierung des Intervalls in jedem Rekursionsschritt die Nachteile der Interpolationssuche zu vermeiden.

Nach dem Muster der Interpolationssuche wird zunächst in jedem rekursiven Schritt die vermutete Position k interpoliert. Anschließend wird – um die Nachteile der Interpolationssuche zu vermeiden – das Intervall der Länge \sqrt{n} gesucht, in dem sich der gesuchte Wert befindet. Auf dieses Intervall wird der nächste rekursive Aufruf der Suche angewendet.

Auf diese Weise verkleinert sich der Suchraum bei gegebener Liste der Länge n bei jedem

rekursiven Schritt auf eine Liste der Länge $n \sqrt{n}$.^a

^ahttps://de.wikipedia.org/wiki/Quadratische_Bin%C3%A4rsuche

Lösungsvorschlag

Das Vorgehen ist folgendermaßen: Man beginnt auf Stufe 0 und falls das Handy nicht kaputt geht, addiert man jeweils Wurzel n . Falls das Handy kaputt geht, geht man linear in Einerschritten das Intervall von der unteren Grenze (ö von der Stufe vor der letzten Addition) bis zur Kaputtstufe ab.^a

^a<http://www.inf.fu-berlin.de/lehre/WS06/HA/skript/vorlesung6.pdf>

46115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 2

- (a) Argumentieren Sie, warum man das Maximum von n Zahlen nicht mit weniger als $n - 1$ Vergleichen bestimmen kann.

Lösungsvorschlag

Wenn die n Zahlen in einem unsortierten Zustand vorliegen, müssen wir alle Zahlen betrachten, um das Maximum zu finden. Wir brauchen dazu $n - 1$ und nicht n Vergleiche, da wir die erste Zahl zu Beginn des Algorithmus als Maximum definieren und anschließend die verbleibenden Zahlen $n - 1$ mit dem aktuellen Maximum vergleichen.

- (b) Geben Sie einen Algorithmus im Pseudocode an, der das Maximum eines Feldes der Länge n mit genau $n - 1$ Vergleichen bestimmt.

Lösungsvorschlag

```
public static int bestimmeMaximum(int[] a) {  
    int max = a[0];  
    for (int i = 1; i < a.length; i++) {  
        if (a[i] > max) {  
            max = a[i];  
        }  
    }  
    return max;  
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java](https://github.com/orgs/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java)

- (c) Wenn man das Minimum und das Maximum von n Zahlen bestimmen will, dann kann das natürlich mit $2n - 2$ Vergleichen erfolgen. Zeigen Sie, dass man bei jedem beliebigen Feld mit deutlich weniger Vergleichen auskommt, wenn man die beiden Werte statt in zwei separaten Durchläufen in einem Durchlauf geschickt bestimmt.

```

/**
 * Diese Methode ist nicht optimiert. Es werden  $2n - 2$  Vergleiche benötigt.
 *
 * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
→ gesucht
 *         werden soll.
 *
 * @return Ein Feld mit zwei Einträgen. Der erste Eintrag enthält das
→ Minimum,
 *         der zweite Eintrag das Maximum.
 */
public static int[] minMaxNaiv(int[] a) {
    int max = a[0];
    int min = a[0];
    for (int i = 1; i < a.length; i++) {
        if (a[i] > max) {
            max = a[i];
        }
        if (a[i] < min) {
            min = a[i];
        }
    }
    return new int[] { min, max };
}

/**
 * Diese Methode ist optimiert. Es werden immer zwei Zahlen paarweise
 * betrachtet. Die Anzahl der Vergleiche reduziert sich auf  $3n/2 + 2$  bzw.
 *  $3(n-1)/2 + 4$  bei einer ungeraden Anzahl an Zahlen im Feld.
 *
 * nach <a href=
 * "https://www.techiedelight.com/find-minimum-maximum-element-array-using-
→ minimum-comparisons/">techiedelight.com</a>
 *
 * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
→ gesucht
 *         werden soll.
 *
 * @return Ein Feld mit zwei Einträgen. Der erste Eintrag enthält das
→ Minimum,
 *         der zweite Eintrag das Maximum.
 */
public static int[] minMaxIterativPaarweise(int[] a) {
    int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
    int n = a.length;

    boolean istUngerade = (n & 1) == 1;
    if (istUngerade) {
        n--;
    }

    for (int i = 0; i < n; i = i + 2) {
        int maximum, minimum;

```

```
        if (a[i] > a[i + 1]) {
            minimum = a[i + 1];
            maximum = a[i];
        } else {
            minimum = a[i];
            maximum = a[i + 1];
        }

        if (maximum > max) {
            max = maximum;
        }

        if (minimum < min) {
            min = minimum;
        }
    }

    if (istUngerade) {
        if (a[n] > max) {
            max = a[n];
        }

        if (a[n] < min) {
            min = a[n];
        }
    }

    return new int[] { min, max };
}

/**
 * Diese Methode ist nach dem Teile-und-Herrsche-Prinzip optimiert. Er
 * funktioniert so ähnlich wie der Mergesort.
 *
 * nach <a href=
 * "https://www.enjoyalgorithms.com/blog/find-the-minimum-and-maximum-
→ value-in-an-array">enjoyalgorithms.com</a>
 *
 * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
 *          gesucht werden soll.
 * @param l Die linke Grenze.
 * @param r Die rechts Grenze.
 *
 * @return Ein Feld mit zwei Einträgen. Der erste Eintrag enthält das
→ Minimum,
 *          der zweite Eintrag das Maximum.
 */
int[] minMaxRekursiv(int[] a, int l, int r) {
    int max, min;
    if (l == r) {
        max = a[l];
        min = a[l];
    } else if (l + 1 == r) {
        if (a[l] < a[r]) {
            max = a[r];

```

```

        min = a[l];
    } else {
        max = a[l];
        min = a[r];
    }
} else {
    int mid = l + (r - l) / 2;
    int[] lErgebnis = minMaxRekursiv(a, l, mid);
    int[] rErgebnis = minMaxRekursiv(a, mid + 1, r);
    if (lErgebnis[0] > rErgebnis[0]) {
        max = lErgebnis[0];
    } else {
        max = rErgebnis[0];
    }
    if (lErgebnis[1] < rErgebnis[1]) {
        min = lErgebnis[1];
    } else {
        min = rErgebnis[1];
    }
}
int[] ergebnis = { max, min };
return ergebnis;
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java)

46115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 3

Gegeben ist ein aufsteigend sortiertes Array A von n ganzen Zahlen und eine ganze Zahl x . Es wird der Algorithmus BinarySearch betrachtet, der A effizient nach dem Wert x absucht. Ergebnis ist der Index i mit $x = A[i]$ oder NIL, falls $x \notin A$.

Funktion BinarySearch(int A, int r)

```

l = 1;
r = A.length;
while r ≥ l do
    if x < A[m] then
        r = m - 1;
    else if x = A[m] then
        return m;
    else
        l = m + 1;
    end
    return NIL;
end

```

- (a) Durchsuchen Sie das folgende Feld jeweils nach den in (i) bis (iii) angegebenen Werten mittels binärer Suche. Geben Sie für jede Iteration die Werte $/,r,m$ und den betretenen if -Zweig an. Geben Sie zudem den Ergebnis-Index bzw. NIL an.

Index

$i[s] \ll \ll 2] 4] \text{ off}$

wen $[ilsfol7] io] w]u]al ale!$

- (i) 10
(ii) 13
(iii) 22
- (b) Betrachten Sie auf das Array aus Teilaufgabe a). Für welche Werte durchläuft der Algorithmus nie den letzten $else$ -Teil in Zeile 11? Hinweis: Unterscheiden Sie auch zwischen enthaltenen und nicht-enthaltenen Werten.
- (c) Wie ändert sich das Ergebnis der binären Suche, wenn im sortierten Eingabefeld zwei aufeinanderfolgende, unterschiedliche Werte vertauscht wurden? Betrachten Sie hierbei die betroffenen Werte, die anderen Feldelemente und nicht enthaltene Werte in Abhängigkeit vom Ort der Vertauschung.
- (d) Angenommen, das Eingabearray A für den Algorithmus für die binäre Suche enthält nur die Zahlen 0 und 1, aufsteigend sortiert. Zudem ist jede der beiden Zahlen mindestens ein Mal vorhanden. Ändern Sie den Algorithmus für die binäre Suche so ab, dass er den bzw. einen Index k zurückgibt, für den gilt: $A[k] = 1$ und $A[k-1] = 0$.

- (e) Betrachten Sie die folgende rekursive Variante von BinarySearch.

1 $\text{int RekBinarySearch}(\text{int}[] A, \text{int } x, \text{int } l, \text{int } r)$

2 $\text{if } l > r$

3 $\text{return } \text{NIL}$ (rekursive Implementierung)

Der initiale Aufruf der rekursiven Variante lautet: $\text{RekBinarySearch}(A, x, 1, A.\text{length})$

66115 / 2020 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 5

Eine Folge von Zahlen ist eine *odd-ascending-even-descending*-Folge, wenn gilt:

Zunächst enthält die Folge alle Schlüssel, die *ungerade* Zahlen sind, und diese Schlüssel sind aufsteigend sortiert angeordnet. Im Anschluss daran enthält die Folge alle Schlüssel, die *gerade* Zahlen sind, und diese Schlüssel sind absteigend sortiert angeordnet.

- (a) Geben Sie die Zahlen 10, 3, 11, 20, 8, 4, 9 als *odd-ascending-even-descending*-Folge an.

3, 9, 11, 20, 10, 8, 4

- (b) Geben Sie einen Algorithmus (z. B. in Pseudocode oder Java) an, der für eine *odd-ascending-even-descending-Folge* F gegeben als Feld und einem Schlüsselwert S prüft, ob S in F vorkommt und **true** im Erfolgsfall und ansonsten **false** liefert. Dabei soll der Algorithmus im Worst-Case eine echt bessere Laufzeit als Linearzeit (in der Größe der Arrays) haben. Erläutern Sie Ihren Algorithmus und begründen Sie die Korrektheit.

Bei dem Algorithmus handelt es sich um einen leicht abgewandelten Code, der eine „klassische“ binären Suche implementiert.

```
public static boolean suche(int[] feld, int schlüssel) {
    int links = 0, rechts = feld.length - 1;
    boolean istGerade = schlüssel % 2 == 0;
    while (links <= rechts) {
        int mitte = links + (rechts - links) / 2;
        if (feld[mitte] == schlüssel) {
            return true;
        }
        // Verschiebe die linke Grenze nach rechts, wenn die gesuchte
        // Zahl gerade ist und die Zahl in der Mitte größer als die
        // gesuchte Zahl ist oder wenn die gesuchte Zahl ungerade ist
        // und die Zahl in der Mitte kleiner.
        if ((istGerade && feld[mitte] > schlüssel) || (!istGerade &&
            ↪ feld[mitte] < schlüssel)) {
            // nach rechts verschieben
            links = mitte + 1;
        } else {
            // nach links verschieben
            rechts = mitte - 1;
        }
    }
    return false;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGerade.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGerade.java)

- (c) Erläutern Sie schrittweise den Ablauf Ihres Algorithmus für die Folge 1, 5, 11, 8, 4, 2 und den Suchschlüssel 4.

Die erste Zeile der Methode `suche` initialisiert die Variable `links` mit 0 und `rechts` mit 5. Da `links` kleiner ist als `rechts`, wird die `while`-Schleife betreten und die Variable `mitte` auf 2 gesetzt. Da der gesuchte Schlüssel gerade ist und `feld[2]` 11 ist, also größer, wird in den `true`-Block der `if`-Bedingung besprungen und die Variable `links` auf 3 gesetzt.

Zu Beginn des 2. Durchlaufs der `while`-Schleife ergeben sich folgende Werte: `links`: 3 `mitte`: 4 `rechts`: 5.

In der anschließenden Bedingten Anweisung wird die `while`-Schleife verlassen und `true` zurückgegeben, da mit `feld[4]` der gewünschte Schlüssel gefunden wurde.

- (d) Analysieren Sie die Laufzeit Ihres Algorithmus für den Worst-Case, geben Sie diese in \mathcal{O} -Notation an und begründen Sie diese.

Lösungsvorschlag

Die Laufzeit des Algorithmuses ist in der Ordnung $\mathcal{O}(\log_2 n)$.

Im schlechtesten Fall muss nicht die gesamte Folge durchsucht werden. Nach dem ersten Teilen der Folge bleiben nur noch $\frac{n}{2}$ Elemente, nach dem zweiten Schritt $\frac{n}{4}$, nach dem dritten $\frac{n}{8}$ usw. Allgemein bedeutet dies, dass im i -ten Durchlauf maximal $\frac{n}{2^i}$ Elemente zu durchsuchen sind. Entsprechend werden $\log_2 n$ Schritte benötigt.

Kompletter Code

```
public class UngeradeGerade {

    public static boolean suche(int[] feld, int schlüssel) {
        int links = 0, rechts = feld.length - 1;
        boolean istGerade = schlüssel % 2 == 0;
        while (links <= rechts) {
            int mitte = links + (rechts - links) / 2;
            if (feld[mitte] == schlüssel) {
                return true;
            }
            // Verschiebe die linke Grenze nach rechts, wenn die gesuchte
            // Zahl gerade ist und die Zahl in der Mitte größer als die
            // gesuchte Zahl ist oder wenn die gesuchte Zahl ungerade ist
            // und die Zahl in der Mitte kleiner.
            if ((istGerade && feld[mitte] > schlüssel) || (!istGerade && feld[mitte] <
                ↪ schlüssel)) {
                // nach rechts verschieben
                links = mitte + 1;
            } else {
                // nach links verschieben
                rechts = mitte - 1;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        System.out.println(suche(new int[] { 1, 5, 11, 8, 4, 2 }, 4));
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGerade.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGerade.java)

Test-Code

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class UngeradeGeradeTest {

    private void assertSucheUnGerade(int[] feld, int suche, boolean ergebnis) {
        assertEquals(ergebnis, UngeradeGerade.suche(feld, suche));
    }

    @Test
    public void assertSucheUnGerade() {
        int[] feld = new int[] { 1, 3, 5, 7, 9, 10, 8, 6, 4, 2 };
        assertSucheUnGerade(feld, 4, true);
        assertSucheUnGerade(feld, 11, false);
        assertSucheUnGerade(feld, 0, false);
        assertSucheUnGerade(feld, 3, true);
    }
}
```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGeradeTest.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGeradeTest.java)

66115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 2

- (a) Argumentieren Sie, warum man das Maximum von n Zahlen nicht mit weniger als $n - 1$ Vergleichen bestimmen kann.

Lösungsvorschlag

Wenn die n Zahlen in einem unsortierten Zustand vorliegen, müssen wir alle Zahlen betrachten, um das Maximum zu finden. Wir brauchen dazu $n - 1$ und nicht n Vergleiche, da wir die erste Zahl zu Beginn des Algorithmus als Maximum definieren und anschließend die verbleibenden Zahlen $n - 1$ mit dem aktuellen Maximum vergleichen.

- (b) Geben Sie einen Algorithmus im Pseudocode an, der das Maximum eines Feldes der Länge n mit genau $n - 1$ Vergleichen bestimmt.

Lösungsvorschlag

```
public static int bestimmeMaximum(int[] a) {
    int max = a[0];
    for (int i = 1; i < a.length; i++) {
        if (a[i] > max) {
            max = a[i];
        }
    }
    return max;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java)

- (c) Wenn man das Minimum und das Maximum von n Zahlen bestimmen will, dann kann das natürlich mit $2n - 2$ Vergleichen erfolgen. Zeigen Sie, dass man bei jedem beliebigen Feld mit deutlich weniger Vergleichen auskommt, wenn man die beiden Werte statt in zwei separaten Durchläufen in einem Durchlauf geschickt bestimmt.

Lösungsvorschlag

```

/**
 * Diese Methode ist nicht optimiert. Es werden  $2n - 2$  Vergleiche benötigt.
 *
 * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
→ gesucht
 *         werden soll.
 *
 * @return Ein Feld mit zwei Einträgen. Der erste Eintrag enthält das
→ Minimum,
 *         der zweite Eintrag das Maximum.
 */
public static int[] minMaxNaiv(int[] a) {
    int max = a[0];
    int min = a[0];
    for (int i = 1; i < a.length; i++) {
        if (a[i] > max) {
            max = a[i];
        }
        if (a[i] < min) {
            min = a[i];
        }
    }
    return new int[] { min, max };
}

/**
 * Diese Methode ist optimiert. Es werden immer zwei Zahlen paarweise
 * betrachtet. Die Anzahl der Vergleiche reduziert sich auf  $3n/2 + 2$  bzw.
 *  $3(n-1)/2 + 4$  bei einer ungeraden Anzahl an Zahlen im Feld.
 *
 * nach <a href=
 * "https://www.techiedelight.com/find-minimum-maximum-element-array-using-
→ minimum-comparisons/">techiedelight.com</a>
 *
 * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
→ gesucht
 *         werden soll.
 *
 * @return Ein Feld mit zwei Einträgen. Der erste Eintrag enthält das
→ Minimum,
 *         der zweite Eintrag das Maximum.
 */
public static int[] minMaxIterativPaarweise(int[] a) {
    int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
    int n = a.length;

    boolean istUngerade = (n & 1) == 1;

```

```
    if (istUngerade) {
        n--;
    }

    for (int i = 0; i < n; i = i + 2) {
        int maximum, minimum;

        if (a[i] > a[i + 1]) {
            minimum = a[i + 1];
            maximum = a[i];
        } else {
            minimum = a[i];
            maximum = a[i + 1];
        }

        if (maximum > max) {
            max = maximum;
        }

        if (minimum < min) {
            min = minimum;
        }
    }

    if (istUngerade) {
        if (a[n] > max) {
            max = a[n];
        }

        if (a[n] < min) {
            min = a[n];
        }
    }

    return new int[] { min, max };
}

/**
 * Diese Methode ist nach dem Teile-und-Herrsche-Prinzip optimiert. Er
 * funktioniert so ähnlich wie der Mergesort.
 *
 * nach <a href=
 * "https://www.enjoyalgorithms.com/blog/find-the-minimum-and-maximum-
→ value-in-an-array">enjoyalgorithms.com</a>
 *
 * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
 *          gesucht werden soll.
 * @param l Die linke Grenze.
 * @param r Die rechts Grenze.
 *
 * @return Ein Feld mit zwei Einträgen. Der erste Eintrag enthält das
→ Minimum,
 *          der zweite Eintrag das Maximum.
 */
int[] minMaxRekursiv(int[] a, int l, int r) {
```

```
int max, min;
if (l == r) {
    max = a[l];
    min = a[l];
} else if (l + 1 == r) {
    if (a[l] < a[r]) {
        max = a[r];
        min = a[l];
    } else {
        max = a[l];
        min = a[r];
    }
} else {
    int mid = l + (r - l) / 2;
    int[] lErgebnis = minMaxRekursiv(a, l, mid);
    int[] rErgebnis = minMaxRekursiv(a, mid + 1, r);
    if (lErgebnis[0] > rErgebnis[0]) {
        max = lErgebnis[0];
    } else {
        max = rErgebnis[0];
    }
    if (lErgebnis[1] < rErgebnis[1]) {
        min = lErgebnis[1];
    } else {
        min = rErgebnis[1];
    }
}
int[] ergebnis = { max, min };
return ergebnis;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java)

66116 / 2017 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 4

Die folgende Seite enthält Software-Quellcode, der einen Algorithmus zur binären Suche implementiert. Dieser ist durch Inspektion zu überprüfen. Im Folgenden sind die Regeln der Inspektion angegeben.

RM1	(Dokumentation)	Jede Quellcode-Datei beginnt mit einem Kommentar, der den Klassennamen, Versionsinformationen, Datum und Urheberrechtsangaben enthält.
RM2	(Dokumentation)	Jede Methode wird kommentiert. Der Kommentar enthält eine vollständige Beschreibung der Signatur so wie eine Design-by-Contract-Spezifikation.
RM3	(Dokumentation)	Deklarationen von Variablen werden kommentiert.
RM4	(Dokumentation)	Jede Kontrollstruktur wird kommentiert.
RM5	(Formatierung)	Zwischen einem Schlüsselwort und einer Klammer steht ein Leerzeichen.
RM6	(Formatierung)	Zwischen binären Operatoren und den Operanden stehen Leerzeichen.
RM7	(Programmierung)	Variablen werden in der Anweisung initialisiert, in der sie auch deklariert werden.
RM8	(Bezeichner)	Klassennamen werden groß geschrieben, Variablennamen klein.

```
/**
 * BinarySearch.java
 *
 * Eine Implementierung der "Binaere Suche"
 * mit einem iterativen Algorithmus
 */
class BinarySearch {

    /**
     * BinaereSuche
     * a: Eingabefeld
     * item: zuzuchendesElement
     * returnValue: der Index des zu suchenden Elements oder -1
     *
     * Vorbedingung:
     * a.length > 0
     * a ist ein linear geordnetes Feld:
     * For all k: (1 <= k < a.length) ==> (a[k-1] <=a [k])
     *
     * Nachbedingung:
     * Wenn item in a, dann gibt es ein k mit a[k] == item und returnValue == k
     * Genau dann wenn returnValue == -1 gibt es kein k mit 0 <= k < a.length
     * und a[k]==item.
     */
    public static int binarySearch(float a[], float item) {

        int End; // exklusiver Index fuer das Ende des
                // zuzuchsuchenden Teils des Arrays
    }
```

```
int start = 1; // inklusiver Index fuer den Anfang der Suche
End = a.length;

// Die Schleife wird verlassen, wenn keine der beiden Haelften das
// Element enthaelt.
while(start < End) {

    // Teilung des Arrays in zwei Haelften
    // untere Haelfte: [0,mid[
    // obere Haelfte: ]mid,End[
    int mid = (start + End) / 2;

    if (item > a[mid]) {
        // Ausschluss der oberen Haelfte
        start = mid + 1;
    } else if(item < a[mid]) {
        // Ausschluss der unteren Haelfte
        End = mid-1;
    } else {
        // Das gesuchte Element wird zurueckgegeben
        return (mid);
    }
} // end of while

// Bei Misserfolg der Suche wird -1 zurueckgegeben
return (-1);
}
```

- (a) Überprüfen Sie durch Inspektion, ob die obigen Regeln für den Quellcode eingehalten wurden. Erstellen Sie eine Liste mit allen Verletzungen der Regeln. Geben Sie für jede Verletzung einer Regel die Zeilennummer, Regelnummer und Kommentar an, z. B. (07, RM4, while nicht kommentiert). Schreiben Sie nicht in den Quellcode.

Lösungsvorschlag

--

Zeile	Regel	Kommentar
3-8	RM1	Fehlen von Versionsinformationen, Datum und Urheberrechtsangaben
11-26	RM2	Fehlen der Invariante in der Design-by-Contract-Spezifikation
36,46	RM5	Fehlen des Leerzeichens vor der Klammer
48	RM6	Um einen binären (zweistellige) Operator handelt es sich im Code-Beispiel um den Subtraktionsoperator: <code>mid-1</code> . Hier fehlen die geforderten Leerzeichen.
32	RM7	Die Variable <code>End</code> wird in Zeile 32 deklariert, aber erst in Zeile initialisiert <code>End = a.length;</code>
32	RM8	Die Variable <code>End</code> muss klein geschrieben werden.

- (b) Entspricht die Methode `binarySearch` ihrer Spezifikation, die durch Vor- und Nachbedingungen angegeben ist? Geben Sie gegebenenfalls Korrekturen der Methode an.

Lösungsvorschlag

Korrektur der Vorbedingung

Die Vorbedingung ist nicht erfüllt, da weder die Länge des Feldes `a` noch die Reihenfolge der Feldeinträge geprüft wurden.

```

if (a.length <= 0) {
    return -1;
}

for (int i = 0; i < a.length; i++) {
    if (a[i] > a[i + 1]) {
        return -1;
    }
}

```

Korrektur der Nachbedingung

`int start` muss mit `0` initialisiert werden, da sonst `a[0]` vernachlässigt wird.

- (c) Beschreiben alle Kommentare ab Zeile 24 die Semantik des Codes korrekt? Geben Sie zu jedem falschen Kommentar einen korrigierten Kommentar mit Zeilennummer an.

Zeile	Kommentar im Code	Korrektur
34-35	<code>// Die Schleife wird v erlassen, wenn keine der beiden Haelften das Elemen t enthaelt.</code>	<code>// Die Schleife wird v erlassen, wenn keine der beiden Haelften das El ement enthaelt oder das Element gefunden wurde.</code>
44	<code>// Ausschluss der oberen Haelfte</code>	<code>// Ausschluss der unteren Haelfte</code>
47	<code>// Ausschluss der unteren Haelfte</code>	<code>// Ausschluss der oberen Haelfte</code>
50	<code>// Das gesuchte Element wird zurueckgegeben</code>	<code>// Der Index des gesuchten Elements wird zurueckgeb en</code>

- (d) Geben Sie den Kontrollflussgraphen für die Methode `binarySearch` an.
- (e) Geben Sie maximal drei Testfälle für die Methode `binarySearch` an, die insgesamt eine vollständige Anweisungsüberdeckung leisten.

Die gegebene Methode: `binarySearch(a[], item)`

Testfall

- (i) Testfall: `a[] = {1, 2, 3}, item = 4`
- (ii) Testfall: `a[] = {1, 2, 3}, item = 2`

Sortieralgorithmen

66116 / 2017 / Herbst / Thema 1 / Aufgabe 4

Das Sortierverfahren *Mergesort*, das nach der Strategie *Divide-and-Conquer* arbeitet, sortiert eine Sequenz, indem die Sequenz in zwei Teile zerlegt wird, die dann einzeln sortiert und wieder zu einer sortierten Sequenz zusammengemischt werden (to merge = zusammenmischen, verschmelzen).

(a) Gegeben seien folgende Methoden:

```
public int[] mergesort(int[] s) {
    int[] left = new int[s.length / 2];
    int[] right = new int[s.length - (s.length / 2)];
    int[] result;

    if (s.length <= 1) {
        result = s;
    } else {
        for (int i = 0; i < s.length / 2; i++) {
            left[i] = s[i];
        }
        int a = 0;
        for (int j = (s.length / 2); j < s.length; j++) {
            right[a++] = s[j];
        }
        result = merge(mergesort(left), mergesort(right));
    }
    return result;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/sortier/Mergesort.java](https://github.com/bschlangaul/aufgaben/aud/sortier/Mergesort.java)

Schreiben Sie die Methode `public int[] merge (int[] s, int[] r)`, die die beiden aufsteigend sortierten Sequenzen `s` und `r` zu einer aufsteigend sortierten Sequenz zusammenmischt.

Lösungsvorschlag

```
public int[] merge(int[] s, int[] r) {
    int[] ergebnis = new int[s.length + r.length];
    int indexLinks = 0;
    int indexRechts = 0;
    int indexErgebnis = 0;

    // Im Reisverschlussverfahren s und r sortiert zusammenfügen.
    while (indexLinks < s.length && indexRechts < r.length) {
        if (s[indexLinks] < r[indexRechts]) {
            ergebnis[indexErgebnis] = s[indexLinks++];
        } else {
            ergebnis[indexErgebnis] = r[indexRechts++];
        }
    }
}
```

```

    indexErgebnis++;
}

// Übrig gebliebene Elemente von s einfügen.
while (indexLinks < s.length) {
    ergebnis[indexErgebnis++] = s[indexLinks++];
}

// Übrig gebliebene Elemente von r einfügen.
while (indexRechts < r.length) {
    ergebnis[indexErgebnis++] = r[indexRechts++];
}

return ergebnis;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/sortier/Mergesort.java](https://github.com/bschlangaul/aufgaben/aud/sortier/Mergesort.java)

(b) Analysieren Sie die Zeitkomplexität von `mergesort`.

Lösungsvorschlag

$$\mathcal{O}(n \cdot \log n)$$

Erklärung

Mergesort ist ein stabiles Sortierverfahren, vorausgesetzt der Merge-Schritt ist korrekt implementiert. Seine Komplexität beträgt im Worst-, Best- und Average-Case in Landau-Notation ausgedrückt stets $\mathcal{O}(n \cdot \log n)$. Für die Laufzeit $T(n)$ von Mergesort bei n zu sortierenden Elementen gilt die Rekursionsformel

$$\begin{aligned}
 T(n) &= \\
 &T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + && \text{Aufwand, 1. Teil zu sortieren} \\
 &T\left(\left\lceil \frac{n}{2} \right\rceil\right) + && \text{Aufwand, 2. Teil zu sortieren} \\
 &\mathcal{O}(n) && \text{Aufwand, beide Teile zu verschmelzen}
 \end{aligned}$$

mit dem Rekursionsanfang $T(1) = 1$.

Nach dem Master-Theorem kann die Rekursionsformel durch

$$2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$$

bzw.

$$2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n$$

approximiert werden mit jeweils der Lösung $T(n) = \mathcal{O}(n \cdot \log n)$.

Mergesort
Quicksort
Bubblesort

66116 / 2017 / Herbst / Thema 1 / Aufgabe 4

Gegeben ist folgende Zahlenfolge:

35, 22, 5, 3, 28, 16, 8, 60, 17, 66, 4, 9, 82, 11, 10, 20

- (a) Sortiere Sie händisch mit Mergesort. Orientieren Sie sich beim Aufschreiben der Zwischenschritte an dieser Darstellung:


- (b) Sortiere Sie händisch mit Quicksort. Wählen Sie als Pivot-Element immer das Element in der Mitte - oder gegebenenfalls das Element direkt links neben der Mitte. Orientieren Sie sich beim Aufschreiben der Zwischenschritte an dieser Darstellung:

66116 / 2017 / Herbst / Thema 1 / Aufgabe 4


Gegeben sei ein Array a , welches die Werte 5, 7, 9, 3, 6, 1, 2, 8 enthält. Sortieren Sie das Array händisch mit:

- (a) Bubblesort


erster Durchgang



5	7	9	3	6	1	2	8
---	---	---	---	---	---	---	---




5	7	9	6	3	1	2	8
---	---	---	---	---	---	---	---




5	7	9	6	1	3	2	8
---	---	---	---	---	---	---	---


Zweiter Durchgang



5	7	9	6	1	2	3	8
---	---	---	---	---	---	---	---




5	7	9	1	6	2	3	8
---	---	---	---	---	---	---	---




5	7	9	1	2	6	3	8
---	---	---	---	---	---	---	---


Dritter Durchgang




5	7	9	1	2	3	6	8
---	---	---	---	---	---	---	---



5	7	1	9	2	3	6	8
---	---	---	---	---	---	---	---



5	7	1	2	9	3	6	8
---	---	---	---	---	---	---	---



5	7	1	2	3	9	6	8
---	---	---	---	---	---	---	---

5 7 1 2 3 6 9 8

Vierter Durchgang

5 7 1 2 3 6 8 9

5 1 7 2 3 6 8 9

5 1 2 7 3 6 8 9

5 1 2 3 7 6 8 9

Fünfter Durchgang

5 1 2 3 6 7 8 9

1 5 2 3 6 7 8 9

1 2 5 3 6 7 8 9

fertig

1 2 3 5 6 7 8 9

```

5  7  9  3  6  1  2  8  Eingabe
5  7  9  3  6  1  2  8  Durchlauf Nr. 1
5  7 >9  3< 6  1  2  8  vertausche [2<>3]
5  7  3 >9  6< 1  2  8  vertausche [3<>4]
5  7  3  6 >9  1< 2  8  vertausche [4<>5]
5  7  3  6  1 >9  2< 8  vertausche [5<>6]
5  7  3  6  1  2 >9  8< vertausche [6<>7]
5  7  3  6  1  2  8  9  Durchlauf Nr. 2
5 >7  3< 6  1  2  8  9  vertausche [1<>2]
5  3 >7  6< 1  2  8  9  vertausche [2<>3]
5  3  6 >7  1< 2  8  9  vertausche [3<>4]
5  3  6  1 >7  2< 8  9  vertausche [4<>5]
5  3  6  1  2  7  8  9  Durchlauf Nr. 3
>5  3< 6  1  2  7  8  9  vertausche [0<>1]
3  5 >6  1< 2  7  8  9  vertausche [2<>3]
3  5  1 >6  2< 7  8  9  vertausche [3<>4]
3  5  1  2  6  7  8  9  Durchlauf Nr. 4
3 >5  1< 2  6  7  8  9  vertausche [1<>2]
3  1 >5  2< 6  7  8  9  vertausche [2<>3]
3  1  2  5  6  7  8  9  Durchlauf Nr. 5
>3  1< 2  5  6  7  8  9  vertausche [0<>1]
```

```

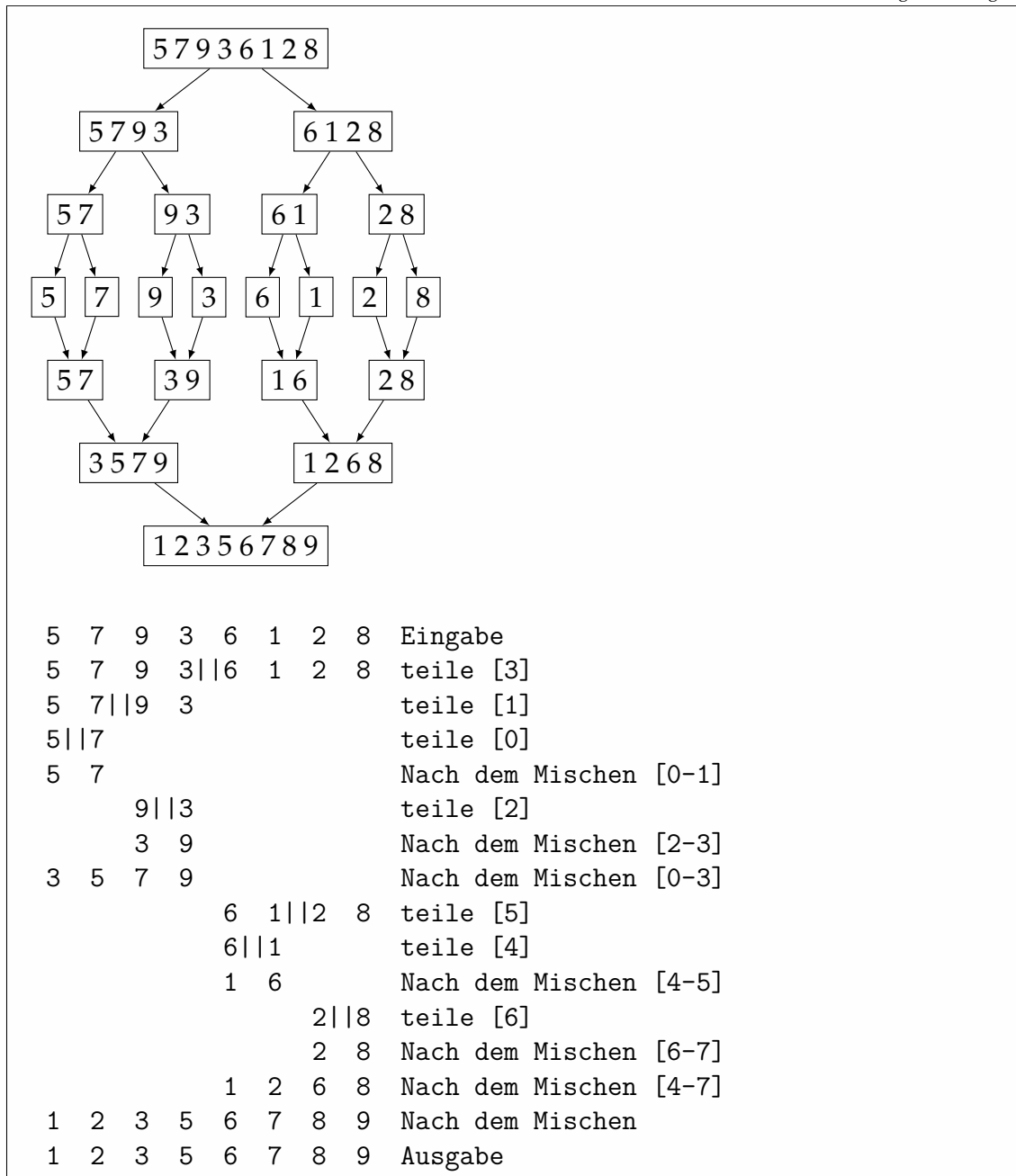
1 >3  2< 5  6  7  8  9  vertausche [1<>2]
1  2  3  5  6  7  8  9  Durchlauf Nr. 6
1  2  3  5  6  7  8  9  Ausgabe

```

Mergesort
Quicksort

(b) Mergesort

Lösungsvorschlag



(c) Quicksort

Lösungsvorschlag

```

5 7 9 3 6 1 2 8 Eingabe
5 7 9 3 6 1 2 8 zerlege
5 7 9 3* 6 1 2 8 markiere [3]

```



```

5 7 9 >3 6 1 2 8< vertausche [3<>7]
>5 7 9 8 6 1< 2 3 vertausche [0<>5]
1 >7 9 8 6 5 2< 3 vertausche [1<>6]
1 2 >9 8 6 5 7 3< vertausche [2<>7]
1 2 zerlege [0-1]
1* 2 markiere [0]
>1 2< vertausche [0<>1]
>2 1< vertausche [0<>1]

8 6 5 7 9 zerlege [3-7]
8 6 5* 7 9 markiere [5]
8 6 >5 7 9< vertausche [5<>7]
>8 6 9 7 5< vertausche [3<>7]
6 9 7 8 zerlege [4-7]
6 9* 7 8 markiere [5]
6 >9 7 8< vertausche [5<>7]
>6< 8 7 9 vertausche [4<>4]
6 >8< 7 9 vertausche [5<>5]
6 8 >7< 9 vertausche [6<>6]
6 8 7 >9< vertausche [7<>7]
6 8 7 zerlege [4-6]
6 8* 7 markiere [5]
6 >8 7< vertausche [5<>6]
>6< 7 8 vertausche [4<>4]
6 >7< 8 vertausche [5<>5]
6 7 >8< vertausche [6<>6]
6 7 zerlege [4-5]
6* 7 markiere [4]
>6 7< vertausche [4<>5]
>7 6< vertausche [4<>5]
1 2 3 5 6 7 8 9 Ausgabe

```

66116 / 2017 / Herbst / Thema 1 / Aufgabe 4

Für diese Aufgabe wird die Vorlage Sortieralgorithmen benötigt, die auf dem Beiblatt genauer erklärt wird.

Die fertigen Methoden sollen in der Lage sein, beliebige Arrays zu sortieren. Im gelben Textfeld des Eingabefensters soll dabei wieder ausführlich und nachvollziehbar angezeigt werden, wie die jeweilige Methode vorgeht. Beispielsweise so:

```

Führe die Methode selectionSort() aus:
Sortiere dieses Feld: 5, 3, 17, 7, 42, 23
Der Marker liegt bei: 5
Das Maximum liegt bei: 4
Diese beiden Elemente werden nun vertauscht.
Ergebnis dieser Runde: 5, 3, 17, 7, 23, 42
Der Marker liegt bei: 4
Das Maximum liegt bei: 4
Diese beiden Elemente werden nun vertauscht.
Ergebnis dieser Runde: 5, 3, 17, 7, 23, 42
Der Marker liegt bei: 3
Das Maximum liegt bei: 2
Diese beiden Elemente werden nun vertauscht.
Ergebnis dieser Runde: 5, 3, 7, 17, 23, 42
Der Marker liegt bei: 2

```

```

Das Maximum liegt bei: 2
Diese beiden Elemente werden nun vertauscht.
Ergebnis dieser Runde: 5, 3, 7, 17, 23, 42
Der Marker liegt bei: 1
Das Maximum liegt bei: 0
Diese beiden Elemente werden nun vertauscht.
Ergebnis dieser Runde: 3, 5, 7, 17, 23, 42
Der Marker liegt bei: 0
Das Maximum liegt bei: 0
Diese beiden Elemente werden nun vertauscht.
Ergebnis dieser Runde: 3, 5, 7, 17, 23, 42

```

```

Führe die Methode bubbleSort() aus:
Sortiere dieses Feld: 5, 3, 17, 7, 42, 23

```

Das Element an der Stelle 0 ist größer als sein Nachfolger.
 Diese beiden werden nun vertauscht.
 Ergebnis dieser Runde: 3, 5, 17, 7, 42, 23
 Das Element an der Stelle 2 ist größer als sein Nachfolger.
 Diese beiden werden nun vertauscht.

Ergebnis dieser Runde: 3, 5, 7, 17, 42, 23
 Das Element an der Stelle 4 ist größer als sein Nachfolger.
 Diese beiden werden nun vertauscht.
 Ergebnis dieser Runde: 3, 5, 7, 17, 23, 42

Selectionsort
 Bubblesort

(a) Vervollständige die Methode `selectionSort()`.

Lösungsvorschlag

```
/**
 * SelectionSort: Sortieren durch Selektion.
 *
 * @param array Ein Feld mit Zahlen.
 *
 * @return Ein sortiertes Feld mit Zahlen.
 */
public int[] selectionSort(int[] array) {
    fenster.schreibeZeile("\nFühre die Methode selectionSort() aus:");
    fenster.schreibe("Sortiere dieses Feld: ");
    fenster.schreibeArray(array);
    fenster.schreibeZeile("");
    int marker = array.length - 1;
    while (marker >= 0) {
        int max = 0;
        for (int i = 0; i <= marker; i++) {
            if (array[i] > array[max]) {
                max = i;
            }
        }
        fenster.schreibeZeile("Der Marker liegt bei: " + marker);
        fenster.schreibeZeile("Das Maximum liegt bei: " + max);
        fenster.schreibeZeile("Diese beiden Elemente werden nun vertauscht.");
        swap(array, marker, max);
        fenster.schreibe("Ergebnis dieser Runde: ");
        fenster.schreibeArray(array);
        fenster.schreibeZeile("");
        marker--;
    }
    return array;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/ab_2/Sortieralgorithmen.java](https://github.com/bschlangaul/aufgaben/aud/ab_2/Sortieralgorithmen.java)

(b) Vervollständige die Methode `bubbleSort()`.

Lösungsvorschlag

```
/**
 * BubbleSort: Sortieren durch Vertauschen.
 *
 * @param array Ein Feld mit Zahlen.
 *
 * @return Ein sortiertes Feld mit Zahlen.
 */
public int[] bubbleSort(int[] array) {
    fenster.schreibeZeile("\nFühre die Methode bubbleSort() aus:");
    fenster.schreibe("Sortiere dieses Feld: ");
```

```

fenster.schreibeArray(array);
fenster.schreibeZeile("");
boolean swapped;
do {
    swapped = false;
    for (int i = 0; i < array.length - 1; i++) {
        if (array[i] > array[i + 1]) {
            fenster.schreibe("Das Element an der Stelle " + i);
            fenster.schreibeZeile(" ist größer als sein Nachfolger.");
            fenster.schreibeZeile("Diese beiden werden nun vertauscht.");
            swap(array, i, i + 1);
            fenster.schreibe("Ergebnis dieser Runde: ");
            fenster.schreibeArray(array);
            fenster.schreibeZeile(" ");
            swapped = true;
        }
    }
} while (swapped);
return array;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/ab_2/Sortieralgorithmen.java](https://github.com/bschlangaul/aufgaben/aud/ab_2/Sortieralgorithmen.java)

46115 / 2013 / Frühjahr / Thema 2 / Aufgabe 6

Aufgabe 6

- (a) Vervollständigen Sie die folgende Sortierung mit MergeSort (Sortieren durch Mischen) — beginnen Sie dabei Ihren „rekursiven Abstieg“ immer im linken Teilfeld:

D | 40 5 89 95 85 84 || 14 25 20 52 7 71 |

Notation: Markieren Sie Zeilen mit D(ivide), in denen das Array zerlegt wird, und mit M(erge), in denen Teilarrays zusammengeführt werden. Beispiel:

D | 82 || 89 44 |

D 82 | 89 || 44 |

M 82 | 44 89 |

M | 44 82 89 |

D		40		5		89		95		85		84		14		25		20		52		7		71	
D		40		5		89		95		85		84													
D		40		5		89																			
D		40		5																					
M		5		40																					
M		5		40		89																			
D								95		85		84													
D								95		85		84													
D								95		85															
M								85		95															
M								84		85		95													
M		5		40		84		85		89		95													
D														14		25		20		52		7		71	

[illegible]

(b) Sortieren Sie mittels HeapSort (Haldensortierung) die folgende Liste weiter: Notation: Markieren Sie die Zeilen wie folgt:

I: Initiale Heap-Eigenschaft hergestellt (größtes Element am Anfang der Liste).

R: Erstes und letztes Element getauscht und letztes „gedanklich entfernt“.

S: Erstes Element nach unten „versickert“ (Heap-Eigenschaft wiederhergestellt).

Lösungsvorschlag

I		99	63	91	4	36	81	76		
R		76	63	91	4	36	81		99	
S		91	63	81	4	36	76		99	
R		76	63	81	4	36		91	99	
S		81	76	63	4	36		91	99	
R		36	76	63	4		81	91	99	
S		76	36	63	4		81	91	99	
R		4	36	63		76	81	91	99	
S		63	4	36		76	81	91	99	
R		4	36		63	76	81	91	99	
S		36	4		63	76	81	91	99	
R		4		36	63	76	81	91	99	
S		4		36	63	76	81	91	99	
R		4	36	63	76	81	91	99		

46115 / 2016 / Frühjahr / Thema 1 / Aufgabe 8

(a) Sortieren Sie das Array mit den Integer Zahlen

25, 1, 12, 27, 30, 9, 33, 34, 18, 16

(i) mit *BubbleSort*

Lösungsvorschlag

25	1	12	27	30	9	33	34	18	16	Eingabe
25	1	12	27	30	9	33	34	18	16	Durchlauf Nr. 1
>25	1<	12	27	30	9	33	34	18	16	vertausche (i 0<>1)
1	>25	12<	27	30	9	33	34	18	16	vertausche (i 1<>2)
1	12	25	27	>30	9<	33	34	18	16	vertausche (i 4<>5)
1	12	25	27	9	30	33	>34	18<	16	vertausche (i 7<>8)
1	12	25	27	9	30	33	18	>34	16<	vertausche (i 8<>9)
1	12	25	27	9	30	33	18	16	34	Durchlauf Nr. 2

```

1  12  25 >27  9<  30  33  18  16  34 vertausche (i 3<>4)
1  12  25  9  27  30 >33 18< 16  34 vertausche (i 6<>7)
1  12  25  9  27  30 18 >33 16< 34 vertausche (i 7<>8)
1  12  25  9  27  30 18 16 33 34 Durchlauf Nr. 3
1  12 >25  9<  27  30 18 16 33 34 vertausche (i 2<>3)
1  12  9  25  27 >30 18< 16 33 34 vertausche (i 5<>6)
1  12  9  25  27 18 >30 16< 33 34 vertausche (i 6<>7)
1  12  9  25  27 18 16 30 33 34 Durchlauf Nr. 4
1 >12  9<  25  27 18 16 30 33 34 vertausche (i 1<>2)
1  9  12  25 >27 18< 16 30 33 34 vertausche (i 4<>5)
1  9  12  25 18 >27 16< 30 33 34 vertausche (i 5<>6)
1  9  12  25 18 16 27 30 33 34 Durchlauf Nr. 5
1  9  12 >25 18< 16 27 30 33 34 vertausche (i 3<>4)
1  9  12 18 >25 16< 27 30 33 34 vertausche (i 4<>5)
1  9  12 18 16 25 27 30 33 34 Durchlauf Nr. 6
1  9  12 >18 16< 25 27 30 33 34 vertausche (i 3<>4)
1  9  12 16 18 25 27 30 33 34 Durchlauf Nr. 7
1  9  12 16 18 25 27 30 33 34 Ausgabe

```

(ii) mit *Quicksort*, wenn als Pivotelement das jeweils erste Element gewählt wird.

Beschreiben Sie die Abläufe der Sortiervverfahren

- (i) bei *BubbleSort* durch eine Angabe der Zwischenergebnisse nach jedem Durchlauf
 - (ii) bei *Quicksort* durch die Angabe der Zwischenergebnisse nach den rekursiven Aufrufen.
- (b) Welche Laufzeit (asymptotisch, in O-Notation) hat BubbleSort bei beliebig großen Arrays mit n Elementen. Begründen Sie Ihre Antwort.

46115 / 2017 / Frühjahr / Thema 2 / Aufgabe 4

Bei Bubblesort wird eine unsortierte Folge von Elementen a_1, a_2, \dots, a_n , von links nach rechts durchlaufen, wobei zwei benachbarte Elemente a_i und a_{i+1} getauscht werden, falls sie nicht in der richtigen Reihenfolge stehen. Dies wird so lange wiederholt, bis die Folge sortiert ist.

- (a) Sortieren Sie die folgende Zahlenfolge mit Bubblesort. Geben Sie die neue Zahlenfolge nach jedem (Tausch-)Schritt an: 3, 2, 4, 1

Lösungsvorschlag

```

3  2  4  1  Eingabe
3  2  4  1  Durchlauf Nr. 1
>3  2<  4  1  vertausche (i 0<>1)
2  3 >4  1<  vertausche (i 2<>3)
2  3  1  4  Durchlauf Nr. 2
2 >3  1<  4  vertausche (i 1<>2)
2  1  3  4  Durchlauf Nr. 3
>2  1<  3  4  vertausche (i 0<>1)
1  2  3  4  Durchlauf Nr. 4

```

1	2	3	4	Ausgabe
---	---	---	---	---------

- (b) Geben Sie den Bubblesort-Algorithmus für ein Array von natürlichen Zahlen in einer Programmiersprache Ihrer Wahl an. Die Funktion `swap (index1, index2)` kann verwendet werden, um zwei Elemente des Arrays zu vertauschen.

Lösungsvorschlag

```
public class BubbleSort {

    public static void swap(int[] array, int index1, int index2) {
        int tmp = array[index1];
        array[index1] = array[index2];
        array[index2] = tmp;
    }

    public static void bubblesort(int[] array) {
        boolean swapped;
        do {
            swapped = false;
            for (int i = 0; i < array.length - 1; i++) {
                if (array[i] > array[i + 1]) {
                    swap(array, i, i + 1);
                    swapped = true;
                }
            }
        } while (swapped);
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2017/fruehjahr/BubbleSort.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2017/fruehjahr/BubbleSort.java)

Test

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class BubbleSortTest {

    @Test
    public void teste() {
        int[] array = new int[] { 3, 2, 4, 1 };
        BubbleSort.bubblesort(array);
        assertEquals(1, array[0]);
        assertEquals(2, array[1]);
        assertEquals(3, array[2]);
        assertEquals(4, array[3]);
    }
}
```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/examen/examen_46115/jahr_2017/fruehjahr/BubbleSortTest.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2017/fruehjahr/BubbleSortTest.java)

- (c) Geben Sie eine obere Schranke für die Laufzeit an. Beschreiben Sie mögliche Eingabedaten, mit denen diese Schranke erreicht wird.

Lösungsvorschlag

$$\mathcal{O}(n^2)$$

Diese obere Schranke wird erreicht, wenn die Zahlenfolgen in der umgekehrten Reihenfolge bereits sortiert ist, z. B. 4, 3, 2, 1.

Aufgabe 7

- (a) Führen Sie „Sortieren durch Einfügen“ lexikographisch aufsteigend und *in-situ* (*in-place*) so in einem Schreibtischlauf auf folgendem Feld (Array) aus, dass gleiche Elemente ihre relative Abfolge jederzeit beibehalten (also dass z. B. A_1 stets vor A_2 im Feld steht). Jede Zeile stellt den Zustand des Feldes dar, nachdem das jeweils nächste Element in die Endposition verschoben wurde. Der bereits sortierte Teilbereich steht vor |||. Gleiche Elemente tragen zwecks Unterscheidung ihre „Objektidentität“ als Index (z. B. `"A1".equals("A2")` aber `"A1" != "A2"`)

L	A ₁	B ₁	F	A ₂	B ₂
---	----------------	----------------	---	----------------	----------------

Lösungsvorschlag

L	A ₁	B ₁	F	A ₂	B ₂
A ₁	L	B ₁	F	A ₂	B ₂
A ₁	B ₁	L	F	A ₂	B ₂
A ₁	B ₁	F	L	A ₂	B ₂
A ₁	A ₂	B ₁	F	L	B ₂
A ₁	A ₂	B ₁	B ₂	F	L

- (b) Ergänzen Sie die folgende Methode so, dass sie die Zeichenketten im Feld `a` lexikographisch aufsteigend durch Einfügen sortiert. Sie muss zum vorangehenden Ablauf passen, ösie muss *iterativ* sowie *in-situ* (*in-place*) arbeiten und die relative Reihenfolge gleicher Elemente jederzeit beibehalten. Sie dürfen davon ausgehen, dass kein Eintrag im Feld null ist.

```
void sortierenDurchEinfuegen(String[] a) {
    // Hilfsvariable:
    String tmp;
}
```

Lösungsvorschlag

```
static void sortierenDurchEinfuegen(String[] a) {
    // Hilfsvariable:
    String tmp;
    for (int i = 1; i < a.length; i++) {
```



```

tmp = a[i];
int j = i;
while (j > 0 && a[j - 1].compareTo(tmp) >= 1) {
    a[j] = a[j - 1];
    j = j - 1;
}
a[j] = tmp;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/beschlangaul/examen/examen_46115/jahr_2017/herbst/InsertionSort.java](https://github.com/src/main/java/org/beschlangaul/examen/examen_46115/jahr_2017/herbst/InsertionSort.java)

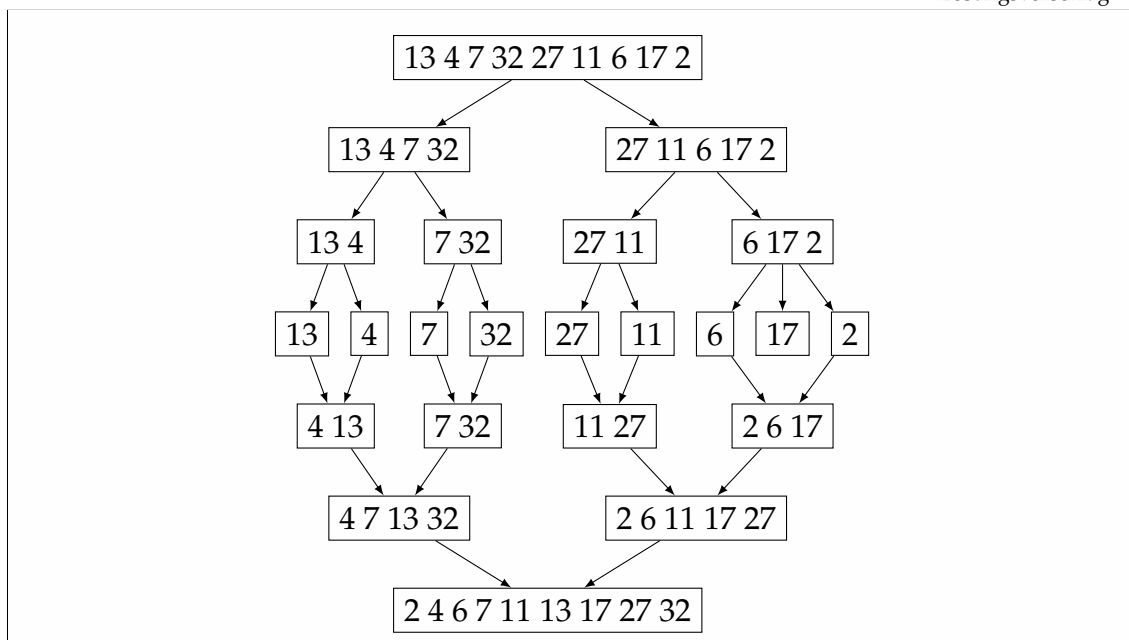
Aufgabe zum Mergesort

(a) Gegeben ist das folgende Array von Zahlen:

[13, 4, 7, 32, 27, 11, 6, 17, 2]

Sortieren Sie das Array mittels Mergesort aufsteigend von links nach rechts. Das Aufteilen einer Liste soll in der Mitte erfolgen und, falls notwendig, die zweite Liste ein Element länger sein als die erste. Listen der Länge zwei dürfen durch direkten Vergleich sortiert werden. Geben Sie die Eingabe und das Ergebnis jedes (rekursiven) Aufrufs an. Geben Sie abschließend die sortierte Liste an.

Lösungsvorschlag



(b) Beantworten Sie folgende Fragen jeweils ohne Begründung oder Beweis.

(i) Welche Worst-Case-Laufzeit (\mathcal{O} -Notation) hat Mergesort für n Elemente?

$\mathcal{O}(n \cdot \log(n))$ im Best-, Average- und Worst-Case

- (ii) Welche Laufzeit hat Mergesort für n Elemente im Best-Case?

$\mathcal{O}(n \cdot \log(n))$ im Best-, Average- und Worst-Case

- (iii) Kann basierend auf paarweisen Vergleichen von Werten schneller (Laufzeitkomplexität) als Mergesort sortiert werden?

Nein. Es lässt sich beweisen, dass ein vergleichsbasiertes Sortierverfahren nicht schneller als $\Omega(n \cdot \log(n))$ sein kann.^a

^a<https://de.wikipedia.org/wiki/Sortierverfahren>

46115 / 2019 / Herbst / Thema 1 / Aufgabe 4

In der folgenden Aufgabe soll ein Feld A von ganzen Zahlen *aufsteigend* sortiert werden. Das Feld habe n Elemente $A[1]$ bis $A[n]$. Der folgende Algorithmus sei gegeben:

```
var A : array[1..n] of integer;

procedure selection_sort
var i, j, smallest, tmp : integer;
begin
  for j := 1 to n-1 do begin
    smallest := j;
    for i := j + 1 to n do begin
      if A[i] < A[smallest] then
        smallest := i;
    end
    tmp = A[j];
    A[j] = A[smallest];
    A[smallest] = tmp;
  end
end
```

- (a) Sortieren Sie das folgende Feld mittels des Algorithmus. Notieren Sie alle Werte, die die Variable *smallest* jeweils beim Durchlauf der inneren Schleife annimmt. Geben Sie die Belegung des Feldes nach jedem Durchlauf der äußeren Schleife in einer neuen Zeile an.

Ausgang

27	32	3	6	17	44	42	29	8	14
----	----	---	---	----	----	----	----	---	----

nach 1. Durchlauf ($j = 1$)

smallest: (1) 3

3	32	27	6	17	44	42	29	8	14
---	----	----	---	----	----	----	----	---	----

nach 2. Durchlauf ($j = 2$)

smallest: (2) 3 4

3	6	27	32	17	44	42	29	8	14
---	---	----	----	----	----	----	----	---	----

nach 3. Durchlauf ($j = 3$)

smallest: (3) 5 9

3	6	8	32	17	44	42	29	27	14
---	---	---	----	----	----	----	----	----	----

nach 4. Durchlauf ($j = 4$)

smallest: (4) 5 10

3	6	8	14	17	44	42	29	27	32
---	---	---	----	----	----	----	----	----	----

nach 5. Durchlauf ($j = 5$)

smallest: (5) -

3	6	8	14	17	44	42	29	27	32
---	---	---	----	----	----	----	----	----	----

nach 6. Durchlauf ($j = 6$)

smallest: (6) 7 8 9

3	6	8	14	17	27	42	29	44	32
---	---	---	----	----	----	----	----	----	----

nach 7. Durchlauf ($j = 7$)

smallest: (7) 8

3	6	8	14	17	27	29	42	44	32
---	---	---	----	----	----	----	----	----	----

nach 8. Durchlauf ($j = 8$)

smallest: (8) 10

3	6	8	14	17	27	29	32	44	42
---	---	---	----	----	----	----	----	----	----

nach 9. Durchlauf ($j = 9$)

smallest: (9) 10

3	6	8	14	17	27	29	32	44	42
---	---	---	----	----	----	----	----	----	----

fertig

3	6	8	14	17	27	29	32	42	44
---	---	---	----	----	----	----	----	----	----

- (b) Der Wert der Variablen *smallest* wird bei jedem Durchlauf der äußeren Schleife mindestens ein Mal neu gesetzt. Wie muss das Feld *A* beschaffen sein, damit der Variablen *smallest* ansonsten niemals ein Wert zugewiesen wird? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Wenn das Feld bereits aufsteigend sortiert ist, dann nimmt die Variable *smallest* in der inneren Schleife niemals einen neuen Wert an.

- (c) Welche Auswirkung auf die Sortierung oder auf die Zuweisungen an die Variable *smallest* hat es, wenn der Vergleich in Zeile 9 des Algorithmus statt $A[i] < A[\text{smallest}]$ lautet $A[i] \leq A[\text{smallest}]$? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Der Algorithmus sortiert dann nicht mehr *stabil*, d.h. die Eingabereihenfolge von Elementen mit *gleichem Wert* wird beim Sortieren nicht mehr *bewahrt*.

- (d) Betrachten Sie den Algorithmus unter der Maßgabe, dass Zeile 9 wie folgt geändert wurde:

```
if A[i] > A[smallest] then
```

Welches Ergebnis berechnet der Algorithmus nun?

Lösungsvorschlag

Der Algorithmus sortiert jetzt absteigend.

- (e) Betrachten Sie die folgende *rekursive* Variante des Algorithmus. Der erste Parameter ist wieder das zu sortierende Feld, der Parameter *n* ist die Größe des Feldes und der Parameter *index* ist eine ganze Zahl. Die Funktion $\text{min_index}(A, x, y)$ berechnet für $1 \leq x \leq y \leq n$ den Index des kleinsten Elements aus der Menge $\{A[x], A[x+1], \dots, A[y]\}$

```
procedure rek_selection_sort(A, n, index : integer)
var k, tmp : integer;
begin
if (Abbruchbedingung) then return;
```

```
k = min_index(A, index, n);
if k <> index then begin
  tmp := A[k];
  A[k] := A[index];
  A[index] := tmp;
end
(rekursiver Aufruf)
end
```

Der initiale Aufruf des Algorithmus lautet: `rek_selection_sort(A, n, 1)`

Vervollständigen Sie die fehlenden Angaben in der Beschreibung des Algorithmus für

- die Abbruchbedingung in Zeile 4 und

Lösungsvorschlag

```
n = index bzw n == index
```

Begründung: Wenn der aktuelle Index so groß ist wie die Anzahl der Elemente im Feld, dann muss / darf abgebrochen werden, denn dann ist das Feld sortiert.

- den rekursiven Aufruf in Zeile 11.

Lösungsvorschlag

```
rek_selection_sort(A, n, index + 1)
```

Am Ende der Methode wurde an die Index-Position *index* das kleinste Element gesetzt, jetzt muss an die nächste Index-Position (*index + 1*) der kleinste Wert, der noch nicht sortieren Zahlen, gesetzt werden.

Begründen Sie Ihre Antworten.

```
import static org.bschlangaul.helfer.Konsole.zeigeZahlenFeld;

public class SelectionSort {

    public static void selectionSort(int[] A) {
        int smallest, tmp;

        for (int j = 0; j < A.length - 1; j++) {
            System.out.println("\nj = " + (j + 1));
            smallest = j;
            for (int i = j + 1; i < A.length; i++) {
                if (A[i] < A[smallest]) {
                    smallest = i;
                    System.out.println(smallest + 1);
                }
            }
            tmp = A[j];
            A[j] = A[smallest];
            A[smallest] = tmp;
            zeigeZahlenFeld(A);
        }
    }
}
```

```

    }
}

public static void rekSelectionSort(int[] A, int n, int index) {
    int k, tmp;

    if (index == n - 1) {
        return;
    }
    k = minIndex(A, index, n);
    if (k != index) {
        tmp = A[k];
        A[k] = A[index];
        A[index] = tmp;
    }
    rekSelectionSort(A, n, index + 1);
}

public static int minIndex(int[] A, int x, int y) {
    int smallest = x;
    for (int i = x; i < y; i++) {
        if (A[i] < A[smallest]) {
            smallest = i;
        }
    }
    return smallest;
}

public static void main(String[] args) {
    int[] A = new int[] { 27, 32, 3, 6, 17, 44, 42, 29, 8, 14 };
    selectionSort(A);

    A = new int[] { 27, 32, 3, 6, 17, 44, 42, 29, 8, 14 };
    rekSelectionSort(A, A.length, 0);
    zeigeZahlenFeld(A);
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/SelectionSort.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/SelectionSort.java)

46115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 1

- (a) Geben Sie für folgende Sortierverfahren jeweils zwei Felder A und B an, so dass das jeweilige Sortierverfahren angewendet auf A seine Best-Case-Laufzeit und angewendet auf B seine Worst-Case-Laufzeit erreicht. (Wir messen die Laufzeit durch die Anzahl der Vergleiche zwischen Elementen der Eingabe.) Dabei soll das Feld A die Zahlen 1,2,...,7 genau einmal enthalten; das Feld B ebenso. Sie bestimmen also nur die Reihenfolge der Zahlen.

Wenden Sie als Beleg für Ihre Aussagen das jeweilige Sortierverfahren auf die Felder A und B an und geben Sie nach jedem größeren Schritt des Algorithmus den Inhalt der Felder an.

Geben Sie außerdem für jedes Verfahren asymptotische Best- und Worst-Case-Laufzeit für ein Feld der Länge n an.

Für drei der Sortierv Verfahren ist der Pseudocode angegeben. Beachten Sie, dass die Feldindizes hier bei 1 beginnen. Die im Pseudocode verwendete Unterroutine $\text{Swap}(A, i, j)$ vertauscht im Feld A die Elemente mit den Indizes i und j miteinander.

- (i) Insertionsort
- (ii) Bubblesort
- (iii) Quicksort

Insertionsort(int[] A) for $i = 2$ to A.length do key = A[i] $i = i - 1$ while $i > 0$ and A[i] > key do A[i + 1] = A[i] $i = i - 1$ A[i + 1] = key

Bubblesort(int[] A) $n := \text{length}(A)$ repeat swapped = false for $i = n$ to 1 do if A[i] > A[i - 1] then Swap(A, i, i - 1)

swapped := true

until not swapped

Quicksort(int[] A, @ = 1, r = A.length) if $2 < r$ then $m = \text{Partition}(A, @, r)$ | Quicksort(A, @, m - 1) Quicksort(A, m + 1, r)

int Partition (int[] A, int @, int r)

pivot = A[r]

$i = @$

for $j = r$ to $@ - 1$ do

if A[j] < pivot then

Swap(A, i, j) $i = i + 1$

Swap(A, i, r)

return i

- (b) Geben Sie die asymptotische Best- und Worst-Case-Laufzeit von Mergesort an.

46116 / 2017 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 4

Ein gängiger Ansatz zur Messung der Qualität von Software ist das automatisierte Testen von Programmen. Im Folgenden werden praktische Testmethoden anhand des nachstehend angegebenen Sortieralgorithmus diskutiert.

Algorithmus 1 Bubble Sort

```
public class BubbleSort {
    void bubblesort(int[] array, int len) {
        for (int i = 0; i < len - 1; i++) { // 1
            for (int j = 0; j < len - 1; j++) { // 2
                if (array[j] > array[j + 1]) { // 3
```

```

    int temp = array[j];           // 4
    array[j] = array[j + 1];       // 5
    array[j + 1] = temp;           // 6
  }
}
}
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/beschlangaul/examen/examen_46116/jahr_2017/herbst/BubbleSort.java](https://github.com/src/main/java/org/beschlangaul/examen/examen_46116/jahr_2017/herbst/BubbleSort.java)

- (a) Nennen Sie eine Art des Black-Box-Testens und beschreiben Sie deren Durchführung anhand des vorgegebenen Algorithmus.

Lösungsvorschlag

Beim Black-Box-Testen sind die Testfälle von Daten getrieben (Data-Driven) und beziehen sich auf die Anforderungen und das spezifizierte Verhalten.)

⇒ Aufruf der Methoden mit verschiedenen Eingangsparametern und Vergleich der erhaltenen Ergebnisse mit den erwarteten Ergebnissen.

Das Ziel ist dabei eine möglichst hohe Anforderungsüberdeckung, wobei man eine minimale Anzahl von Testfällen durch Äquivalenzklassenzerlegung (1) und Grenzwertanalyse (2) erhält.

zu (1): Man identifiziert Bereiche von Eingabewerten, die jeweils dieselben Ergebnisse liefern. Dies sind die sog. Äquivalenzklassen. Aus diesen wählt man nun je einen Repräsentanten und nutzt diesen für den Testfall.

zu (2): Bei der Grenzwertanalyse identifiziert man die Grenzbereiche der Eingabedaten und wählt Daten aus dem nahen Umfeld dieser für seine Testfälle.

Angewendet auf den gegebenen Bubblesort-Algorithmus würde die Grenzwertanalyse bedeuten, dass man ein bereits aufsteigend sortiertes Array und ein absteigend sortiertes Array übergibt.

- (b) Zeichnen Sie ein mit Zeilennummern beschriftetes Kontrollflussdiagramm für den oben angegebenen Sortieralgorithmus.

Lösungsvorschlag

Zur Erinnerung: Eine im Code enthaltene Wiederholung mit for muss wie folgt im Kontrollflussgraphen „zerlegt“ werden:

- (c) Erklären Sie, ob eine vollständige Pfadüberdeckung für die gegebene Funktion möglich und sinnvoll ist.

Lösungsvorschlag

Eine vollständige Pfadüberdeckung (C_1 -Test) kann nicht erreicht werden, da die Bedingung der inneren Wiederholung immer wahr ist, wenn die Bedingung der äußeren Wiederholung wahr ist. D. h., der Pfad S-1-1-2-2-1“ kann nie gegangen werden. Dies wäre aber auch nicht sinnvoll, weil jeder Eintrag

mit jedem anderen verglichen werden soll und im Fall true \rightarrow false ein Durchgang ausgelassen.

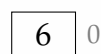
AVL-Baum

66115 / 2006 / Herbst / Thema 1 / Aufgabe 4

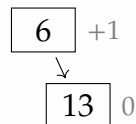
- (a) Gegeben sei die folgende Folge ganzer Zahlen: 6, 13, 4, 8, 11, 9, 10.
- (i) Fügen Sie obige Zahlen der Reihe nach in einen anfangs leeren AVL-Baum ein und stellen Sie den Baum nach jedem Einfügeschritt dar!

Lösungsvorschlag

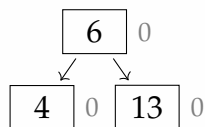
Nach dem Einfügen von „6“:



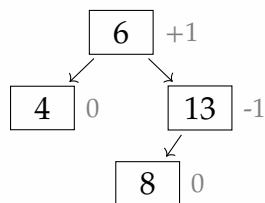
Nach dem Einfügen von „13“:



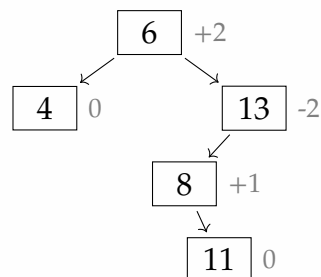
Nach dem Einfügen von „4“:



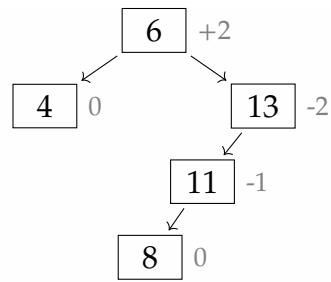
Nach dem Einfügen von „8“:



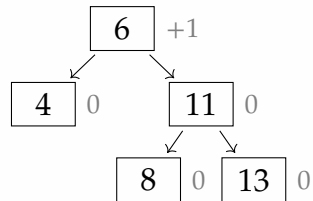
Nach dem Einfügen von „11“:



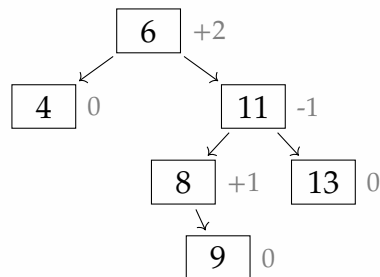
Nach der Linksrotation:



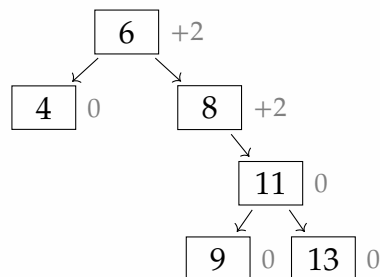
Nach der Rechtsrotation:



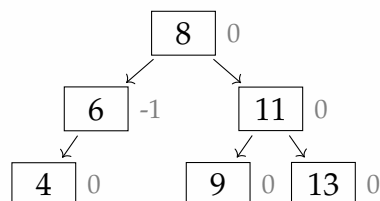
Nach dem Einfügen von „9“:



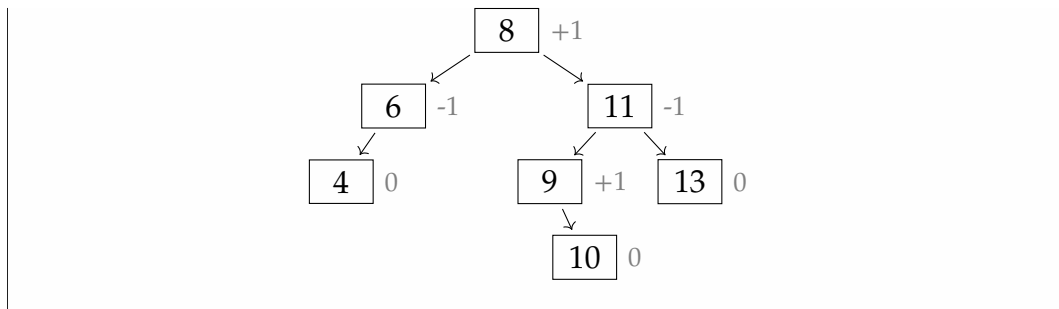
Nach der Rechtsrotation:



Nach der Linksrotation:

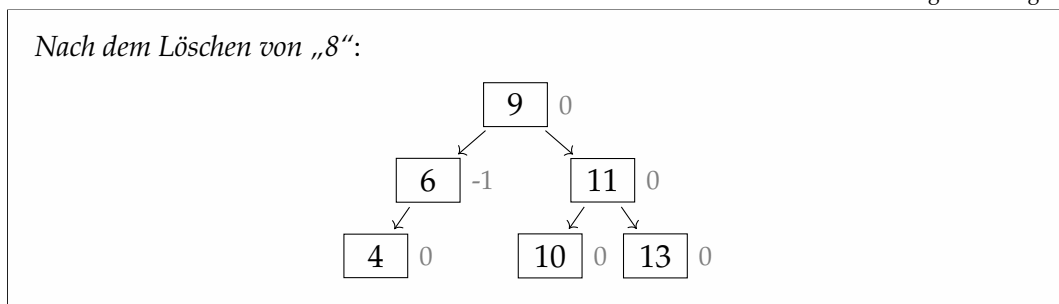


Nach dem Einfügen von „10“:



- (ii) Löschen Sie das Wurzelement des entstandenen AVL-Baums und stellen Sie die AVL-Eigenschaft wieder her!

Lösungsvorschlag



- (b) Gegeben sei der folgende gerichtete und gewichtete Graph:
- Bestimmen Sie mit Hilfe des Algorithmus von Dijkstra die kürzesten Wege vom Knoten A zu allen anderen Knoten! Geben Sie dabei nach jedem Verarbeitungsschritt den Zustand der Hilfsdatenstruktur an!
 - Skizzieren Sie einen Algorithmus für den Tiefendurchlauf von gerichteten Graphen, wobei jede Kante nur einmal verwendet werden darf!
- (c) Ein wesentlicher Nachteil der Standardimplementierung des QUICKSORT Algorithmus ist dessen rekursiver Aufruf. Implementieren Sie den Algorithmus QUICKSORT ohne den rekursiven Prozeduraufruf!

66115 / 2014 / Herbst / Thema 2 / Aufgabe 6

Gegeben sei ein einfacher Sortieralgorithmus, der ein gegebenes Feld A dadurch sortiert, dass er das *Minimum* m von A findet, dann das Minimum von A ohne das Element m usw.

- (a) Geben Sie den Algorithmus in Java an. Implementieren Sie den Algorithmus *in situ*, d.h., dass er außer dem Eingabefeld nur konstanten Extraspeicher benötigt. Es steht eine Testklasse zur Verfügung.

Lösungsvorschlag

```

public class SortierungDurchAuswaehlen {
    static void vertausche(int[] zahlen, int index1, int index2) {
        int tmp = zahlen[index1];
        zahlen[index1] = zahlen[index2];
        zahlen[index2] = tmp;
    }
}
  
```

```
}

static void sortiereDurchAuswählen(int[] zahlen) {
    // Am Anfang ist die Markierung das erste Element im Zahlen-Array.
    int markierung = 0;
    while (markierung < zahlen.length) {
        // Bestimme das kleinste Element.
        // 'min' ist der Index des kleinsten Elements.
        // Am Anfang auf das letzte Element setzen.
        int min = zahlen.length - 1;
        // Wir müssen nicht bis letzten Index gehen, da wir 'min' auf das
        // ↪ letzte Element
        // setzen.
        for (int i = markierung; i < zahlen.length - 1; i++) {
            if (zahlen[i] < zahlen[min]) {
                min = i;
            }
        }

        // Tausche zahlen[markierung] mit gefundenem Element.
        vertausche(zahlen, markierung, min);
        // Die Markierung um eins nach hinten verlegen.
        markierung++;
    }
}

public static void main(String[] args) {
    int[] zahlen = { 5, 2, 7, 1, 6, 3, 4 };
    sortiereDurchAuswählen(zahlen);
    for (int i = 0; i < zahlen.length; i++) {
        System.out.print(zahlen[i] + " ");
    }
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/SortierungDurchAuswaehlen.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/herbst/SortierungDurchAuswaehlen.java)

(b) Analysieren Sie die Laufzeit Ihres Algorithmus.

Beim ersten Durchlauf des *Selectionsort*-Algorithmus muss $n - 1$ mal das Minimum durch Vergleich ermittelt werden, beim zweiten Mal $n - 2$. Mit Hilfe der *Gaußschen Summenformel* kann die Komplexität gerechnet werden:

$$(n - 1) + (n - 2) + \dots + 3 + 2 + 1 = \frac{(n - 1) \cdot n}{2} = \frac{n^2}{2} - \frac{n}{2} \approx \frac{n^2}{2} \approx n^2$$

Da es bei der Berechnung der Komplexität um die Berechnung der asymptotischen oberen Grenze geht, können Konstanten und die Addition, Subtraktion, Multiplikation und Division mit Konstanten z. B. $\frac{n^2}{2}$ vernachlässigt werden.

Der *Selectionsort*-Algorithmus hat deshalb die Komplexität $\mathcal{O}(n^2)$, er ist von

der Ordnung $\mathcal{O}(n^2)$.

- (c) Geben Sie eine Datenstruktur an, mit der Sie Ihren Algorithmus beschleunigen können.

Lösungsvorschlag

Der *Selectionsort*-Algorithmus kann mit einer Min- (in diesem Fall) bzw. einer Max-Heap beschleunigt werden. Mit Hilfe dieser Datenstruktur kann sehr schnell das Minimum gefunden werden. So kann auf die vielen Vergleiche verzichtet werden. Die Komplexität ist dann $\mathcal{O}(n \log n)$.

66115 / 2015 / Herbst / Thema 2 / Aufgabe 2

Gegeben sei folgende Klasse:

```
class W {
    int t;
    String f;
    // ...
}
```

Dazu gibt es verschiedene Comparatoren, zum Beispiel:

```
// ascending order for field W.t
class ComparatorAscByFieldT implements Comparator<W> {
    // Returns a negative integer, zero, or a positive integer as the
    // first argument is less than, equal to, or greater than the second.
    @Override
    public int compare(W o1, W o2) { // ...
}
```

Außerdem steht Ihnen die vorgegebene Methode swap zur Verfügung:

```
void swap(W[] w, int a, int b) { // ...
}
```

- (a) Phase 1: Die Haldensortierung beginnt mit der Herstellung der Max-Heap-Eigenschaft von rechts nach links. Diese ist für alle Feldelemente im dunklen Bereich bereits erfüllt. Geben Sie die Positionen (IDs) derjenigen Elemente des Feldes an, die das Verfahren im „Versickerschritt“ für das nächste Element mit Hilfe des `ComparatorAscByFieldT` miteinander vergleicht:

IDsangeben> 0 1 2 3 4 5 6 <iDs angeben

Nach dem Vergleichen werden gegebenenfalls Werte mit swap vertauscht. Geben Sie das Resultat (in obiger Array-Darstellung) nach diesem Schritt an.

- (b) Phase 2: Das folgende Feld enthält den bereits vollständig aufgebauten MaxHeap:

Qo 1 2 3 4 5 6

71/6)/5,,3 7) 14,04 2

Die Haldensortierung verschiebt das maximale Element in den sortierten (dunklen) Bereich:

Q 1 2 3 4

2|6|/5/3/1/o

Geben Sie das Ergebnis des nachfolgenden „Versickerns“ (erneut in derselben Array-Darstellung) an, bei dem die Heap-Eigenschaft wiederhergestellt wird.

- (c) Ergänzen Sie die rekursive Methode `reheap`, die die Max-Heap-Eigenschaft im Feld `w` zwischen den Indizes `i` und `k` (jeweils einschließlich) in $O(\log(k-i))$ gemäß `Comparator<W> c` wiederherstellt, indem sie das Element `w[i]` „versickert“. `k` bezeichnet das Ende des unsortierten Bereichs.

```
// restores the max-heap property in w[i to k] using c
void reheap(W[] w, Comparator<W> c, int i, int k) {
    int leftId = 2 * i + 1;
    int rightId = leftId + 1;
    int kidId;
    // ToDo: Code hier ergaenzen
}
```

- (d) Implementieren Sie nun die eigentliche Haldensortierung. Sie dürfen hier die Methode `reheap` verwenden.

```
// sorts w in-situ according to the order imposed by c
void heapSort(W[] w, Comparator<W> c) {
    int n = w.length;

    // Phase 1: Max-Heap-Eigenschaft herstellen
    // (siehe Teilaufgabe a)
    // ToDo: Code hier ergaenzen

    // Phase 2: jeweils Maximum entnehmen und sortierte Liste am Ende
    // → aufbauen
    // (siehe Teilaufgabe b)
    // ToDo: Code hier ergaenzen
}
```

66115 / 2016 / Frühjahr / Thema 1 / Aufgabe 6

Sortieren Sie die Werte

1 45 8 53 9 2 17 10

mit Quicksort.

Lösungsvorschlag

Sortieralgorithmus nach Saake

1	45	8	53	9	2	17	10	zerlege
1	45	8	53*	9	2	17	10	markiere (i 3)
1	45	8	>53	9	2	17	10<	vertausche (i 3<>7)
>1<	45	8	10	9	2	17	53	vertausche (i 0<>0)
1	>45<	8	10	9	2	17	53	vertausche (i 1<>1)
1	45	>8<	10	9	2	17	53	vertausche (i 2<>2)

```

1  45  8  >10< 9   2  17  53  vertausche (i 3<>3)
1  45  8   10 >9< 2  17  53  vertausche (i 4<>4)
1  45  8   10  9 >2< 17  53  vertausche (i 5<>5)
1  45  8   10  9  2  >17< 53  vertausche (i 6<>6)
1  45  8   10  9  2  17 >53< vertausche (i 7<>7)
1  45  8   10  9  2  17      zerlege
1  45  8   10* 9  2  17      markiere (i 3)
1  45  8   >10  9  2  17<    vertausche (i 3<>6)
>1< 45  8   17  9  2  10      vertausche (i 0<>0)
1  >45  8<  17  9  2  10      vertausche (i 1<>2)
1  8  >45  17  9<  2  10      vertausche (i 2<>4)
1  8  9  >17  45  2<  10      vertausche (i 3<>5)
1  8  9  2  >45  17  10<    vertausche (i 4<>6)
1  8  9  2
1  8*  9  2
1  >8  9  2<
>1< 2  9  8
1  >2< 9  8
1  2  >9  8<
1  2
1*  2
>1  2<
>2  1<

          17  45      zerlege
          17* 45      markiere (i 5)
          >17 45<    vertausche (i 5<>6)
          >45 17<    vertausche (i 5<>6)

```

Sortieralgorithmus nach Horare

```

1  45  8   53  9   2  17  10  zerlege
1  45  8   53* 9   2  17  10  markiere (i 3)
1  45  8   >53  9   2  17  10< vertausche (i 3<>7)
1  45  8   10  9   2  17      zerlege
1  45  8   10* 9   2  17      markiere (i 3)
1  >45  8   10  9   2<  17      vertausche (i 1<>5)
1  2  8   >10  9<  45  17      vertausche (i 3<>4)
1  2  8   9
1  2*  8   9
1  2
1*  2
      8   9      zerlege
      8*  9      markiere (i 2)
          10 45 17  zerlege
          10 45* 17 markiere (i 5)
          10 >45 17< vertausche (i 5<>6)

```


10	17	zerlege
10*	17	markiere (i 4)

66115 / 2016 / Herbst / Thema 2 / Aufgabe 7

- (a) Gegeben ist die Ausgabe der Methode **Partition** (s. Pseudocode), rekonstruieren Sie die Eingabe.

Konkret sollen Sie das Array $A = (_, _, 1, _, _)$ so vervollständigen, dass der Aufruf `Partition(A, 1, 5)` die Zahl 3 zurückgibt und nach dem Aufruf gilt, dass $A = (1, 2, 3, 4, 5)$ ist.

Geben Sie A nach jedem Durchgang der for-Schleife in **Partition** an.

Lösungsvorschlag

```

2  4  1  5  3  Eingabe
2  4  1  5  3  zerlege
2  4  1  5  3* markiere (i 4)
>2< 4  1  5  3  vertausche (i 0<>0)
2  >4  1< 5  3  vertausche (i 1<>2)
2  1  >4  5  3< vertausche (i 2<>4)
2  1
2  1*
>2  1<
      5  4 zerlege
      5  4* markiere (i 4)
      >5  4< vertausche (i 3<>4)
1  2  3  4  5  Ausgabe

```

- (b) Beweisen Sie die Korrektheit von **Partition** (z. B. mittels einer Schleifeninvarianten)!
- (c) Geben Sie für jede natürliche Zahl n eine Instanz I_n , der Länge n an, so dass `QuickSort(I_n)` $\Omega(n^2)$ Zeit benötigt. Begründen Sie Ihre Behauptung.

Lösungsvorschlag

$$I_n = 1, 2, 3, \dots, n$$

Die Methode **Partition** wird n mal aufgerufen, weil bei jedem Aufruf der Methode nur eine Zahl, nämlich die größte Zahl, abgespalten wird.

- `Partition(A, 1, n)`
- `Partition(A, 1, n - 1)`
- `Partition(A, 1, n - 2)`
- `Partition(A, 1, ...)`
- `Partition(A, 1, 1)`

In der For-Schleife der Methode Partition wird bei jeder Wiederholung ein Vertauschvorgang durchgeführt (Die Zahlen werden mit sich selbst getauscht.)

```

1 2 3 4 5 6 7 zerlege
1 2 3 4 5 6 7* markiere (i 6)
>1< 2 3 4 5 6 7 vertausche (i 0<>0)
1 >2< 3 4 5 6 7 vertausche (i 1<>1)
1 2 >3< 4 5 6 7 vertausche (i 2<>2)
1 2 3 >4< 5 6 7 vertausche (i 3<>3)
1 2 3 4 >5< 6 7 vertausche (i 4<>4)
1 2 3 4 5 >6< 7 vertausche (i 5<>5)
1 2 3 4 5 6 >7< vertausche (i 6<>6)
1 2 3 4 5 6 zerlege
1 2 3 4 5 6* markiere (i 5)
>1< 2 3 4 5 6 vertausche (i 0<>0)
1 >2< 3 4 5 6 vertausche (i 1<>1)
1 2 >3< 4 5 6 vertausche (i 2<>2)
1 2 3 >4< 5 6 vertausche (i 3<>3)
1 2 3 4 >5< 6 vertausche (i 4<>4)
1 2 3 4 5 >6< vertausche (i 5<>5)
1 2 3 4 5 zerlege
1 2 3 4 5* markiere (i 4)
>1< 2 3 4 5 vertausche (i 0<>0)
1 >2< 3 4 5 vertausche (i 1<>1)
1 2 >3< 4 5 vertausche (i 2<>2)
1 2 3 >4< 5 vertausche (i 3<>3)
1 2 3 4 >5< vertausche (i 4<>4)
1 2 3 4 zerlege
1 2 3 4* markiere (i 3)
>1< 2 3 4 vertausche (i 0<>0)
1 >2< 3 4 vertausche (i 1<>1)
1 2 >3< 4 vertausche (i 2<>2)
1 2 3 >4< vertausche (i 3<>3)
1 2 3 zerlege
1 2 3* markiere (i 2)
>1< 2 3 vertausche (i 0<>0)
1 >2< 3 vertausche (i 1<>1)
1 2 >3< vertausche (i 2<>2)
1 2 zerlege
1 2* markiere (i 1)
>1< 2 vertausche (i 0<>0)
1 >2< vertausche (i 1<>1)

```

- (d) Was müsste Partition (in Linearzeit) leisten, damit QuickSort Instanzen der Länge n in $\mathcal{O}(n \cdot \log n)$ Zeit sortiert? Zeigen Sie, dass Partition mit der von Ihnen geforderten Eigenschaft zur gewünschten Laufzeit von QuickSort führt.

Exkurs: Master-Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

a = Anzahl der rekursiven Aufrufe, Anzahl der Unterprobleme in der Rekursion ($a \geq 1$).

$\frac{1}{b}$ = Teil des Originalproblems, welches wiederum durch alle Unterprobleme repräsentiert wird, Anteil an der Verkleinerung des Problems ($b > 1$).

$f(n)$ = Kosten (Aufwand, Nebenkosten), die durch die Division des Problems und die Kombination der Teillösungen entstehen. Eine von $T(n)$ unabhängige und nicht negative Funktion.

Dann gilt:

1. Fall: $T(n) \in \Theta\left(n^{\log_b a}\right)$

falls $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$ für $\varepsilon > 0$

2. Fall: $T(n) \in \Theta\left(n^{\log_b a} \cdot \log n\right)$

falls $f(n) \in \Theta\left(n^{\log_b a}\right)$

3. Fall: $T(n) \in \Theta(f(n))$

falls $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ für $\varepsilon > 0$ und ebenfalls für ein c mit $0 < c < 1$ und alle hinreichend großen n gilt: $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$

Lösungsvorschlag

Die Methode **Partition** müsste die Instanzen der Länge n in zwei gleich große Teile spalten ($\frac{n-1}{2}$).

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

2

Anteil Verkleinerung des Problems (b):

um $\frac{1}{2}$ also $b = 2$

Laufzeit der rekursiven Funktion ($f(n)$):

n

Ergibt folgende Rekursionsgleichung:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$$

1. Fall: $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$:

für $\varepsilon = 4$:

$$f(n) = n \notin \mathcal{O}\left(n^{\log_2 2 - \varepsilon}\right)$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$f(n) = n \in \Theta(n^{\log_2 2}) = \Theta(n)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \epsilon})$:

$$f(n) = n \notin \Omega(n^{\log_2 2 + \epsilon})$$

$$\Rightarrow T(n) \in \Theta(n^{\log_2 2} \cdot \log n) = \Theta(n \cdot \log n)$$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

Funktion Quicksort($A, l = 1, r = A.length$)

if $l < r$ **then**

$m = \text{Partition}(A, l, r)$;

 Quicksort($A, l, m - 1$);

 Quicksort($A, m + 1, r$);

end

Funktion Partition($A, \text{int } l, \text{int } r$)

$\text{pivot} = A[r]$;

$i = l$;

for $j = l$ **to** $r - 1$ **do**

if $A[j] \leq \text{pivot}$ **then**

 Swap(A, i, j);

$i = i + 1$;

end

end

Funktion Swap($A, \text{int } l, \text{int } r$)

$\text{temp} = A[l]$;

$A[l] = A[r]$;

$A[r] = \text{temp}$;

66115 / 2017 / Frühjahr / Thema 1 / Aufgabe 2

In dieser Aufgabe sei vereinfachend angenommen, dass sich Top-Level-Domains (TLD) ausschließlich aus zwei oder drei der 26 Kleinbuchstaben des deutschen Alphabets ohne Umlaute zusammensetzen. Im Folgenden sollen TLDs lexikographisch aufsteigend sortiert werden, eine TLD (s_1, s_2) mit zwei Buchstaben (z. B. „co“ für Kolumbien) wird also vor einer TLD (t_1, t_2, t_3) der Länge drei (z. B. „com“) einsortiert, wenn $s_1 < t_1 \vee (s_1 = t_1 \wedge s_2 \leq t_2)$ gilt.

- (a) Sortieren Sie zunächst die Reihung [„de“, „com“, „uk“, „org“, „co“, „net“, „fr“, „ee“] schrittweise unter Verwendung des Radix-Sortierverfahrens (Bucketsort). Erstellen Sie dazu eine Tabelle wie das folgende Muster und tragen Sie dabei in das Feld „Stelle“ die Position des Buchstabens ein, nach dem im jeweiligen Durchgang sortiert wird (das Zeichen am TLD-Anfang habe dabei die „Stelle“ 1).

Exkurs: Alphabet

abcdefghijklmnopqrstuvwxyz

Lösungsvorschlag

Stelle	Reihung							
	de_	com	uk_	org	co_	net	fr_	ee_
3	de_	uk_	co_	fr_	ee_	org	com	net
2	de_	ee_	net	uk_	co_	com	fr_	org
1	co_	com	de_	ee_	fr_	net	org	uk_

- (b) Sortieren Sie nun die gleiche Reihung wieder schrittweise, diesmal jedoch unter Verwendung des Mergesort-Verfahrens (Sortieren durch Mischen). Erstellen Sie dazu eine Tabelle wie das folgende Muster und vermerken Sie in der ersten Spalte jeweils welche Operation durchgeführt wurde: Wenn Sie die Reihung geteilt haben, schreiben Sie in die linke Spalte ein T und markieren Sie die Stelle, an der Sie die Reihung geteilt haben, mit einem senkrechten Strich „|“. Wenn Sie zwei Teilreihungen durch Mischen zusammengeführt haben, schreiben Sie ein M in die linke Spalte und unterstreichen Sie die zusammengemischten Einträge. Beginnen Sie mit dem rekursiven Abstieg immer in der linken Hälfte einer (Teil-)Reihung.

0	Reihung							
T	de_	com	uk_	org		co_	net	fr_ ee_
T	de_	com		uk_	org			
T	de_		com					
M	com	de_						
T			uk_		org			
M			org	uk_				
M	com	de_	org	uk_				
T					co_	net		fr_ ee_
T					co_		net	
M					co_	net		
T							fr_	ee_
T							ee_	fr_
M					co_	ee_	fr_	net
M	co_	com	de_	ee_	fr_	net	org	uk_

- (c) Implementieren Sie das Sortierverfahren Quicksort für String-TLDs in einer gängigen Programmiersprache Ihrer Wahl. Ihr Programm (Ihre Methode) wird mit drei Parametern gestartet: dem String-Array mit den zu sortierenden TLDs selbst sowie jeweils der Position des ersten und des letzten zu sortierenden Eintrags im Array.

Lösungsvorschlag

```
public class Quicksort {

    public static void swap(String[] array, int index1, int index2) {
        String tmp = array[index1];
        array[index1] = array[index2];
        array[index2] = tmp;
    }

    public static int partition(String[] array, int first, int last) {
        int pivotIndex = (last + first) / 2;
        String pivotValue = array[pivotIndex];
        int pivotIndexFinal = first;
        swap(array, pivotIndex, last);
        for (int i = first; i < last; i++) {
            if (array[i].compareTo(pivotValue) < 0) {
                swap(array, i, pivotIndexFinal);
                pivotIndexFinal++;
            }
        }
        swap(array, last, pivotIndexFinal);
        return pivotIndexFinal;
    }

    public static void sort(String[] array, int first, int last) {
        if (first < last) {
            int pivotIndex = partition(array, first, last);
            sort(array, first, pivotIndex - 1);
            sort(array, pivotIndex + 1, last);
        }
    }

    public static void main(String[] args) {
        String[] array = new String[] { "de", "com", "uk", "org", "co", "net",
            ↪ "fr", "ee" };
        sort(array, 0, array.length - 1);
        for (int i = 0; i < array.length; i++) {
            System.out.println(array[i]);
        }
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2017/fruehjahr/Quicksort.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2017/fruehjahr/Quicksort.java)

66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 7

- (a) Gegeben ist das folgende Array von Zahlen: [23, 5, 4, 67, 30, 15, 25, 21].

Sortieren Sie das Array mittels Quicksort in-situ aufsteigend von links nach rechts. Geben Sie die (Teil-)Arrays nach jeder Swap-Operation (auch wenn Elemente mit sich selber getauscht werden) und am Anfang jedes Aufrufs der rekursiven Methode an. Verwenden Sie als Pivotelement jeweils das rechteste Element im Teilarray und markieren Sie dieses entsprechend. Teilarrays der Länge ≤ 2 dürfen im rekursiven Aufruf durch direkten Vergleich sortiert werden. Geben Sie am Ende das sortierte Array an.

- (b) Welche Worst-Case-Laufzeit (O-Notation) hat Quicksort für n Elemente? Geben Sie ein Array mit fünf Elementen an, in welchem die Quicksort-Variante aus (a) diese Worst-Case-Laufzeit benötigt (ohne Begründung).

66115 / 2018 / Herbst / Thema 2 / Aufgabe 8

Gegeben sei das folgende Feld A mit 7 Schlüsseln:

[15, 4, 10, 7, 1, 8, 10]

- (a) Sortieren Sie das Feld mittels des Sortierverfahrens *Bubblesort*. Markieren Sie jeweils, welche zwei Feldwerte verglichen werden und geben Sie den Zustand des gesamten Feldes jeweils neu an, wenn Sie eine Vertauschung durchgeführt haben.

Lösungsvorschlag

15	4	10	7	1	8	10	Eingabe
15	4	10	7	1	8	10	Durchlauf Nr. 1
>15	4<	10	7	1	8	10	vertausche (i 0<>1)
4	>15	10<	7	1	8	10	vertausche (i 1<>2)
4	10	>15	7<	1	8	10	vertausche (i 2<>3)
4	10	7	>15	1<	8	10	vertausche (i 3<>4)
4	10	7	1	>15	8<	10	vertausche (i 4<>5)
4	10	7	1	8	>15	10<	vertausche (i 5<>6)
4	10	7	1	8	10	15	Durchlauf Nr. 2
4	>10	7<	1	8	10	15	vertausche (i 1<>2)
4	7	>10	1<	8	10	15	vertausche (i 2<>3)
4	7	1	>10	8<	10	15	vertausche (i 3<>4)
4	7	1	8	10	10	15	Durchlauf Nr. 3
4	>7	1<	8	10	10	15	vertausche (i 1<>2)
4	1	7	8	10	10	15	Durchlauf Nr. 4
>4	1<	7	8	10	10	15	vertausche (i 0<>1)
1	4	7	8	10	10	15	Durchlauf Nr. 5
1	4	7	8	10	10	15	Ausgabe

- (b) Sortieren Sie das Feld mittels des Sortierverfahrens *Selectionsort*. Markieren Sie jeweils, welche zwei Feldwerte verglichen werden und geben Sie den Zustand des gesamten Feldes jeweils neu an, wenn Sie eine Vertauschung durchgeführt haben.

15	4	10	7	1	8	10	Eingabe
15	4	10	7	1	8	10*	markiere (i 6)
>15	4	10	7	1	8	10<	vertausche (i 0<>6)
10	4	10	7	1	8*	15	markiere (i 5)
>10	4	10	7	1	8<	15	vertausche (i 0<>5)
8	4	10	7	1*	10	15	markiere (i 4)
8	4	>10	7	1<	10	15	vertausche (i 2<>4)
8	4	1	7*	10	10	15	markiere (i 3)
>8	4	1	7<	10	10	15	vertausche (i 0<>3)
7	4	1*	8	10	10	15	markiere (i 2)
>7	4	1<	8	10	10	15	vertausche (i 0<>2)
1	4*	7	8	10	10	15	markiere (i 1)
1	>4	7	8	10	10	15	vertausche (i 1<>1)
1*	4	7	8	10	10	15	markiere (i 0)
>1	4	7	8	10	10	15	vertausche (i 0<>0)
1	4	7	8	10	10	15	Ausgabe

- (c) Vergleichen Sie beide Sortiervverfahren hinsichtlich ihres Laufzeitverhaltens im *best case*. Welches Verfahren ist in dieser Hinsicht besser, wenn das zu sortierende Feld anfangs bereits sortiert ist? Begründen Sie Ihre Antwort.

Der Bubblesort-Algorithmus hat im *best case* eine Laufzeit von $\mathcal{O}(n)$, der Selectionsort-Algorithmus $\mathcal{O}(n^2)$.

Bubblesort steuert seine äußere bedingte Wiederholung in vielen Implementationen über eine boolsche Hilfsvariable `getauscht`, die beim Betreten der Schleife erstmals auf falsch gesetzt wird. Erst wenn Vertauschungen vorgenommen werden müssen, wird diese Variable auf wahr gesetzt und die äußere Schleife läuft ein weiteres Mal ab. Ist das zu sortierende Feld bereits sortiert, durchsucht der Algorithmus des Bubblesort das Feld einmal und terminiert dann.

Der Selectionsort-Algorithmus hingegen ist mit zwei ineinander verschränkten Schleifen umgesetzt, deren Wiederholungsanzahl sich starr nach der Anzahl der Elemente im Feld richtet.

Bubblesort

```
int durchlaufNr = 0;
boolean getauscht;
do {
    durchlaufNr++;
    berichte.feld("Durchlauf Nr. " + durchlaufNr);
    getauscht = false;
    for (int i = 0; i < zahlen.length - 1; i++) {
        if (zahlen[i] > zahlen[i + 1]) {
            // Elemente vertauschen
        }
    }
}
```

```

        vertausche(i, i + 1);
        getauscht = true;
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/sortier/BubbleIterativ.java](https://github.com/orgs/bschlangaul/repositories)

Selectionsort

```

// Am Anfang ist die Markierung das letzte Element im Zahlen-Array.
int markierung = zahlen.length - 1;
while (markierung >= 0) {
    berichte.feldMarkierung(markierung);
    // Bestimme das größtes Element.
    // max ist der Index des größten Elements.
    int max = 0;
    // Wir vergleichen zuerst die Zahlen mit der Index-Number
    // 0 und 1, dann 1 und 2, etc. bis zur Markierung
    for (int i = 1; i <= markierung; i++) {
        if (zahlen[i] > zahlen[max]) {
            max = i;
        }
    }

    // Tausche zahlen[markierung] mit dem gefundenem Element.
    vertausche(markierung, max);
    // Die Markierung um eins nach vorne verlegen.
    markierung--;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/sortier/SelectionRechtsIterativ.java](https://github.com/orgs/bschlangaul/repositories)

66115 / 2019 / Herbst / Thema 1 / Aufgabe 5

In der folgenden Aufgabe soll ein Feld A von ganzen Zahlen aufsteigend sortiert werden. Das Feld habe n Elemente A[0] bis A[n-1]. Der folgende Algorithmus (in der Notation des Informatik-Duden) sei gegeben:

```

procedure quicksort(links, rechts : integer)
var i, j, x : integer;
begin
    i := links;
    j := rechts;
    if j > i then begin
        x := A[links];
        repeat
            while A[i] < x do i := i+1;
            while A[j] > x do j := j-1;
            if i < j then begin
                tmp := A[i]; A[i] := A[j]; A[j] := tmp;
                i := i+1; j := j-1;
            end
        until i > j;
    end
end

```

```
    quicksort(links, j);
    quicksort(i, rechts);
end
end
```

Umsetzung in Java:

```
public static void quicksort(int[] A, int links, int rechts) {
    System.out.println("quick");
    int i = links;
    int j = rechts;
    if (j > i) {
        int x = A[links];
        do {
            while (A[i] < x) {
                i = i + 1;
            }
            while (A[j] > x) {
                j = j - 1;
            }
            if (i <= j) {
                int tmp = A[i];
                A[i] = A[j];
                A[j] = tmp;
                i = i + 1;
                j = j - 1;
            }
            // Java verfügt über keine do-until Schleife.
            // Wir verwenden eine do-while-Schleife mit einem umgedrehten Test
            // until i > j -> while (i <= j)
        } while (i <= j);
        quicksort(A, links, j);
        quicksort(A, i, rechts);
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/beschlangaul/examen/examen_66115/jahr_2019/herbst/QuickSort.java](https://github.com/src/main/java/org/beschlangaul/examen/examen_66115/jahr_2019/herbst/QuickSort.java)

Der initiale Aufruf der Prozedur lautet:

`quicksort(0,n-1)`

- (a) Sortieren Sie das folgende Feld der Länge 7 mittels des Algorithmus. Notieren Sie jeweils alle Aufrufe der Prozedur quicksort mit den konkreten Parameterwerten. Geben Sie zudem für jeden Aufruf der Prozedur den Wert des in Zeile 7 gewählten Elements an.

27 13 21 3 6 17 44 42

Lösungsvorschlag

```
quicksort(0, 6)
27 32 3 6 17 44 42
x: 27
quicksort(0, 2)
17 6 3 32 27 44 42
x: 17
quicksort(0, 1)
```

```
3 6 17 32 27 44 42
x: 3
quicksort(0, -1)
3 6 17 32 27 44 42
quicksort(1, 1)
3 6 17 32 27 44 42
quicksort(2, 2)
3 6 17 32 27 44 42
quicksort(3, 6)
3 6 17 32 27 44 42
x: 32
quicksort(3, 3)
3 6 17 27 32 44 42
quicksort(4, 6)
3 6 17 27 32 44 42
x: 32
quicksort(4, 3)
3 6 17 27 32 44 42
quicksort(5, 6)
3 6 17 27 32 44 42
x: 44
quicksort(5, 5)
3 6 17 27 32 42 44
quicksort(6, 6)
3 6 17 27 32 42 44
3 6 17 27 32 42 44
```

- (b) Angenommen, die Bedingung $j > i$ in Zeile 6 des Algorithmus wird ersetzt durch die Bedingung $j \geq i$. Ist der Algorithmus weiterhin korrekt? Begründen Sie Ihre Antwort.

geht, dauert aber länger

```
quicksort(0, 6)
27 32 3 6 17 44 42
x: 27
quicksort(0, 2)
17 6 3 32 27 44 42
x: 17
quicksort(0, 1)
3 6 17 32 27 44 42
x: 3
quicksort(0, -1)
3 6 17 32 27 44 42
quicksort(1, 1)
3 6 17 32 27 44 42
x: 6
quicksort(1, 0)
3 6 17 32 27 44 42
quicksort(2, 1)
3 6 17 32 27 44 42
quicksort(2, 2)
3 6 17 32 27 44 42
x: 17
```

```
quicksort(2, 1)
3 6 17 32 27 44 42
quicksort(3, 2)
3 6 17 32 27 44 42
quicksort(3, 6)
3 6 17 32 27 44 42
x: 32
quicksort(3, 3)
3 6 17 27 32 44 42
x: 27
quicksort(3, 2)
3 6 17 27 32 44 42
quicksort(4, 3)
3 6 17 27 32 44 42
quicksort(4, 6)
3 6 17 27 32 44 42
x: 32
quicksort(4, 3)
3 6 17 27 32 44 42
quicksort(5, 6)
3 6 17 27 32 44 42
x: 44
quicksort(5, 5)
3 6 17 27 32 42 44
x: 42
quicksort(5, 4)
3 6 17 27 32 42 44
quicksort(6, 5)
3 6 17 27 32 42 44
quicksort(6, 6)
3 6 17 27 32 42 44
x: 44
quicksort(6, 5)
3 6 17 27 32 42 44
quicksort(7, 6)
3 6 17 27 32 42 44
3 6 17 27 32 42 44
```

- (c) Angenommen, die Bedingung $i \leq j$ in Zeile 11 des Algorithmus wird ersetzt durch die Bedingung $i < j$. Ist der Algorithmus weiterhin korrekt? Begründen Sie Ihre Antwort.

Lösungsvorschlag

bleibt hängen

```
quicksort(0, 6)
27 32 3 6 17 44 42
x: 27
quicksort(0, 2)
17 6 3 32 27 44 42
x: 17
quicksort(0, 1)
3 6 17 32 27 44 42
x: 3
```

- (d) Wie muss das Feld A gestaltet sein, damit der Algorithmus mit der geringsten Anzahl von Schritten terminiert? Betrachten Sie dazu vor allem Zeile 7. Begründen Sie Ihre Antwort und geben Sie ein Beispiel.

Lösungsvorschlag

Im Worst Case (schlechtesten Fall) wird das Pivotelement stets so gewählt, dass es das größte oder das kleinste Element der Liste ist. Dies ist etwa der Fall, wenn als Pivotelement stets das Element am Ende der Liste gewählt wird und die zu sortierende Liste bereits sortiert vorliegt. Die zu untersuchende Liste wird dann in jedem Rekursionsschritt nur um eins kleiner und die Zeitkomplexität wird beschrieben durch $\mathcal{O}(n^2)$. Die Anzahl der Vergleiche ist in diesem Fall $\frac{n \cdot (n+1)}{2} - 1 = \frac{n^2}{2} + \frac{n}{2} - 1$.

Die Länge der jeweils längeren Teilliste beim rekursiven Aufrufe ist nämlich im Schnitt $\frac{2}{n} \sum_{i=\frac{n}{2}}^{n-1} i = \frac{3}{4}n - \frac{2}{4}$ und die Tiefe der Rekursion damit in $\mathcal{O}(\log(n))$.

Im Average Case ist die Anzahl der Vergleiche etwa $2 \cdot \log(2) \cdot (n+1) \cdot \log_2(n) \approx 1,39 \cdot (n+1) \cdot \log_2(n)$.

- (e) Die rekursiven Aufrufe in den Zeilen 16 und 17 des Algorithmus werden zur Laufzeit des Computers auf dem Stack verwaltet. Die Anzahl der Aufrufe von quicksort auf dem Stack abhängig von der Eingabegröße n sei mit $s(n)$ bezeichnet. Geben Sie die Komplexitätsklasse von $s(n)$ für den schlimmsten möglichen Fall an. Begründen Sie Ihre Antwort.

66115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 1

- (a) Geben Sie für folgende Sortierverfahren jeweils zwei Felder A und B an, so dass das jeweilige Sortierverfahren angewendet auf A seine Best-Case-Laufzeit und angewendet auf B seine Worst-Case-Laufzeit erreicht. (Wir messen die Laufzeit durch die Anzahl der Vergleiche zwischen Elementen der Eingabe.) Dabei soll das Feld A die Zahlen $1, 2, \dots, 7$ genau einmal enthalten; das Feld B ebenso. Sie bestimmen also nur die Reihenfolge der Zahlen.

Wenden Sie als Beleg für Ihre Aussagen das jeweilige Sortierverfahren auf die Felder A und B an und geben Sie nach jedem größeren Schritt des Algorithmus den Inhalt der Felder an.

Geben Sie außerdem für jedes Verfahren asymptotische Best- und Worst-Case-Laufzeit für ein Feld der Länge n an.

Die im Pseudocode verwendete Unterroutine $\text{Swap}(A, i, j)$ vertauscht im Feld A die jeweiligen Elemente mit den Indizes i und j miteinander.

- (i) **Insertionsort**

Lösungsvorschlag

Best-Case

1	2	3	4	5	6	7
---	---	---	---	---	---	---

```

1  2  3  4  5  6  7  Eingabe
1  2* 3  4  5  6  7  markiere (i 1)
1  2  3* 4  5  6  7  markiere (i 2)
1  2  3  4* 5  6  7  markiere (i 3)
1  2  3  4  5* 6  7  markiere (i 4)
1  2  3  4  5  6* 7  markiere (i 5)
1  2  3  4  5  6  7* markiere (i 6)
1  2  3  4  5  6  7  Ausgabe

```

Worst-Case

7	6	5	4	3	2	1
---	---	---	---	---	---	---

```

7  6  5  4  3  2  1  Eingabe
7  6* 5  4  3  2  1  markiere (i 1)
>7  7< 5  4  3  2  1  vertausche (i 0<>1)
6  7  5* 4  3  2  1  markiere (i 2)
6 >7  7< 4  3  2  1  vertausche (i 1<>2)
>6  6< 7  4  3  2  1  vertausche (i 0<>1)
5  6  7  4* 3  2  1  markiere (i 3)
5  6 >7  7< 3  2  1  vertausche (i 2<>3)
5 >6  6< 7  3  2  1  vertausche (i 1<>2)
>5  5< 6  7  3  2  1  vertausche (i 0<>1)
4  5  6  7  3* 2  1  markiere (i 4)
4  5  6 >7  7< 2  1  vertausche (i 3<>4)
4  5 >6  6< 7  2  1  vertausche (i 2<>3)
4 >5  5< 6  7  2  1  vertausche (i 1<>2)
>4  4< 5  6  7  2  1  vertausche (i 0<>1)
3  4  5  6  7  2* 1  markiere (i 5)
3  4  5  6 >7  7< 1  vertausche (i 4<>5)
3  4  5 >6  6< 7  1  vertausche (i 3<>4)
3  4 >5  5< 6  7  1  vertausche (i 2<>3)
3 >4  4< 5  6  7  1  vertausche (i 1<>2)
>3  3< 4  5  6  7  1  vertausche (i 0<>1)
2  3  4  5  6  7  1* markiere (i 6)
2  3  4  5  6 >7  7< vertausche (i 5<>6)
2  3  4  5 >6  6< 7  vertausche (i 4<>5)
2  3  4 >5  5< 6  7  vertausche (i 3<>4)
2  3 >4  4< 5  6  7  vertausche (i 2<>3)
2 >3  3< 4  5  6  7  vertausche (i 1<>2)
>2  2< 3  4  5  6  7  vertausche (i 0<>1)
1  2  3  4  5  6  7  Ausgabe

```

(ii) Standardversion von **Quicksort** (Pseudocode s.u., Feldindizes beginnen bei

1), bei der das letzte Element eines Teilfeldes als Pivot-Element gewählt wird. Quicksort

Funktion Quicksort($A, l = 1, r = A.length$)
<pre>if $l < r$ then $m = \text{Partition}(A, l, r);$ Quicksort($A, l, m - 1$); Quicksort($A, m + 1, r$); end</pre>

Funktion Partition($A, \text{int } l, \text{int } r$)
<pre>pivot = $A[r]$; $i = l$; for $j = l$ to $r - 1$ do if $A[j] < \text{pivot}$ then Swap(A, i, j); $i = i + 1$; end end</pre>

Best-Case

1	3	2	6	5	7	4
---	---	---	---	---	---	---

```

1 3 2 6 5 7 4 zerlege
1 3 2 6 5 7 4* markiere (i 6)
>1< 3 2 6 5 7 4 vertausche (i 0<>0)
1 >3< 2 6 5 7 4 vertausche (i 1<>1)
1 3 >2< 6 5 7 4 vertausche (i 2<>2)
1 3 2 >6 5 7 4< vertausche (i 3<>6)
1 3 2
1 3 2* zerlege
>1< 3 2 markiere (i 2)
1 >3 2< vertausche (i 0<>0)
1 >3 2< vertausche (i 1<>2)
5 7 6 zerlege
5 7 6* markiere (i 6)
>5< 7 6 vertausche (i 4<>4)
5 >7 6< vertausche (i 5<>6)

```

Worst-Case

7	6	5	4	3	2	1
---	---	---	---	---	---	---

```

1 2 3 4 5 6 7 zerlege
1 2 3 4 5 6 7* markiere (i 6)
>1< 2 3 4 5 6 7 vertausche (i 0<>0)
1 >2< 3 4 5 6 7 vertausche (i 1<>1)
1 2 >3< 4 5 6 7 vertausche (i 2<>2)
1 2 3 >4< 5 6 7 vertausche (i 3<>3)
1 2 3 4 >5< 6 7 vertausche (i 4<>4)
1 2 3 4 5 >6< 7 vertausche (i 5<>5)
1 2 3 4 5 6 >7< vertausche (i 6<>6)
1 2 3 4 5 6 zerlege
1 2 3 4 5 6* markiere (i 5)
>1< 2 3 4 5 6 vertausche (i 0<>0)
1 >2< 3 4 5 6 vertausche (i 1<>1)
1 2 >3< 4 5 6 vertausche (i 2<>2)
1 2 3 >4< 5 6 vertausche (i 3<>3)
1 2 3 4 >5< 6 vertausche (i 4<>4)
1 2 3 4 5 >6< vertausche (i 5<>5)
1 2 3 4 5 zerlege
1 2 3 4 5* markiere (i 4)
>1< 2 3 4 5 vertausche (i 0<>0)
1 >2< 3 4 5 vertausche (i 1<>1)
1 2 >3< 4 5 vertausche (i 2<>2)
1 2 3 >4< 5 vertausche (i 3<>3)
1 2 3 4 >5< vertausche (i 4<>4)
1 2 3 4 zerlege
1 2 3 4* markiere (i 3)
>1< 2 3 4 vertausche (i 0<>0)
1 >2< 3 4 vertausche (i 1<>1)
1 2 >3< 4 vertausche (i 2<>2)
1 2 3 >4< vertausche (i 3<>3)
1 2 3 zerlege
1 2 3* markiere (i 2)
>1< 2 3 vertausche (i 0<>0)
1 >2< 3 vertausche (i 1<>1)
1 2 >3< vertausche (i 2<>2)
1 2 zerlege
1 2* markiere (i 1)
>1< 2 vertausche (i 0<>0)
1 >2< vertausche (i 1<>1)

```

- (iii) **QuicksortVar**: Variante von Quicksort, bei der immer das mittlere Element eines Teilfeldes als Pivot-Element gewählt wird (Pseudocode s.u., nur eine Zeile neu).

Bei einem Aufruf von PartitionVar auf ein Teilfeld $A[l \dots r]$ wird also erst mithilfe der Unteroutine Swap $A \left[\left\lfloor \frac{l+r-1}{2} \right\rfloor \right]$ mit $A[r]$ vertauscht.

Funktion QuicksortVar($A, l = 1, r = A.length$)

```

if  $l < r$  then
     $m = \text{PartitionVar}(A, l, r);$ 
     $\text{QuicksortVar}(A, l, m - 1);$ 
     $\text{QuicksortVar}(A, m + 1, r);$ 
end

```

Funktion PartitionVar($A, \text{int } l, \text{int } r$)

```

 $\text{Swap}(A, \lfloor \frac{l+r-1}{2} \rfloor, r);$ 
 $\text{pivot} = A[r];$ 
 $i = l;$ 
for  $j = l$  to  $r - 1$  do
    if  $A[j] < \text{pivot}$  then
         $\text{Swap}(A, i, j);$ 
         $i = i + 1;$ 
    end
end

```

Lösungsvorschlag

Best-Case

1	2	3	4	5	6	7
---	---	---	---	---	---	---

```

1 2 3 4 5 6 7 zerlege
1 2 3 4* 5 6 7 markiere (i 3)
1 2 3 >4 5 6 7< vertausche (i 3<>6)
>1< 2 3 7 5 6 4 vertausche (i 0<>0)
1 >2< 3 7 5 6 4 vertausche (i 1<>1)
1 2 >3< 7 5 6 4 vertausche (i 2<>2)
1 2 3 >7 5 6 4< vertausche (i 3<>6)
1 2 3
1 2* 3 zerlege
1 >2 3< markiere (i 1)
>1< 3 2 vertausche (i 1<>2)
1 >3 2< vertausche (i 0<>0)
1 >3 2< vertausche (i 1<>2)
5 6 7 zerlege
5 6* 7 markiere (i 5)
5 >6 7< vertausche (i 5<>6)
>5< 7 6 vertausche (i 4<>4)
5 >7 6< vertausche (i 5<>6)
1 2 3 4 5 6 7 Ausgabe

```

Worst-Case

2	4	6	7	1	5	3
---	---	---	---	---	---	---

```

2 4 6 7 1 5 3 zerlege
2 4 6 7* 1 5 3 markiere (i 3)
2 4 6 >7 1 5 3< vertausche (i 3<>6)
>2< 4 6 3 1 5 7 vertausche (i 0<>0)
2 >4< 6 3 1 5 7 vertausche (i 1<>1)
2 4 >6< 3 1 5 7 vertausche (i 2<>2)
2 4 6 >3< 1 5 7 vertausche (i 3<>3)
2 4 6 3 >1< 5 7 vertausche (i 4<>4)
2 4 6 3 1 >5< 7 vertausche (i 5<>5)
2 4 6 3 1 5 >7< vertausche (i 6<>6)
2 4 6 3 1 5 zerlege
2 4 6* 3 1 5 markiere (i 2)
2 4 >6 3 1 5< vertausche (i 2<>5)
>2< 4 5 3 1 6 vertausche (i 0<>0)
2 >4< 5 3 1 6 vertausche (i 1<>1)
2 4 >5< 3 1 6 vertausche (i 2<>2)
2 4 5 >3< 1 6 vertausche (i 3<>3)
2 4 5 3 >1< 6 vertausche (i 4<>4)
2 4 5 3 1 >6< vertausche (i 5<>5)
2 4 5 3 1 zerlege
2 4 5* 3 1 markiere (i 2)
2 4 >5 3 1< vertausche (i 2<>4)
>2< 4 1 3 5 vertausche (i 0<>0)
2 >4< 1 3 5 vertausche (i 1<>1)
2 4 >1< 3 5 vertausche (i 2<>2)
2 4 1 >3< 5 vertausche (i 3<>3)
2 4 1 3 >5< vertausche (i 4<>4)
2 4 1 3 zerlege
2 4* 1 3 markiere (i 1)
2 >4 1 3< vertausche (i 1<>3)
>2< 3 1 4 vertausche (i 0<>0)
2 >3< 1 4 vertausche (i 1<>1)
2 3 >1< 4 vertausche (i 2<>2)
2 3 1 >4< vertausche (i 3<>3)
2 3 1 zerlege
2 3* 1 markiere (i 1)
2 >3 1< vertausche (i 1<>2)
>2< 1 3 vertausche (i 0<>0)
2 >1< 3 vertausche (i 1<>1)
2 1 >3< vertausche (i 2<>2)
2 1 zerlege
2* 1 markiere (i 0)

```

>2 1<	vertausche (i 0<>1)
>1< 2	vertausche (i 0<>0)
1 >2<	vertausche (i 1<>1)

Mergesort
Datenflussorientiertes Testen
Bubblesort
Datenfluss-annotierter
Kontrollflussgraph

(b) Geben Sie die asymptotische Best- und Worst-Case-Laufzeit von **Mergesort** an.

Lösungsvorschlag

Best-Case: $\mathcal{O}(n \cdot \log(n))$
Worst-Case: $\mathcal{O}(n^2)$

66116 / 2016 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 3

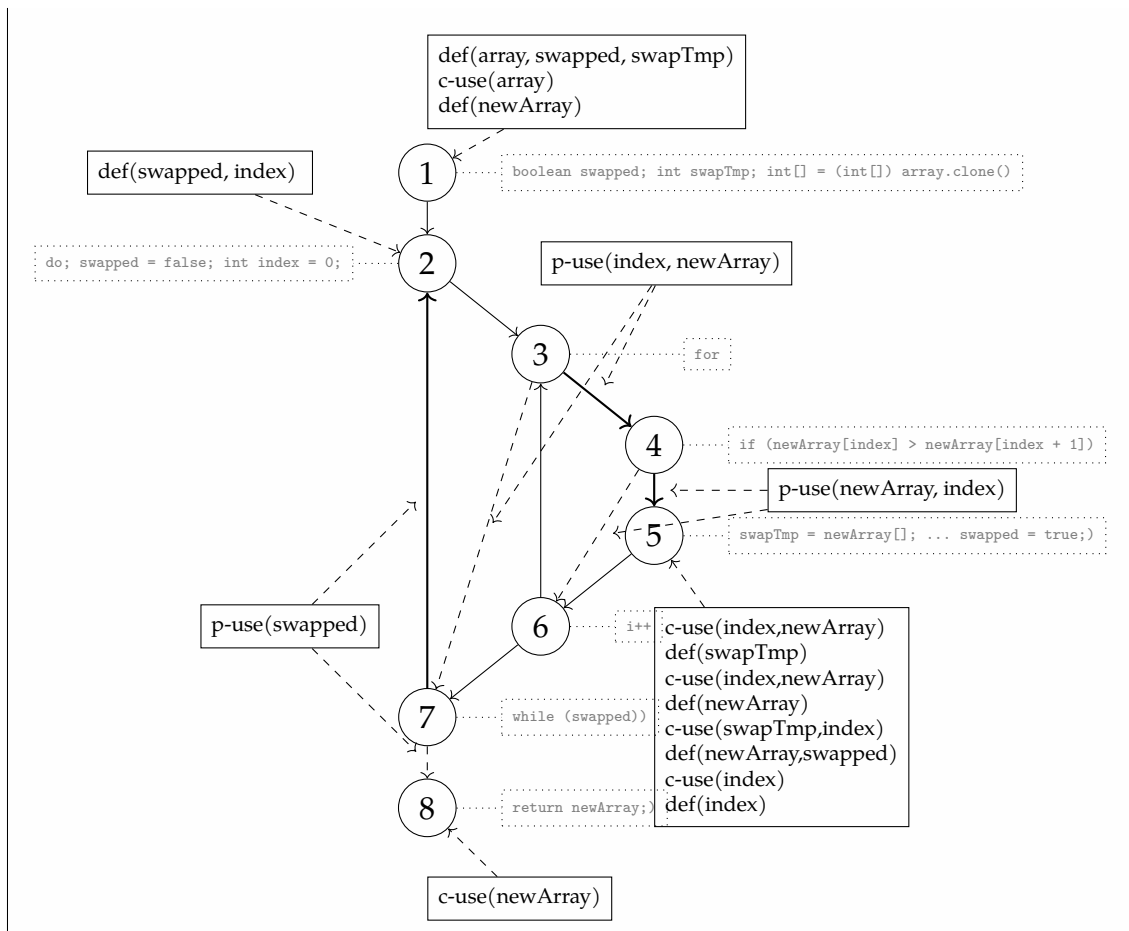
Gegeben Sei folgende Java-Methode `sort` zum Sortieren eines Feldes ganzer Zahlen:

```
public static int[] sort(int[] array) {
    boolean swapped;
    int swapTmp;
    int[] newArray = (int[]) array.clone();
    do {
        swapped = false;
        for (int index = 0; index < newArray.length - 1; index++) {
            if (newArray[index] > newArray[index + 1]) {
                swapTmp = newArray[index];
                newArray[index] = newArray[index + 1];
                newArray[index + 1] = swapTmp;
                swapped = true;
            }
        }
    } while (swapped);
    return newArray;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2016/herbst/BubbleSort.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2016/herbst/BubbleSort.java)

(a) Konstruieren Sie den Kontrollflussgraphen des obigen Code-Fragments und annotieren Sie an den Knoten und Kanten die zugehörigen Datenflussinformationen (Definitionen bzw. berechnende oder prädikative Verwendung von Variablen).

Lösungsvorschlag



Zyklomatische Komplexität
nach McCabe
C1-Test Zweigüberdeckung
(Branch Coverage)
all uses

- (b) Nennen Sie die maximale Anzahl linear unabhängiger Programmpfade, also die zyklomatische Komplexität nach McCabe.

Lösungsvorschlag

Der Graph hat 8 Knoten und 10 Kanten. Daher ist die zyklomatische Komplexität nach McCabe gegeben durch $10 - 8 + 2 = 4$.

- (c) Geben Sie einen möglichst kleinen Testdatensatz an, der eine 100%-ige Verzweigungsüberdeckung dieses Moduls erzielt.

Lösungsvorschlag

Die Eingabe muss mindestens ein Feld der Länge 3 sein. Ansonsten wäre das Feld schon sortiert bzw. bräuchte nur eine Vertauschung und die innere if-Bedingung wäre nicht zu 100% überdeckt. Daher wählt man beispielsweise `array = [1,3,2]`.

- (d) Beschreiben Sie kurz, welche Eigenschaften eine Testfallmenge allgemein haben muss, damit das datenflussorientierte Überdeckungskriterium „all-uses“ erfüllt.

Das Kriterium all-uses ist das Hauptkriterium des datenflussorientierten Testens, denn es testet den kompletten prädikativen und berechnenden Datenfluss. Konkret: von jedem Knoten mit einem globalen $\text{def}(x)$ einer Variable x

existiert ein definitions-freier Pfad bzgl. x ($\text{def-clear}(x)$) zu jedem erreichbaren Knoten mit einem $\text{c-use}(x)$ oder $\text{p-use}(x)$.

Algorithmische Komplexität (O-Notation)

66116 / 2016 / Herbst / Thema 1 / Aufgabe 3

Seien **A** und **B** zwei Algorithmen, die dasselbe Problem lösen. Zur Lösung von Problemen der Eingabegröße n benötigt Algorithmus **A** $500 \cdot n^2 - 16 \cdot n$ Elementoperationen und **B** $\frac{1}{2} \cdot n^3 + \frac{11}{2} \cdot n + 7$ Elementoperationen.

$$A(n) = 500 \cdot n^2 - 16 \cdot n$$

$$B(n) = \frac{1}{2} \cdot n^3 + \frac{11}{2} \cdot n + 7$$

- (a) Wenn Sie ein Problem für die Eingabegröße 256 lösen wollen, welchen Algorithmus würden Sie dann wählen?

Lösungsvorschlag

Wir können die Aufgabe durch Einsetzen lösen, d.h. wir berechnen explizit die Anzahl benötigter Elementoperationen und vergleichen. Algorithmus **A** benötigt $500 \cdot n^2 - 16 \cdot n$ Operationen bei einer Eingabe der Größe n , also bei $n = 256$ genau

$$\begin{aligned} A(256) &= 500 \cdot 256^2 - 16 \cdot 256 \\ &= 32763904 \end{aligned}$$

In der gleichen Art können wir den Aufwand von Algorithmus **B** berechnen:

$$\begin{aligned} B(256) &= \frac{1}{2} \cdot 256^3 + \frac{11}{2} \cdot 256 + 7 \\ &= 8390023 \end{aligned}$$

In diesem Fall benötigt Algorithmus **B** also deutlich weniger Elementoperationen.

- (b) Wenn Sie ein Problem lösen wollen, deren Eingabegröße immer mindestens 1024 ist, welchen Algorithmus würden Sie wählen? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Da in der Aufgabenstellung von Eingaben der Größe mindestens 1024 die Rede ist, stellen wir uns die Frage, für welche n Algorithmus **A** schneller als **B** ist, also für welche n

$$500 \cdot n^2 - 16 \cdot n < \frac{1}{2} \cdot n^3 + \frac{11}{2} \cdot n + 7$$

gilt. Dies lässt sich äquivalent umformen zu

$$\frac{1}{2} \cdot n^3 - 500 \cdot n^2 + \frac{43}{2} \cdot n + 7 > 0$$

Diese Ungleichung ist erfüllt, wenn allein $\frac{1}{2} \cdot n^3 - 500 \cdot n^2 > 0$ gilt, denn es gilt $\frac{43}{2} \cdot n + 7 > 0$. Das Problem reduziert sich also zu

$$\frac{1}{2} \cdot n^3 - 500 \cdot n^2 > 0 \Leftrightarrow n > 1000$$

Auf jeden Fall ist A schneller als B für $n > 1000$ (man erinnere sich, dass das bei $n = 256$ noch andersherum war). Wie ist dieses Verhalten zu erklären?

Obwohl der Aufwand von A im O-Kalkül $\mathcal{O}(n^2)$ und der von B $\mathcal{O}(n^3)$ ist, man also geneigt sein könnte zu sagen, A ist immer schneller als B, stimmt das nicht immer. Im Einzelfall können es durchaus große Konstanten (wie in diesem Fall die 500) sein, die dafür sorgen, dass n erst einmal sehr groß werden muss, damit sich die Laufzeiten tatsächlich so verhalten, wie erwartet. Wenn nur kleine Eingaben verarbeitet werden sollen, kann es manchmal also durchaus lohnenswert sein, einen $\mathcal{O}(n^3)$ -Algorithmus anstatt eines $\mathcal{O}(n^2)$ -Algorithmus zu verwenden, wenn die im O-Kalkül unterschlagenen Konstanten zuungunsten des eigentlich langsameren Algorithmus sprechen.

66116 / 2016 / Herbst / Thema 1 / Aufgabe 3

Der Konstruktor `QueueElement(...)` und die Methode `setNext(...)` sowie `getNext(...)` haben $\mathcal{O}(1)$. Geben Sie die Zeitkomplexität der Methode `append(int contents)` an, die einer Schlange ein neues Element anhängt.

```
public void append(int contents) {
    QueueElement newElement = new QueueElement(contents) ;
    if (first == 0) {
        first = newElement;
        last = newElement;
    } else {
        // Ein neues Element hinten anhängen.
        last.setNext(newElement);
        // Das angehängte Element als Letztes setzen.
        last = last.getNext();
    }
}
```

Lösungsvorschlag

Das Anhängen eines neuen Elements in die gegebene Warteschlange hat die konstanten Rechenzeitbedarf von $\mathcal{O}(1)$, egal wie lange die Schlange ist, da wir das letzte

Element direkt ansprechen können.

66116 / 2016 / Herbst / Thema 1 / Aufgabe 3

Welche Komplexität hat das Programmfragment?

```
public void magicStaff(int[] array) {
    for (int i = 0; i < array.length; i++) {
        int counter = 0;
        if (array[i] % 3 == 0) {
            break;
        }
        do {
            if (array[i] % 2 == 0) {
                array[i] += array[counter];
            }
        } while (counter++ < array.length);
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/komplexitaet/Komplexitaet.java](https://github.com/src/main/java/org/bschlangaul/aufgaben/aud/komplexitaet/Komplexitaet.java)

Bestimmen Sie in Abhängigkeit von n die Komplexität des Programmabschnitts im

(a) Best-Case.

Lösungsvorschlag

$\mathcal{O}(1)$: Wenn die erste Zahl im Feld `array` ohne Rest durch 3 teilbar ist, wird sofort aus der for-Schleife ausgestiegen (wegen der `break` Anweisung).

(b) Worst-Case.

Lösungsvorschlag

$\mathcal{O}(n^2)$: Wenn keine Zahl aus `array` ohne Rest durch 3 teilbar ist, werden zwei Schleifen (`for` und `do while`) über die Anzahl n der Elemente des Felds durchlaufen.

66116 / 2016 / Herbst / Thema 1 / Aufgabe 3

Gegeben sind die zwei Funktionen. Gilt $f(n) \in \Theta(g(n))$?¹

$$f(n) = 3n^5 + 4n^3 + 15$$

$$g(n) = n^5$$

¹https://www.informatik.hu-berlin.de/de/forschung/gebiete/wbi/teaching/archive/SS17/ue_algodat/schaefer01.pdf

(a) Zu zeigen: $f(n) \in \mathcal{O}(g(n)) \Leftrightarrow (\exists c, n_0 > 0 \forall n_0 \geq n_0 : 3n^5 + 4n^3 + 15 \leq c \cdot n^5)$

Wähle z. B. $c = 3 + 4 + 15 = 22$,

dann gilt $\forall n \geq 1 : 3n^5 + 4n^3 + 15 \leq 3n^5 + 4n^5 + 15n^5 \leq 22n^5$

$\Rightarrow f(n) \in \mathcal{O}(n^5)$

(b) Zu zeigen: $f(n) \in \Omega(g(n)) \Leftrightarrow (\exists c', n_0 > 0 \forall n_0 \geq n_0 : 3n^5 + 4n^3 + 15 \leq c' \cdot n^5)$

Wähle z. B. $c' = 3$,

dann gilt $\forall n \geq 1 : 3n^5 = 3n^5 + 4n^3 + 15 \leq 3n^5$

$\Rightarrow f(n) \in \Omega(n^5)$

$\Rightarrow f(n) \in \Theta(g(n))$

66116 / 2016 / Herbst / Thema 1 / Aufgabe 3

Geben Sie die Komplexität folgender Funktionen in der \mathcal{O} -Notation an!

(a) $x(n) = 4 \cdot n$

Lösungsvorschlag

 $\mathcal{O}(n)$

(b) $a(n) = n^2$

Lösungsvorschlag

 $\mathcal{O}(n^2)$

(c) $k(n) = 5 + n$

Lösungsvorschlag

 $\mathcal{O}(n)$

(d) $p(n) = 4$

Lösungsvorschlag

 $\mathcal{O}(1)$

(e) $j(n) = 4^n$

Lösungsvorschlag

 $\mathcal{O}(4^n)$

(f) $b(n) = \frac{n}{8}$

$\mathcal{O}(n)$

(g) $m(n) = \frac{1}{n}$

 $\mathcal{O}(1)$ **46115 / 2016 / Herbst / Thema 2 / Aufgabe 2**

Geben Sie jeweils die kleinste, gerade noch passende Laufzeitkomplexität folgender Java-Methoden im O-Kalkül (Landau-Notation) in Abhängigkeit von n und ggf. der Länge der Arrays an.

(a)

```
int matrixSumme(int n, int[][] feld) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            sum += feld[i][j];
        }
    }
    return sum;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/Komplexitaet.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2016/herbst/Komplexitaet.java)

Die Laufzeit liegt in $\mathcal{O}(n^2)$.

Begründung (nicht verlangt): Die äußere Schleife wird n -mal durchlaufen. Die innere Schleife wird dann jeweils wieder n -mal durchlaufen. Die Größe des Arrays spielt hier übrigens keine Rolle, da die Schleifen ohnehin immer nur bis zum Wert n ausgeführt werden.

(b)

```
int find(int key, int[][] keys) {
    int a = 0, o = keys.length;
    while (o - a > 1) {
        int m = (a + o) / 2;
        if (keys[m][0] > key)
            o = m;
        else
            a = m;
    }
    return a;
}
```

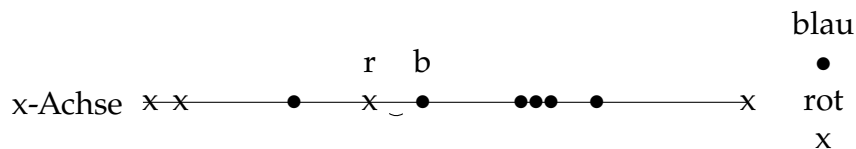
Lösungsvorschlag

Die Laufzeit liegt in $\mathcal{O}(\log(\text{keys.length}))$. Dabei ist keys.length die Größe des Arrays bezüglich seiner ersten Dimension.

Begründung (nicht verlangt): Der Grund für diese Laufzeit ist derselbe wie bei der binären Suche. Die Größe des Arrays bezüglich seiner zweiten Dimension spielt hier übrigens keine Rolle, da diese Dimension hier ja nur einen einzigen festen Wert annimmt.

46115 / 2020 / Frühjahr / Thema 2 / Aufgabe 6

Gegeben seien zwei nichtleere Mengen R und B von roten bzw. blauen Punkten auf der x -Achse. Gesucht ist der minimale euklidische Abstand $d(r, b)$ über alle Punktepaaire (r, b) mit $r \in R$ und $b \in B$. Hier ist eine Beispielinstantz:



Die Eingabe wird in einem Feld A übergeben. Jeder Punkt $A[i]$ mit $1 \leq i \leq n$ hat eine x -Koordinate $A[i].x$ und eine Farbe $A[i].color \in \{\text{rot}, \text{blau}\}$. Das Feld A ist nach x -Koordinate sortiert, des gilt $A[1].x < A[2].x < \dots < A[n].x$, wobei $n = |R| + |B|$.

- (a) Geben Sie in Worten einen Algorithmus an, der den gesuchten Abstand in $\mathcal{O}(n)$ Zeit berechnet.

Lösungsvorschlag

Pseudo-Code

Algorithmus 1: Minimaler Euklidischer Abstand

```

 $d_{min} := \max;$  // Setze  $d_{min}$  zuerst auf einen maximalen Wert.
for  $i$  in  $0 \dots \text{vorletzter Index}$  do ; // Iteriere über die Indizes des Punkte-Arrays
     $P$  bis zum vorletzten Index  $P[n-1]$ 

    if  $P[n].color \neq P[n+1].color$  then ; // Berechne den Abstand nur, wenn die
        Punkte unterschiedliche Farben haben

         $d = P[n+1].x - P[n].x$ 
        if  $d < d_{min}$  then
             $d_{min} = d$ 
        end
    end
end

```

Java

```

public double findMinimalDistance() {
    double distanceMin = Double.MAX_VALUE;
    for (int i = 0; i < latestIndex - 1; i++) {
        if (points[i].color != points[i + 1].color) {
            double distance = points[i + 1].x - points[i].x;
            if (distance < distanceMin) {
                distanceMin = distance;
            }
        }
    }
    return distanceMin;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/fruehjahr/RedBluePairCollection.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/fruehjahr/RedBluePairCollection.java)

- (b) Begründen Sie kurz die Laufzeit Ihres Algorithmus.

Lösungsvorschlag

Da das Array der Länge n nur einmal durchlaufen wird, ist die Laufzeit $\mathcal{O}(n)$ sichergestellt.

- (c) Begründen Sie die Korrektheit Ihres Algorithmus.

Lösungsvorschlag

In d_{min} steht am Ende der gesuchte Wert (sofern nicht $d_{min} = Integer.MAX_VALUE$ geblieben ist)

- (d) Wir betrachten nun den Spezialfall, dass alle blauen Punkte links von allen roten Punkten liegen. Beschreiben Sie in Worten, wie man in dieser Situation den gesuchten Abstand in $\mathcal{O}(n)$ Zeit berechnen kann. (Ihr Algorithmus darf also insbesondere nicht Laufzeit $\Theta(n)$ haben.)

Lösungsvorschlag

Zuerst müssen wir den letzten blauen Punkt finden. Das ist mit einer binären Suche möglich. Wir beginnen mit dem ganzen Feld als Suchbereich und betrachten den mittleren Punkt. Wenn er blau ist, wiederholen wir die Suche in der zweiten Hälfte des Suchbereichs, sonst in der ersten, bis wir einen blauen Punkt gefolgt von einem roten Punkt gefunden haben.

Der gesuchte minimale Abstand ist dann der Abstand zwischen dem gefundenen blauen und dem nachfolgenden roten Punkt. Die Binärsuche hat eine Worst-case-Laufzeit von $\mathcal{O}(\log n)$.

46115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 1

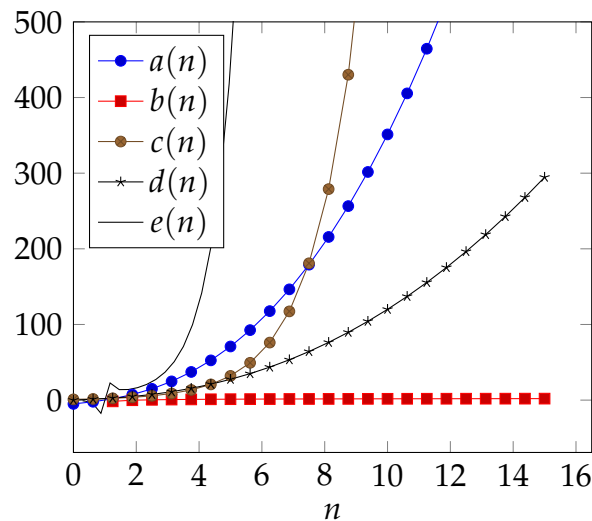
Sortieren Sie die unten angegebenen Funktionen der O-Klassen $\mathcal{O}(a)$, $\mathcal{O}(b)$, $\mathcal{O}(c)$, $\mathcal{O}(d)$ und $\mathcal{O}(e)$ bezüglich ihrer Teilmengenbeziehungen. Nutzen Sie ausschließlich die echte

Teilmenge \subsetneq sowie die Gleichheit $=$ für die Beziehung zwischen den Mengen. Folgendes Beispiel illustriert diese Schreibweise für einige Funktionen f_1 bis f_5 . (Diese haben nichts mit den unten angegebenen Funktionen zu tun.)

$$\mathcal{O}(f_4) \subsetneq \mathcal{O}(f_3) = \mathcal{O}(f_5) \subsetneq \mathcal{O}(f_1) = \mathcal{O}(f_2)$$

Die angegebenen Beziehungen müssen weder bewiesen noch begründet werden.

- $a(n) = \sqrt{n^5} + 4n - 5$
- $b(n) = \log_2(\log_2(n))$
- $c(n) = 2^n$
- $d(n) = n^2 \log(n) + 2n$
- $e(n) = \frac{4^n}{\log_2 n}$



66115 / 2011 / Frühjahr / Thema 1 / Aufgabe 1

Bestimmen Sie mit Hilfe des Master-Theorems für die folgenden Rekursionsgleichungen möglichst scharfe asymptotische untere und obere Schranken, falls das Master-Theorem anwendbar ist! Geben Sie andernfalls eine kurze Begründung, warum das Master-Theorem nicht anwendbar ist!

Exkurs: Master-Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

a = Anzahl der rekursiven Aufrufe, Anzahl der Unterprobleme in der Rekursion ($a \geq 1$).

$\frac{1}{b}$ = Teil des Originalproblems, welches wiederum durch alle Unterprobleme repräsentiert wird, Anteil an der Verkleinerung des Problems ($b > 1$).

$f(n)$ = Kosten (Aufwand, Nebenkosten), die durch die Division des Problems und die Kombination der Teillösungen entstehen. Eine von $T(n)$ unabhängige und nicht negative Funktion.

Dann gilt:

1. Fall: $T(n) \in \Theta(n^{\log_b a})$

falls $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$ für $\varepsilon > 0$

2. Fall: $T(n) \in \Theta(n^{\log_b a} \cdot \log n)$

falls $f(n) \in \Theta(n^{\log_b a})$

3. Fall: $T(n) \in \Theta(f(n))$

falls $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ für $\varepsilon > 0$ und ebenfalls für ein c mit $0 < c < 1$ und alle hinreichend großen n gilt: $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$

(a) $T(n) = 16 \cdot T(\frac{n}{2}) + 40n - 6$

Lösungsvorschlag

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T(\frac{n}{b}) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

$$16$$

Anteil Verkleinerung des Problems (b):

$$\text{um } \frac{1}{2} \text{ also } b = 2$$

Laufzeit der rekursiven Funktion ($f(n)$):

$$40n - 6$$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 16 \cdot T(\frac{n}{2}) + 40n - 6$$

1. Fall: $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$:

für $\varepsilon = 14$:

$$f(n) = 40n - 6 \in \mathcal{O}(n^{\log_2 16 - 14}) = \mathcal{O}(n^{\log_2 2}) = \mathcal{O}(n)$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$f(n) = 40n - 6 \notin \Theta(n^{\log_2 16}) = \Theta(n^4)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \varepsilon})$:

$$f(n) = 40n - 6 \notin \Omega(n^{\log_2 16 + \varepsilon})$$

$$\Rightarrow T(n) \in \Theta(n^4)$$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

(b) $T(n) = 27 \cdot T(\frac{n}{3}) + 3n^2 \log n$

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

$$27$$

Anteil Verkleinerung des Problems (b):

$$\text{um } \frac{1}{3} \text{ also } b = 3$$

Laufzeit der rekursiven Funktion (f(n)):

$$3n^2 \log n$$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 27 \cdot T\left(\frac{n}{3}\right) + 3n^2 \log n$$

1. Fall: $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$:

$$f(n) = 3n^2 \log n = n \in \mathcal{O}(n^{\log_3 27 - 24}) = \mathcal{O}(n^{\log_3 3}) = \mathcal{O}(n)$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$f(n) = 3n^2 \log n = n \notin \Theta(n^{\log_3 27}) = \Theta(n^3)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \varepsilon})$:

$$f(n) = 3n^2 \log n = n \notin \Omega(n^{\log_3 27 + \varepsilon})$$

$$\Rightarrow T(n) \in \Theta(n^3)$$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

$$(c) \quad T(n) = 4 \cdot T\left(\frac{n}{2}\right) + 3n^2 + \log n$$

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

4

Anteil Verkleinerung des Problems (b):um $\frac{1}{2}$ also $b = 2$ **Laufzeit der rekursiven Funktion ($f(n)$):**

$$3n^2 + \log n$$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + 3n^2 + \log n$$

1. Fall: $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$:

$$f(n) = 3n^2 + \log n = n^2 = n \notin \mathcal{O}(n^{\log_2 4 - \epsilon})$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$f(n) = 3n^2 + \log n = n^2 = n \in \Theta(n^{\log_2 4}) = \Theta(n^2)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \epsilon})$:

$$f(n) = 3n^2 + \log n = n^2 = n \notin \Omega(n^{\log_2 4 + \epsilon})$$

$$\Rightarrow T(n) \in \Theta(n^2 \log n)$$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

(d) $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + 100 \log n + \sqrt{2n} + n^{-2}$

Lösungsvorschlag

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

$$4$$

Anteil Verkleinerung des Problems (b):

$$\text{um } \frac{1}{2} \text{ also } b = 2$$

Laufzeit der rekursiven Funktion (f(n)):

$$100 \log n + \sqrt{2n} + n^{-2}$$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + 100 \log n + \sqrt{2n} + n^{-2}$$

1. Fall: $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$:

$$f(n) = 100 \log n + \sqrt{2n} + n^{-2} = n \in \mathcal{O}(n^{\log_2 4 - 2}) = \mathcal{O}(n)$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$f(n) = 100 \log n + \sqrt{2n} + n^{-2} = n \notin \Theta(n^{\log_2 4}) = \Theta(n^2)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \epsilon})$:

$$f(n) = 100 \log n + \sqrt{2n} + n^{-2} = n \notin \Omega(n^{\log_2 4 + \epsilon})$$

$$\Rightarrow T(n) \in \Theta(n^2)$$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

66115 / 2012 / Herbst / Thema 2 / Aufgabe 6

Gegeben seien die Funktionen $f: \mathbb{N} \rightarrow \mathbb{N}$ und $g: \mathbb{N} \rightarrow \mathbb{N}$, wobei $f(n) = (n-1)^3$ und $g(n) = (2n+3)(3n+2)$. Geben Sie an, welche der folgenden Aussagen gelten. Beweisen Sie Ihre Angaben.

(a) $f(n) \in \mathcal{O}(g(n))$

(b) $g(n) \in \mathcal{O}(f(n))$ **Exkurs: Regel von L'Hospital**

Die Regel von de L'Hospital ist ein Hilfsmittel zum Berechnen von Grenzwerten bei Brüchen $\frac{f}{g}$ von Funktionen f und g , wenn Zähler und Nenner entweder beide gegen 0 oder beide gegen (+ oder -) unendlich gehen. Wenn in einem solchen Fall auch der Grenzwert des Bruches der Ableitungen existiert, so hat dieser denselben Wert wie der ursprüngliche Grenzwert: ^a

$$\lim_{x \rightarrow x_0} \frac{f(x)}{g(x)} = \lim_{x \rightarrow x_0} \frac{f'(x)}{g'(x)}$$

^a<https://de.serlo.org/mathe/funktionen/grenzwerte-stetigkeit-differenzierbarkeit/grenzwert/regel-l-hospital>

Lösungsvorschlag

Es gilt Aussage (b), da $f(n) \in \mathcal{O}(n^3)$ und $g(n) \in \mathcal{O}(n^2)$ und der Grenzwert \lim bei größer werdendem n gegen ∞ geht. Damit wächst $f(n)$ stärker als $g(n)$, sodass nur Aussage (b) gilt und nicht (a). Dafür nutzen wir die formale Definition des \mathcal{O} -Kalküls, indem wir den Grenzwert $\frac{f}{g}$ bzw. $\frac{g}{f}$ bilden:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{(n-1)^3}{(2n+3)(3n+2)} = \lim_{n \rightarrow \infty} \frac{3(n-1)^2}{(2n+3) \cdot 3 + 2 \cdot (3n+2)} = \lim_{n \rightarrow \infty} \frac{6(n-1)}{12} = \infty$$

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \lim_{n \rightarrow \infty} \frac{(2n+3)(3n+2)}{(n-1)^3} = \lim_{n \rightarrow \infty} \frac{(2n+3) \cdot 3 + 2 \cdot (3n+2)}{3(n-1)^2} = \lim_{n \rightarrow \infty} \frac{12}{6(n-1)} = 0$$

Hinweis: Hierbei haben wir die Regel von L'Hospital angewendet.

66115 / 2014 / Herbst / Thema 2 / Aufgabe 5

Gegeben sei eine Standarddatenstruktur Stapel (Stack) mit den Operationen

- `void push(Element e)`
- `Element pop()`,
- `boolean isEmpty()`.

sowie dem Standardkonstruktor `Stapel()`, der einen leeren Stapel zur Verfügung stellt.

- (a) Geben Sie eine Methode `Stapel merge(Stapel s, Stapel t)` an, die einen aufsteigend geordneten Stapel zurückgibt, unter der Bedingung, dass die beiden übergebenen Stapel aufsteigend sortiert sind, d.h. `S.pop()` liefert das größte Element in `s` zurück und `T.pop()` liefert das größte Element in `t` zurück. Als Hilfsdatenstruktur dürfen Sie nur Stapel verwenden, keine Felder oder Listen.

Hinweis: Nehmen Sie an, dass Objekte der Klasse `Element`, die auf dem Stapel liegen mit `compareTo()` verglichen werden können. Zum Testen haben wir Ihnen

eine Klasse `StapelTest` zur Verfügung gestellt, sie können Ihre Methode hier einfügen und testen, ob die Stapel korrekt sortiert werden. Überlegen Sie auch, was geschieht, wenn einer der Stapel (oder beide) leer ist!

```

public static Stapel merge(Stack s, Stack t) {
    // Die beiden Stapel unsortiert aneinander hängen.
    Stack mergedStack = new Stack();
    while (!s.isEmpty()) {
        mergedStack.push(s.pop());
    }
    while (!t.isEmpty()) {
        mergedStack.push(t.pop());
    }
    // https://www.geeksforgeeks.org/sort-stack-using-temporary-stack/
    Stack tmpStack = new Stack();
    while (!mergedStack.isEmpty()) {
        Element tmpElement = mergedStack.pop();
        while (!tmpStack.isEmpty() && tmpStack.top().getValue() >
            tmpElement.getValue()) {
            mergedStack.push(tmpStack.pop());
        }
        tmpStack.push(tmpElement);
    }
    return tmpStack;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/Stapel.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/herbst/Stapel.java)

Komplette Klasse Stapel

```

/**
 * https://www.studon.fau.de/file2860857_download.html
 */
public class Stapel {
    public Element top;

    public Stapel() {
        top = null;
    }

    /**
     * @param element Das Element, dass hinzugefügt werden soll zur Stapel.
     */
    public void push(Element element) {
        element.setNext(top);
        top = element;
    }

    /**
     * @return Das Element oder null, wenn der Stapel leer ist.
     */
    public Element pop() {

```



```
        if (top == null) {
            return null;
        }
        Element element = top;
        top = top.getNext();
        return element;
    }

    /**
     * @return Wahr wenn der Stapel leer ist.
     */
    public boolean isEmpty() {
        return top == null;
    }

    /**
     * @param s Stapel s
     * @param t Stapel t
     *
     * @return Ein neuer Stapel.
     */
    public static Stapel merge(Stapel s, Stapel t) {
        // Die beiden Stapel unsortiert aneinander hängen.
        Stapel mergedStack = new Stapel();
        while (!s.isEmpty()) {
            mergedStack.push(s.pop());
        }
        while (!t.isEmpty()) {
            mergedStack.push(t.pop());
        }
        // https://www.geeksforgeeks.org/sort-stack-using-temporary-stack/
        Stapel tmpStack = new Stapel();
        while (!mergedStack.isEmpty()) {
            Element tmpElement = mergedStack.pop();
            while (!tmpStack.isEmpty() && tmpStack.top.getValue() >
                tmpElement.getValue()) {
                mergedStack.push(tmpStack.pop());
            }
            tmpStack.push(tmpElement);
        }
        return tmpStack;
    }

    public static void main(String[] args) {
        Stapel sa = new Stapel();
        sa.push(new Element(1));
        sa.push(new Element(2));
        sa.push(new Element(4));
        sa.push(new Element(5));
        sa.push(new Element(7));
        sa.push(new Element(8));
        Stapel sb = new Stapel();
        sb.push(new Element(2));
        sb.push(new Element(3));
    }
}
```

```
sb.push(new Element(6));
sb.push(new Element(9));
sb.push(new Element(10));

Stapel sc = Stapel.merge(sa, sb);

while (!sc.isEmpty()) {
    System.out.print(sc.pop().getValue() + ", ");
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/Stapel.java](https://github.com/bschlangaul/examen/blob/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/Stapel.java)

Komplette Klasse Element

```
/**
 * https://www.studon.fau.de/file2860856_download.html
 */
public class Element {
    public int value;

    public Element next;

    public Element() {
        this.next = null;
    }

    public Element(int value, Element element) {
        this.value = value;
        this.next = element;
    }

    public Element(int value) {
        this.value = value;
        this.next = null;
    }

    public int getValue() {
        return value;
    }

    public Element getNext() {
        return next;
    }

    public void setNext(Element element) {
        next = element;
    }

    public int compareTo(Element element) {
        if (getValue() > element.getValue()) {
```

```
        return 1;
    } else if (element.getValue() == getValue()) {
        return 0;
    } else {
        return -1;
    }
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/Element.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/herbst/Element.java)

Test-Klasse

```
import static org.junit.Assert.*;
import org.junit.Test;

/**
 * https://www.studon.fau.de/file2860850_download.html
 */
public class StapelTest {

    @Test
    public void testeMethodenPushPop() {
        Stapel stapel = new Stapel();
        stapel.push(new Element(1));
        stapel.push(new Element(2));
        stapel.push(new Element(3));

        assertEquals(3, stapel.pop().value);
        assertEquals(2, stapel.pop().value);
        assertEquals(1, stapel.pop().value);
    }

    @Test
    public void testeMethodeMerge() {
        Stapel sa = new Stapel();
        sa.push(new Element(1));
        sa.push(new Element(3));
        sa.push(new Element(5));

        Stapel sb = new Stapel();
        sb.push(new Element(2));
        sb.push(new Element(4));

        Stapel sc = Stapel.merge(sa, sb);

        assertEquals(5, sc.pop().getValue());
        assertEquals(4, sc.pop().getValue());
        assertEquals(3, sc.pop().getValue());
        assertEquals(2, sc.pop().getValue());
        assertEquals(1, sc.pop().getValue());
    }
}
```

```
}

@Test
public void testeMethodeMergeMehrWerte() {
    Stapel sa = new Stapel();
    sa.push(new Element(1));
    sa.push(new Element(2));
    sa.push(new Element(4));
    sa.push(new Element(5));
    sa.push(new Element(7));
    sa.push(new Element(8));
    Stapel sb = new Stapel();
    sb.push(new Element(2));
    sb.push(new Element(3));
    sb.push(new Element(6));
    sb.push(new Element(9));
    sb.push(new Element(10));

    Stapel sc = Stapel.merge(sa, sb);

    assertEquals(10, sc.pop().getValue());
    assertEquals(9, sc.pop().getValue());
    assertEquals(8, sc.pop().getValue());
    assertEquals(7, sc.pop().getValue());
    assertEquals(6, sc.pop().getValue());
    assertEquals(5, sc.pop().getValue());
    assertEquals(4, sc.pop().getValue());
    assertEquals(3, sc.pop().getValue());
    assertEquals(2, sc.pop().getValue());
    assertEquals(2, sc.pop().getValue());
    assertEquals(1, sc.pop().getValue());
}

@Test
public void testeMethodeMergeBLEer() {
    Stapel sa = new Stapel();
    sa.push(new Element(1));
    sa.push(new Element(3));
    sa.push(new Element(5));

    Stapel sb = new Stapel();

    Stapel sc = Stapel.merge(sa, sb);

    assertEquals(5, sc.pop().getValue());
    assertEquals(3, sc.pop().getValue());
    assertEquals(1, sc.pop().getValue());
}

@Test
public void testeMethodeMergeALEer() {
    Stapel sa = new Stapel();

    Stapel sb = new Stapel();
```

```

sb.push(new Element(2));
sb.push(new Element(4));

Stapel sc = Stapel.merge(sa, sb);

assertEquals(4, sc.pop().getValue());
assertEquals(2, sc.pop().getValue());
}

}

```

Code-Beispiel auf Github ansehen: src/test/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/StapelTest.java

(b) Analysieren Sie die Laufzeit Ihrer Methode.

Lösungsvorschlag

Best case: $\mathcal{O}(1)$ Worst case: $\mathcal{O}(n^2)$ **66115 / 2015 / Frühjahr / Thema 2 / Aufgabe 5**

Gegeben seien die Standardstrukturen Stapel (Stack) und Schlange (Queue) mit folgenden Standardoperationen:

Stapel	Schlange
<code>boolean isEmpty()</code>	<code>boolean isEmpty()</code>
<code>void push(int e)</code>	<code>enqueue(int e)</code>
<code>int pop()</code>	<code>int dequeue()</code>
<code>int top()</code>	<code>int head()</code>

Beim Stapel gibt die Operation `top()` das gleiche Element wie `pop()` zurück, bei der Schlange gibt `head()` das gleiche Element wie `dequeue()` zurück. Im Unterschied zu `pop()`, beziehungsweise `dequeue()`, wird das Element bei `top()` und `head()` nicht aus der Datenstruktur entfernt.

- (a) Geben Sie in Pseudocode einen Algorithmus `sort(Stapel s)` an, der als Eingabe einen Stapel `s` mit `n` Zahlen erhält und die Zahlen in `s` sortiert. (Sie dürfen die Zahlen wahlweise entweder aufsteigend oder absteigend sortieren.) Verwenden Sie als Hilfsdatenstruktur ausschließlich eine Schlange `q`. Sie erhalten volle Punktzahl, wenn Sie außer `s` und `q` keine weiteren Variablen benutzen. Sie dürfen annehmen, dass alle Zahlen in `s` verschieden sind.

Lösungsvorschlag

```

q := neue Schlange
while s not empty:
    q.enqueue(s.pop())
while q not empty:
    while s not empty and s.top() < q.head():
        q.enqueue(s.pop())
    s.push(q.dequeue)

```

Als Java-Code

```
/**
 * So ähnlich wie der <a href=
 * "https://www.geeksforgeeks.org/sort-stack-using-temporary-
 → stack/">Stapel-Sortiert-Algorithmus
 * der nur Stapel verwendet</a>, nur mit einer Warteschlange.
 *
 * @param s Der Stapel, der sortiert wird.
 */
public static void sort(Stack s) {
    Schlange q = new Schlange();
    while (!s.isEmpty()) {
        q.enqueue(s.pop());
    }
    while (!q.isEmpty()) {
        // Sortiert aufsteigend. Für absteigend einfach das „kleiner“
        // Zeichen umdrehen.
        while (!s.isEmpty() && s.top() < q.head()) {
            q.enqueue(s.pop());
        }
        s.push(q.dequeue());
    }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Sort.java

Klasse Sort

```
public class Sort {

    /**
     * So ähnlich wie der <a href=
     * "https://www.geeksforgeeks.org/sort-stack-using-temporary-
     → stack/">Stapel-Sortiert-Algorithmus
     * der nur Stapel verwendet</a>, nur mit einer Warteschlange.
     *
     * @param s Der Stapel, der sortiert wird.
     */
    public static void sort(Stack s) {
        Schlange q = new Schlange();
        while (!s.isEmpty()) {
            q.enqueue(s.pop());
        }
        while (!q.isEmpty()) {
            // Sortiert aufsteigend. Für absteigend einfach das „kleiner“
            // Zeichen umdrehen.
            while (!s.isEmpty() && s.top() < q.head()) {
                q.enqueue(s.pop());
            }
            s.push(q.dequeue());
        }
    }
}
```

```
public static Stapel stapelBefüllen(int[] zahlen) {
    Stapel s = new Stapel();
    for (int i : zahlen) {
        s.push(i);
    }
    return s;
}

public static void zeigeStapel(Stack s) {
    while (!s.isEmpty()) {
        System.out.print(s.pop() + " ");
    }
    System.out.println();
}

public static void main(String[] args) {
    Stapel s1 = stapelBefüllen(new int[] { 4, 2, 1, 5, 3 });
    sort(s1);
    zeigeStapel(s1);

    Stapel s2 = stapelBefüllen(new int[] { 1, 2, 6, 3, 9, 11, 4 });
    sort(s2);
    zeigeStapel(s2);
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Sort.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Sort.java)

Klasse Schlange

```
public class Schlange {

    public Element head;

    public Schlange() {
        head = null;
    }

    public int head() {
        if (head.getNext() == null) {
            return head.getValue();
        }
        Element element = head;
        Element previous = head;
        while (element.getNext() != null) {
            previous = element;
            element = element.getNext();
        }
        element = previous.getNext();
        return element.getValue();
    }
}
```

```
/**
 * @param value Eine Zahl, die zur Schlange hinzugefügt werden soll.
 */
public void enqueue(int value) {
    Element element = new Element(value);
    element.setNext(head);
    head = element;
}

/**
 * @return Das Element oder null, wenn der Schlange leer ist.
 */
public int dequeue() {
    if (head.getNext() == null) {
        int result = head.getValue();
        head = null;
        return result;
    }
    Element element = head;
    Element previous = null;
    while (element.getNext() != null) {
        previous = element;
        element = element.getNext();
    }
    element = previous.getNext();
    previous.setNext(null);
    return element.getValue();
}

/**
 * @return Wahr wenn der Schlange leer ist.
 */
public boolean isEmpty() {
    return head == null;
}

public static void main(String[] args) {
    Schlange s = new Schlange();
    s.enqueue(1);
    s.enqueue(2);
    s.enqueue(3);
    System.out.println(s.head());
    System.out.println(s.dequeue());
    System.out.println(s.head());
    System.out.println(s.dequeue());
    System.out.println(s.head());
    System.out.println(s.dequeue());
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Schlange.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Schlange.java)

Klasse Element

```
public class Element {
    public int value;

    public Element next;

    public Element() {
        this.next = null;
    }

    public Element(int value, Element element) {
        this.value = value;
        this.next = element;
    }

    public Element(int value) {
        this.value = value;
        this.next = null;
    }

    public int getValue() {
        return value;
    }

    public Element getNext() {
        return next;
    }

    public void setNext(Element element) {
        next = element;
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Element.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Element.java)

Test-Klasse

```
import static org.junit.Assert.*;

import org.junit.Test;

public class TestCase {

    @Test
    public void testeStapel() {
        Stapel s = new Stapel();
        s.push(1);
        s.push(2);
        s.push(3);

        assertEquals(false, s.isEmpty());
    }
}
```

```

    assertEquals(3, s.top());
    assertEquals(3, s.pop());

    assertEquals(2, s.top());
    assertEquals(2, s.pop());

    assertEquals(1, s.top());
    assertEquals(1, s.pop());
    assertEquals(true, s.isEmpty());
}

@Test
public void testeSchlange() {
    Schlange s = new Schlange();
    s.enqueue(1);
    s.enqueue(2);
    s.enqueue(3);

    assertEquals(false, s.isEmpty());

    assertEquals(1, s.head());
    assertEquals(1, s.dequeue());

    assertEquals(2, s.head());
    assertEquals(2, s.dequeue());

    assertEquals(3, s.head());
    assertEquals(3, s.dequeue());
    assertEquals(true, s.isEmpty());
}
}

```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/TestCase.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/TestCase.java)

(b) Analysieren Sie die Laufzeit Ihrer Methode in Abhängigkeit von n .

Lösungsvorschlag

Zeitkomplexität: $\mathcal{O}(n^2)$, da es zwei ineinander verschachtelte **while**-Schleifen gibt, die von der Anzahl der Elemente im Stapel abhängen.

66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 6

Der Hauptsatz der Laufzeitfunktionen ist bekanntlich folgendermaßen definiert:

Exkurs: Master-Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

a = Anzahl der rekursiven Aufrufe, Anzahl der Unterprobleme in der Rekursion ($a \geq 1$).

$\frac{1}{b}$ = Teil des Originalproblems, welches wiederum durch alle Unterprobleme repräsentiert wird, Anteil an der Verkleinerung des Problems ($b > 1$).

$f(n)$ = Kosten (Aufwand, Nebenkosten), die durch die Division des Problems und die Kombination der Teillösungen entstehen. Eine von $T(n)$ unabhängige und nicht negative Funktion.

Dann gilt:

1. Fall: $T(n) \in \Theta(n^{\log_b a})$

falls $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für $\epsilon > 0$

2. Fall: $T(n) \in \Theta(n^{\log_b a} \cdot \log n)$

falls $f(n) \in \Theta(n^{\log_b a})$

3. Fall: $T(n) \in \Theta(f(n))$

falls $f(n) \in \Omega(n^{\log_b a + \epsilon})$ für $\epsilon > 0$ und ebenfalls für ein c mit $0 < c < 1$ und alle hinreichend großen n gilt: $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$

- (a) Betrachten Sie die folgende Methode `m` in Java, die initial mit `m(r, 0, r.length)` für das Array `r` aufgerufen wird. Geben Sie dazu eine Rekursionsgleichung $T(n)$ an, welche die Anzahl an Rechenschritten von `m` in Abhängigkeit von der Länge $n = r.length$ berechnet.

```
public static int m(int[] r, int lo, int hi) {
    if (lo < 8 || hi <= 10 || lo >= r.length || hi > r.length) {
        throw new IllegalArgumentException();
    }

    if (hi - lo == 1) {
        return r[lo];
    } else if (hi - lo == 2) {
        return Math.max(r[lo], r[lo + 1]); // O(1)
    } else {
        int s = (hi - lo) / 3;
        int x = m(r, lo, lo + s);
        int y = m(r, lo + s, lo + 2 * s);
        int z = m(r, lo + 2 * s, hi);
        return Math.max(Math.max(x, y), z); // O(1)
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2018/fruehjahr/MasterTheorem.java](https://github.com/org/bschlangaul/examen/examen_66115/jahr_2018/fruehjahr/MasterTheorem.java)

Lösungsvorschlag

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

3

Anteil Verkleinerung des Problems (b):

um $\frac{1}{3}$ also $b = 3$

Laufzeit der rekursiven Funktion $(f(n))$:

$$\mathcal{O}(1)$$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 3 \cdot T\left(\frac{n}{3}\right) + \mathcal{O}(1)$$

- (b) Ordnen Sie die rekursive Funktion $T(n)$ aus (a) einem der drei Fälle des Mastertheorems zu und geben Sie die resultierende Zeitkomplexität an. Zeigen Sie dabei, dass die Voraussetzung des Falles erfüllt ist.

Lösungsvorschlag

1. Fall: $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$:

$$f(n) \in \mathcal{O}(n^{\log_3 3 - \varepsilon}) = \mathcal{O}(n^{1 - \varepsilon}) = \mathcal{O}(1) \text{ für } \varepsilon = 1$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$f(n) \notin \Theta(n^{\log_3 3}) = \Theta(n^1)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \varepsilon})$:

$$f(n) \notin \Omega(n^{\log_3 3 + \varepsilon}) = \Omega(n^{1 + \varepsilon})$$

$$\text{Also: } T(n) \in \Theta(n^{\log_b a})$$

66115 / 2019 / Herbst / Thema 1 / Aufgabe 6

- (a) Sortieren Sie die unten angegebenen Funktionen der O-Klassen $\mathcal{O}(a(n))$, $\mathcal{O}(b(n))$, $\mathcal{O}(c(n))$, $\mathcal{O}(d(n))$ und $\mathcal{O}(e(n))$ bezüglich ihrer Teilmengenbeziehungen. Nutzen Sie ausschließlich die echte Teilmenge \subset sowie die Gleichheit $=$ für die Beziehung zwischen den Mengen. Folgendes Beispiel illustriert diese Schreibweise für einige Funktionen f_1 bis f_5 (diese haben nichts mit den unten angegebenen Funktionen zu tun):²

$$\mathcal{O}(f_4(n)) \subset \mathcal{O}(f_3(n)) = \mathcal{O}(f_5(n)) \subset \mathcal{O}(f_1(n)) = \mathcal{O}(f_2(n))$$

Die angegebenen Beziehungen müssen weder bewiesen noch begründet werden.

- $a(n) = n^2 \cdot \log_2(n) + 42$
- $b(n) = 2^n + n^4$
- $c(n) = 2^{2 \cdot n}$
- $d(n) = 2^{n+3}$
- $e(n) = \sqrt{n^5}$

²[http://www.s-inf.de/Skripte/DaStru.2012-SS-Katoen.\(KK\).Klausur1MitLoesung.pdf](http://www.s-inf.de/Skripte/DaStru.2012-SS-Katoen.(KK).Klausur1MitLoesung.pdf)

$$\begin{aligned}
 a(n) &= n^2 \cdot \log_2(n) + 42 & = n \\
 b(n) &= 2^n + n^4 & = 2^n \\
 c(n) &= 2^{2 \cdot n} & = 2^{2 \cdot n} \\
 d(n) &= 2^{n+3} & = 2^n \\
 e(n) &= \sqrt{n^5}
 \end{aligned}$$

$$\mathcal{O}(a(n)) \subset \mathcal{O}(e(n)) \subset \mathcal{O}(b(n)) = \mathcal{O}(d(n)) \subset \mathcal{O}(c(n))$$

$$\mathcal{O}(n^2 \cdot \log_2(n) + 42) \subset \mathcal{O}(\sqrt{n^5}) \subset \mathcal{O}(2^n + n^4) = \mathcal{O}(2^{n+3}) \subset \mathcal{O}(2^{2 \cdot n})$$

(b) Beweisen Sie die folgenden Aussagen formal nach den Definitionen der O-Notation oder widerlegen Sie sie.

(i) $\mathcal{O}(n \cdot \log_2 n) \subseteq \mathcal{O}(n \cdot (\log_2 n)^2)$

Die Aussage gilt. Für $n \geq 16$ haben wir

$$(\log_2 n)^2 \leq n \Leftrightarrow \log_2 n \leq \sqrt{n}$$

und dies ist eine wahre Aussage für $n \geq 16$. Also gilt die Aussage mit $n_0 = 16$ und $c = 1$.

(ii) $2^{(n+1)} \in \mathcal{O}(n \cdot \log_2 n)$

(c) Bestimmen Sie eine asymptotische Lösung (in Θ -Schreibweise) für die folgende Rekursionsgleichung:

(i) $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2$

(ii) $T(n) = T\left(\frac{n}{2}\right) + \frac{n}{2}n^2 + n$

66115 / 2019 / Herbst / Thema 2 / Aufgabe 6

Der Hauptsatz der Laufzeitfunktionen ist bekanntlich folgendermaßen definiert:

1. Fall: $T(n) \in \Theta(n^{\log_b a})$

falls $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$ für $\varepsilon > 0$

2. Fall: $T(n) \in \Theta(n^{\log_b a} \cdot \log n)$

falls $f(n) \in \Theta(n^{\log_b a})$

3. Fall: $T(n) \in \Theta(f(n))$

falls $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ für $\varepsilon > 0$ und ebenfalls für ein c mit $0 < c < 1$ und alle hinreichend großen n gilt: $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$

Bestimmen und begründen Sie formal mit Hilfe dieses Satzes welche Komplexität folgende Laufzeitfunktionen haben.

(a) $T(n) = 8 \cdot T(\frac{n}{2}) + 5n^2$

Lösungsvorschlag

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T(\frac{n}{b}) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

8

Anteil Verkleinerung des Problems (b):

um $\frac{1}{2}$ also $b = 2$

Laufzeit der rekursiven Funktion ($f(n)$):

$5n^2$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 8 \cdot T(\frac{n}{2}) + 5n^2$$

1. Fall: $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$:

für $\varepsilon = 4$:

$$f(n) = 5n^2 \in \mathcal{O}(n^{\log_2 8 - 4}) = \mathcal{O}(n^{\log_2 4}) = \mathcal{O}(n^2)$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$f(n) = 5n^2 \notin \Theta(n^{\log_2 8}) = \Theta(n^3)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \varepsilon})$:

$$f(n) = 5n^2 \notin \mathcal{O}(n^{\log_2 8 + \varepsilon})$$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

(b) $T(n) = 9 \cdot T(\frac{n}{3}) + 5n^2$

Lösungsvorschlag

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T(\frac{n}{b}) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

9

Anteil Verkleinerung des Problems (b):um $\frac{1}{3}$ also $b = 3$ **Laufzeit der rekursiven Funktion ($f(n)$):**

$$5n^2$$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 9 \cdot T\left(\frac{n}{3}\right) + 5n^2$$

1. Fall: $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$:

$$f(n) = 5n^2 \notin \mathcal{O}(n^{\log_3 9 - \varepsilon}) \text{ für } \varepsilon > 0$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$f(n) = 5n^2 \in \Theta(n^{\log_3 9}) = \Theta(n^2)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \varepsilon})$:

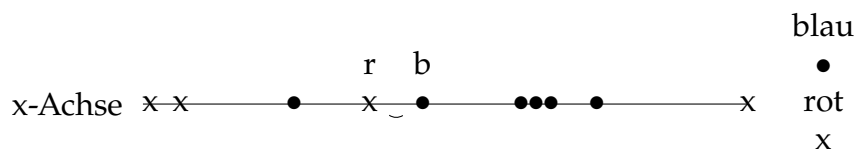
$$f(n) = 5n^2 \notin \mathcal{O}(n^{\log_3 9 + \varepsilon}) \text{ für } \varepsilon > 0$$

$$\Rightarrow T(n) \in \Theta(n^2 \cdot \log n)$$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

66115 / 2020 / Frühjahr / Thema 2 / Aufgabe 8

Gegeben seien zwei nichtleere Mengen R und B von roten bzw. blauen Punkten auf der x -Achse. Gesucht ist der minimale euklidische Abstand $d(r, b)$ über alle Punktepaare (r, b) mit $r \in R$ und $b \in B$. Hier ist eine Beispielinstantz:



Die Eingabe wird in einem Feld A übergeben. Jeder Punkt $A[i]$ mit $1 \leq i \leq n$ hat eine x -Koordinate $A[i].x$ und eine Farbe $A[i].color \in \{\text{rot}, \text{blau}\}$. Das Feld A ist nach x -Koordinate sortiert, d.h. gilt $A[1].x < A[2].x < \dots < A[n].x$, wobei $n = |R| + |B|$.

- (a) Geben Sie in Worten einen Algorithmus an, der den gesuchten Abstand in $\mathcal{O}(n)$ Zeit berechnet.

Lösungsvorschlag

Pseudo-Code

Algorithmus 2: Minimaler Euklidischer Abstand

```

 $d_{min} := \max ;$  // Setze  $d_{min}$  zuerst auf einen maximalen Wert.
for  $i$  in  $0 \dots \text{vorletzter Index}$  do ; // Iteriere über die Indizes des Punkte-Arrays
     $P$  bis zum vorletzten Index  $P[n-1]$ 

    if  $P[n].color \neq P[n+1].color$  then ; // Berechne den Abstand nur, wenn die
        Punkte unterschiedliche Farben haben

         $d = P[n+1].x - P[n].x$ 
        if  $d < d_{min}$  then
             $d_{min} = d$ 
        end
    end
end

```

Java

```

public double findMinimalDistance() {
    double distanceMin = Double.MAX_VALUE;
    for (int i = 0; i < latestIndex - 1; i++) {
        if (points[i].color != points[i + 1].color) {
            double distance = points[i + 1].x - points[i].x;
            if (distance < distanceMin) {
                distanceMin = distance;
            }
        }
    }
    return distanceMin;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/fruehjahr/RedBluePairCollection.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/fruehjahr/RedBluePairCollection.java)

- (b) Begründen Sie kurz die Laufzeit Ihres Algorithmus.

Lösungsvorschlag

Da das Array der Länge n nur einmal durchlaufen wird, ist die Laufzeit $\mathcal{O}(n)$ sichergestellt.

- (c) Begründen Sie die Korrektheit Ihres Algorithmus.

Lösungsvorschlag

In d_{min} steht am Ende der gesuchte Wert (sofern nicht $d_{min} = \text{Integer.MAX_VALUE}$ geblieben ist)

- (d) Wir betrachten nun den Spezialfall, dass alle blauen Punkte links von allen roten Punkten liegen. Beschreiben Sie in Worten, wie man in dieser Situation den gesuchten Abstand in $o(n)$ Zeit berechnen kann. (Ihr Algorithmus darf also insbesondere nicht Laufzeit $\Theta(n)$ haben.)

Zuerst müssen wir den letzten blauen Punkt finden. Das ist mit einer binären Suche möglich. Wir beginnen mit dem ganzen Feld als Suchbereich und betrachten den mittleren Punkt. Wenn er blau ist, wiederholen wir die Suche in der zweiten Hälfte des Suchbereichs, sonst in der ersten, bis wir einen blauen Punkt gefolgt von einem roten Punkt gefunden haben.

Der gesuchte minimale Abstand ist dann der Abstand zwischen dem gefundenen blauen und dem nachfolgenden roten Punkt. Die Binärsuche hat eine Worst-case-Laufzeit von $\mathcal{O}(\log n)$.

66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 4

(a) Betrachten Sie das folgende Code-Beispiel (in Java-Notation):

```
int mystery(int n) {
    int a = 0, b = 0;
    int i = 0;
    while (i < n) {
        a = b + i;
        b = a;
        i = i + 1;
    }
    return a;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/o_notation/Mystery1.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/o_notation/Mystery1.java)

Bestimmen Sie die asymptotische worst-case Laufzeit des Code-Beispiels in \mathcal{O} -Notation bezüglich der Problemgröße n . Begründen Sie Ihre Antwort.

Lösungsvorschlag

Die asymptotische worst-case Laufzeit des Code-Beispiels in \mathcal{O} -Notation ist $\mathcal{O}(n)$.

Die `while`-Schleife wird genau n mal ausgeführt. In der Schleife wird die Variable `i` in der Zeile `i = i + 1`; inkrementiert. `i` wird mit 0 initialisiert. Die `while`-Schleife endet, wenn `i` gleich groß ist als `n`.

(b) Betrachten Sie das folgende Code-Beispiel (in Java-Notation):

```
int mystery(int n) {
    int r = 0;
    while (n > 0) {
        int y = n;
        int x = n;
        for (int i = 0; i < y; i++) {
            for (int j = 0; j < i; j++) {
                r = r + 1;
            }
        }
    }
}
```

```

    r = r - 1;
  }
  n = n - 1;
}
return r;

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/o_notation/Mystery2.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/o_notation/Mystery2.java)

Bestimmen Sie für das Code-Beispiel die asymptotische worst-case Laufzeit in \mathcal{O} -Notation bezüglich der Problemgröße n . Begründen Sie Ihre Antwort.

Lösungsvorschlag

```

while: n-mal
1. for: n, n - 1, ..., 2, 1
2. for: 1, 2, ..., n - 1, n
 $n \times n \times n = \mathcal{O}(n^3)$ 

```

- (c) Bestimmen Sie eine asymptotische Lösung (in Θ -Schreibweise) für die folgende Rekursionsgleichung:

$$T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$$

Exkurs: Master-Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

a = Anzahl der rekursiven Aufrufe, Anzahl der Unterprobleme in der Rekursion ($a \geq 1$).

$\frac{1}{b}$ = Teil des Originalproblems, welches wiederum durch alle Unterprobleme repräsentiert wird, Anteil an der Verkleinerung des Problems ($b > 1$).

$f(n)$ = Kosten (Aufwand, Nebenkosten), die durch die Division des Problems und die Kombination der Teillösungen entstehen. Eine von $T(n)$ unabhängige und nicht negative Funktion.

Dann gilt:

1. Fall: $T(n) \in \Theta\left(n^{\log_b a}\right)$

falls $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$ für $\varepsilon > 0$

2. Fall: $T(n) \in \Theta\left(n^{\log_b a} \cdot \log n\right)$

falls $f(n) \in \Theta\left(n^{\log_b a}\right)$

3. Fall: $T(n) \in \Theta(f(n))$

falls $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ für $\varepsilon > 0$ und ebenfalls für ein c mit $0 < c < 1$ und alle hinreichend großen n gilt: $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$

Allgemeine Rekursionsgleichung:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Anzahl der rekursiven Aufrufe (a):

$$1$$

Anteil Verkleinerung des Problems (b):

$$\text{um } \frac{1}{2} \text{ also } b = 2$$

Laufzeit der rekursiven Funktion (f(n)):

$$\frac{1}{2}n^2 + n$$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$$

$$\text{Nebenrechnung: } \log_b a = \log_2 1 = 0$$

1. Fall: $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$:

$$\frac{1}{2}n^2 + n \notin \mathcal{O}(n^{-1})$$

2. Fall: $f(n) \in \Theta(n^{\log_b a})$:

$$\frac{1}{2}n^2 + n \notin \Theta(1)$$

3. Fall: $f(n) \in \Omega(n^{\log_b a + \varepsilon})$:

$$\varepsilon = 2$$

$$\frac{1}{2}n^2 + n \in \Omega(n^2)$$

Für eine Abschätzung suchen wir eine Konstante, damit gilt:

$$1 \cdot f\left(\frac{n}{2}\right) \leq c \cdot f(n)$$

$$\frac{1}{2} \cdot \frac{1}{4}n^2 + \frac{1}{2}n \leq c \cdot \left(\frac{1}{2} \cdot n^2 + n\right)$$

$$\text{Damit folgt } c = \frac{1}{4}$$

$$\text{und } 0 < c < 1$$

$$\Rightarrow \Theta\left(\frac{1}{2}n^2 + n\right)$$

$$\Rightarrow \Theta(n^2)$$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

Master-Theorem

Algorithmenmuster

66115 / 2020 / Herbst / Thema 1 / Aufgabe 4

- (a) Nehmen Sie an, es stehen beliebig viele 5-Cent, 2-Cent und 1-Cent-Münzen zur Verfügung. Die Aufgabe besteht darin, für einen gegebenen Cent-Betrag möglichst wenig Münzen zu verbrauchen. Entwerfen Sie eine Methode

```
public void wechselgeld (int n)
```

die diese Aufgabe mit einem Greedy-Algorithmus löst und für den Betrag von n Cent die Anzahl $c5$ der 5-Cent-Münzen, die Anzahl $c2$ der 2-Cent-Münzen und die Anzahl $c1$ der 1-Cent-Münzen berechnet und diese auf der Konsole ausgibt. Sie können dabei den Operator $/$ für die ganzzahlige Division und den Operator $\%$ für den Rest bei der ganzzahligen Division verwenden.¹

Lösungsvorschlag

```
public static void wechsle(int betrag) {
    int rest;
    int c5 = betrag / 5;
    rest = betrag % 5;
    int c2 = rest / 2;
    int c1 = rest % 2;

    System.out.println(String.format("Für den Betrag von %s Cent werden \n" +
        ↳ "%s Fünf-Cent-Münzen, \n"
        + "%s Zwei-Cent-Münzen und \n" + "%s Ein-Cent-Münzen ausgegeben.",
        ↳ betrag, c5, c2, c1));
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/muster/greedy/Muenzwechsler.java](https://github.com/bschlangaul/aufgaben/aud/muster/greedy/Muenzwechsler.java)

- (b) Es kann gezeigt werden, dass der Greedy-Algorithmus für den obigen Fall der Münzwerte 5, 2 und 1 optimal ist, dass er immer die Gesamtzahl der Münzen minimiert. Nehmen Sie nun an, es gibt die Münzwerte 5 und 1. Ist es dann möglich, einen dritten Münzwert so zu wählen, dass der Greedy-Algorithmus mit den drei Münzen nicht mehr optimal ist? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Falls der dritte Münzwert 4 ist, ist der Greedy-Algorithmus nicht mehr optimal. Der Greedy-Algorithmus benutzt zunächst so viele 5-Cent-Münzen wie möglich und dann so viele 4-Cent-Münzen wie möglich. Ein Betrag von 8 Cent wird also in eine 5-Cent und drei 1-Cent-Münzen aufgeteilt. Optimal ist aber die Aufteilung in zwei 4-Cent-Münzen.

¹Quelle möglicherweise von <https://www.yumpu.com/de/document/read/17936760/uebungen-zum-prasenzmodul-algorithmen-und-datenstrukturen>

66115 / 2020 / Herbst / Thema 1 / Aufgabe 4

Als Beispiel nehmen wir die Herausgabe von Wechselgeld auf Beträge unter 1€. Verfügbar sind die Münzen mit den Werten 50ct, 10ct, 5ct, 2ct, 1ct. Unser Ziel ist, so wenig Münzen wie möglich in das Portemonnaie zu bekommen. Ein Beispiel: $78\text{ct} = 50 + 2 \cdot 10 + 5 + 2 + 1$. Es wird jeweils immer die größte Münze unter dem Zielwert genommen und von diesem abgezogen. Das wird so lange durchgeführt, bis der Zielwert Null ist.

Formalisierung

Gesucht ist ein Algorithmus der folgende Eigenschaften beschreibt. Bei der *Eingabe* muss gelten:

- (a) dass die eingegebene Zahl eine natürliche Zahl ist, also $\text{betrag} > 0$
- (b) dass eine Menge von Münzwerten zur Verfügung steht $\text{münzen} = \{c_1, \dots, c_n\}$ z. B. $\{1, 2, 5, 10, 20, 50\}$

Die *Ausgabe* besteht dann aus ganzen Zahlen $\text{wechselgeld}[1], \dots, \text{wechselgeld}[n]$. Dabei ist $\text{wechselgeld}[i]$ die Anzahl der Münzen des Münzwertes für c_i für $i = 1, \dots, n$ und haben die Eigenschaften:

- (a) $\text{wechselgeld}[1] \cdot c_1 + \dots + \text{wechselgeld}[n] \cdot c_n = \text{betrag}$
- (b) $\text{wechselgeld}[1] + \dots + \text{wechselgeld}[n]$ ist minimal unter allen Lösungen für 1.

Lösungsvorschlag

```
/**
 * <a href=
 *
 * → "https://de.wikiversity.org/wiki/Kurs:Algorithmen_und_Datenstrukturen/Vorlesung/Greedyalgorithmen
 * und Datenstrukturen/Vorlesung/Greedyalgorithmen Wechselgeldalgorithmus">
 */
public class Wechselgeld {

    public static int[] berechneWechselgeld(int[] münzen, int betrag) {
        int[] wechselgeld = new int[münzen.length];
        int aktuelleMünze = münzen.length - 1;
        while (betrag > 0) {
            while (betrag < münzen[aktuelleMünze] && aktuelleMünze > 0)
                aktuelleMünze--;
            if (betrag >= münzen[aktuelleMünze] && aktuelleMünze >= 0) {
                betrag -= münzen[aktuelleMünze];
                wechselgeld[aktuelleMünze]++;
            } else
                return null;
        }
        return wechselgeld;
    }

    public static void main(String[] args) {
        int[] münzen = { 1, 2, 5, 10, 20, 50 };
        int betrag = 78;
    }
}
```

```

int[] wechselgeld = berechneWechselgeld(münzen, betrag);

System.out.println(String.format("Der Betrag von %s Cent wird gewechselt in:",
    ↪ betrag));

for (int i = 0; i < wechselgeld.length; i++) {
    System.out.println(String.format("%s x %s Cent", wechselgeld[i],
    ↪ münzen[i]));
}
}
}

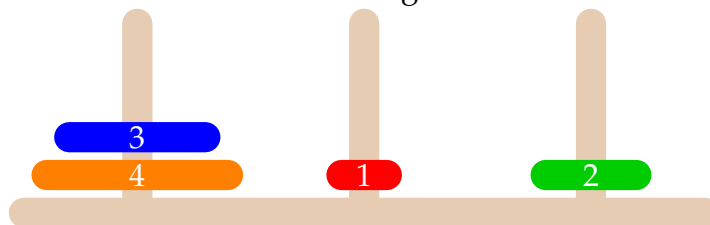
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/muster/Wechselgeld.java](https://github.com/org/bschlangaul/muster/Wechselgeld.java)

66115 / 2020 / Herbst / Thema 1 / Aufgabe 4

Betrachten wir das folgende Spiel (Türme von Hanoi), das aus drei Stäben 1, 2 und 3 besteht, die senkrecht im Boden befestigt sind. Weiter gibt es n kreisförmige Scheiben mit einem Loch im Mittelpunkt, so dass man sie auf die Stäbe stecken kann. Dabei haben die Scheiben verschiedene Radien, alle sind unterschiedlich groß. Zu Beginn stecken alle Scheiben auf dem Stab 1, wobei immer eine kleinere auf einer größeren liegt. Das Ziel des Spiels ist es nun, die Scheiben so umzuordnen, dass sie in der gleichen Reihenfolge auf dem Stab 3 liegen. Dabei darf immer nur eine Scheibe bewegt werden und es darf nie eine größere auf einer kleineren Scheibe liegen. Stab 2 darf dabei als Hilfsstab verwendet werden.

Ein Beispiel für 4 Scheiben finden Sie in folgendem Bild:



Entwerfen Sie mit Hilfe der Vorlage eine varibale Simulation der Türme von Hanoi.

- Ein ELEMENT hat immer einen Wert (Integer) und kennt das Nachfolgende Element, wobei immer nur das jeweilige Element auf seinen Wert und seinen Nachfolger zugreifen darf
- Ein Turm ist einem Stack (Kellerspeicher) nachempfunden und kennt somit nur das erste Element. Hinweis: Beachten Sie, dass nur kleinere Elemente auf den bisherigen Stack gelegt werden können
- In der Klasse HANOI müssen Sie nur die Methode `public void hanoi (int n, TURM quelle, TURM ziel, TURM hilfe)` implementieren. Die anderen Methoden sind zur Veranschaulichung des Spiels! Entwerfen Sie eine rekursive Methode die einen Turm der Höhe n vom Stab `quelle` auf den Stab `ziel` transportiert und den Stab `hilfe` als Hilfsstab verwendet.

66115 / 2020 / Herbst / Thema 1 / Aufgabe 4

Betrachten Sie das folgende Gitter mit $m + 1$ Zeilen und $n + 1$ Spalten ($m \geq 1$ und $n \geq 1$):² [geeksforgeeks](https://www.geeksforgeeks.org/count-possible-paths-top-left-bottom-right-nxm-matrix/)³

Angenommen, Sie befinden sich zu Beginn am Punkt $(0,0)$ und wollen zum Punkt (m,n) .

Für die Anzahl $A(i,j)$ aller verschiedenen Wege vom Punkt $(0,0)$ zum Punkt (i,j) lassen sich folgende drei Fälle unterscheiden (es geht jeweils um die kürzesten Wege ohne Umweg!):

- $1 \leq i \leq m$ und $j = 0$:

Es gibt genau einen Weg von $(0,0)$ nach $(i,0)$ für $1 \leq i \leq m$.

- $i = 0$ und $1 \leq j \leq n$:

Es gibt genau einen Weg von $(0,0)$ nach $(0,j)$ für $1 \leq j \leq n$.

- $1 \leq i \leq m$ und $1 \leq j \leq n$:

auf dem Weg zu (i,j) muss als vorletzter Punkt entweder $(i-1,j)$ oder $(i,j-1)$ besucht worden sein.

Daraus ergibt sich folgende Rekursionsgleichung:

$$A(i,j) = \begin{cases} 1 & \text{falls } (1 \leq i \leq m \text{ und } j = 0) \text{ oder } (i = 0 \text{ und } 1 \leq j \leq n) \\ A(i-1,j) + A(i,j-1) & \text{falls } 1 \leq i \leq m \text{ und } 1 \leq j \leq n \end{cases}$$

Implementieren Sie die Java-Klasse `Gitter` mit der Methode

```
public int berechneAnzahlWege(),
```

die ausgehend von der Rekursionsgleichung durch dynamische Programmierung die Anzahl aller Wege vom Punkt $(0,0)$ zum Punkt (m,n) berechnet. Die Überprüfung, ob $m \leq 1$ und $n \leq 1$ gilt, können Sie der Einfachheit halber weglassen.

Lösungsvorschlag

```
public int berechneAnzahlWege() {
    int i, j;
    for (i = 1; i <= m; i++) {
        anzahlWege[i][0] = 1;
    }
    for (j = 1; j <= n; j++) {
        anzahlWege[0][j] = 1;
    }
    for (i = 1; i <= m; i++) {
        for (j = 1; j <= n; j++) {
            anzahlWege[i][j] = anzahlWege[i-1][j] + anzahlWege[i][j-1];
        }
    }
}
```

²Quelle möglicherweise von <https://www.yumpu.com/de/document/read/17936760/ubungen-zum-prasenzmodul-algorithmen-und-datenstrukturen>

³<https://www.geeksforgeeks.org/count-possible-paths-top-left-bottom-right-nxm-matrix/>


```

    }
}
return anzahlWege[m][n];
}

```

Implementierung in Java
BacktrackingCode-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/muster/dp/Gitter.java](https://github.com/bschlangaul/aufgaben/blob/master/aud/muster/dp/Gitter.java)

66115 / 2020 / Herbst / Thema 1 / Aufgabe 4

Implementieren sie mittels Backtracking einen Algorithmus, der acht Damen auf einem Schachbrett so aufgestellt, dass keine zwei Damen einander gemäß ihren in den Schachregeln definierten Zugmöglichkeiten schlagen können. Für Damen heißt dies konkret: Es dürfen keine zwei Damen auf derselben Reihe, Linie oder Diagonale stehen. Es gibt 92 mögliche Lösungen für das 8×8 Feld.

Lösungsvorschlag

```

public class Damenproblem {
    static int n = 8;
    static int[][] spielBrett = new int[n][n];
    static int DAME = 1;
    static int LEER = 0;

    public static boolean istGültig(int zeile, int spalte) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (spielBrett[i][j] == 1) {
                    if (i == zeile || j == spalte) {
                        return false;
                    }
                }
            }
        }
        for (int i = 0; i < n; i++) {
            if (zeile + i < n && spalte + i < n && spielBrett[zeile + i][spalte + i] == 1)
                return false;
            if (zeile - i > -1 && spalte - i > -1 && spielBrett[zeile - i][spalte - i] == 1)
                return false;
            if (zeile + i < n && spalte - i > -1 && spielBrett[zeile + i][spalte - i] == 1)
                return false;
            if (zeile - i > -1 && spalte + i < n && spielBrett[zeile - i][spalte + i] == 1)
                return false;
        }
        return true;
    }

    public static boolean löse(int zeile) {
        if (zeile == n) {
            return true;
        }
    }
}

```

```

for (int i = 0; i < n; i++) {
    if (istGültig(zeile, i) == true) {
        spielBrett[zeile][i] = DAME;

        if (löse(zeile + 1) == true) {
            return true;
        }
        spielBrett[zeile][i] = LEER;
    }
}

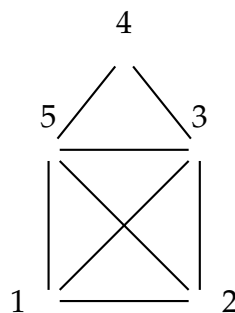
return false;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/muster/backtracking/damenproblem/Damenproblem.java](https://github.com/bschlangaul/aufgaben/blob/master/backtracking/damenproblem/Damenproblem.java)

66115 / 2020 / Herbst / Thema 1 / Aufgabe 4

Hier ist das „Haus des Nikolaus“ mit einer bestimmten Nummerierung der Eckpunkte vorgegeben. Es sollen alle Lösungen zum Zeichnen der Figur in einem Zug gefunden werden. Eine Lösung könnte dann in der Form 123451352 ausgegeben werden. Das Programm soll eine einfache Anpassung an andere Graphen ermöglichen. Der Ausschluss von gespiegelten Lösungen ist nicht gefordert.



Exkurs: Backtracking

Eine Lösung lässt sich nach dem Prinzip *Versuch und Testen* ermitteln. Eine vermutete Teillösung muss wieder verworfen werden, wenn ein Test ihre Ungültigkeit nachgewiesen hat. Man nennt diesen Ansatz deshalb auch *Rückverfolgen* oder *Backtracking*. Mit diesem Ansatz lassen sich eine ganze Reihe von Problemen in der Informatik sehr elegant formulieren und lösen. Hier eine kleine Auswahl (Genaueres dazu später):

Acht-Damen-Problem: Acht Damen sollen so auf ein Schachbrett gestellt werden, dass keine Dame eine andere bedroht.

Vier-Farben-Problem: Eine Landkarte soll mit vier Farben so gefärbt werden, dass benachbarte Länder immer unterschiedliche Farben bekommen.

Labyrinth-Problem: Ein Labyrinth mit Sackgassen und Verzweigungen ist zu durchlaufen, um den Ausgang zu finden.

Konkreter:

- (a) Man versucht, eine Kante (Verbindungsstrecke) zu zeichnen, wenn sie zulässig ist oder noch nicht gezeichnet wurde.
- (b) Ist das nicht möglich, muss die zuletzt gezeichnete Kante gelöscht werden.
- (c) Ist es möglich, dann hat man das Problem um eine Stufe vereinfacht.
- (d) Hat man durch dieses Verfahren insgesamt 8 Kanten zeichnen können, hat man eine Lösung gefunden. Jetzt löscht man wieder die zuletzt gezeichnete Kante und sucht nach weiteren Lösungen.

Realisierung des Programms

Datenstrukturen

Die folgende Tabelle gibt an, welche Verbindungslinien zulässig sind (durch X markiert). Die erste Zeile bedeutet also, dass von Punkt 1 zu den Punkten 2, 3 und 5 Strecken gezeichnet werden dürfen. Eine solche Tabelle heißt auch Adjazenzmatrix (von adjazieren; lat.: anwohnen, anliegen). Eine solche Tabelle lässt sich durch `boolean[] [] kanteZulaessig`; in einem zweidimensionalen Feld speichern. Eine entsprechende Tabelle `boolean[] [] kanteGezeichnet`; erfasst dann die schon gezeichneten Kanten. In einem weiteren eindimensionalen Feld wird jeweils eine Lösung erfasst.

Methoden

Es bieten sich folgende Methoden zur Strukturierung des Programmes an:

- (a) `void initialisiereFelder()`
- (b) `void zeichneKante(int von, int nach)`
- (c) `void löscheKante(int von, int nach)`
- (d) `void gibLösungAus()`
- (e) `void versucheKanteZuZeichnen(int start)`: Die rekursive Methode soll vom Punkt start weitere Kanten zeichnen.
- (f) Das Hauptprogramm:

```
public static void main(String[] arg) {
    initialisiereFelder();
    for (int punktNr = 1; punktNr <= maxPunktAnzahl; punktNr++) {
        lösungsWeg[0] = punktNr; // Startpunkt eintragen
        versucheKanteZuZeichnen(punktNr);
    }
    System.out.println();
    System.out.println("Es ergaben sich " + lösungsAnzahl + " Loesungen.");
}
```

```
/**
 * Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen:
 * Aufgabenblatt 3: Algorithmenmuster.
 *
 * <a href="https://www.studon.fau.de/file2521908_download.html">Angabe: AB_3
 * Greedy_DP_Backtracking.pdf</a>
 * <a href="https://www.studon.fau.de/file2521907_download.html">Lösung: AB_3
 * Greedy_DP_Backtracking_Lsg.pdf</a>
 */
public class Nikolaus {
    static final int maxPunktAnzahl = 5;
    static final int maxKantenAnzahl = 8;
    static boolean[][] kanteZulässig;
    static boolean[][] kanteGezeichnet;
    static int[] lösungsWeg;
    static int aktuelleKantenAnzahl = 0;
    static int lösungsAnzahl = 0;

    /**
     * Zulässige Kanten für das „Haus des Nikolaus“ eintragen. Der Nummerierung
     * liegt das Bild in main zu Grunde. Eine Anpassung an andere Graphen ist leicht
     * möglich.
     */
    static void initialisiereFelder() {
        kanteZulässig = new boolean[maxKantenAnzahl + 1][maxKantenAnzahl + 1];
        kanteGezeichnet = new boolean[maxKantenAnzahl + 1][maxKantenAnzahl + 1];
        lösungsWeg = new int[maxKantenAnzahl + 2]; // mit Startpunkt
        // Erst mal alles auf false ;
        for (int i = 1; i <= maxPunktAnzahl; i++) {
            for (int k = 1; k <= maxPunktAnzahl; k++) {
                kanteZulässig[i][k] = false;
                kanteGezeichnet[i][k] = false;
            }
        }

        kanteZulässig[1][2] = true; // von 1 nach 2 zulässig
        kanteZulässig[2][1] = true;

        kanteZulässig[1][3] = true;
        kanteZulässig[3][1] = true;

        kanteZulässig[1][5] = true;
        kanteZulässig[5][1] = true;

        kanteZulässig[2][3] = true;
        kanteZulässig[3][2] = true;

        kanteZulässig[2][5] = true;
        kanteZulässig[5][2] = true;

        kanteZulässig[3][4] = true;
        kanteZulässig[4][3] = true;

        kanteZulässig[3][5] = true;
        kanteZulässig[5][3] = true;
    }
}
```

```
kanteZulässig[4][5] = true;
kanteZulässig[5][4] = true;
for (int i = 0; i <= maxKantenAnzahl; i++) {
    lösungsWeg[i] = 0;
}
}

static void zeichneKante(final int von, final int nach) {
    kanteGezeichnet[von][nach] = true;
    kanteGezeichnet[nach][von] = true;
    // Anzahl bereits gezeichneter Kanten erhöhen
    aktuelleKantenAnzahl++;
    // neuen Wegpunkt in Lösung aufnehmen
    lösungsWeg[aktuelleKantenAnzahl] = nach;
}

static void löscheKante(final int von, final int nach) {
    kanteGezeichnet[von][nach] = false;
    kanteGezeichnet[nach][von] = false;
    aktuelleKantenAnzahl--;
}

static boolean fertig() {
    return (aktuelleKantenAnzahl == maxKantenAnzahl);
}

static void gibLösungAus() {
    for (int i = 0; i <= maxKantenAnzahl; i++) {
        System.out.print(lösungsWeg[i]);
        System.out.print(" ");
        lösungsAnzahl++;
        if (lösungsAnzahl % 8 == 0) {
            System.out.println();
        }
    }
}

static void versucheKanteZuZeichnen(final int start) {
    for (int ziel = 1; ziel <= maxPunktAnzahl; ziel++) {
        if (kanteZulässig[start][ziel] && !kanteGezeichnet[start][ziel]) {
            zeichneKante(start, ziel);
            if (!fertig()) {
                versucheKanteZuZeichnen(ziel);
            } else {
                gibLösungAus();
            }
            löscheKante(start, ziel);
        }
    }
}

public static void main(final String[] arg) {
    initialisiereFelder();
    System.out
```

```

↪ .println("Das Programm bestimmt alle Lösungen des Problems, das Haus des Nikolaus in einem
System.out.println("      4      ");
System.out.println("      . .      ");
System.out.println("      . .      ");
System.out.println("  5-----3  ");
System.out.println(" | . . |  ");
System.out.println(" | . . |  ");
System.out.println(" | . . |  ");
System.out.println(" | . . |  ");
System.out.println("  1-----2  ");
for (int punktNr = 1; punktNr <= maxPunktAnzahl; punktNr++) {
    lösungsWeg[0] = punktNr;
    versucheKanteZuZeichnen(punktNr);
}
System.out.println();
System.out.println("Es ergaben sich " + lösungsAnzahl + " Lösungen.");
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/muster/backtracking/Nikolaus.java](https://github.com/bschlangaul/aufgaben/aud/muster/backtracking/Nikolaus.java)

46114 / 2008 / Herbst / Thema 2 / Aufgabe 3

Quicksort ist ein Sortierungsverfahren, das nach dem Divide-and-Conquer-Prinzip (Teile und Herrsche) arbeitet. Wir betrachten im Folgenden die Anwendung dieses Verfahrens zum Sortieren von Integerzahlen. Die Sortierung soll in aufsteigender Reihenfolge der Werte erfolgen. Wir nehmen dabei an, dass die zu sortierenden Zahlen in einem Feld fester Länge abgelegt sind.

- (a) Beschreiben Sie die Arbeitsweise des Divide-and-Conquer-Prinzips im allgemeinen Fall. Geben Sie dabei die Bedeutung der Schritte divide, conquer und combine an.
- (b) Beschreiben Sie die Arbeitsweise des Algorithmus Quicksort. Geben Sie dabei an, worin die Schritte divide, conquer und combine im konkreten Fall bestehen.
- (c) Geben Sie in C, C++ oder Java eine Implementierung des Algorithmus Quicksort an. Formulieren Sie die Implementierung als rekursive Funktion quicksort() und verwenden Sie das jeweils erste Element des (Teil-)Feldes für die Aufteilung. Verwenden Sie für Ihre Implementierung von quicksort() «lei Parameter:
 - (i) das Feld, in dem die zu sortierenden Zahlen abgelegt sind;
 - (ii) den Index des am weitesten links gelegenen Elementes des zu sortierenden Teilfeldes;
 - (iii) den Index des am weitesten rechts gelegenen Elementes des zu sortierenden Teilfeldes.

Erläutern Sie die Arbeitsweise Ihrer Implementierung. Kennzeichnen Sie die Schritte divide, conquer und combine des zugrundeliegenden Divide-and-Conquer-Prinzips.

46115 / 2016 / Herbst / Thema 2 / Aufgabe 4

Mittels Dynamischer Programmierung (auch Memoization genannt) kann man insbesondere rekursive Lösungen auf Kosten des Speicherbedarf beschleunigen, indem man Zwischenergebnisse „abspeichert“ und bei (wiederkehrendem) Bedarf „abrufen“, ohne sie erneut berechnen zu müssen.

Gegeben sei folgende geschachtelt-rekursive Funktion für $n, m \geq 0$:

$$a(n, m) = \begin{cases} n + \lfloor \frac{n}{2} \rfloor & \text{falls } m = 0 \\ a(1, m - 1), & \text{falls } n = 0 \wedge m \neq 0 \\ a(n + \lfloor \sqrt{a(n - 1, m)} \rfloor, m - 1), & \text{sonst} \end{cases}$$

- (a) Implementieren Sie die obige Funktion $a(n, m)$ zunächst ohne weitere Optimierungen als Prozedur/Methode in einer Programmiersprache Ihrer Wahl.

Lösungsvorschlag

```
public static long a(int n, int m) {
    if (m == 0) {
        return n + (n / 2);
    } else if (n == 0 && m != 0) {
        return a(1, m - 1);
    } else {
        return a(n + ((int) Math.sqrt(a(n - 1, m))), m - 1);
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/beschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java](https://github.com/src/main/java/org/beschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java)

- (b) Geben Sie nun eine DP-Implementierung der Funktion $a(n, m)$ an, die $a(n, m)$ für $0 \leq n \leq 100000$ und $0 \leq m \leq 25$ höchstens einmal gemäß obiger rekursiver Definition berechnet. Beachten Sie, dass Ihre Prozedur trotzdem auch weiterhin mit $n > 100000$ und $m > 25$ aufgerufen werden können soll.

Lösungsvorschlag

```
static long[] [] tmp = new long[100001][26];

public static long aDp(int n, int m) {
    if (n <= 100000 && m <= 25 && tmp[n][m] != -1) {
        return tmp[n][m];
    } else {
        long merker;
        if (m == 0) {
            merker = n + (n / 2);
        } else if (n == 0 && m != 0) {
            merker = aDp(1, m - 1);
        } else {
            merker = aDp(n + ((int) Math.sqrt(aDp(n - 1, m))), m - 1);
        }
        if (n <= 100000 && m <= 25) {
            tmp[n][m] = merker;
        }
    }
}
```

```

        return merker;
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java)

Lösungsvorschlag

Kompletter Code

```

public class DynamischeProgrammierung {
    public static long a(int n, int m) {
        if (m == 0) {
            return n + (n / 2);
        } else if (n == 0 && m != 0) {
            return a(1, m - 1);
        } else {
            return a(n + ((int) Math.sqrt(a(n - 1, m))), m - 1);
        }
    }

    static long[][] tmp = new long[100001][26];

    public static long aDp(int n, int m) {
        if (n <= 100000 && m <= 25 && tmp[n][m] != -1) {
            return tmp[n][m];
        } else {
            long merker;
            if (m == 0) {
                merker = n + (n / 2);
            } else if (n == 0 && m != 0) {
                merker = aDp(1, m - 1);
            } else {
                merker = aDp(n + ((int) Math.sqrt(aDp(n - 1, m))), m - 1);
            }
            if (n <= 100000 && m <= 25) {
                tmp[n][m] = merker;
            }
            return merker;
        }
    }

    public static void main(String[] args) {
        for (int i = 0; i < 100001; i++) {
            for (int j = 0; j < 26; j++) {
                tmp[i][j] = -1;
            }
        }
        System.out.println("schnell mit DP: " + aDp(7,7));
        System.out.println("langsam ohne DP: " + a(7,7));
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java)

46115 / 2017 / Herbst / Thema 2 / Aufgabe 3

Die Methode `pKR` berechnet die n -te Primzahl ($n \geq 1$) kaskadenartig rekursiv und äußerst ineffizient:

```
static long pKR(int n) {
    long p = 2;
    if (n >= 2) {
        p = pKR(n - 1); // beginne die Suche bei der vorhergehenden Primzahl
        int i = 0;
        do {
            p++; // pruefe, ob die jeweils naechste Zahl prim ist, d.h. ...
            for (i = 1; i < n && p % pKR(i) != 0; i++) {
            } // pruefe, ob unter den kleineren Primzahlen ein Teiler ist
        } while (i != n); // ... bis nur noch 1 und p Teiler von p sind
    }
    return p;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2017/herbst/PrimzahlDP.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2017/herbst/PrimzahlDP.java)

Überführen Sie `pKR` mittels *dynamischer Programmierung* (hier also *Memoization*) und mit möglichst *wenigen Änderungen* so in die *linear* rekursive Methode `pLR`, dass `pLR(n, new long[n + 1])` ebenfalls die n -te Primzahl ermittelt:

```
private long pLR(int n, long[] ps) {
    ps[1] = 2;
    // ...
}
```

Lösungsvorschlag

Lösungsvorschlag

Exkurs: Kaskadenartig rekursiv

Kaskadenförmige Rekursion bezeichnet den Fall, in dem mehrere rekursive Aufrufe nebeneinander stehen.

Lösungsvorschlag

Exkurs: Linear rekursiv

Die häufigste Rekursionsform ist die lineare Rekursion, bei der in jedem Fall der rekursiven Definition höchstens ein rekursiver Aufruf vorkommen darf.

```
static long pLR(int n, long[] ps) {
    ps[1] = 2;
    long p = 2;
    if (ps[n] != 0) // Fall die Primzahl bereits gefunden / berechnet wurde,
        return ps[n]; // gib die berechnete Primzahl zurück.
    if (n >= 2) {
        // der einzige rekursive Aufruf steht hier, damit die Methode linear
        //   ↳ rekursiv
        //   ↳ ist.
        p = pLR(n - 1, ps);
    }
```

```
int i = 0;
do {
    p++;
    // Hier wird auf das gespeicherte Feld zurückgegriffen.
    for (i = 1; i < n && p % ps[i] != 0; i++) {
    }
} while (i != n);
}
ps[n] = p; // Die gesuchte Primzahl im Feld speichern.
return p;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2017/herbst/PrimzahlDP.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2017/herbst/PrimzahlDP.java)

Der komplette Quellcode

```
/**
 * Berechne die n-te Primzahl.
 *
 * Eine Primzahl ist eine natürliche Zahl, die größer als 1 und ausschließlich
 * durch sich selbst und durch 1 teilbar ist.
 *
 * <ul>
 * <li>1. Primzahl: 2
 * <li>2. Primzahl: 3
 * <li>3. Primzahl: 5
 * <li>4. Primzahl: 7
 * <li>5. Primzahl: 11
 * <li>6. Primzahl: 13
 * <li>7. Primzahl: 17
 * <li>8. Primzahl: 19
 * <li>9. Primzahl: 23
 * <li>10. Primzahl: 29
 * </ul>
 */
public class PrimzahlDP {

    /**
     * Die Methode pKR berechnet die n-te Primzahl ({@code n >= 1}) Kaskadenartig
     * → Rekursiv.
     *
     * @param n Die Nummer (n-te) der gesuchten Primzahl. Die Primzahl 2 ist die
     *          erste Primzahl. Die Primzahl 3 ist die zweite Primzahl etc.
     *
     * @return Die gesuchte n-te Primzahl.
     */
    static long pKR(int n) {
        long p = 2;
        if (n >= 2) {
            p = pKR(n - 1); // beginne die Suche bei der vorhergehenden Primzahl
            int i = 0;
            do {
                p++; // prüfe, ob die jeweils naechste Zahl prim ist, d.h. ...
                for (i = 1; i < n && p % pKR(i) != 0; i++) {

```

```

        } // pruefe, ob unter den kleineren Primzahlen ein Teiler ist
    } while (i != n); // ... bis nur noch 1 und p Teiler von p sind
    }
    return p;
}

/**
 * Die Methode pLR berechnet die n-te Primzahl ({@code n >= 1}) Linear Rekursiv.
 *
 * @param n Die Nummer (n-te) der gesuchten Primzahl. Die Primzahl 2 ist die
 *          erste Primzahl. Die Primzahl 3 ist die zweite Primzahl etc.
 * @param ps Primzahl Speicher. Muss mit n + 1 initialisiert werden.
 *
 * @return Die gesuchte n-te Primzahl.
 */
static long pLR(int n, long[] ps) {
    ps[1] = 2;
    long p = 2;
    if (ps[n] != 0) // Fall die Primzahl bereits gefunden / berechnet wurde,
        return ps[n]; // gib die berechnete Primzahl zurück.
    if (n >= 2) {
        // der einzige rekursive Aufruf steht hier, damit die Methode linear
        ↪ rekursiv
        // ist.
        p = pLR(n - 1, ps);
        int i = 0;
        do {
            p++;
            // Hier wird auf das gespeicherte Feld zurückgegriffen.
            for (i = 1; i < n && p % ps[i] != 0; i++) {
            }
        } while (i != n);
    }
    ps[n] = p; // Die gesuchte Primzahl im Feld speichern.
    return p;
}

static void debug(int n) {
    ↪ System.out.println(String.format("%d. Primzahl: %d (kaskadenartig rekursiv berechnet)",
    ↪ n, pKR(n)));

    ↪ System.out.println(String.format("%d. Primzahl: %d (linear rekursiv berechnet)",
    ↪ n, pLR(n, new long[n + 1])));
}

public static void main(String[] args) {
    System.out.println(pKR(10));
    System.out.println(pLR(10, new long[11]));

    for (int i = 1; i <= 10; i++) {
        debug(i);
    }
}

```

}

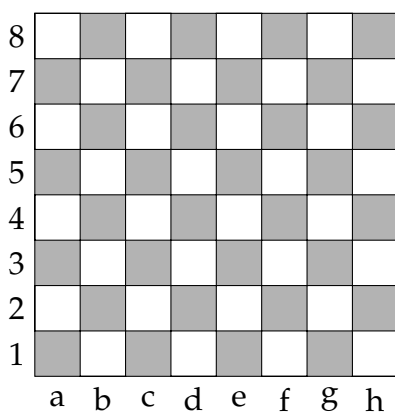
Backtracking

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2017/herbst/PrimzahlDP.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2017/herbst/PrimzahlDP.java)

46115 / 2018 / Herbst / Thema 2 / Aufgabe 5

Das *Springerproblem* ist ein kombinatorisches Problem, das darin besteht, für einen Springer auf einem leeren Schachbrett eine Route von einem gegebenen Startfeld aus zu finden, auf der dieser jedes Feld des Schachbretts genau einmal besucht.

Ein Schachbrett besteht aus 8×8 Feldern. Ein Springer kann bei einem Zug von einem Ausgangsfeld aus eines von maximal 8 Folgefelder betreten, wie dies in der folgenden Abbildung dargestellt ist. Der Springer darf selbstverständlich nicht über den Rand des Schachbretts hinauspringen.



Eine Lösung des Springerproblems mit Startfeld **h1** sieht wie folgt aus. Die Felder sind in ihrer Besuchsreihenfolge durchnummeriert. Der Springer bewegt sich also von **h1** nach **f2**, dann von **f2** nach **h3** usw.

41	10	29	26	49	12	31	16
28	25	40	11	30	15	50	13
9	42	27	56	61	48	17	32
24	39	58	47	64	53	14	51
43	8	55	62	57	60	33	18
38	23	46	59	54	63	52	3
7	44	21	36	5	2	19	34
22	37	6	45	20	35	4	1

Formulieren Sie einen rekursiven Algorithmus zur Lösung des Springerproblems von einem vorgegebenen Startfeld aus. Es sollen dabei alle möglichen Lösungen des Springerproblems gefunden werden. Die Lösungen sollen durch Backtracking gefunden werden. Hierbei werden alle möglichen Teilrouten systematisch durchprobiert, und Teilrouten, die nicht zu einer Lösung des Springerproblems führen können, werden nicht

weiterverfolgt. Dies ist durch rekursiven Aufruf einer Lösungsfunktion `huepf(z, y, z)` zu realisieren, wobei

- `x` und `y` die Koordinaten des als nächstes anzuspringenden Feldes sind, und
- `z` die aktuelle Rekursionstiefe enthält. Wenn die Rekursionstiefe 64 erreicht und das betreffende Feld noch unbesucht ist, ist eine Lösung des Springerproblems gefunden.

Der initiale Aufruf Ihres Algorithmus kann beispielsweise über den Aufruf

`huepf(1, 8, 1)`

erfolgen.

Wählen Sie geeignete Datenstrukturen zur Verwaltung der unbesuchten Felder und zum Speichern gefundener (Teil)Lösungen. Der Algorithmus soll eine gefundene Lösung in der oben angegebenen Form ausdrucken, also als Matrix mit der Besuchsreihenfolge pro Feld.

```
/**
 * Nach <a href=
 * "https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-
 * → 1">geeksforgeeks.org</a>
 */
public class Springerproblem {
    static int felderAnzahl = 8;

    static int lösung[] [];

    static int xBewegungen[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
    static int yBewegungen[] = { 1, 2, 2, 1, -1, -2, -2, -1 };

    static void druckeLösung() {
        for (int x = 0; x < felderAnzahl; x++) {
            for (int y = 0; y < felderAnzahl; y++) {
                System.out.print(lösung[x][y] + " ");
            }
            System.out.println();
        }
    }

    /**
     * Versuche zum angegebenen Feld zu hüpfen.
     *
     * @param x Die x-Koordinate des als nächstes anzuspringenden Feldes.
     * @param y Die y-Koordinate des als nächstes anzuspringenden Feldes.
     * @param z Die aktuelle Rekursionstiefe.
     *
     * @return Wahr wenn das „angehüpfte“ Feld in der Lösung ist, sonst falsch.
     */
    static boolean huepf(int x, int y, int z) {
        // nächste x-Koordinate
        int xN;
        // nächste y-Koordinate
```

```

int yN;
if (z == felderAnzahl * felderAnzahl) {
    return true;
}

for (int i = 0; i < 8; i++) {
    xN = x + xBewegungen[i];
    yN = y + yBewegungen[i];
    if (xN >= 0 && xN < felderAnzahl && yN >= 0 && yN < felderAnzahl &&
        ↪ lösung[xN][yN] == -1) {
        lösung[xN][yN] = z;
        if (huepf(xN, yN, z + 1)) {
            return true;
        } else {
            // backtracking
            lösung[xN][yN] = -1;
        }
    }
}
return false;
}

static boolean löseSpringerproblem(int x, int y) {
    lösung = new int[8][8];

    for (int i = 0; i < felderAnzahl; i++) {
        for (int j = 0; j < felderAnzahl; j++) {
            lösung[i][j] = -1;
        }
    }

    lösung[x][y] = 0;

    if (!huepf(x, y, 1)) {
        System.out.println("Es konnte keine Lösung gefunden werden.");
        return false;
    } else {
        druckeLösung();
    }

    return true;
}

public static void main(String args[]) {
    löseSpringerproblem(0, 0);
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2018/herbst/Springerproblem.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2018/herbst/Springerproblem.java)

66115 / 2007 / Frühjahr / Thema 2 / Aufgabe 1

- (a) Beschreiben Sie in Pseudocode oder einer Programmiersprache Ihrer Wahl einen Greedy-Algorithmus, der einen Betrag von n Cents mit möglichst wenigen Cent-

Münzen herausgibt. Bei $n = 29$ wäre die erwartete Antwort etwa $1 \times 20\text{ct}$, $1 \times 5\text{ct}$, $2 \times 2\text{ct}$.

- (b) Beweisen Sie die Korrektheit Ihres Verfahrens, also dass tatsächlich die Anzahl der Münzen minimiert wird.
- (c) Nehmen wir an, Bayern führe eine Sondermünze im Wert von 7ct ein. Dann liefert der naheliegende Greedy-Algorithmus nicht immer die minimale Zahl von Münzen. Geben Sie für dieses Phänomen ein konkretes Beispiel an und führen Sie aus, warum Ihr Beweis aus Aufgabenteil a) in dieser Situation nicht funktioniert.

66115 / 2009 / Herbst / Thema 2 / Aufgabe 6

Die Wäscheleinaufgabe besteht darin, n Wäschestücke der Breiten b_1, b_2, \dots, b_n auf Wäscheleinen der Breite b aufzuhängen. Idealerweise sollte die Zahl der benutzten Leinen möglichst klein werden. Formal ist eine Aufhängung der Wäsche auf l Leinen also eine Einteilung der Menge $\{1, \dots, n\}$ in l Klassen L_1, \dots, L_l , sodass für alle $j = 1 \dots l$ gilt $\sum_{i \in L_j} b_i \leq b$. Eine Lösung der Wäscheleinaufgabe ist dann eine Zahl l und eine Aufhängung der Wäsche auf l Leinen. Eine Lösung ist umso besser, je kleiner l ist.

- (a) Beschreiben Sie einen sinnvollen Greedy-Algorithmus für das Wäscheleinenproblem. (Also nicht einfach für jedes Wäschestück eine neue Leine)
- (b) Geben Sie ein Beispiel einer Wäscheladung (Instanz des Wäscheleinenproblems), für die Ihr Algorithmus mehr als die minimal mögliche Zahl von Leinen verbraucht.
- (c) Nennen Sie ein Beispiel einer Problemstellung, die mit einem Greedy-Algorithmus optimal gelöst werden kann.

66115 / 2012 / Herbst / Thema 1 / Aufgabe 4

Gegeben ist ein Array a von ganzen Zahlen der Länge n , z. B. :

i	0	1	2	3	4	5	6	7	8	9
a_i	5	-6	4	2	-5	7	-2	-7	3	5

Im Beispiel ist also $n = 10$. Es soll die maximale Teilsumme berechnet werden, also der Wert des Ausdrucks

$$\max_{i,j \leq n} \sum_{k=i}^{j-1} a_k$$

Im Beispiel ist dieser Wert 8 und wird für $i = 8, j = 10$ erreicht. Entwerfen Sie ein Divide-And-Conquer Verfahren, welches diese Aufgabenstellung in Zeit $\mathcal{O}(n \log n)$ löst. Skizzieren Sie Ihre Lösung hinreichend detailliert.

Tipp: Sie sollten ein geringfügig allgemeineres Problem lösen, welches neben der maximalen Teilsumme auch noch die beiden „maximalen Randsummen“ berechnet. Die werden dann bei der Endausgabe verworfen.


```
/**
 * Klasse zur Berechnung der maximalen Teilsumme einer Zahlenfolge.
 *
 * nach Teilsumme.java Klasse mit Algorithmen für die Berechnung des größten
 * gemeinsamen Teilers zweier Ganzzahlen Algorithmen und Datenstrukturen,
 * Auflage 4, Kapitel 2.1
 *
 * nach Prof. Grude, Prof. Solymosi, (c) 2000-2008: 22. April 2008
 * <a href="http://public.beuth-hochschule.de/oo-
 * ↪ plug/A&D/prog/kap21/Teilsumme.java">Teilsumme.java</a>
 */
public class Teilsumme {

    /**
     * Berechne die maximale Teilsumme an der rechten Grenze. Die Eingabeparameter
     * müssen diese Werte aufweisen: 0 <= links <= rechts <= folge.length.
     *
     * @param folge Die Zahlenfolge, in der die maximale Zahlensumme gerechnet
     *              werden soll.
     * @param links Die Index-Nummer der linken Grenze.
     * @param rechts Die Index-Nummer der rechten Grenze.
     *
     * @return Die maximale Teilsumme.
     */
    private static int berechneRandRechts(int[] folge, int links, int rechts) {
        int max = 0;
        int sum = 0;
        for (int i = rechts; i >= links; i--) {
            sum += folge[i];
            max = Math.max(max, sum);
        }
        return max;
    }

    /**
     * Berechne die maximale Teilsumme an der linken Grenze. Die Eingabeparameter
     * müssen diese Werte aufweisen: 0 <= links <= rechts <= folge.length.
     *
     * @param folge Die Zahlenfolge, in der die maximale Zahlensumme gerechnet
     *              werden soll.
     * @param links Die Index-Nummer der linken Grenze.
     * @param rechts Die Index-Nummer der rechten Grenze.
     *
     * @return Die maximale Teilsumme.
     */
    private static int berechneRandLinks(int[] folge, int links, int rechts) {
        int max = 0;
        int sum = 0;
        for (int i = links; i <= rechts; i++) {
            sum += folge[i];
            max = Math.max(max, sum);
        }
        return max;
    }
}
```

```

/**
 * Berechne die maximale Teilsumme in der Zahlenfolge zwischen einer gegebenen
 * linken und rechten Grenze. Die Eingabeparameter müssen diese Werte aufweisen:
 * 0 <= links <= rechts <= folge.length.
 *
 * @param folge Die Zahlenfolge, in der die maximale Zahlensumme gerechnet
 *              werden soll.
 * @param links Die Index-Nummer der linken Grenze.
 * @param rechts Die Index-Nummer der rechten Grenze.
 *
 * @return Die maximale Teilsumme.
 */
private static int berechne(int[] folge, int links, int rechts) {
    if (links == rechts) // nur ein Element
        return Math.max(0, folge[links]);
    else {
        final int mitte = (rechts + links) / 2;
        final int maxLinks = berechne(folge, links, mitte);
        final int maxRechts = berechne(folge, mitte + 1, rechts);
        final int maxGrenzeRechts = berechneRandRechts(folge, links, mitte);
        // linke Hälfte
        final int maxGrenzeLinks = berechneRandLinks(folge, mitte + 1, rechts);
        // rechte Hälfte
        return Math.max(maxRechts, Math.max(maxLinks, maxGrenzeRechts +
            ↪ maxGrenzeLinks));
    }
}

/**
 * Berechne die maximale Teilsumme einer Zahlenfolge rekursiv mit
 * logarithmischer Zeitkomplexität.
 *
 * @param folge Die Zahlenfolge, in der die maximale Zahlensumme gerechnet
 *              werden soll.
 *
 * @return Die maximale Teilsumme.
 */
public static int berechne(int[] folge) {
    return berechne(folge, 0, folge.length - 1);
}

public static void main(String[] args) {
    int[] folge = { 5, -6, 4, 2, -5, 7, -2, -7, 3, 5 };
    int ergebnis = berechne(folge);
    System.out.println(ergebnis);
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2012/herbst/Teilsumme.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_66115/jahr_2012/herbst/Teilsumme.java)

66115 / 2016 / Herbst / Thema 1 / Aufgabe 4

Es sei $A[0 \dots n - 1]$ ein Array von paarweise verschiedenen ganzen Zahlen.

Wir interessieren uns für die Zahl der Inversionen von A ; das sind Paare von Indices

(i, j) , sodass $i < j$ aber $A[i] > A[j]$. Die Inversionen im Array $[2, 3, 8, 6, 1]$ sind $(0, 4)$, da $A[0] > A[4]$ und weiter $(1, 4)$, $(2, 3)$, $(2, 4)$, $(3, 4)$. Es gibt also 5 Inversionen.

- (a) Wie viel Inversionen hat das Array $[3, 7, 1, 4, 5, 9, 2]$?

Lösungsvorschlag

- $(0, 1): 3 > 1$
- $(0, 6): 3 > 2$
- $(1, 2): 7 > 1$
- $(1, 3): 7 > 4$
- $(1, 4): 7 > 5$
- $(1, 6): 7 > 2$
- $(3, 6): 4 > 2$
- $(4, 6): 5 > 2$

- (b) Welches Array mit den Einträgen $\{1, \dots, n\}$ hat die meisten Inversionen, welches hat die wenigsten?

Lösungsvorschlag

Folgt nach der 1 eine absteigend sortierte Folge, so hat sie am meisten Inversionen, z. B. $\{1, 7, 6, 5, 4, 3, 2\}$. Eine aufsteigend sortierte Zahlenfolge hat keine Inversionen, z. B. $\{1, 2, 3, 4, 5, 6, 7\}$.

- (c) Entwerfen Sie eine Prozedur `int merge(int[] a, int i, int h, int j);` welche das Teilarray $a[i..j]$ sortiert und die Zahl der in ihm enthaltenen Inversionen zurückliefert, wobei die folgenden Vorbedingungen angenommen werden:

- $0 \leq i \leq h \leq j < n$, wobei n die Länge von a ist ($n = a.length$).
- $a[i \dots h]$ und $a[h + 1 \dots j]$ sind aufsteigend sortiert.
- Die Einträge von $a[i \dots j]$ sind paarweise verschieden.

Ihre Prozedur soll in linearer Zeit, also $\mathcal{O}(j - i)$ laufen. Orientieren Sie sich bei Ihrer Lösung an der Mischoperation des bekannten Mergesort-Verfahrens.

- (d) Entwerfen Sie nun ein Divide-and-Conquer-Verfahren zur Bestimmung der Zahl der Inversionen, indem Sie angelehnt an das Mergesort-Verfahren einen Algorithmus [ZI](#) beschreiben, der ein gegebenes Array in sortierter Form liefert und gleichzeitig dessen Inversionsanzahl berechnet. Im Beispiel wäre also

$$ZI([2, 3, 8, 6, 1]) = ([1, 2, 3, 6, 8], 5)$$

Die Laufzeit Ihres Algorithmus auf einem Array der Größe n soll $\mathcal{O}(n \log(n))$ sein.

Sie dürfen die Hilfsprozedur `merge` aus dem vorherigen Aufgabenteil verwenden, auch, wenn Sie diese nicht gelöst haben.

- (e) Begründen Sie, dass Ihr Algorithmus die Laufzeit $\mathcal{O}(n \log(n))$ hat.
- (f) Geben Sie die Lösungen folgender asymptotischer Rekurrenzen (in O-Notation) an:

- (i) $T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(\log n)$
- (ii) $T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n^2)$
- (iii) $T(n) = 3 \cdot T(\frac{n}{2}) + \mathcal{O}(n)$

66115 / 2017 / Herbst / Thema 1 / Aufgabe 8

Sei $X = (I_1, \dots, I_n)$ eine Menge von n (geschlossenen) Intervallen über den reellen Zahlen \mathbb{R} . Das Intervall I_j sei dabei gegeben durch seine linke Intervallgrenze $l_j \in \mathbb{R}$ sowie seine rechte Intervallgrenze $r_j \in \mathbb{R}$ mit $r_j > l_j$, $I_j = [l_j, r_j]$.

Wir nehmen in dieser Aufgabe der Einfachheit halber an, dass die Zahlen alle paarweise verschieden sind.

Zwei Intervalle I_j, I_k überlappen sich gdw. sie mindestens einen Punkt gemeinsam haben, d.h. falls für (o.B.d.A.) $l_j < r_k$, auch $l_k < r_j$ gilt. Eine gültige Färbung von X mit $c \in \mathbb{N}$ Farben ist eine Funktion $F : X \rightarrow \{1, 2, \dots, c\}$ mit der Eigenschaft, dass für jedes Paar I_j, I_k von überlappenden Intervallen $F(I_j) \neq F(I_k)$ gilt.

Abbildung 1: Eine gültige Färbung von X

Eine minimale gültige Färbung von X ist eine gültige Färbung mit einer minimalen Anzahl an Farben. Die Anzahl von Farben in einer minimalen gültigen Färbung von X bezeichnen wir mit $\chi(X)$. Wir gehen im Folgenden davon aus, dass für X eine minimale gültige Färbung F^* gefunden wurde.

- (a) Nehmen wir an, dass aus X alle Intervalle einer bestimmten Farbe von F^* gelöscht werden. Ist die so aus F^* entstandene Färbung der übrigen Intervalle in jedem Fall immer noch eine minimale gültige Färbung? Begründen Sie Ihre Antwort.
- (b) Nehmen wir an, dass aus X ein beliebiges Intervall gelöscht wird. Ist die so aus F^* entstehende Färbung der übrigen Intervalle in jedem Fall immer noch eine minimale gültige Färbung? Begründen Sie Ihre Antwort.
- (c) Mit $u_j(X)$ bezeichnen wir die maximale Anzahl von Intervallen in X , die sich paarweise überlappen. Zeigen Sie, dass $\chi(X) = u_j(X)$ ist. Wir betrachten nun folgenden Algorithmus, der die Menge $X = (I_1, \dots, I_n)$ von n Intervallen einfärbt:
- Zunächst sortieren wir die Intervalle von X aufsteigend nach ihren linken Intervallgrenzen. Die Intervalle werden jetzt in dieser Reihenfolge nacheinander eingefärbt; ist ein Intervall dabei erst einmal eingefärbt, ändert sich seine Farbe nie wieder. Angenommen die sortierte Reihenfolge der Intervalle sei I_1, I_2, \dots, I_n .
 - Das erste Intervall I_1 erhält die Farbe 1. Für $1 < i \leq n$ verfahren wir im i -ten Schritt zum Färben des i -ten Intervalls wie folgt:

Bestimme die Menge C_j aller Farben der bisher schon eingefärbten Intervalle die $/_{j-1}$ überlappen. Färbe $/_{j-1}(j)$ dann mit der Farbe $c_j = \min(\{1, 2, \dots, n\} \setminus C_j)$.
Fortsetzung nächste Seite!

- (d) Begründen Sie, warum der Algorithmus immer eine gültige Färbung von X findet (Hinweis: Induktion).
- (e) Zeigen Sie, dass die Anzahl an Farben, die der Algorithmus für das Einfärben benötigt, mindestens $\chi(X)$ ist.
- (f) Zeigen Sie, dass die Anzahl an Farben, die der Algorithmus für das Einfärben benötigt, höchstens $\chi(X)$ ist.
- (g) Begründen Sie mit Hilfe der o.g. Eigenschaften, warum der Algorithmus korrekt ist, d.h. immer eine minimale gültige Färbung von X findet.
- (h) Wir betrachten folgende Implementierung des Algorithmus in Pseudocode:
Was ist die asymptotische Laufzeit dieses Algorithmus? Was ist der asymptotische Speicherbedarf dieses Algorithmus? Begründen Sie Ihre Antworten.

66115 / 2018 / Herbst / Thema 2 / Aufgabe 6

Ein sehr bekanntes Optimierungsproblem ist das sogenannte Rucksackproblem: Gegeben ist ein Rucksack mit der Tragfähigkeit B . Weiterhin ist eine endliche Menge von Gegenständen mit Werten und Gewichten gegeben. Nun soll eine Teilmenge der Gegenstände so ausgewählt werden, dass ihr Gesamtwert maximal ist, aber ihr Gesamtgewicht die Tragfähigkeit des Rucksacks nicht überschreitet.

Mathematisch exakt kann das Rucksackproblem wie folgt formuliert werden:

Gegeben ist eine endliche Menge von Objekten U . Durch eine Gewichtsfunktion $w: U \rightarrow \mathbb{R}^+$ wird den Objekten ein Gewicht und durch eine Nutzenfunktion $v: U \rightarrow \mathbb{R}^+$ ein festgelegter Nutzwert zugeordnet.

Des Weiteren gibt es eine vorgegebene Gewichtsschranke $B \in \mathbb{R}^+$. Gesucht ist eine Teilmenge $K \subseteq U$, die die Bedingung $\sum_{u \in K} w(u) \leq B$ einhält und die Zielfunktion $\sum_{u \in K} v(u)$ maximiert.

Das Rucksackproblem ist NP-vollständig (Problemgröße ist die Anzahl der Objekte), sodass es an dieser Stelle wenig Sinn macht, über eine effiziente Lösung nachzudenken. Lösen Sie das Rucksackproblem daher mittels Backtracking und formulieren Sie einen entsprechenden Algorithmus. Gehen Sie davon aus, dass die Gewichtsschranke B sowie die Anzahl an Objekten N beliebig, aber fest vorgegeben sind.

Das Programm soll folgende Ausgaben liefern:

- (a) Maximaler Nutzwert, der durch eine Objektauswahl unter Einhaltung der Gewichtsschranke B erreicht werden kann.
- (b) Das durch die maximierende Objektmenge erreichte Gesamtgewicht.
- (c) Diejenigen Objekte (Objektnummern) aus U , die zur Maximierung des Nutzwerts beigetragen haben.


```
/**
 * https://stackoverflow.com/a/14186622
 */
public class Rucksack {
    // static int[] werte = new int[] { 894, 260, 392, 281, 27 };
    // static int[] gewichte = new int[] { 8, 6, 4, 0, 21 };
    // static int[] werte = new int[] { 4, 2, 10, 1, 2 };
    // static int[] gewichte = new int[] { 12, 1, 4, 1, 2 };
    static int werte[] = new int[] { 60, 100, 120 };
    static int gewichte[] = new int[] { 10, 20, 30 };

    /**
     * Gewichtsschranke
     */
    //static int B = 30;
    //static int B = 15;
    static int B = 50;

    /**
     * Diejenigen Objekte aus U, die zur Maximierung des Nutzwerts beigetragen
     * haben.
     */
    static boolean[] auswahl = new boolean[werte.length];

    private static int berechne(int i, int W) {
        if (i < 0) {
            return 0;
        }
        int alt = berechne(i - 1, W);
        if (gewichte[i] > W) {
            // Backtracking!
            auswahl[i] = false;
            return alt;
        } else {

            int neu = berechne(i - 1, W - gewichte[i]) + werte[i];
            if (alt >= neu) {
                // Backtracking!
                auswahl[i] = false;
                return alt;
            } else {
                auswahl[i] = true;
                return neu;
            }
        }
    }

    static void werteAus() {
        System.out.println(berechne(werte.length - 1, B));

        int gesamtGewicht = 0;
        int gesamtWert = 0;
    }
}
```

```

for (int i = 0; i < auswahl.length; i++) {
    if (auswahl[i]) {
        gesamtGewicht += gewichte[i];
        gesamtWert += werte[i];
        System.out.println("Objekt-Nr. " + i + " Gewicht: " + gewichte[i] +
            → " Wert: " + werte[i]);
    }
}

System.out.println("Gesamtgewicht: " + gesamtGewicht);
System.out.println("Gesamtwert: " + gesamtWert);
}

public static void main(String[] args) {
    werteAus();
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2018/herbst/Rucksack.java](https://github.com/org/bschlangaul/examen/examen_66115/jahr_2018/herbst/Rucksack.java)

66115 / 2019 / Frühjahr / Thema 1 / Aufgabe 6

Aus dem Känguru-Wettbewerb 2017 — Klassenstufen 3 und 4.

Luna hat für den Kuchenbasar Muffins mitgebracht: 10 Apfelmuffins, 18 Nussmuffins, 12 Schokomuffins und 9 Blaubeermuffins. Sie nimmt immer 3 verschiedene Muffins und legt sie auf einen Teller. Welches ist die kleinste Zahl von Muffins, die dabei übrig bleiben können?

A: 1, B: 3, C: 4, D: 7, E: 8

- (a) Geben Sie die richtige Antwort auf die im Känguru-Wettbewerb gestellte Frage und begründen Sie sie.

Lösungsvorschlag

4^a

^a<https://www.youtube.com/watch?v=ceJW9kAp1VY>

- (b) Lunas Freundin empfiehlt den jeweils nächsten Teller immer aus den drei aktuell häufigsten Muffinsorten zusammenzustellen. Leiten Sie aus dieser Idee einen effizienten GreedyAlgorithmus her, der die Fragestellung für beliebige Anzahlen von Muffins löst (nach wie vor soll es nur vier Sorten und je drei pro Teller geben). Skizzieren Sie in geeigneter Form, wie Ihr Algorithmus die Beispielinstantz von oben richtig löst.

Lösungsvorschlag

```

public static int berechneRest4Sorten3ProTeller() {
    int[] muffins = new int[] { 10, 18, 12, 9 };
    int n = muffins.length;
    sortiere(muffins);
}

```



```
// Wir nehmen uns 3 verschiedene Muffins solange, wie die dritthäufigste
// Muffinsorte noch Muffins hat.
while (muffins[n - 3] > 0) {
    muffins[n - 1]--;
    muffins[n - 2]--;
    muffins[n - 3]--;
    sortiere(muffins);
}
return berechneGesamtzahl(muffins);
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/fruehjahr/Muffin.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2019/fruehjahr/Muffin.java)

- (c) Beschreiben Sie eine mögliche und sinnvolle Verallgemeinerung Ihrer Lösung auf n Muffinsorten und k Muffins pro Teller für $n > 4$ und $k > 3$.

Lösungsvorschlag

```
public static int berechneRestAllgemein(int[] muffins, int k) {
    int n = muffins.length;
    sortiere(muffins);
    while (muffins[n - k] > 0) {
        for (int i = 1; i <= k; i++) {
            muffins[n - i]--;
        }
        sortiere(muffins);
    }
    return berechneGesamtzahl(muffins);
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/fruehjahr/Muffin.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2019/fruehjahr/Muffin.java)

- (d) Diskutieren Sie, wie man die Korrektheit des Greedy-Algorithmus zeigen könnte, also dass er tatsächlich immer eine optimale Lösung findet. Ein kompletter, rigoroser Beweis ist nicht verlangt.

66115 / 2020 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 4

Das GUTSCHEIN-Problem ist gegeben durch eine Folge w_1, \dots, w_n von Warenwerten (wobei $w \in \mathbb{N}_0$ für $i = 1, \dots, n$) und einem Gutscheinbetrag $G \in \mathbb{N}_0$.

Da Gutscheine nicht in Bargeld ausgezahlt werden können, ist die Frage, ob man eine Teilfolge der Waren findet, sodass man genau den Gutschein ausnutzt. Formal ist dies die Frage, ob es eine Menge von Indizes I mit $I \subseteq \{1, \dots, n\}$ gibt, sodass $\sum_{i \in I} w_i = G$.

Exkurs: Teilsommenproblem

Das **Teilsommenproblem** (SUBSET SUM oder SSP) ist ein spezielles Rucksackproblem. Gegeben sei eine Menge von ganzen Zahlen $I = \{w_1, w_2, \dots, w_n\}$. Gesucht ist eine Untermenge, deren Elementsumme maximal, aber nicht größer als eine gegebene obere Schranke c ist.

- (a) Sei $w_1 = 10, w_2 = 30, w_3 = 40, w_4 = 20, w_5 = 15$ eine Folge von Warenwerten.
- (i) Geben Sie einen Gutscheinbetrag $40 < G < 115$ an, sodass die GUTSCHEIN-Instanz eine Lösung hat. Geben Sie auch die lösende Menge $I \subseteq \{1, 2, 3, 4, 5\}$ von Indizes an.

Lösungsvorschlag

50
 $I = \{1, 3\}$

- (ii) Geben Sie einen Gutscheinbetrag G mit $40 < G < 115$ an, sodass die GUTSCHEIN-Instanz keine Lösung hat.

Lösungsvorschlag

51

- (b) Sei $table$ eine $(n \times (G + 1))$ -Tabelle mit Einträgen $table[i, k]$, für $1 \leq i \leq n$ und $0 \leq k \leq G$, sodass

$$table[i, k] = \begin{cases} \mathbf{true} & \text{falls es } I \subseteq \{1, \dots, n\} \text{ mit } \sum_{i \in I} w_i = G \text{ gibt} \\ \mathbf{false} & \text{sonst} \end{cases}$$

Geben Sie einen Algorithmus in Pseudo-Code oder Java an, der die Tabelle $table$ mit *dynamischer Programmierung* in Worst-Case-Laufzeit $\mathcal{O}(n \times G)$ erzeugt. Begründen Sie die Korrektheit und die Laufzeit Ihres Algorithmus. Welcher Eintrag in $table$ löst das GUTSCHEIN-Problem?

Lösungsvorschlag

Algorithmus 3: Gutschein-Problem

```

table = boolean array  $n + 1 \times (G + 1)$  ;    // Initialisiere ein boolsches Feld
    mit  $n + 1$  Zeilen für jeden Warenwert und 0 für keinen Warenwert und mit  $G + 1$  Spalten
    für alle Gutscheinbetrag bis  $G$  und 0 für keinen Gutscheinbetrag

for  $k$  in  $1 \dots G$  do ;           // Wenn der Gutscheinbetrag größer als 0 ist und es keine
    Warenwerte gibt,  $\forall n = 0$ , kann der Gutschein nicht eingelöst werden.

    | table[0][ $k$ ] = false
end

for  $i$  in  $0 \dots n$  do ;           // Ist der Gutscheinbetrag 0, dann kann er immer eingelöst
    werden.

    | table[i][0] = true
end
for  $i$  in  $1 \dots n$  do ;           // Durchlaufe jede Zeile der Warenwerte

    for  $k$  in  $1 \dots G$  do ;       // Durchlaufe jede Spalte der Gutscheinbeträge in dieser
        Zeile

        | table[i][ $k$ ] = table[i - 1][ $k$ ] ;    // Übernehme erstmals das Ergebnis der
            Zelle der vorhergehenden Zeile in der gleichen Spalte
        if  $k \geq w_i$  und table[i][ $k$ ] noch nicht markiert then ;           // Wenn der
            aktuelle Gutscheinbetrag größer als der aktuelle Warenwert und die aktuelle
            Zelle noch nicht als wahr markiert ist

            | table[i][ $k$ ] = table[i - 1][ $k - w_i$ ] ;    // übernimmt das Ergebnis des
                aktuellen Gutscheinbetrags minus des aktuellen Warenwerts
            ;
        end
    end
end

```

```

/**
 * Nach <a href="https://www.geeksforgeeks.org/subset-sub-problem-dp-25">
 * → Subset
 * Sum Problem auf geeksforgeeks.org</a>
 */
public class Gutschein {
    /**
     * @param G Die Indizes der GUTSCHEIN-Beträge.
     *
     * @param W Das GUTSCHEIN-Problem ist gegeben durch eine Folge  $w_1, \dots, w_n$ 
     * → von
     *
     * Warenwerten.
     */
}

```

```

*
* @return Wahr, wenn der Gutscheinbetrag vollständig in Warenwerten
→ eingelöst
*       werden kann, falsch wenn der Betrag nicht vollständig eingelöst
*       werden kann.
*/
public static boolean gutscheinDP(int G, int W[]) {
    // Der Eintrag in der Tabelle tabelle[i][k] ist wahr,
    // wenn es eine Teilsumme der
    // W[0..i-1] gibt, die gleich k ist.
    int n = W.length;
    boolean table[][] = new boolean[n + 1][G + 1];

    // Wenn der Gutschein-Betrag größer als 0 ist und es keine
    // Warenwerte (n = 0) gibt, kann der Gutschein nicht eingelöst
    // werden.
    for (int k = 1; k <= G; k++) {
        table[0][k] = false;
    }

    // Ist der Gutscheinbetrag 0, dann kann er immer eingelöst werden.
    for (int i = 0; i <= n; i++) {
        table[i][0] = true;
    }

    for (int i = 1; i <= n; i++) {
        for (int k = 1; k <= G; k++) {
            table[i][k] = table[i - 1][k];
            // Warenwert
            int w = W[i - 1];
            if (k >= w && !table[i][k]) {
                table[i][k] = table[i - 1][k - w];
            }
        }
    }
    return table[n][G];
}

public static void main(String[] args) {
    System.out.println(gutscheinDP(50, new int[] { 10, 30, 40, 20, 15 }));
    System.out.println(gutscheinDP(41, new int[] { 10, 30, 40, 20, 15 }));

    System.out.println(gutscheinDP(3, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(5, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(6, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(2, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(1, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(7, new int[] { 1, 2, 3 }));
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Gutschein.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/Gutschein.java)

Die äußere for-Schleife läuft n mal und die innere for-Schleife G mal.

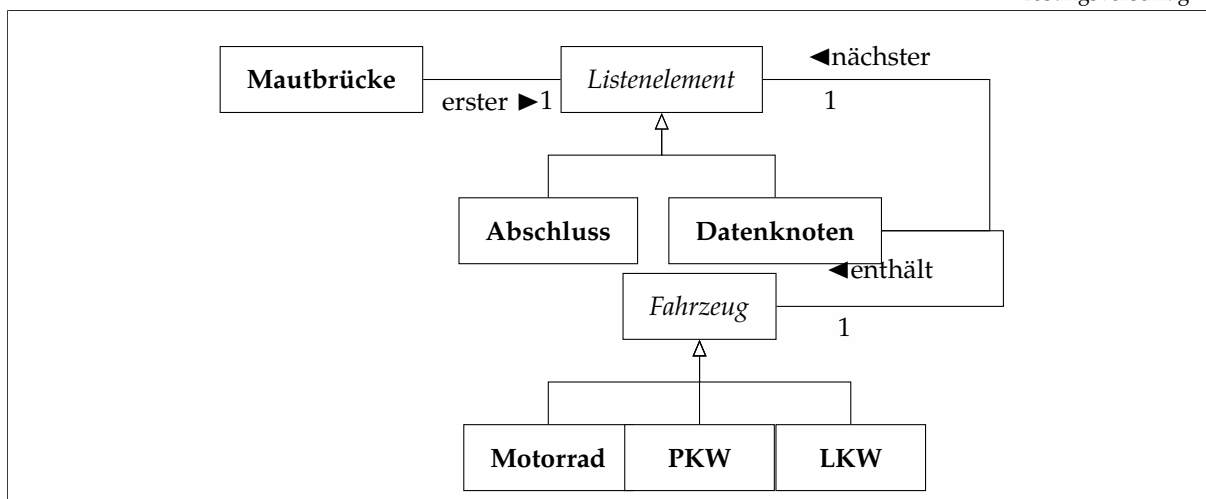
Der letzte Eintrag in der Tabelle, also der Wert in der Zelle `table[W.length]` `[G]`, löst das Gutscheinproblem. Steht hier `true`, dann gibt es eine Teilfolge der Waren, die den Gutscheinbetrag genau ausnutzt.

Listen

66115 / 2020 / Herbst / Thema 2 / Aufgabe 4

Für die Umsetzung der Maut auf deutschen Autobahnen soll eine Java-basierte Lösung entworfen werden. Dazu sollen alle Fahrzeuge, die von einer Mautbrücke erfasst werden, in einer *einfach verketteten Liste* abgelegt werden. Um einen besseren Überblick über die Einnahmen zu erhalten, soll zwischen *LKWs*, *PKWs* und *Motorrädern* unterschieden werden. Als Informatiker schlagen Sie eine *heterogene Liste* zur Realisierung vor. Notieren Sie unter Verwendung des *Entwurfsmusters Kompositum* ein entsprechendes *Klassendiagramm* zur Realisierung der Lösung für eine Mautbrücke. Auf die Angabe von Attributen und Methoden kann verzichtet werden. Kennzeichnen Sie in Ihrem Klassendiagramm die *abstrakten Klassen* und benennen Sie die bestehenden *Beziehungen*.

Lösungsvorschlag



66115 / 2020 / Herbst / Thema 2 / Aufgabe 4

Erstellen Sie ein Deutsch-Englisch Wörterbuch. Verwenden Sie dazu eine einfach verkettete Liste mit Kompositum. Identifizieren Sie die benötigten Klassen, legen Sie das Wörterbuch an und implementieren Sie anschließend die geforderten Methoden.

- Ein Listenelement, welches immer jeweils auf seinen Nachfolger verweisen kann, enthält jeweils einen Eintrag des Wörterbuchs. Ein Eintrag besteht aus dem deutschen und dem zugehörigen englischen Wort. Diese können natürlich jeweils zurückgegeben werden.
- Mit der Methode `einfuegen (String deutsch, String englisch)` soll ein neuer Eintrag in das Wörterbuch eingefügt werden können. Wie in jedem Wörterbuch müssen die (deutschen) Einträge jedoch alphabetisch sortiert sein, sodass nicht an einer beliebigen Stelle eingefügt werden kann. Um die korrekte Einfügeposition zu finden, ist das Vergleichen von Strings notwendig. Recherchieren Sie dazu, wie die Methode `compareTo()` in Java funktioniert!

- Der Aufruf der Methode `uebersetze(String deutsch)` auf der Liste soll nun für ein übergebenes deutsches Wort die englische Übersetzung ausgeben.

Lösungsvorschlag

Klasse WörterbuchEintrag

Die abstrakte Klasse im Kompositumentwurfsmuster von der sowohl die primitive Klasse als auch die Behälterklasse erben.

```
public abstract class WoerterbuchEintrag {
    protected WortPaar nächstes;

    protected WortPaar gibNächstes () {
        return nächstes;
    }

    protected void setzeNächstes (WortPaar wortPaar) {
        nächstes = wortPaar;
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/woerterbuch/WoerterbuchEintrag.java](https://github.com/bschlangaul/aufgaben/aud/listen/woerterbuch/WoerterbuchEintrag.java)

Klasse WortPaar

Das Listenelement (die primitive Klasse im Kompositumentwurfsmuster).

```
public class WortPaar extends WoerterbuchEintrag {
    private final String deutsch;

    private final String englisch;

    public WortPaar(String deutsch, String englisch) {
        this.deutsch = deutsch;
        this.englisch = englisch;
    }

    public String gibDeutschesWort() {
        return deutsch;
    }

    public String gibEnglischesWort() {
        return englisch;
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/woerterbuch/WortPaar.java](https://github.com/bschlangaul/aufgaben/aud/listen/woerterbuch/WortPaar.java)

Klasse Wörterbuch

Die Behälterklasse im Kompositumentwurfsmuster.

```

*/
public class Woerterbuch extends WoerterbuchEintrag {

    public void einfügen(String deutsch, String englisch) {
        WortPaar wort = new WortPaar(deutsch, englisch);

        // Spezialbehandlung, wenn vor das erste Wortpaar des Wörterbuchs eingefügt
        // werden muss.
        WortPaar kopf = gibNächstes();
        if (kopf == null || kopf.gibDeutschesWort().compareTo(wort.gibDeutschesWort())
            ↪ >= 0) {
            wort.setzeNächstes(kopf);
            setzeNächstes(wort);
            return;
        }
        WortPaar vergleichsWort = gibNächstes();
        while (vergleichsWort.gibNächstes() != null
            &&
            ↪ vergleichsWort.gibNächstes().gibDeutschesWort().compareTo(wort.gibDeutschesWort())
            ↪ < 0) {
            vergleichsWort = vergleichsWort.gibNächstes();
        }
        wort.setzeNächstes(vergleichsWort.gibNächstes());
        vergleichsWort.setzeNächstes(wort);
    }

    public String übersetze(String deutsch) {
        if (gibNächstes() == null) {
            return "Noch keine Wörter im Wörterbuch.";
        }
        WortPaar wort = gibNächstes();
        while (wort != null) {
            if (wort.gibDeutschesWort().equals(deutsch)) {
                return wort.gibEnglischesWort();
            }
            wort = wort.gibNächstes();
        }
        return "Es konnte keine passende Übersetzung gefunden werden";
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/woerterbuch/Woerterbuch.java](https://github.com/bschlangaul/aufgaben/aud/listen/woerterbuch/Woerterbuch.java)

Test

```

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class WoerterbuchTest {

    @Test
    public void methodeÜbersetze() {
        Woerterbuch wörterbuch = new Woerterbuch();
    }
}

```



```
assertEquals("Noch keine Wörter im Wörterbuch.",
    ↪ wörterbuch.übersetze("Wassermelone"));
}

@Test
public void methodeEinfügen() {
    Woerterbuch wörterbuch = new Woerterbuch();
    wörterbuch.einfügen("Wassermelone", "Watermelon");
    assertEquals("Watermelon", wörterbuch.übersetze("Wassermelone"));
}

@Test
public void sortierung() {
    Woerterbuch wörterbuch = new Woerterbuch();
    wörterbuch.einfügen("Wassermelone", "Watermelon");
    wörterbuch.einfügen("Apfel", "Apple");
    wörterbuch.einfügen("Zitrone", "Lemon");
    wörterbuch.einfügen("Birne", "Pear");
    wörterbuch.einfügen("Klementine", "Clementine");

    WortPaar paar;
    paar = wörterbuch.gibNächstes();
    assertEquals("Apfel", paar.gibDeutschesWort());

    paar = paar.gibNächstes();
    assertEquals("Birne", paar.gibDeutschesWort());

    paar = paar.gibNächstes();
    assertEquals("Klementine", paar.gibDeutschesWort());

    paar = paar.gibNächstes();
    assertEquals("Wassermelone", paar.gibDeutschesWort());

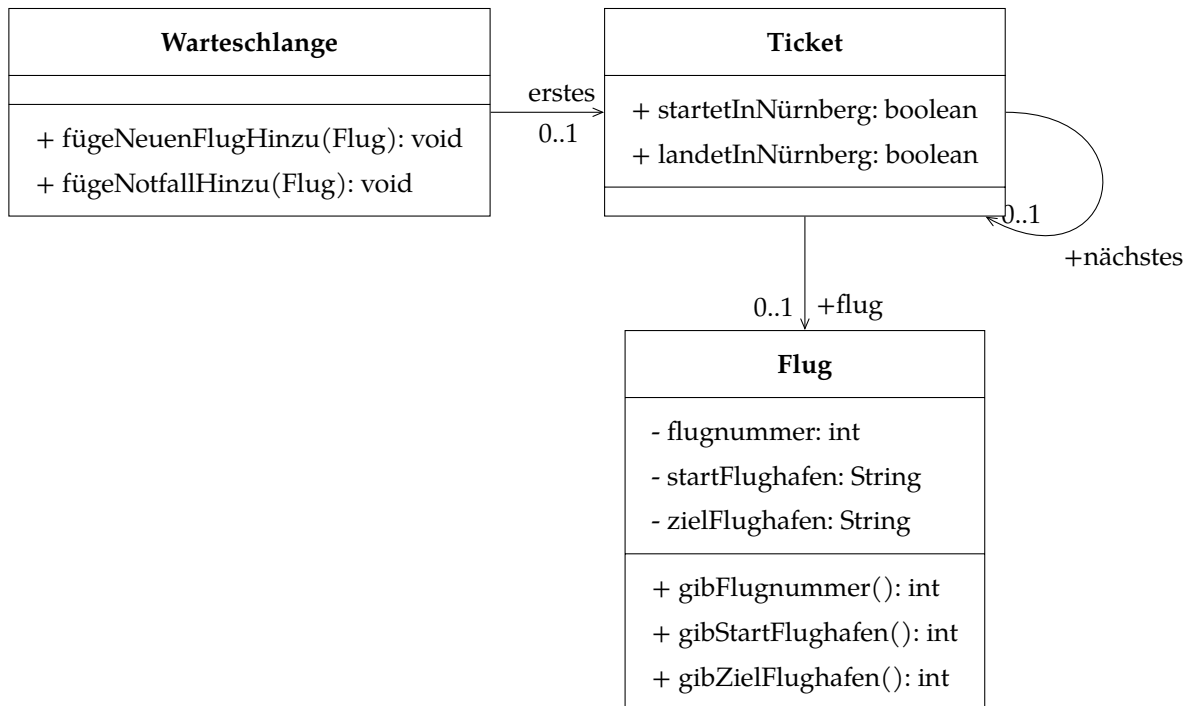
    paar = paar.gibNächstes();
    assertEquals("Zitrone", paar.gibDeutschesWort());
}
}
```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/aufgaben/aud/listen/woerterbuch/WoerterbuchTest.java](https://github.com/bschlangaul/aufgaben/aud/listen/woerterbuch/WoerterbuchTest.java)

66115 / 2020 / Herbst / Thema 2 / Aufgabe 4

Am Nürnberger Flughafen starten und landen täglich viele Flugzeuge. Der Flughafen verfügt jedoch nur über eine einzige Start- bzw. Landebahn, sodass Starts und Landungen gemeinsam koordiniert werden müssen. Für die interne Verwaltung, welcher Flieger als nächstes bearbeitet werden soll, wird eine neue Software entwickelt. Sie haben die Aufgabe einen Teil dieser Software zu erstellen.

Das folgende UML-Klassendiagramm gibt einen Überblick über den für Sie relevanten Teil der Software:



Implementieren Sie die beiden Methoden `fügeNeuenFlugHinzu(Flug)` und `fügeNotfallHinzu(Flug)` der Klasse **Warteschlange**!

- In der Methode `fügeNeuenFlugHinzu()`
 - soll ein neu übergebener Flug der Warteschlange hinzugefügt werden.
- In der Methode `fügeNotfallHinzu()`
 - soll für den übergebenen Flug überprüft werden, ob der Flug bereits in der Warteschlange vorkommt oder ob es sich um einen komplett neuen Flug handelt.
 - soll der Notfall die erste Priorität in der Warteschlange erhalten, özwingend als nächstes abgearbeitet werden.
 - soll die korrekte Funktionalität der Warteschlange weiterhin gegeben sein.

Jeder Flug wird in der Warteschlange mit einem eigenen Ticket verwaltet. Die Funktionalitäten der Tickets und der Flüge entnehmen Sie dem Quelltext.

Lösungsvorschlag

Klasse „Flug“

```

public class Flug {
    private int flugnummer;
    private String startFlughafen;
    private String zielFlughafen;

    public Flug(int flugnummer, String startFlughafen, String zielFlughafen) {
        this.flugnummer = flugnummer;
        this.startFlughafen = startFlughafen;
    }
  
```

```
    this.zielFlughafen = zielFlughafen;
}

public int gibFlugnummer() {
    return flugnummer;
}

public String gibStartFlughafen() {
    return startFlughafen;
}

public String gibZielFlughafen() {
    return zielFlughafen;
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/flughafen/Flug.java](https://github.com/bschlangaul/aufgaben/aud/listen/flughafen/Flug.java)

Klasse „Ticket“

```
public class Ticket {
    public Flug flug;
    public Ticket nächstes;
    public boolean startetInNürnberg = false;
    public boolean landetInNürnberg = false;

    public Ticket(Flug flug) {
        this.flug = flug;
        if (flug.gibStartFlughafen().equals("NUE")) {
            startetInNürnberg = true;
        } else {
            landetInNürnberg = true;
        }
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/flughafen/Ticket.java](https://github.com/bschlangaul/aufgaben/aud/listen/flughafen/Ticket.java)

Klasse „Warteschlange“

```
public class Warteschlange {
    Ticket erstes = null;

    public void fügeNeuenFlugHinzu(Flug flug) {
        Ticket zähler = erstes;
        if (erstes == null) {
            erstes = new Ticket(flug);
        } else {
            while (zähler.nächstes != null) {
                zähler = zähler.nächstes;
            }
            zähler.nächstes = new Ticket(flug);
        }
    }
}
```

```

    }

    public void fügeNotfallHinzu(Flug flug) {
        if (erstes.flug.gibFlugnummer() == flug.gibFlugnummer()) {
            return;
        }
        Ticket zähler = erstes;
        Ticket zähler2 = erstes.nächstes;
        Ticket notfall = null;
        while (zähler2 != null) {
            if (zähler2.flug.gibFlugnummer() == flug.gibFlugnummer()) {
                notfall = zähler2;
                zähler.nächstes = zähler2.nächstes;
                break;
            } else {
                zähler = zähler2;
                zähler2 = zähler2.nächstes;
            }
        }
        if (notfall == null) {
            notfall = new Ticket(flug);
        }
        notfall.nächstes = erstes;
        erstes = notfall;
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/listen/flughafen/Warteschlange.java](https://github.com/bschlangaul/aufgaben/aud/listen/flughafen/Warteschlange.java)

66115 / 2020 / Herbst / Thema 2 / Aufgabe 4

Im Restaurant der Biberschule gibt es normalerweise zwei Warteschlangen: In der einen holen sich die kleinen Biber ihre hohen grünen Teller, in der anderen holen sich die großen Biber ihre flachen braunen Teller. Wegen Bauarbeiten kann es heute nur eine Warteschlange für alle Biber geben. Die Küchenbiber müssen deshalb einen Tellerstapel vorbereiten, der zur Schlange passt: Sie müssen die grünen und braunen Teller so stapeln, dass jeder Biber in der Schlange den passenden Teller bekommt. Schau dir zum Beispiel diese Warteschlange an. Für diese Warteschlange müssen die Teller so gestapelt sein.

Daten, die mit Computerprogrammen verarbeitet werden sollen, müssen passend organisiert sein. Informatiker beschäftigen sich deshalb intensiv mit Datenstrukturen. Zwei einfache Datenstrukturen sind „Schlange“ (Queue) und „Stapel“ (Stack). Bei einer „Schlange“ kann man nur auf die zuerst eingereichten Daten zugreifen (nach dem Prinzip FIFO: „first in, first out“). Bei einem „Stapel“ kann man nur auf die zuletzt eingereichten Daten zugreifen (nach dem Prinzip LIFO: „last in, first out“). Die Datenstruktur der wartenden Biber ist eine „Schlange“. Die Datenstruktur der Teller ist ein „Stapel“.

66115 / 2020 / Herbst / Thema 2 / Aufgabe 4

Der Güterzug der Biberbahn wurde in der Wagenreihung D-E-B-C-A abgestellt: Die Lok kann vorwärts und rückwärts fahren und dabei beliebig viele Waggon ziehen und schieben. Jedes Mal, wenn ein Waggon angekoppelt oder ein Waggon abgekoppelt wird, zählt das als eine Rangieroperation. Wie viele Rangieroperationen sind mindestens nötig, um die Wagenreihung A-B-C-D-E herzustellen?

Die Anzahl 8 ist richtig: Um einen Zug mit nur zwei Waggon umzuordnen, muss jeder der beiden Waggon einmal an- und einmal abgekoppelt werden, das sind vier Operationen. Bei dieser Aufgabe kann man die bereits geordneten Zugteile D-E und B-C als einzelne Waggon behandeln. Die ersten beiden umzuordnen, etwa D-E und B-C, erfordert also vier Operationen. Den so gewonnenen Zugteil B-C-D-E und den verbleibenden Waggon A umzuordnen erfordert weitere vier Operationen. Die Reihenfolge der Schritte mag variieren, aber nur mit mehr Gleisen könnten Operationen eingespart werden.

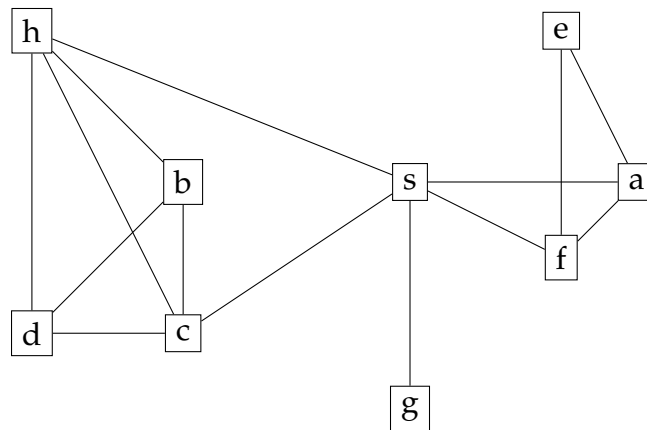
Die zwei Abstellgleise können als Stapelspeicher (stacks) angesehen werden. Man kann Objekte hineintun und wieder herausholen – aber nicht in beliebiger Reihenfolge. Was zuletzt hineinkam (push), muss zuerst wieder heraus (pop). Stapelspeicher, manchmal auch Kellerspeicher genannt, werden von der Informatik in Programmen und Hardwareschaltungen für vielfältige Zwecke eingesetzt.

66115 / 2020 / Herbst / Thema 2 / Aufgabe 4

In einem Sackbahnhof mit drei Gleisen befinden sich in den Gleisen S1 und S2 zwei Züge jeweils mit Waggon für Zielbahnhof A und B. Gleis S3 ist leer. Stellen Sie die Züge zusammen, die nur Waggon für einen Zielbahnhof enthalten! Betrachten Sie S1, S2 und S3 als Stapel und entwerfen Sie einen Algorithmus (Pseudocode genügt), der die Züge so umordnet, dass anschließend alle Waggon für A in S1 und alle Waggon für B in S2 stehen.

46115 / 2014 / Herbst / Thema 1 / Aufgabe 8**Aufgabe 8**

- (a) Führen Sie auf dem folgenden ungerichteten Graphen G eine Tiefensuche ab dem Knoten s aus (graphische Umsetzung). Unbesuchte Nachbarn eines Knotens sollen dabei in *alphabetischer Reihenfolge* abgearbeitet werden. Die Tiefensuche soll auf Basis eines *Stacks* umgesetzt werden. Geben Sie die Reihenfolge der besuchten Knoten, also die *dfs-number* der Knoten, und den Inhalt des *Stacks* in jedem Schritt an.



Lösungsvorschlag

In der Musterlösung auf Seite 3 lautet das Ergebnis s, a, e, f, c, b, d, h, g. Ich glaube jedoch diese Lösung ist richtig:

fett: Knoten, der entnommen wird.

kursiv: Knoten, die zum Stapel hinzugefügt werden.

Reihenfolge	Stapel	besucht
1	s	s
2	<i>a, c, f, g, h</i>	h
3	<i>a, c, f, g, b, d</i>	d
4	<i>a, c, f, g, b</i>	b
5	<i>a, c, f, g</i>	g
6	<i>a, c, f</i>	f
7	<i>a, c, e</i>	e
8	<i>a, c</i>	c
9	a	a

- (b) Führen Sie nun eine Breitensuche auf dem gegebenen Graphen aus, diese soll mit einer Queue umgesetzt werden. Als Startknoten wird wieder s verwendet. Geben Sie auch hier die Reihenfolge der besuchten Knoten und den Inhalt der Queue bei jedem Schritt an.

Lösungsvorschlag

fett: Knoten, der entnommen wird.

kursiv: Knoten, die zur Warteschlange hinzugefügt werden.

Reihenfolge	Warteschlange	besucht
1	s	s
2	a, c, f, g, h	a
3	c, f, g, h, e	c
4	f, g, h, e, b, d	f
5	g, h, e, b, d	g
6	h, e, b, d	h
7	e, b, d	e
8	b, d	b
9	d	d

Stapel (Stack)

- (c) Geben Sie in Pseudocode den Ablauf von Tiefen- und Breitensuche an, wenn diese wie beschrieben mit einem Stack bzw. einer Queue implementiert werden.

46115 / 2019 / Herbst / Thema 1 / Aufgabe 6

Gegeben sei die Implementierung eines Stacks ganzer Zahlen mit folgender Schnittstelle:

```
import java.util.Stack;

/**
 * Um schnell einen lauffähigen Stack zu bekommen, verwenden wir den Stack aus
 * der Java Collection.
 */
public class IntStack {
    private Stack<Integer> stack = new Stack<Integer>();

    /**
     * Legt Element i auf den Stack.
     *
     * @param i Eine Zahl, die auf dem Stack gelegt werden soll.
     */
    public void push(int i) {
        stack.push(i);
    }

    /**
     * Gibt oberstes Element vom Stack.
     *
     * @return Das oberste Element auf dem Stapel.
     */
    public int pop() {
        return stack.pop();
    }
}
```

```
/**
 * Fragt ab, ob Stack leer ist.
 *
 * @return Wahr, wenn der Stapel leer ist.
 */
public boolean isEmpty() {
    return stack.empty();
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/IntStack.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/IntStack.java)

Betrachten Sie nun die Realisierung der folgenden Datenstruktur `Mystery`, die zwei Stacks benutzt.

```
public class Mystery {
    private IntStack a = new IntStack();
    private IntStack b = new IntStack();

    public void foo(int item) {
        a.push(item);
    }

    public int bar() {
        if (b.isEmpty()) {
            while (!a.isEmpty()) {
                b.push(a.pop());
            }
        }
        return b.pop();
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/Mystery.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/Mystery.java)

- (a) Skizzieren Sie nach jedem Methodenaufruf der im folgenden angegebenen Befehlssequenz den Zustand der beiden Stacks eines Objekts `m` der Klasse `Mystery`. Geben Sie zudem bei jedem Aufruf der Methode `bar` an, welchen Wert diese zurückliefert.

```
Mystery m = new Mystery();
m.foo(3);
m.foo(5);
m.foo(4);
m.bar();
m.foo(7);
m.bar();
m.foo(2);
m.bar();
m.bar();
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/Mystery.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/Mystery.java)

Code	Stack b	Stack a	Rückgabewert
<code>m.foo(3);</code>	{ 3 }	{ }	
<code>m.foo(5);</code>	{ 5, 3 }	{ }	
<code>m.foo(4);</code>	{ 4, 5, 3 }	{ }	
<code>m.bar();</code>	{ }	{ 5, 4 }	3
<code>m.foo(7);</code>	{ 7 }	{ 5, 4 }	
<code>m.bar();</code>	{ 7 }	{ 4 }	5
<code>m.foo(2);</code>	{ 2, 7 }	{ }	
<code>m.bar();</code>	{ 2, 7 }	{ }	4
<code>m.bar();</code>	{ }	{ 2 }	7

- (b) Sei n die Anzahl der in einem Objekt der Klasse `Mystery` gespeicherten Werte. Im folgenden wird gefragt, wieviele Aufrufe von Operationen der Klasse `IntStack` einzelne Aufrufe von Methoden der Klasse `Mystery` verursachen. Begründen Sie jeweils Ihre Antwort.

- (i) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `foo(x)` im besten Fall?

Lösungsvorschlag

Einen Aufruf, nämlich `a.push(i)`

- (ii) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `foo(x)` im schlechtesten Fall?

Lösungsvorschlag

Einen Aufruf, nämlich `a.push(i)`

- (iii) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `bar()` im besten Fall?

Lösungsvorschlag

Wenn der Stack `b` nicht leer ist, dann werden zwei Aufrufe benötigt, nämlich `b.isEmpty()` und `b.pop()`

- (iv) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `bar()` im schlechtesten Fall?

Lösungsvorschlag

Wenn der Stack `b` leer ist, dann liegen all n Objekte im Stack `a`. Die Objekte im Stack `a` werden in der `while`-Schleife nach `b` verschoben. Pro Objekt sind drei Aufrufe nötig, also $3 \cdot n$. `b.isEmpty()` (erste Zeile in der Methode) und `b.pop()` (letzte Zeile in der Methode) wird immer aufgerufen. Wenn alle Objekt von `a` nach `b` verschoben wurden, wird

zusätzlich noch einmal in der Bedingung der `while`-Schleife `a.isEmpty()` aufgerufen. Im schlechtesten Fall werden also $3 \cdot n + 3$ Operationen der Klasse `IntStack` aufgerufen.

- (c) Welche allgemeinen Eigenschaften werden durch die Methoden `foo` und `bar` realisiert? Unter welchem Namen ist diese Datenstruktur allgemein bekannt?

Lösungsvorschlag

`foo()` Legt das Objekt auf den Stack `a`. Das Objekt wird in die Warteschlange eingereiht. Die Methode müsste eigentlich `enqueue()` heißen.

`bar()` Verschiebt alle Objekte vom Stack `a` in umgekehrter Reihenfolge in den Stack `b`, aber nur dann, wenn Stack `b` leer ist. Entfernt dann den obersten Wert aus dem Stack `b` und gibt ihn zurück. Das zuerst eingereihte Objekt wird aus der Warteschlange entnommen. Die Methode müsste eigentlich `dequeue()` heißen.

Die Datenstruktur ist unter dem Namen Warteschlange oder Queue bekannt

46115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 2

Gegeben sei die folgende Java-Implementierung eines Stacks.

```
class Stack {
    private Item head;

    public Stack() {
        head = null;
    }

    public void push(int val) {
        if (head == null) {
            head = new Item(val, null);
        } else {
            head = new Item(val, head);
        }
    }

    public int pop() {
        // ...
    }

    public int size() {
        // ...
    }

    public int min() {
        // ...
    }
}

class Item {
    private int val;
```

```
private Item next;

public Item(int val, Item next) {
    this.val = val;
    this.next = next;
}
}
```

- (a) Implementieren Sie die Methode `pop` in einer objektorientierten Programmiersprache Ihrer Wahl, die das erste Item des Stacks entfernt und seinen Wert zurückgibt. Ist kein Wert im Stack enthalten, so soll dies mit einer `IndexOutOfBoundsException` oder Ähnlichem gemeldet werden.

Beschreiben Sie nun jeweils die notwendigen Änderungen an den bisherigen Implementierungen, die für die Realisierung der folgenden Methoden notwendig sind.

Lösungsvorschlag

```
public int pop() {
    if (head != null) {
        int val = head.val;
        size--;
        head = head.next;
        return val;
    } else {
        throw new IndexOutOfBoundsException("The stack is empty");
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2021/fruehjahr/Stack.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2021/fruehjahr/Stack.java)

- (b) `size` gibt in Laufzeit $\mathcal{O}(1)$ die Anzahl der enthaltenen Items zurück.

Lösungsvorschlag

```
public void push(int val) {
    if (head == null) {
        head = new Item(val, null);
    } else {
        head = new Item(val, head);
    }
    if (min > val) {
        min = val;
    }
    size++;
}

public int pop() {
    if (head != null) {
        int val = head.val;
        size--;
        head = head.next;
        return val;
    } else {
        throw new IndexOutOfBoundsException("The stack is empty");
    }
}
```

```

        throw new IndexOutOfBoundsException("The stack is empty");
    }
}

public int size() {
    return size;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2021/fruehjahr/Stack.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2021/fruehjahr/Stack.java)

- (c) `min` gibt (zu jedem Zeitpunkt) in Laufzeit $\mathcal{O}(1)$ den Wert des kleinsten Elements im Stack zurück.

Lösungsvorschlag

```

public void push(int val) {
    if (head == null) {
        head = new Item(val, null);
    } else {
        head = new Item(val, head);
    }
    if (min > val) {
        min = val;
    }
    size++;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2021/fruehjahr/Stack.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2021/fruehjahr/Stack.java)

```

public int min() {
    return min;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2021/fruehjahr/Stack.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2021/fruehjahr/Stack.java)

Sie dürfen jeweils alle anderen angegebenen Methoden der Klasse verwenden, auch wenn Sie diese nicht implementiert haben. Sie können anstelle von objekt-orientiertem Quellcode auch eine informelle Beschreibung Ihrer Änderungen angeben.

46116 / 2010 / Frühjahr / Thema 1 / Aufgabe 1

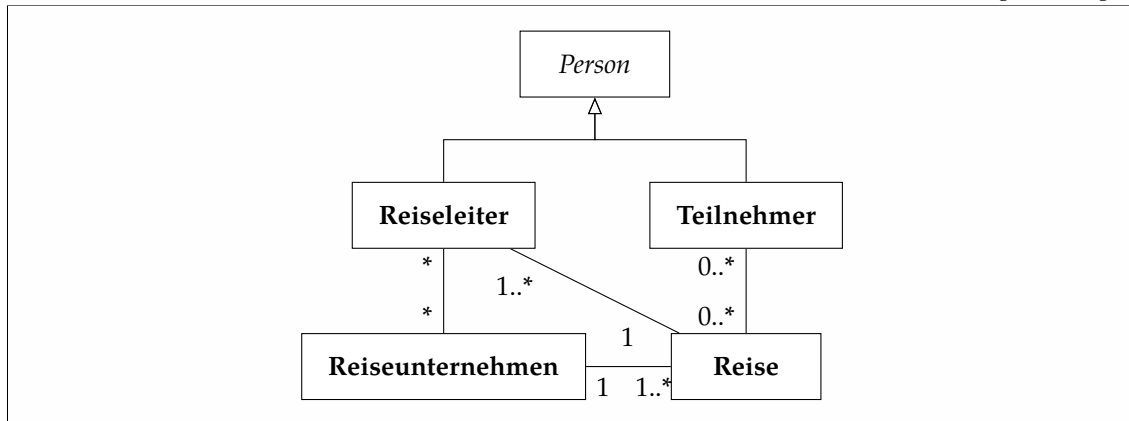
Es sei folgender Sachverhalt gegeben:

Ein Reiseunternehmen bietet verschiedene Reisen an. Dazu beschäftigt es eine Reihe von Reiseleitern, wobei eine Reise von mindestens einem Reiseleiter geleitet wird. Da Reiseleiter freiberuflich arbeiten, können sie bei mehreren Reiseunternehmen Reisen leiten.

An einer Reise können mehrere Teilnehmer teilnehmen, ein Teilnehmer kann auch an verschiedenen Reisen teilnehmen.

- (a) Modellieren Sie diesen Sachverhalt in einem UML-Klassendiagramm. Für Teilnehmer und Reiseleiter sollen Sie dabei eine abstrakte Oberklasse definieren. Achten Sie dabei auf die Multiplizitäten der Assoziationen. Sie müssen keine Attribute bzw. Methoden angeben.

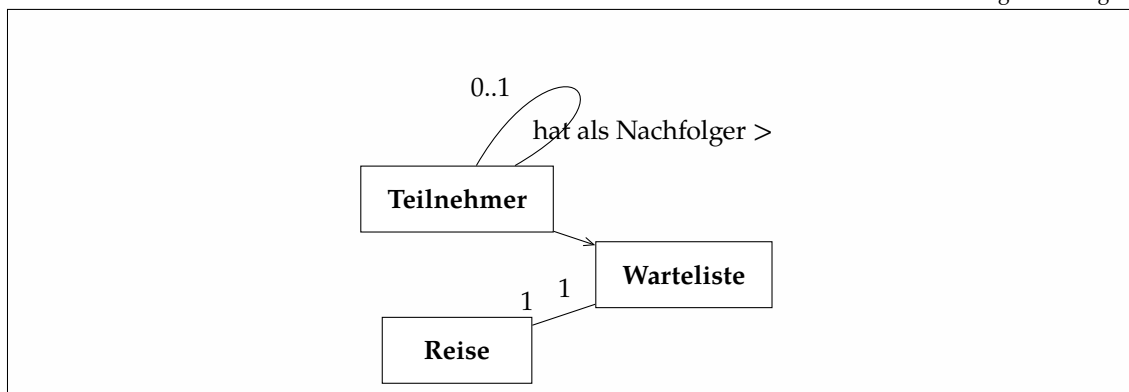
Lösungsvorschlag



- (b) Eine Reise kann jedoch nur mit einer begrenzten Kapazität angeboten werden, das heißt, zu einer bestimmten Reise kann nur eine begrenzte Anzahl von Teilnehmern assoziiert werden. Als Ausgleich soll pro Reise eine Warteliste verwaltet werden.

Modellieren Sie diesen erweiterten Sachverhalt in einem neuen Diagramm. Nicht veränderte Klassen brauchen nicht noch einmal angegeben werden. Beachten Sie dabei, dass die Reihenfolge bei einer Warteliste eine Rolle spielt.

Lösungsvorschlag



- (c) Implementieren Sie die in Aufgabenteil b) modellierten Klassen in Java. Fügen Sie eine Methode hinzu, die einen Teilnehmer von einer Reise entfernt. Dabei soll automatisch der erste Platz der Warteliste zu einem ReisetTeilnehmer werden, wenn die Warteliste nicht leer ist. Achten Sie auf die Navigierbarkeit Ihrer Assoziationen. Sie können davon ausgehen, dass die Methode nur mit Teilnehmern aufgerufen wird, die in der Tat Teilnehmer der Reise sind.

```

public class Teilnehmer {
    Teilnehmer nächster;
}
  
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46116/jahr_2010/fruehjahr/reiseunternehmen/Teilnehmer.java

```
/**
 * Diese Klasse ist eine Implementation einer einfach verketteten Liste. Sie
 * wird einerseits in der Klasse {@link Reise} genutzt, um die Reiseteilnehmer zu
 * speichern, andererseits um eine Warteliste darauf aufbauen zu können.
 */
public class TeilnehmerListe {

}
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_46116/jahr_2010/fruehjahr/reiseunternehmen/TeilnehmerListe.java

```
/**
 * Eine Reise kann jedoch nur mit einer begrenzten Kapazität angeboten
 * werden, das heißt, zu einer bestimmten Reise kann nur eine begrenzte
 * Anzahl von Teilnehmern assoziiert werden. Als Ausgleich soll pro
 * Reise eine Warteliste verwaltet werden.
 *
 * Modellieren Sie diesen erweiterten Sachverhalt in einem neuen
 * Diagramm. Nicht veränderte Klassen brauchen nicht noch einmal
 * angegeben werden. Beachten Sie dabei, dass die Reihenfolge bei einer
 * Warteliste eine Rolle spielt.
 *
 * Implementieren Sie die in Aufgabenteil b) modellierten Klassen in
 * Java. Fügen Sie eine Methode hinzu, die einen Teilnehmer von einer
 * Reise entfernt. Dabei soll automatisch der erste Platz der Warteliste
 * zu einem Reiseteilnehmer werden, wenn die Warteliste nicht leer ist.
 * Achten Sie auf die Navigierbarkeit Ihrer Assoziationen. Sie können
 * davon ausgehen, dass die Methode nur mit Teilnehmern aufgerufen wird,
 * die in der Tat Teilnehmer der Reise sind.
 */
public class Warteliste extends TeilnehmerListe {

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46116/jahr_2010/fruehjahr/reiseunternehmen/Warteliste.java

```
public class Reise {

    Teilnehmer teilnehmer;

    Warteliste warteliste;

    void entferneTeilnehmer(Teilnehmer teilnehmer) {

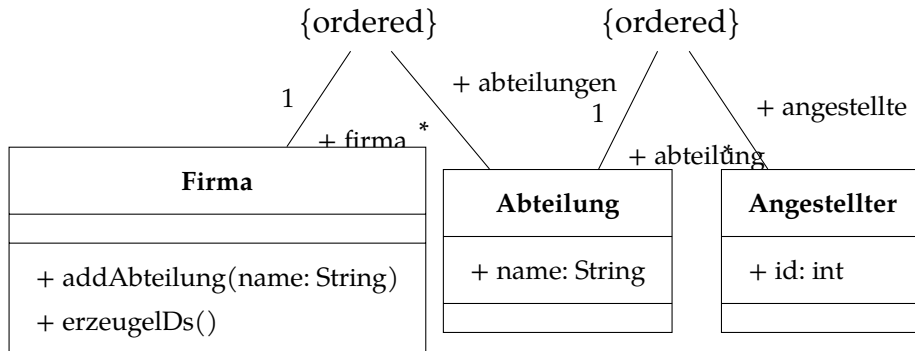
    }

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46116/jahr_2010/fruehjahr/reiseunternehmen/Reise.java

46116 / 2011 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 1

Firmenstruktur

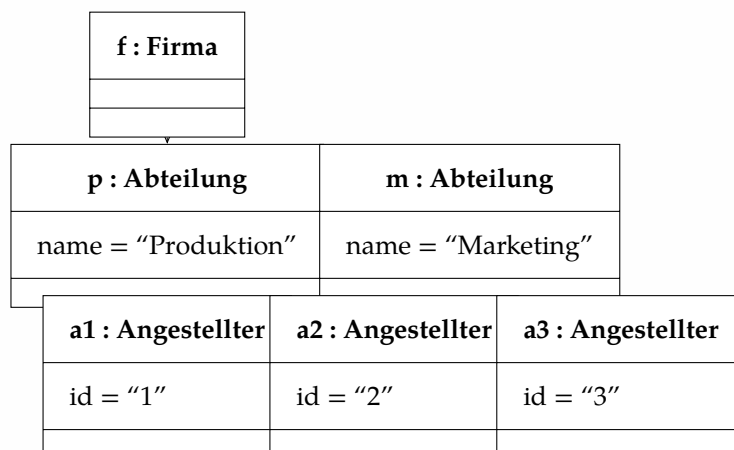


Eine Firma besteht aus **null** oder mehr Abteilungen, von denen jede **null** oder mehr Angestellte hat. Da sowohl die Abteilungen als auch deren Angestellten geordnet sind, sind Angestellte insgesamt geordnet. Sie haben durchgehende, ganzzahlige IDs, die bei 1 beginnen.

- (a) Erstellen Sie exemplarisch ein Objektdiagramm: Stellen Sie eine Firma mit dem Instanznamen **f** und den zwei Abteilungen „Produktion“ (Name **p**) und „Marketing“ (Name **m**) dar. Die Produktion hat zwei Angestellte, Marketing hat einen Angestellten. Die Angestellten haben die Namen **a1**, **a2**, und **a3**.

Lösungsvorschlag

Die Instanzbezeichnung müsste noch unterstrichen werden. Das geht aber leider mit TikZ-UML nicht.



- (b) Implementieren Sie das Klassendiagramm in Java oder in einer anderen geeigneten objektorientierten Programmiersprache Ihrer Wahl. Beachten Sie, dass die Assoziationen bidirektional und geordnet sind. Die beiden Methoden der Klasse **Firma** sollen dabei folgendes Verhalten haben:

Die Methode `erzeugeIDs` sorgt dafür, dass die IDs wieder korrekt zugewiesen sind. Die alten IDs können beliebig geändert werden, solange das Endergebnis wieder den obenstehenden Kriterien genügt.

Lösungsvorschlag

```
import java.util.ArrayList;
import java.util.List;

public class Firma {

    List<Abteilung> abteilungen;

    public Firma() {
        abteilungen = new ArrayList<Abteilung>();
    }

    public void addAbteilung(String name) {
        for (Abteilung abteilung : abteilungen) {
            if (abteilung.name.equals(name)) {

                ⇨ System.out.println("Eine Abteilung mit diesem Namen gibt es bereits schon.");
                return;
            }
            abteilungen.add(new Abteilung(name));
        }
    }

    public void erzeugeIDs() {
        int idZähler = 1;
        for (Abteilung abteilung : abteilungen) {
            for (Angestellter angestellter : abteilung.angestellte) {
                angestellter.id = idZähler;
                idZähler++;
            }
        }
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46116/jahr_2011/fruehjahr/Firma.java](https://github.com/bschlangaul/examen/examen_46116/jahr_2011/fruehjahr/Firma.java)

```
import java.util.ArrayList;
import java.util.List;

public class Abteilung {
    public String name;

    public List<Angestellter> angestellte;

    public Abteilung(String name) {
        this.name = name;
        angestellte = new ArrayList<Angestellter>();
    }
}
```


Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46116/jahr_2011/fruehjahr/Abteilung.java](#)

```
import java.util.List;

public class Angestellter {
    public int id;

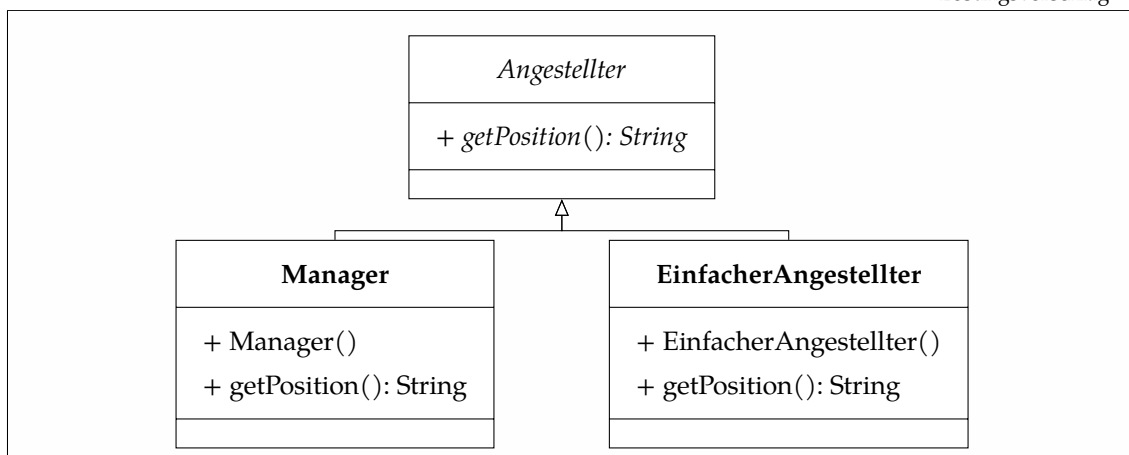
    public Firma firma;

    public List<Angestellter> angestellte;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46116/jahr_2011/fruehjahr/Angestellter.java](#)Implementierung in Java
Warteschlange (Queue)

- (c) Angestellte sollen in Manager und einfache Angestellte unterteilt werden. Zeichnen Sie ein Klassendiagramm mit der Oberklasse `Angestellter` und den zwei Unterklassen `Manager` und `EinfacherAngestellter`. Die Klasse `Angestellter` soll nicht instantiierbar sein und erzwingen, dass die Methode `getPosition()` (öffentlich, ohne Argumente, Rückgabewert `String`) von allen konkreten Unterklassen implementiert wird. `Manager` und `EinfacherAngestellter` sollen instantiierbar sein.

Lösungsvorschlag



- (d) Wie lautet der Fachbegriff dafür, dass eine Methode in einer Klasse und in deren Unterklassen dieselbe Signatur hat, aber in den Unterklassen unterschiedlich implementiert ist?

Lösungsvorschlag

Abstrakte Methode

66115 / 2007 / Frühjahr / Thema 1 / Aufgabe 7

Implementieren Sie die angegebenen Methoden einer Klasse `Queue` für Warteschlangen. Eine Warteschlange soll eine unbeschränkte Anzahl von Elementen aufnehmen können. Elemente sollen am Ende der Warteschlange angefügt und am Anfang aus ihr entfernt werden. Sie können davon ausgehen, dass eine Klasse `QueueElement` mit der folgenden Schnittstelle bereits implementiert ist.

```
class QueueElement {  
  
    private QueueElement next;  
    private Object contents;  
  
    QueueElement(Object contents) {  
        this.contents = contents;  
    }  
  
    Object getContents() {  
        return contents;  
    }  
  
    QueueElement getNext() {  
        return next;  
    }  
  
    void setNext(QueueElement next) {  
        this.next = next;  
    }  
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/QueueElement.java

Von der Klasse `Queue` ist folgendes gegeben:

```
class Queue {  
    QueueElement first;  
    QueueElement last;
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java

- (a) Schreiben Sie eine Methode `void append (Object contents)`, die ein neues Objekt in der Warteschlange einfügt.

Lösungsvorschlag

```
public void append(Object contents) {  
    QueueElement newElement = new QueueElement(contents);  
    if (first == null) {  
        first = newElement;  
        last = newElement;  
    } else {  
        // neues Element hinten anhängen  
        last.setNext(newElement);  
        // angehängtes Element ist Letztes  
        last = last.getNext();  
    }  
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java

- (b) Schreiben Sie eine Methode `Object remove()`, die ein Element aus der Warteschlange entfernt und dessen Inhalt zurückliefert. Berücksichtigen Sie, dass die Warteschlange leer sein könnte.

```
public Object remove() {
    Object tmp = null;
    if (first != null) {
        // Dein Inhalt des ersten Elements temporär speichern
        tmp = first.getContents();
        // Das erste Element aus der Schlange nehmen
        first = first.getNext();
    }
    // Den Inhalt des gelöschten Elements ausgeben bzw . null
    return tmp;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java)

- (c) Schreiben Sie eine Methode `boolean isEmpty()`, die überprüft, ob die Warteschlange leer ist.

```
public boolean isEmpty() {
    return (first == null);
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java)

Klasse Queue

```
class Queue {
    QueueElement first;
    QueueElement last;

    public void append(Object contents) {
        QueueElement newElement = new QueueElement(contents);
        if (first == null) {
            first = newElement;
            last = newElement;
        } else {
            // neues Element hinten anhängen
            last.setNext(newElement);
            // angehängtes Element ist Letztes
            last = last.getNext();
        }
    }

    public Object remove() {
        Object tmp = null;
        if (first != null) {
            // Dein Inhalt des ersten Elements temporär speichern
            tmp = first.getContents();
            // Das erste Element aus der Schlange nehmen
            first = first.getNext();
        }
    }
}
```

```
// Den Inhalt des gelöschten Elements ausgeben bzw . null
return tmp;
}

public boolean isEmpty() {
    return (first == null);
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java)

Tests

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class QueueTest {

    @Test
    public void methodAppend() {
        Queue queue = new Queue();
        assertEquals(true, queue.isEmpty());
        queue.append(1);
        assertEquals(false, queue.isEmpty());
    }

    @Test
    public void methodRemove() {
        Queue queue = new Queue();
        queue.append(1);
        queue.append(2);
        queue.append(3);

        assertEquals(1, queue.remove());
        assertEquals(2, queue.remove());
        assertEquals(3, queue.remove());
        assertEquals(null, queue.remove());
    }

    @Test
    public void methodIsEmpty() {
        Queue queue = new Queue();
        assertEquals(true, queue.isEmpty());
    }
}
```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/QueueTest.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/QueueTest.java)

66115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 2

Gegeben sei die folgende Java-Implementierung einer doppelt-verketteten Liste.

```
class DoubleLinkedList {
    private Item head;

    public DoubleLinkedList() {
        head = null;
    }

    public Item append(Object val) {
        if (head == null) {
            head = new Item(val, null, null);
            head.prev = head;
            head.next = head;
        } else {
            Item item = new Item(val, head.prev, head);
            head.prev.next = item;
            head.prev = item;
        }
        return head.prev;
    }

    public Item search(Object val) {
        // ...
    }

    public void delete(Object val) {
        // ...
    }
}

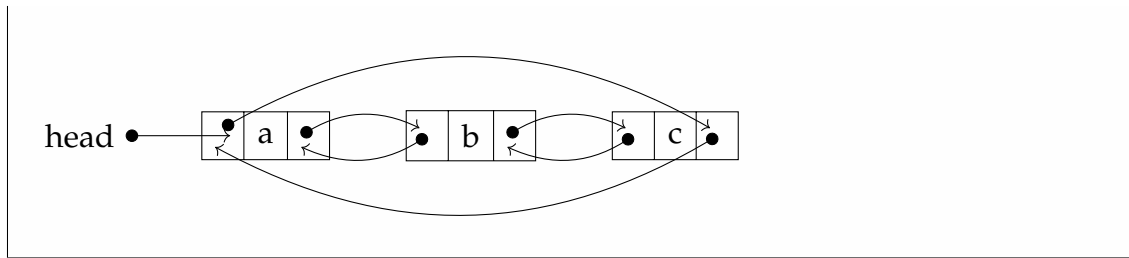
class Item {
    private Object val;
    private Item prev;
    private Item next;

    public Item(Object val, Item prev, Item next) {
        this.val = val;
        this.prev = prev;
        this.next = next;
    }
}
```

- (a) Skizzieren Sie den Zustand der Datenstruktur nach Aufruf der folgenden Befehlssequenz. Um Variablen mit Zeigern auf Objekte darzustellen, können Sie mit dem Variablennamen beschriftete Pfeile verwenden.

```
DoubleLinkedList list = new DoubleLinkedList();
list.append("a");
list.append("b");
list.append("c");
```

Lösungsvorschlag



- (b) Implementieren Sie in der Klasse `DoubleLinkedList` die Methode `search`, die zu einem gegebenen Wert das Item der Liste mit dem entsprechenden Wert, oder `null` falls der Wert nicht in der Liste enthalten ist, zurückgibt.

Lösungsvorschlag

```
public Item search(Object val) {
    Item item = null;
    if (head != null) {
        item = head;
        do {
            if (item.val.equals(val)) {
                return item;
            }
            item = item.next;
        } while (!item.equals(head));
    }
    return null;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/DoubleLinkedList.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/DoubleLinkedList.java)

- (c) Implementieren Sie in der Klasse `DoubleLinkedList` die Methode `delete`, die das erste Vorkommen eines Wertes aus der Liste entfernt. Ist der Wert nicht in der Liste enthalten, terminiert die Methode „stillschweigend“, ohne Änderung der Liste und ohne Fehlermeldung. Sie dürfen die Methode `search` aus Teilaufgabe b) verwenden, auch wenn Sie sie nicht implementiert haben.

Lösungsvorschlag

```
public void delete(Object val) {
    Item item = search(val);
    if (item != null) {
        if (head.next.equals(head)) {
            head = null;
        } else {
            if (item.equals(head)) {
                head = item.next;
            }
            item.prev.next = item.next;
            item.next.prev = item.prev;
        }
    }
}
```

- (d) Beschreiben Sie die notwendigen Änderungen an der Datenstruktur und an den bisherigen Implementierungen, um eine Methode `size`, die die Anzahl der enthaltenen Items zurück gibt, mit Laufzeit $\mathcal{O}(1)$ zu realisieren.

Lösungsvorschlag

In der Klasse wird ein Zähler eingefügt, der bei jedem Aufruf der Methode `append` um eins nach oben gezählt wird und bei jedem erfolgreichen Löschen in der `delete`-Methode um eins nach unten gezählt wird. Mit `return` kann der Zählerstand in $\mathcal{O}(1)$ ausgegeben werden. Dazu müsste ein Getter zum Ausgeben implementiert werden. Die Datenstruktur bleibt unverändert.

66116 / 2012 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 1

Ein Getränkelieferservice verwaltet die Bestellungen verschiedener Kunden. Die folgenden Teilaufgaben sind in einer objektorientierten Programmiersprache zu lösen (die verwendete Sprache ist vorab anzugeben.).

- (a) Implementieren Sie eine Klasse `Kasten` zur Beschreibung eines Getränkekastens mit den folgenden Eigenschaften. Entscheiden Sie dabei jeweils ob eine Realisierung als Objekt- oder Klassenfeld sinnvoll ist.
- Es existiert ein einheitliches Kastenpfand in Höhe von 1,50 Euro.
 - Für alle Flaschen in einem Kasten gelte ein einheitliches Flaschenpfand, das jedoch von Kasten zu Kasten verschieden sein kann.
 - Während das Flaschenpfand für alle Flaschen eines Kastens gleich ist, sind die Einzelpreise der Flaschen je nach Inhalt unterschiedlich. Die Einzelpreise (ohne Flaschenpfand) der im Kasten enthaltenen Flaschen sollen in einem 2-dimensionalen Array abgelegt werden.

Geben Sie für die Klasse `Kasten` einen geeigneten Konstruktor an. Ergänzen Sie in der Klasse `Kasten` eine Objektmethode zur Berechnung des Gesamtpreises des Getränkekastens inklusive Kasten- und Flaschenpfand.

- (b) Schreiben Sie eine Klasse `Bestellung`. Jeder Bestellung soll eine eindeutige Bestellnummer zugeordnet werden, die über den Konstruktoraufruf erstellt wird. Außerdem soll zu jeder Bestellung der Name des Kunden gespeichert werden, sowie eine einfach verkettete Liste der bestellten Getränkekästen. Die Klasse `Bestellung` soll weiterhin eine Methode beinhalten, die den Gesamtpreis der Bestellung ermittelt.
- (c) Schreiben Sie ein kleines Testprogramm, das eine Bestellung erstellt, die zwei Getränkekästen umfasst. Der erste Kasten soll ein 1×1 Getränkekasten mit einer Flasche zu 0,75 Euro sein, der zweite Kasten soll - wie in Abbildung 1 dargestellt - ein 3×3 Getränkekasten mit 3 Flaschen zu 0,7 Euro auf der Diagonalen und

3 weiteren Flaschen zu je 1 Euro sein. Das Flaschenpfand beider Kästen beträgt 0,15 Euro pro Flasche, das Kastenpfand 1,50 Euro. Anschließend soll der Preis der Bestellung berechnet und auf der Standardausgabe ausgegeben werden.

1,0	1,0	0,7
1,0	0,7	0
0,7	0	0


```
/**
 * „Implementieren Sie eine Klasse Kasten zur Beschreibung eines
 → Getränkekastens
 * mit den folgenden Eigenschaften. Entscheiden Sie dabei jeweils ob eine
 * Realisierung als Objekt- oder Klassenfeld sinnvoll ist.“
 */
public class Kasten {
    /**
     * Wir verwenden static, also ein sogenanntes Klassenfeld, da das
 → Kastenpfand
     * für alle Kästen gleich ist: „Es existiert ein einheitliches Kastenpfand
 → in
     * Höhe von 1,50 Euro.“
     */
    static double kastenPfad = 1.5;

    /**
     * Wir verwenden ein Objektfeld, d.h. ein nicht statisches Feld: „Für alle
     * Flaschen in einem Kasten gelte ein einheitliches Flaschenpfand, das
 → jedoch
     * von Kasten zu Kasten verschieden sein kann.“
     */
    double flaschenPfad;

    /**
     * „Während das Flaschenpfand für alle Flaschen eines Kastens gleich ist,
 → sind
     * die Einzelpreise der Flaschen je nach Inhalt unterschiedlich. Die
     * Einzelpreise (ohne Flaschenpfand) der im Kasten enthaltenen Flaschen
 → sollen
     * in einem 2-dimensionalen Array abgelegt werden.“
     */
    double[][] flaschen;

    /**
     * „Sowie eine einfach verkettete Liste der bestellten Getränkekästen.“
     */
    Kasten nächsterKasten = null;

    /**
     * „Geben Sie für die Klasse Kasten einen geeigneten Konstruktor an.“
     *
     * @param flaschen Die Belegung des Kastens mit Flaschen als
```

```

    *           zweidimensionales Feld der Flaschenpreise ohne
    *           Flaschenpfand.
    * @param flaschenPfad Die Höhe des Flaschenpfads, dass für alle Flaschen
    ↪ in
    *           diesem Kasten gleich ist.
    */
    public Kasten(double[][] flaschen, double flaschenPfad) {
        this.flaschen = flaschen;
        this.flaschenPfad = flaschenPfad;
    }

    /**
     * „Ergänzen Sie in der Klasse Kasten eine Objektmethode zur Berechnung des
     * Gesamtpreises des Getränkekastens inklusive Kasten- und Flaschenpfand.“
     *
     * @return Der Gesamtpreis des Getränkekastens inklusive Kasten- und
     *         Flaschenpfand.
     */
    double berechneGesamtPreis() {
        double gesamtPreis = kastenPfad;
        for (int i = 0; i < flaschen.length; i++) {
            double[] reihe = flaschen[i];
            for (int j = 0; j < reihe.length; j++) {
                double flaschenPreis = flaschen[i][j];
                // Nur im Kasten vorhandene Flaschen kosten auch Flaschenpfand.
                if (flaschenPreis > 0)
                    gesamtPreis += flaschenPfad + flaschen[i][j];
            }
        }
        return gesamtPreis;
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2012/herbst/getraenke/Kasten.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2012/herbst/getraenke/Kasten.java)

```

import java.text.DecimalFormat;

/**
 * „Schreiben Sie eine Klasse Bestellung“
 */
public class Bestellung {
    /**
     * „Jeder Bestellung soll eine eindeutige Bestellnummer zugeordnet werden,
     ↪ die
     * über den Konstruktoraufruf erstellt wird.“
     */
    int bestellNummer;

    /**
     * „Außerdem soll zu jeder Bestellung der Name des Kunden gespeichert
     ↪ werden.“
     */
    String kundenName;
}

```

```
/**
 * „sowie eine einfach verkettete Liste der bestellten Getränkekästen. "
 */
Kasten kästen = null;

/**
 * „Jeder Bestellung soll eine eindeutige Bestellnummer zugeordnet werden,
→ die
 * über den Konstruktoraufruf erstellt wird. Außerdem soll zu jeder
→ Bestellung
 * der Name des Kunden gespeichert werden."
 *
 * @param bestellNummer Die Nummer der Getränkebestellung.
 * @param kundenName Der Name des/der KundenIn.
 */
public Bestellung(int bestellNummer, String kundenName) {
    this.bestellNummer = bestellNummer;
    this.kundenName = kundenName;
}

/**
 * „Die Klasse Bestellung soll weiterhin eine Methode beinhalten, die den
 * Gesamtpreis der Bestellung ermittelt."
 *
 * @return Der Gesamtpreis der Getränkebestellung.
 */
double berechneGesamtPreis() {
    double gesamtPreis = 0;
    Kasten kasten = kästen;
    while (kasten != null) {
        gesamtPreis += kasten.berechneGesamtPreis();
        kasten = kasten.nächsterKasten;
    }
    return gesamtPreis;
}

/**
 * Nicht verlangt. Könnte auch in die Test-Methode geschrieben werden.
 *
 * @param flaschen Die Belegung des Kasten mit Flaschen als
 * zweidimensionales Feld der Flaschenpreise ohne
 * Flaschenpfad.
 * @param flaschenPfad Die Höhe des Flaschenpfads, dass für alle Flaschen
→ in
 * diesem Kasten gleich ist.
 */
void bestelleKasten(double[][] flaschen, double flaschenPfad) {
    Kasten bestellterKasten = new Kasten(flaschen, flaschenPfad);
    if (kästen == null) {
        kästen = bestellterKasten;
        return;
    }
    Kasten kasten = kästen;
```

```

Kasten letzterKasten = null;
while (kasten != null) {
    letzterKasten = kasten;
    kasten = kasten.nächsterKasten;
}
letzterKasten.nächsterKasten = bestellterKasten;
}

/**
 * Kleines Schmankerl. Nicht verlangt. Damit wir nicht 9.899999999999999
→ als
 * Aufgabe bekommen.
 *
 * @param preis Ein Preis als Gleitkommazahl.
 *
 * @return Der Preis als Text mit zwei Stellen nach dem Komma.
 */
static String runde(double preis) {
    DecimalFormat df = new DecimalFormat("#.##");
    df.setMinimumFractionDigits(2);
    return df.format(preis);
}

/**
 * Die main-Methode soll hier als Testmethode verwendet werden:
 *
 * „Schreiben Sie ein kleines Testprogramm, das eine Bestellung erstellt,
→ die
 * zwei Getränkekästen umfasst. Der erste Kasten soll ein 1 x 1
→ Getränkekasten
 * mit einer Flasche zu 0,75 Euro sein, der zweite Kasten soll - wie in
 * Abbildung 1 dargestellt - ein 3 x 3 Getränkekasten mit 3 Flaschen zu 0,7
→ Euro
 * auf der Diagonalen und 3 weiteren Flaschen zu je 1 Euro sein. Das
 * Flaschenpfand beider Kästen beträgt 0,15 Euro pro Flasche, das
→ Kastenpfand
 * 1,50 Euro. Anschließend soll der Preis der Bestellung berechnet und auf
→ der
 * Standardausgabe ausgegeben werden."
 *
 * @param args Kommandozeilenargumente, die uns nicht zu interessieren
→ brauchen.
 */
public static void main(String[] args) {
    Bestellung bestellung = new Bestellung(1, "Hermine Bschlangaul");

    // Müsste eigentlich nicht mehr gesetzt werden, da wir es schon in der
    // Klassendefinition gesetzt haben.
    Kasten.kastenPfad = 1.50;

    bestellung.bestelleKasten(new double[][] { { 0.75 } }, 0.15);
    bestellung.bestelleKasten(new double[][] { { 1.0, 1.0, 0.7 }, { 1.0, 0.7,
→ 0 }, { 0.7, 0, 0 } }, 0.15);

```

```

// Oder kürzer
// bestellung.bestelleKasten(new double[][] { { 1, 1, .7 }, { 1, .7 }, {
↪ .7 } }, .15);

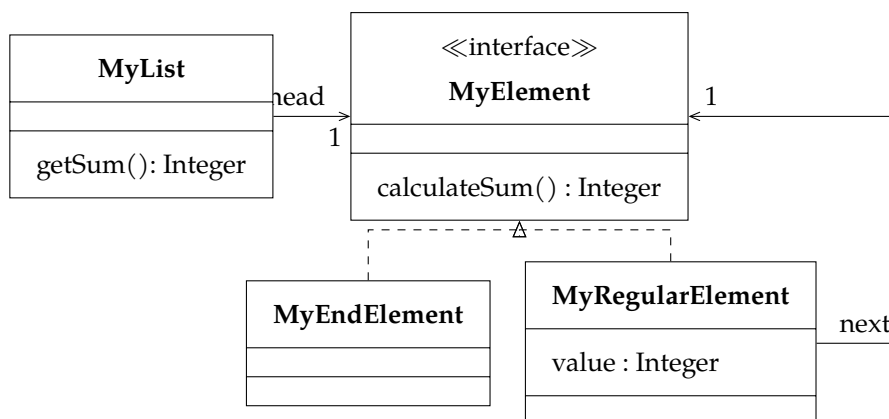
// Gegenrechnung:
// 1 x 0.75 = 0.75
// 3 x 1.00 = 3.00
// 3 x 0.70 = 2.10
// 7 x 0.15 = 1.05 (Flaschenpfad)
// 3 x 1.50 = 3.00 (Kastenpfand)
// ----
// 9.90
System.out.println("Der Gesamtpreis der Getränkebestellung beträgt: " +
↪ runde(bestellung.berechneGesamtPreis()) + " €");
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2012/herbst/getraenke/Bestellung.java](https://github.com/orgs/bschlangaul/examen/examen_66116/jahr_2012/herbst/getraenke/Bestellung.java)

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 5

Die folgende Abbildung stellt den Entwurf der Implementierung einer verketteten Liste dar, welche Integer-Werte als Elemente enthalten kann.



Die Klasse **MyList** stellt die Methode **getSum()** zur Verfügung, welche die Summe über alle in einer Liste befindlichen Elemente berechnet. Ein Ausschnitt der Implementierung sieht folgendermaßen aus:

```

public class MyList {
    private MyElement head;

    public MyList() {
        this.head = new MyEndElement();
    }

    public int getSum() {
        // ..
    }
}

```

Gehen Sie im Folgenden davon aus, dass bereits Methoden existieren, welche Elemente in die Liste einfügen können.

- (a) Implementieren Sie in einer objektorientierten Programmiersprache Ihrer Wahl, z. B. Java, die Methode `calculateSum()` der Klassen `MyEndElement` und `MyRegularElement`, so dass rekursiv die Summe der Elemente der Liste berechnet wird. Als Abbruchbedingung darf hierbei nicht das Feld `MyRegularElement.next` auf den Wert `null` überprüft werden.

Hinweis: Gehen Sie davon aus, die Implementierung von `MyList` garantiert, dass `MyRegularElement.next` niemals den Wert `null` annimmt, sondern das letzte hinzugefügte `MyRegularElement` auf eine Instanz der Klasse `MyEndElement` verweist. Es gibt immer nur eine Instanz der Klasse `MyEndElement` in einer Liste.

Hinweis: Achten Sie auf die Angabe einer korrekten Methodensignatur.

Lösungsvorschlag

```
int calculateSum() {  
    return value + next.calculateSum();  
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2021/fruehjahr/my_list/MyElement.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2021/fruehjahr/my_list/MyElement.java)

```
int calculateSum() {  
    return 0;  
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66116/jahr_2021/fruehjahr/my_list/MyEndElement.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2021/fruehjahr/my_list/MyEndElement.java)

- (b) Nennen Sie den Namen des Entwurfsmusters, auf welchem das oben gegebene Klassendiagramm basiert, und ordnen Sie dieses in eine der Kategorien von Entwurfsmustern ein.

Hinweis: Es genügt die Angabe eines Musters, falls Sie mehrere Muster identifizieren sollten.

Lösungsvorschlag

Kompositium (Strukturmuster)

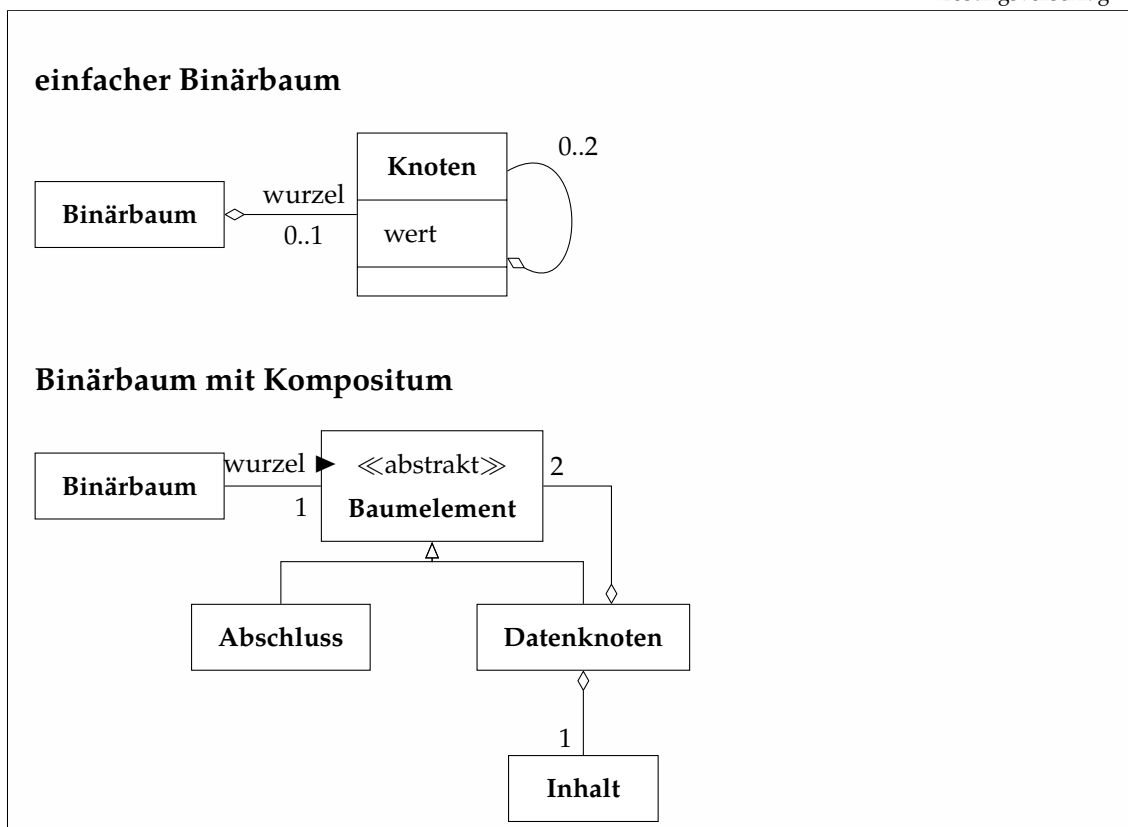
Bäume

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

Klassendiagramm und Implementierung

(a) Erstellen Sie ein Klassendiagramm für einen Binärbaum.

Lösungsvorschlag



(b) Entwerfen Sie eine mögliche Implementierung zur Erzeugung eines binären Baumes in Java.

Lösungsvorschlag

einfacher Binärbaum

```

public class Binaerbaum {
    public Knoten wurzel;

    public void fügeEin(int wert) {
        if (wurzel == null) {
            wurzel = new Knoten(wert);
        } else {
            if (wert <= wurzel.gibWert()) {

```

```
        if (wurzel.gibLinks() != null) {
            wurzel.gibLinks().fügeEin(wert);
        } else {
            wurzel.setzeLinks(new Knoten(wert));
        }
    } else {
        if (wurzel.gibRechts() != null) {
            wurzel.gibRechts().fügeEin(wert);
        } else {
            wurzel.setzeRechts(new Knoten(wert));
        }
    }
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/baum/einfach/Binaerbaum.java](https://github.com/bschlangaul/aufgaben/aud/baum/einfach/Binaerbaum.java)

```
public class Knoten {

    private Knoten links;
    private Knoten rechts;
    private int wert;

    public Knoten(int wert) {
        this.wert = wert;
    }

    public void setzeWert(int w) {
        wert = w;
    }

    public int gibWert() {
        return wert;
    }

    public void setzeLinks(Knoten l) {
        links = l;
    }

    public void setzeRechts(Knoten r) {
        rechts = r;
    }

    public Knoten gibLinks() {
        return links;
    }

    public Knoten gibRechts() {
        return rechts;
    }

    public void fügeEin(int wert) {
```



```
        if (wert <= this.gibWert()) {
            if (this.gibLinks() != null) {
                this.gibLinks().fügeEin(wert);
            } else {
                this.setzeLinks(new Knoten(wert));
            }
        } else {
            if (this.gibRechts() != null) {
                this.gibRechts().fügeEin(wert);
            } else {
                this.setzeRechts(new Knoten(wert));
            }
        }
    }
}
```

Code-Beispiel auf Github ansehen: <src/main/java/org/bschlangaul/aufgaben/aud/baum/einfach/Knoten.java>

Binärbaum mit Kompositum

```
class Binaerbaum {
    private Bauelement wurzel;

    public Binaerbaum() {
        wurzel = new Abschluss();
    }

    public void setzeWurzel(Bauelement wurzel) {
        this.wurzel = wurzel;
    }

    public Bauelement gibWurzel() {
        return wurzel;
    }

    public int gibAnzahl() {
        return wurzel.gibAnzahl();
    }

    public void gibAus() {
        wurzel.gibAus();
    }
}
```

Code-Beispiel auf Github ansehen: <src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Binaerbaum.java>

```
abstract class Bauelement {
    public abstract void setzeLinks(Bauelement nl);

    public abstract void setzeRechts(Bauelement nr);
}
```

```
public abstract Bauelement gibLinks();

public abstract Bauelement gibRechts();

public abstract Datenelement gibInhalt();

public abstract int gibAnzahl();

public abstract void gibAus();
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Bauelement.java](https://github.com/bschlangaul/aufgaben/aud/baum/kompositum/Bauelement.java)

```
class Abschluss extends Bauelement {

    public void setzteLinks(Bauelement links) {
        System.out.println("Ein Abschluss hat kein linkes Element!");
    }

    public void setzeRechts(Bauelement rechts) {
        System.out.println("Ein Abschluss hat kein rechts Element!");
    }

    public Bauelement gibLinks() {
        System.out.println("Linkes Element nicht bekannt!");
        return this;
    }

    public Bauelement gibRechts() {
        System.out.println("Linkes Element nicht bekannt!");
        return this;
    }

    public Datenelement gibInhalt() {
        return null;
    }

    public int gibAnzahl() {
        return 0;
    }

    public void gibAus() {
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Abschluss.java](https://github.com/bschlangaul/aufgaben/aud/baum/kompositum/Abschluss.java)

```
class Datenknoten extends Bauelement {
    private Bauelement links, rechts;
    private Datenelement inhalt;

    public Datenknoten(Bauelement links, Bauelement rechts, Datenelement
↵ inhalt) {
        this.links = links;
    }
}
```

```
        this.rechts = rechts;
        this.inhalt = inhalt;
    }

    public void setzteLinks(Bauelement links) {
        this.links = links;
    }

    public void setzeRechts(Bauelement rechts) {
        this.rechts = rechts;
    }

    public void inhaltSetzen(Datenelement inhalt) {
        this.inhalt = inhalt;
    }

    public Bauelement gibLinks() {
        return links;
    }

    public Bauelement gibRechts() {
        return rechts;
    }

    public Datenelement gibInhalt() {
        return inhalt;
    }

    public int gibAnzahl() {
        return 1 + links.gibAnzahl() + rechts.gibAnzahl();
    }

    public void gibAus() {
        System.out.print(" [");
        links.gibAus();
        inhalt.gibAus();
        rechts.gibAus();
        System.out.print("] ");
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Datenknoten.java](#)

```
abstract class Datenelement {
    public abstract void gibAus();
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Datenelement.java](#)

```
class Inhalt extends Datenelement {
    private String inhalt;

    public Inhalt(String inhalt) {
        this.inhalt = inhalt;
    }
}
```

```
}

public String gibInhalt() {
    return inhalt;
}

public void gibAus() {
    System.out.print(inhalt);
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Inhalt.java](https://github.com/bschlangaul/aufgaben/blob/main/java/org/bschlangaul/aud/baum/kompositum/Inhalt.java)

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class BinaerbaumTest {

    @Test
    public void teste(){
        Binaerbaum baum = new Binaerbaum();
        Inhalt[] inhalte = new Inhalt[16];
        Datenknoten[] datenknoten = new Datenknoten[16];
        inhalte[0] = new Inhalt("Inhalt 1");

        inhalte[1] = new Inhalt("Inhalt 2");
        inhalte[2] = new Inhalt("Inhalt 3");

        inhalte[3] = new Inhalt("Inhalt 4");
        inhalte[4] = new Inhalt("Inhalt 5");

        for (int i = 0; i < 5; i++) {
            datenknoten[i] = new Datenknoten(new Abschluss(), new Abschluss(),
                ↪ inhalte[i]);
        }
        baum.setzeWurzel(datenknoten[0]);

        datenknoten[0].setzteLinks(datenknoten[1]);
        datenknoten[0].setzeRechts(datenknoten[2]);

        datenknoten[1].setzteLinks(datenknoten[3]);
        datenknoten[1].setzeRechts(datenknoten[4]);

        assertEquals(5, baum.gibAnzahl());

        Inhalt inhalt = (Inhalt)
            ↪ baum.gibWurzel().gibLinks().gibLinks().gibInhalt();
        assertEquals("Inhalt 4", inhalt.gibInhalt());
    }
}
```

Code-Beispiel auf Github ansehen: [src/test/java/org/bschlangaul/aufgaben/aud/baum/kompositum/BinaerbaumTest.java](https://github.com/bschlangaul/aufgaben/blob/main/java/org/bschlangaul/aud/baum/kompositum/BinaerbaumTest.java)

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

Gegeben sei folgender AVL-Baum:

Dabei sind die Blätter der Übersichtlichkeit halber weggelassen worden und über jedem Knoten v ist folgender Wert angegeben:

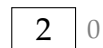
Höhe linker Teilbaum von v – Höhe rechter Teilbaum von v

- (a) Fügen Sie in diesen Baum den Schlüssel 1 ein.
- (b) Fügen Sie in diesen Baum den Schlüssel 11 ein.

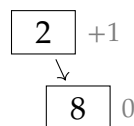
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

Fügen Sie die Zahlen 2, 8, 10, 1, 4, 5, 11 in der vorgegebenen Reihenfolge in einen AVL-Baum ein. Wie sieht der finale AVL-Baum aus?

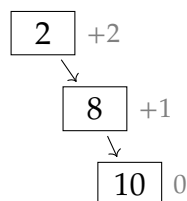
Nach dem Einfügen von „2“:



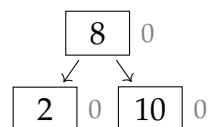
Nach dem Einfügen von „8“:



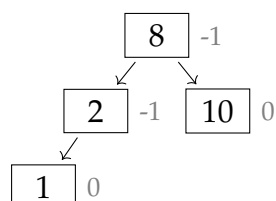
Nach dem Einfügen von „10“:



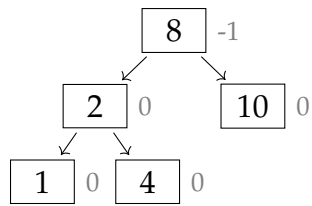
Nach der Linksrotation:



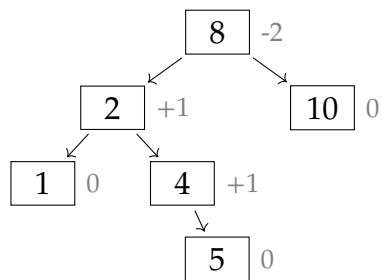
Nach dem Einfügen von „1“:



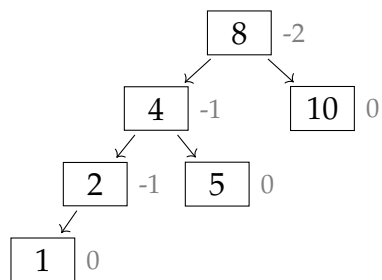
Nach dem Einfügen von „4“:



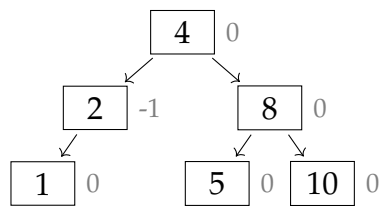
Nach dem Einfügen von „5“:



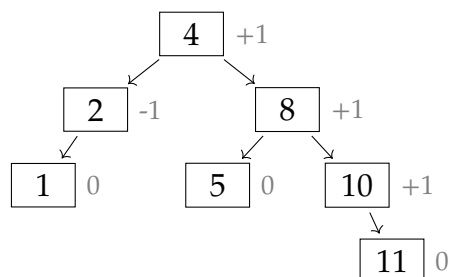
Nach der Linksrotation:



Nach der Rechtsrotation:

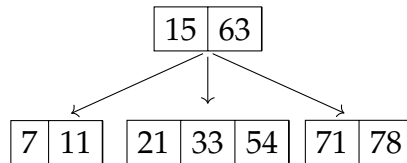


Nach dem Einfügen von „11“:



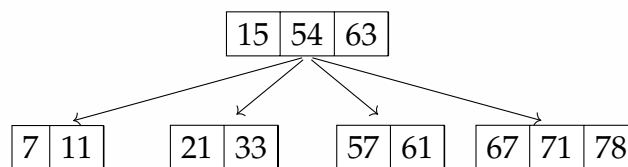
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

- (a) Gegeben ist der unten vereinfacht dargestellte B-Baum der Klasse der Ordnung 2. Fügen Sie die Schlüsselwerte 67, 57, 61, 75, 5, 13, 2, 91, 9, 17, 10 und 8 ein. Geben Sie in jedem Einfügeschritt die verwendete Maßnahme (einfaches Einfügen in einen Knoten, Splitten) an und zeichnen Sie den Baum nach jedem Knotensplit neu.

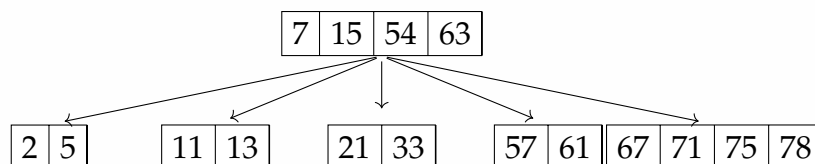


Lösungsvorschlag

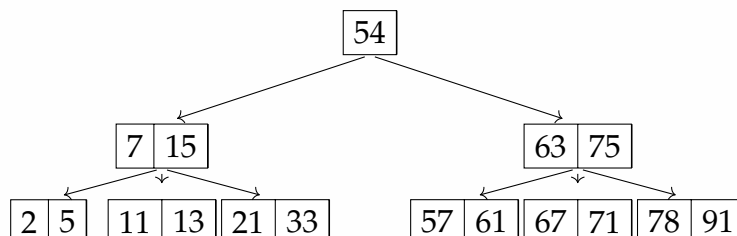
- Schlüsselwert 67 (einfaches Einfügen)
- Schlüsselwert 57 (einfaches Einfügen)
- Schlüsselwert 61 (Splitten)



- Schlüsselwert 75 (einfaches Einfügen)
- Schlüsselwert 5 (einfaches Einfügen)
- Schlüsselwert 13 (einfaches Einfügen)
- Schlüsselwert 2 (Splitten)

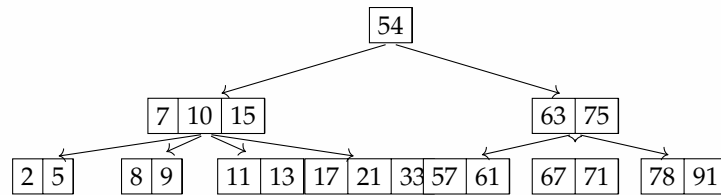


- Schlüsselwert 91 (Splitten)

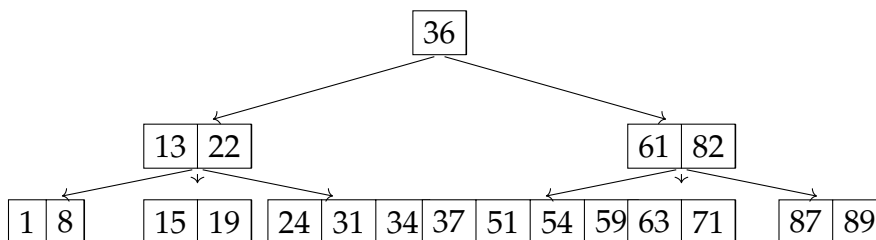


- Schlüsselwert 9 (einfaches Einfügen)
- Schlüsselwert 17 (einfaches Einfügen)
- Schlüsselwert 10 (einfaches Einfügen)

- Schlüsselwert 8 (Splitt)

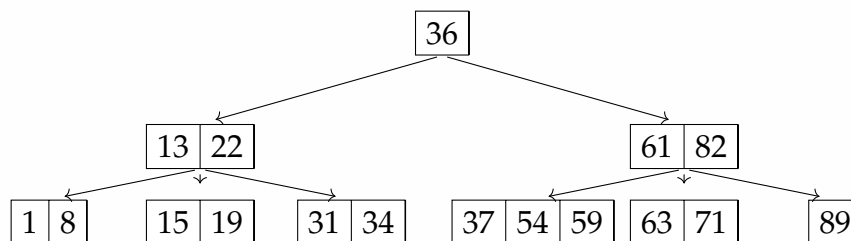


- (b) Gegeben ist der unten dargestellte B-Baum der Ordnung 2. Löschen Sie die Schlüsselwerte 24, 51, 87, 34, 71, 19, 31 und 8. Geben Sie in jedem Löschschritt die verwendete Maßnahme (einfaches Löschen, Mischen, Ausgleichen) an und zeichnen Sie den Baum nach jeder Veränderung der Knotenstruktur (Mischen, Ausgleichen) neu. Für Ausgleichsoperationen sollen nur unmittelbare Nachbarknoten herangezogen werden.



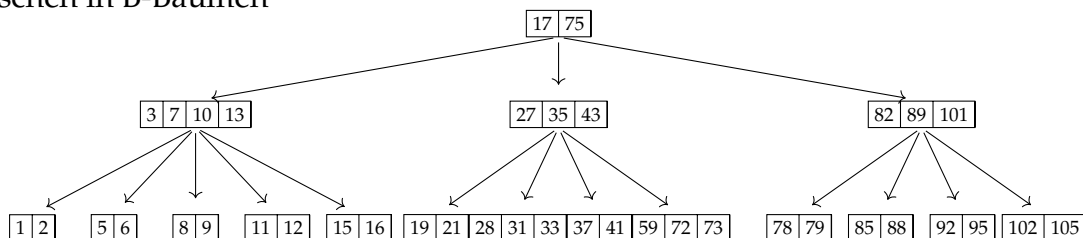
Lösungsvorschlag

- Schlüsselwert 24 (einfaches Löschen)
- Schlüsselwert 51 (einfaches Löschen)



66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

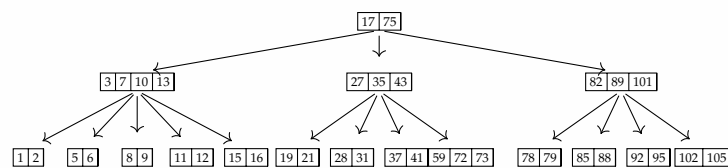
Löschen in B-Bäumen ¹



- (a) Löschen 33

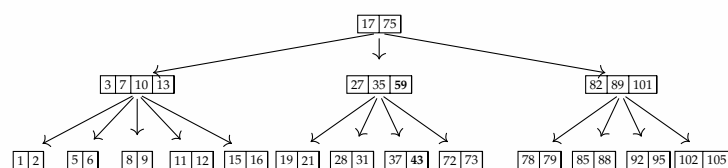
¹https://www.youtube.com/watch?v=in_JgH-XUyY

Löschen der 33 führt zu keinem Unterlauf.



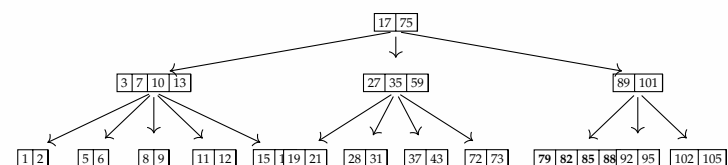
(b) Löschen 41

Ausgleichen Rotieren nach links



(c) Löschen 78

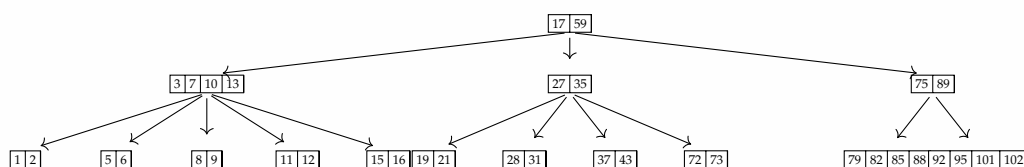
Mischen der Zahlen 79 82 85 88



(d) Löschen 105

Mischen der Zahlen 92 95 101 102

Rotieren 59 75



66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

(a) Ist $h(k) = k^2 \bmod 11$ eine gut gewählte Hashfunktion? Begründen Sie Ihre Antwort.

Tipp: Berechnen Sie zunächst $h(k)$ für $0 \leq k < 11$. Überlegen Sie dann, welche Werte $h(k')$ für $k' = a \cdot 11 + k$ mit $a > 0$ und $0 \leq k < 11$ annehmen kann.

Nein, h ist keine gute Hashfunktion. Betrachten wir zunächst die Wertetabelle von h für $0 \leq k < 11$. Wir erhalten

k	Nebenrechnung	$h(k)$
0	$0^2 \bmod 11 = 0 \bmod 11$	0
1	$1^2 \bmod 11 = 1 \bmod 11$	1
2	$2^2 \bmod 11 = 4 \bmod 11$	4
3	$3^2 \bmod 11 = 9 \bmod 11$	9
4	$4^2 \bmod 11 = 16 \bmod 11$	5
5	$5^2 \bmod 11 = 25 \bmod 11$	3
6	$6^2 \bmod 11 = 36 \bmod 11$	3
7	$7^2 \bmod 11 = 49 \bmod 11$	5
8	$8^2 \bmod 11 = 64 \bmod 11$	9
9	$9^2 \bmod 11 = 81 \bmod 11$	4
10	$10^2 \bmod 11 = 100 \bmod 11$	1

Wir sehen, dass nie die Werte 2, 6, 7, 8 und 10 eingenommen werden. Man könnte nun noch hoffen, dass das vielleicht für irgendein größeres k der Fall ist, dem ist jedoch nicht so. Wir können uns leicht davon überzeugen, dass für ein beliebiges $k' = a \cdot 11 + k$ mit $a > 0$ und $0 \leq k < 11$ folgendes gilt:

$$\begin{aligned}
 h(k') &= (k')^2 \bmod 11 \\
 &= (a \cdot 11 + k)^2 \bmod 11 \\
 &= (a^2 \cdot 11^2 + 2ak \cdot 11 + k^2) \bmod 11 \\
 &= (k^2) \bmod 11 \\
 &= h(k)
 \end{aligned}$$

Somit haben wir die Berechnung des Hashwertes für ein beliebiges k' auf die Berechnung des Hashwertes für ein $k < 11$ zurückgeführt, was impliziert, dass kein Schlüssel jemals auf etwas anderes als 0, 1, 3, 4, 5 oder 9 abgebildet werden kann.

- (b) Die Schlüssel 23, 57, 26, 6, 77, 43, 74, 60, 9, 91 sollen in dieser Reihenfolge mit der Hashfunktion $h(k) = k \bmod 17$ in eine Hashtabelle der Länge 17 eingefügt werden.

Exkurs: Sondieren

separate Verkettung Kollisionsauflösung durch Verkettung (separate chaining): Jedes Bucket speichert mit Hilfe einer dynamischen Datenstruktur (Liste, Baum, weitere Streutabelle, ...) alle Elemente mit dem entsprechenden Hashwert.

lineares Sondieren es wird um ein konstantes Intervall verschoben nach einer freien Stelle gesucht. Meistens wird die Intervallgröße auf 1 festgelegt.

quadratisches Sondieren Nach jedem erfolglosen Suchschritt wird das Intervall quadriert.

(i) Verwenden Sie separate Verkettung zur Kollisionsauflösung.

Nebenrechnung:

$$17 \cdot 1 = 17$$

$$17 \cdot 2 = 34$$

$$17 \cdot 3 = 51$$

$$17 \cdot 4 = 68$$

$$17 \cdot 5 = 85$$

Modulo-Berechnung der gegebenen Zahlen:

$$23 \bmod 17 = 6 \text{ da } 23 : 17 = 1, \text{ Rest } 6 \text{ da } 23 = 1 \cdot 17 + 6$$

$$57 \bmod 17 = 57 - 3 \cdot 17 = 57 - 51 = 6$$

$$26 \bmod 17 = 26 - 17 = 9$$

$$6 \bmod 17 = 6 - 0 \cdot 17 = 6$$

$$77 \bmod 17 = 77 - 4 \cdot 17 = 77 - 68 = 9$$

$$43 \bmod 17 = 9$$

$$74 \bmod 17 = 6$$

$$60 \bmod 17 = 9$$

$$9 \bmod 17 = 9$$

$$91 \bmod 17 = 6$$

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Schlüssel							23			26							
							57			77							
							6			43							
							74			60							
							91			9							

(ii) Verwenden Sie lineares Sondieren zur Kollisionsauflösung.

Die Hashfunktion lautet:

$$h'(k) = k \bmod 17$$

Die verwendete Hashfunktion beim linearen Sondieren:

$$h(k, i) = (h'(k) - i) \bmod 17$$

Mit Schrittweite -1 ergeben sich folgende Sondierungsfolgen:

Schlüssel	Index
23	6
57	6 5
26	9
6	6 5 4
77	9 8
43	9 8 7
74	6 5 4 3
60	9 8 7 6 5 4 3 2
9	9 8 7 6 5 4 3 2 1
91	6 5 4 3 2 1

Damit ergibt sich folgende Hashtabelle:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Schlüssel	91	9	60	74	6	57	23	43	77	26							

(iii) Verwenden Sie quadratisches Sondieren zur Kollisionsauflösung.

Die Hashfunktion lautet:

$$h'(k) = k \bmod 17$$

Die verwendete Hashfunktion beim quadratischen Sondieren:

$$h(k, i) = (h'(k) + i^2) \bmod 17$$

Am Beispiel von zwei Schlüsseln werden die Sondierungsfolgen berechnet:

$$h'(23) = 6$$

i. Sondierungsfolge:

$$h(23, 0) = (h'(23) + 0^2) \bmod 17 = (6 + 0) \bmod 17 = 6 \bmod 17 = 6$$

ii. Sondierungsfolge:

$$h(23, 1) = (h'(23) + 1^2) \bmod 17 = (6 + 1) \bmod 17 = 7 \bmod 17 = 7$$

iii. Sondierungsfolge:

$$h(23, 2) = (h'(23) + 2^2) \bmod 17 = (6 + 4) \bmod 17 = 10 \bmod 17 = 10$$

iv. Sondierungsfolge:

$$h(23,3) = (h'(23) + 3^2) \bmod 17 = (6 + 9) \bmod 17 = 15 \bmod 17 = \mathbf{15}$$

v. Sondierungsfolge:

$$h(23,4) = (h'(23) + 4^2) \bmod 17 = (6 + 16) \bmod 17 = 22 \bmod 17 = \mathbf{5}$$

$$h'(26) = 9$$

i. Sondierungsfolge:

$$h(26,0) = (h'(26) + 0^2) \bmod 17 = (9 + 0) \bmod 17 = 9 \bmod 17 = \mathbf{9}$$

ii. Sondierungsfolge:

$$h(26,1) = (h'(26) + 1^2) \bmod 17 = (9 + 1) \bmod 17 = 10 \bmod 17 = \mathbf{10}$$

iii. Sondierungsfolge:

$$h(26,2) = (h'(26) + 2^2) \bmod 17 = (9 + 4) \bmod 17 = 13 \bmod 17 = \mathbf{13}$$

iv. Sondierungsfolge:

$$h(26,3) = (h'(26) + 3^2) \bmod 17 = (9 + 9) \bmod 17 = 18 \bmod 17 = \mathbf{1}$$

v. Sondierungsfolge:

$$h(26,4) = (h'(26) + 4^2) \bmod 17 = (9 + 16) \bmod 17 = 25 \bmod 17 = \mathbf{8}$$

vi. Sondierungsfolge:

$$h(26,5) = (h'(26) + 5^2) \bmod 17 = (9 + 25) \bmod 17 = 34 \bmod 17 = \mathbf{0}$$

Es ergeben sich folgende Sondierungsfolgen:

Schlüssel	Index
23	6
57	6 7
26	9
6	6 7 10
77	9 10 13
43	9 10 13 1
74	6 7 10 15
60	9 10 13 1 8
9	9 10 13 1 8 0
91	6 7 10 15 5

Damit ergibt sich folgende Hashtabelle:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Schlüssel	9	43				91	23	57	60	26	6			77		74	

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

2

²nach Foliensatz der RWTH Aachen, Seite 19 <https://moves.rwth-aachen.de/wp-content/uploads/SS15/dsal/lec13.pdf>

$$\begin{aligned}
 h'(k) &= k \bmod 11 \\
 h(k, i) &= (h'(k) + i + 3i^2) \bmod 11 \\
 h'(17) &= 17 \bmod 11 = 6
 \end{aligned}$$

Sondierungsfolgen

$$\begin{aligned}
 h(17, 0) &= (17 + 0 + 3 \cdot 0^2) \bmod 11 = 6 \\
 h(17, 1) &= (17 + 1 + 3 \cdot 1^2) \bmod 11 = 21 \bmod 11 = 10 \\
 h(17, 2) &= (17 + 2 + 3 \cdot 2^2) \bmod 11 = 31 \bmod 11 = 31 - 2 \cdot 11 = 9 \\
 h(17, 3) &= (17 + 3 + 3 \cdot 3^2) \bmod 11 = 47 \bmod 11 = 47 - 4 \cdot 11 = 3
 \end{aligned}$$

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

3

Formel

$$\begin{aligned}
 h(k, i) &:= h'(k) + (-1)^{i+1} \cdot \left\lfloor \frac{i+1}{2} \right\rfloor^2 \bmod m \\
 k, k + 1^2, k - 1^2, k + 2^2, k - 2^2, \dots, k + \left(\frac{m-1}{2}\right)^2, k - \left(\frac{m-1}{2}\right)^2 &\bmod m
 \end{aligned}$$

Werte

$m = 19$, d. h. das Feld (die Tabelle) hat die Index-Nummern 0 bis 18. $k = h(x) = 7$

Sondierungsfolgen

i	Rechnung	Ergebnis	Index in der Tabelle
0	$7 + 0^2$	7	7
1	$7 + 1^2$	8	8
1	$7 - 1^2$	6	6
2	$7 + 2^2$	11	11
2	$7 - 2^2$	3	2
3	$7 + 3^2 = 7 + 9$	16	16
3	$7 - 3^2 = 7 - 9$	-2	17 <small>(19 - 2 = 10) oder (0 → 0, -1 → 18, -2 → 17)</small>
4	$7 + 4^2 = 7 + 16$	23	4 <small>(23 - 19 = 4) oder (19 → 0, 20 → 1, 21 → 2, 22 → 3, 23 → 4)</small>
4	$7 - 4^2 = 7 - 16$	-9	10 <small>(19 - 9 = 10) oder (0 → 0, -1 → 18, -2 → 17, ..., -9 → 10)</small>
5	$7 + 5^2 = 7 + 25$	32	13 <small>(32 - 19 = 13)</small>
5	$7 - 5^2 = 7 - 25$	-18	1 <small>(19 - 18 = 1)</small>

³nach Foliensatz der TU Braunschweig Seite 25 <https://www.ibr.cs.tu-bs.de/courses/ws0708/aud/skript/hash.np.pdf>

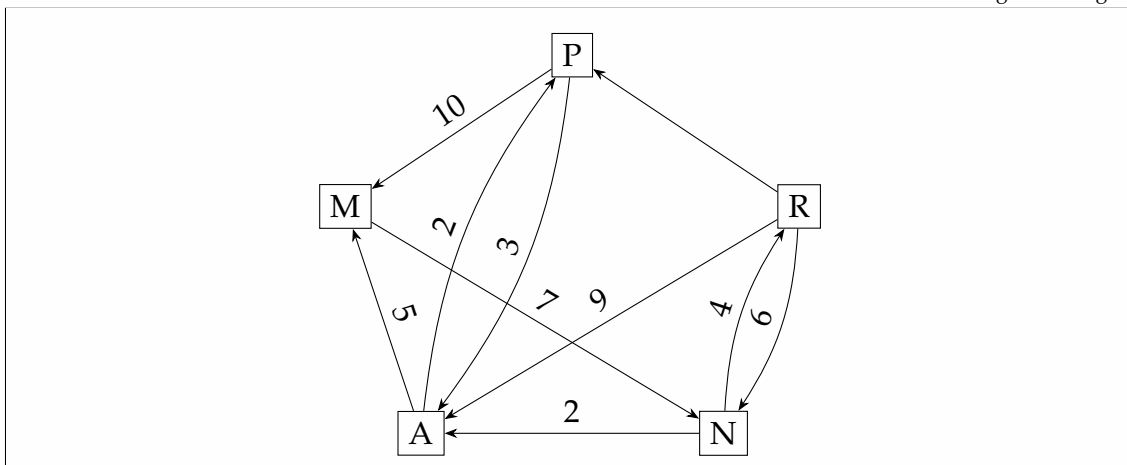
66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 5

Ein gerichteter Distanzgraph sei durch seine Adjazenzmatrix gegeben (in einer Zeile stehen die Längen der von dem Zeilenkopf ausgehenden Wege.)

$$\begin{array}{c}
 \begin{array}{ccccc}
 & A & M & N & P & R \\
 A & \left(\begin{array}{ccccc}
 * & 5 & - & 2 & - \\
 - & * & 7 & - & - \\
 2 & - & * & - & 4 \\
 3 & 10 & - & * & - \\
 9 & - & 6 & 1 & *
 \end{array} \right) \\
 M \\
 N \\
 P \\
 R
 \end{array}
 \end{array}$$

- (a) Stellen Sie den Graph in der üblichen Form dar.

Lösungsvorschlag



- (b) Bestimmen Sie mit dem Algorithmus von Dijkstra ausgehend von *M* die kürzeste Wege zu allen anderen Knoten.

Lösungsvorschlag

Nr.	besucht	A	M	N	P	R
0		∞	0	∞	∞	∞
1	M	∞	0	7	∞	∞
2	N	9		7	∞	11
3	A	9			11	11
4	P				11	11
5	R					11

nach	Entfernung	Reihenfolge	Pfad
$M \rightarrow A$	9	3	$M \rightarrow N \rightarrow A$
$M \rightarrow M$	0	1	
$M \rightarrow N$	7	2	$M \rightarrow N$
$M \rightarrow P$	11	4	$M \rightarrow N \rightarrow A \rightarrow P$
$M \rightarrow R$	11	5	$M \rightarrow N \rightarrow R$

- (c) Beschreiben Sie wie ein Heap als Prioritätswarteschlange in diesem Algorithmus verwendet werden kann.

Lösungsvorschlag

Ein Heap kann in diesem Algorithmus dazu verwendet werden, den nächsten Knoten mit der kürzesten Distanz zum Startknoten auszuwählen.

- (d) Geben Sie die Operation „Entfernen des Minimums“ für einen Heap an. Dazu gehört selbstverständlich die Restrukturierung des Heaps.

Lösungsvorschlag

```

static int NO_VALUE = Integer.MIN_VALUE;

static void heapify(int a[], int i, int last) {
    int smallest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l <= last && a[l] != NO_VALUE && a[l] < a[smallest]) {
        smallest = l;
    }

    if (r <= last && a[r] != NO_VALUE && a[r] < a[smallest]) {
        smallest = r;
    }

    if (smallest != i) {
        int swap = a[i];
        a[i] = a[smallest];
        a[smallest] = swap;
        heapify(a, smallest, last);
    }
}

static int removeMin(int a[]) {
    int result = a[0];
    int last = a.length - 1;
    while (last > 0 && a[last] == NO_VALUE) {
        last--;
    }
    a[0] = a[last];
    a[last] = NO_VALUE;
    heapify(a, 0, last);
}

```



```
    return result;  
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/baum/Heap.java](https://github.com/bschlangaul/aufgaben/blob/main/aud/baum/Heap.java)

Binärbaum
AVL-Baum

46115 / 2010 / Frühjahr / Thema 1 / Aufgabe 5

5. Datenstrukturen und Algorithmen: Binäre Suchbäume und AVL-Bäume

- (a) Geben Sie jeweils eine Definition für binäre Suchbäume und AVL-Bäume an.

Lösungsvorschlag

Binärer Suchbaum Für jeden Knoten gilt: Ein Knoten enthält einen Schlüsselwert. Ein Knoten hat zwei Kindknoten: einen linken und einen rechten Kindknoten. Alle Schlüsselwerte des linken Teilbaums sind kleiner, alle Schlüsselwerte des rechten Teilbaums sind größer als der Wert des Knotens.

AVL-Baum Alle Eigenschaften des oben definierten binären Suchbaums gelten auch im AVL-Baum. Zusätzlich gilt: Die Differenz der Höhen des rechten und linken Teilbaums darf sich nie mehr als um eins unterscheiden.

- (b) Zeichnen Sie einen AVL-Baum, der die folgenden Schlüssel enthält: 11, 1, 5, 37, 17, 29, 31, 3.

Nach dem Einfügen von „11“:

11 0

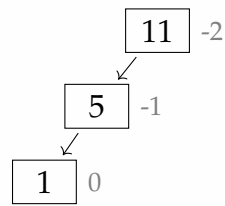
Nach dem Einfügen von „1“:

11 -1
↓
1 0

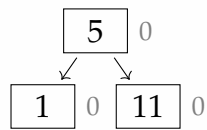
Nach dem Einfügen von „5“:

11 -2
↓
1 +1
↓
5 0

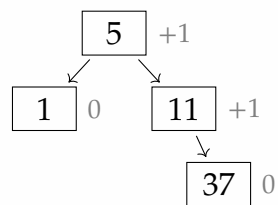
Nach der Linksrotation:



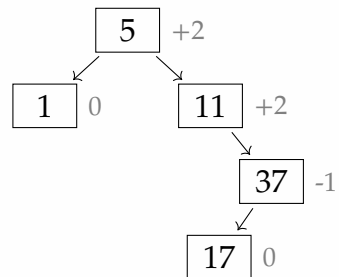
Nach der Rechtsrotation:



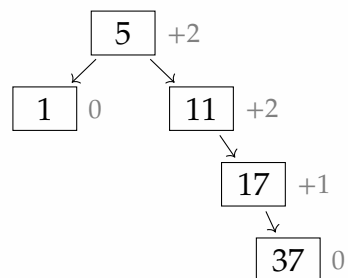
Nach dem Einfügen von „37“:



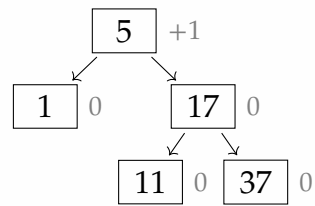
Nach dem Einfügen von „17“:



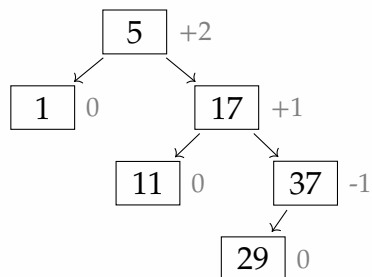
Nach der Rechtsrotation:



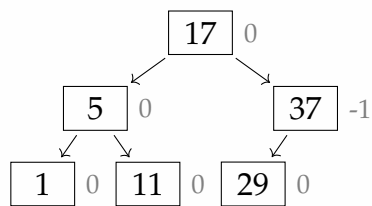
Nach der Linksrotation:



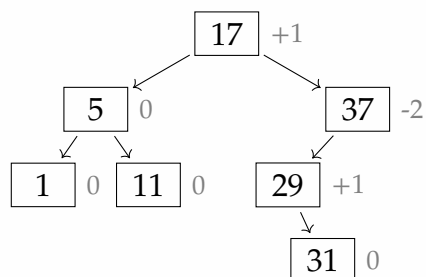
Nach dem Einfügen von „29“:



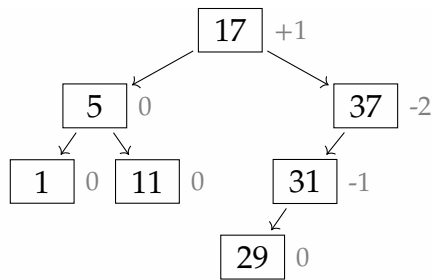
Nach der Linksrotation:



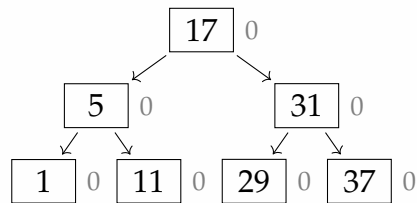
Nach dem Einfügen von „31“:



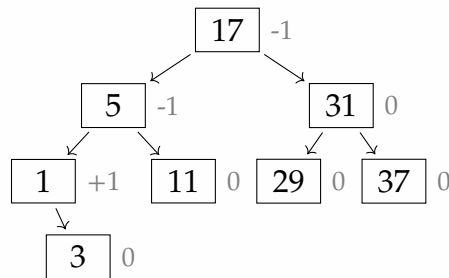
Nach der Linksrotation:



Nach der Rechtsrotation:



Nach dem Einfügen von „3“:



- (c) Welche Zeitkomplexität haben die Operationen *Einfügen*, *Löschen* und *Suchen* auf AVL-Bäumen? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Für alle Operationen wird jeweils nur ein Pfad von der Wurzel bis zum gewünschten Knoten beschriftet. Für alle Operation ist deshalb die maximale Höhe des Baums entscheidend. Da AVL-Bäume höhenbalanciert sind.

- (d) Implementieren Sie die Datenstruktur AVL-Baum mit Schlüsseln vom Typ `int` (für Java oder entsprechende Datentypen für die anderen Programmiersprachen) in entweder Java, C++, Smalltalk oder Eiffel. Ihre Implementierung muss als einzige Operation die Methode `suche` bereitstellen, die einen Schlüssel als Parameter bekommt und

- `true` zurückliefert, wenn der gesuchte Schlüssel im Baum enthalten ist,
- ansonsten `false`.

Ihre Implementierung muss keine Konstruktoren oder andere Methoden zur Initialisierung bereitstellen. Desweiteren soll die Implementierung nur Schlüsselknoten berücksichtigen.

B-Baum
AVL-Baum
B-Baum

Kommentieren Sie Ihre Implementierung ausführlich. Geben Sie an, welche Programmiersprache Sie gewählt haben.

```
public class AVLBaum {  
  
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2010/fruehjahr/AVLBaum.java](https://github.com/org/bschlangaul/examen/examen_46115/jahr_2010/fruehjahr/AVLBaum.java)

46115 / 2011 / Frühjahr / Thema 1 / Aufgabe 3

Aufgabe 2-3-4-B-Baum und AVL-Baum

- (a) Fügen Sie in einen anfangs leeren 2-3-4-Baum (B-Baum der Ordnung 4)⁴ der Reihe nach die folgenden Schlüssel ein:

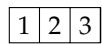
1, 2, 3, 5, 7, 8, 9, 4, 11, 12, 13, 6.

Dokumentieren Sie die Zwischenschritte so, dass die Entstehung des Baumes und nicht nur das Endergebnis nachvollziehbar ist.

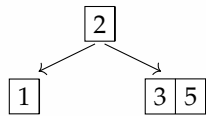
Lösungsvorschlag

⁴ein Baum, für den folgendes gilt: Er besitzt in einem Knoten max. 3 Schlüssel-Einträge und 4 Kindknoten und minimal einen Schlüssel und 2 Nachfolger

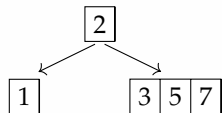
(i) 1, 2, 3 (Einfaches Einfügen):



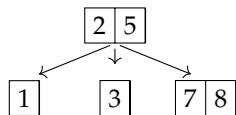
(ii) 5 (Split):



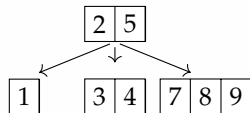
(iii) 7 (Einfaches Einfügen):



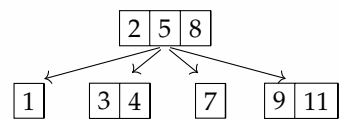
(iv) 8 (Split):



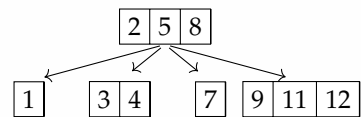
(v) 9, 4 (Einfaches Einfügen):



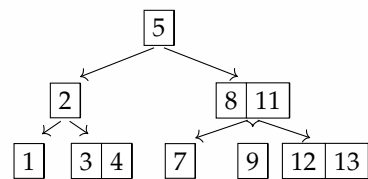
(vi) 11 (Split):



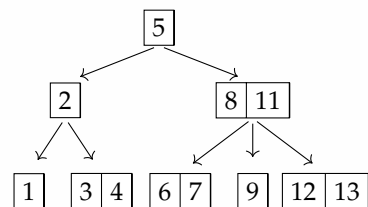
(vii) 12 (Einfaches Einfügen):



(viii) 13 (zwei Splits):

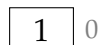


(ix) 6 (Einfaches Einfügen):

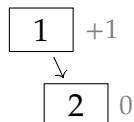


(b) Zeichnen Sie einen Rot-Schwarz-Baum oder einen AVL-Baum, der dieselben Einträge enthält.

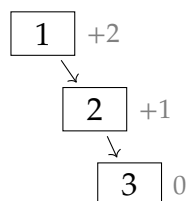
Nach dem Einfügen von „1“:



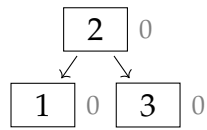
Nach dem Einfügen von „2“:



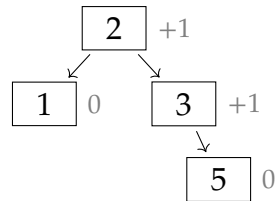
Nach dem Einfügen von „3“:



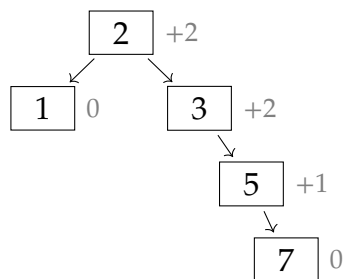
Nach der Linksrotation:



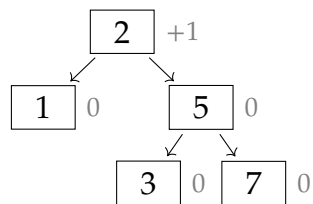
Nach dem Einfügen von „5“:



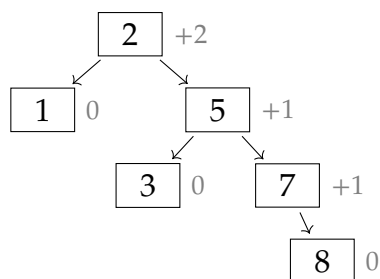
Nach dem Einfügen von „7“:



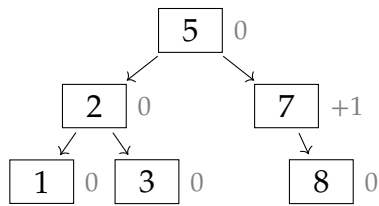
Nach der Linksrotation:



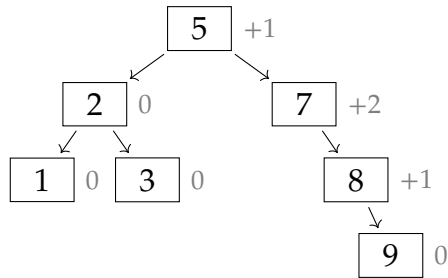
Nach dem Einfügen von „8“:



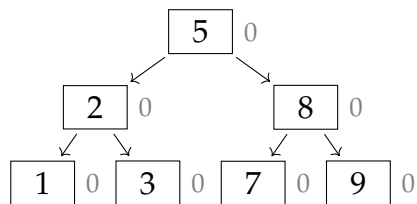
Nach der Linksrotation:



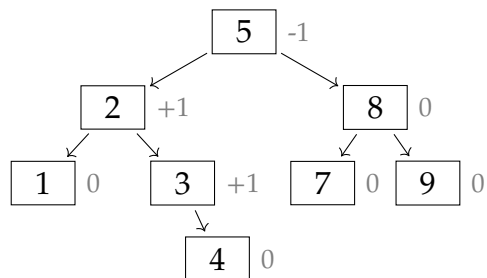
Nach dem Einfügen von „9“:



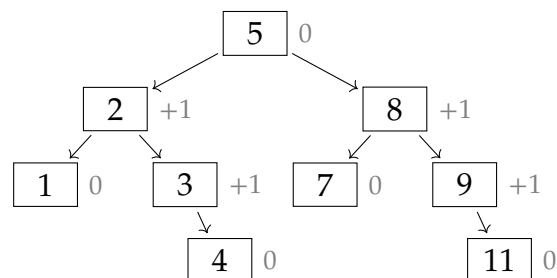
Nach der Linksrotation:



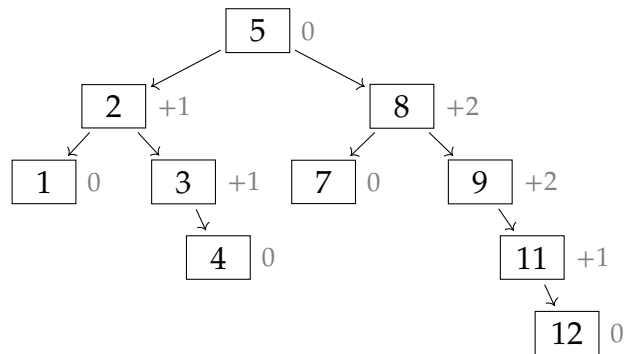
Nach dem Einfügen von „4“:



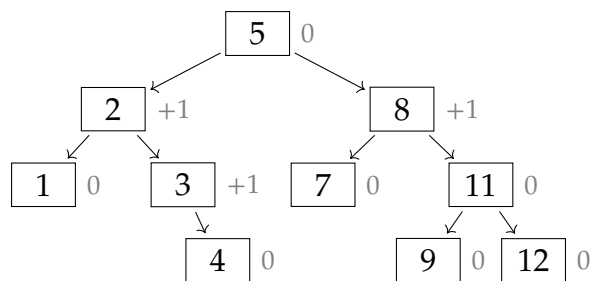
Nach dem Einfügen von „11“:



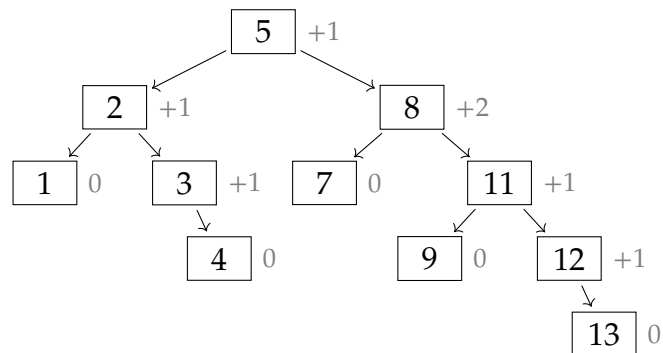
Nach dem Einfügen von „12“:



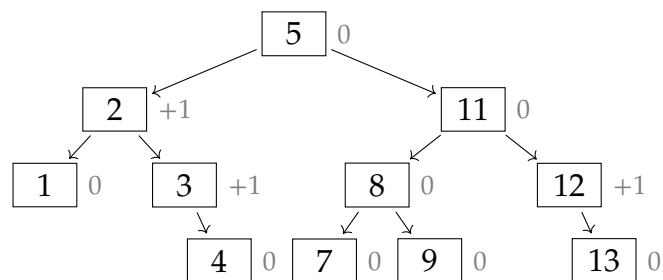
Nach der Linksrotation:



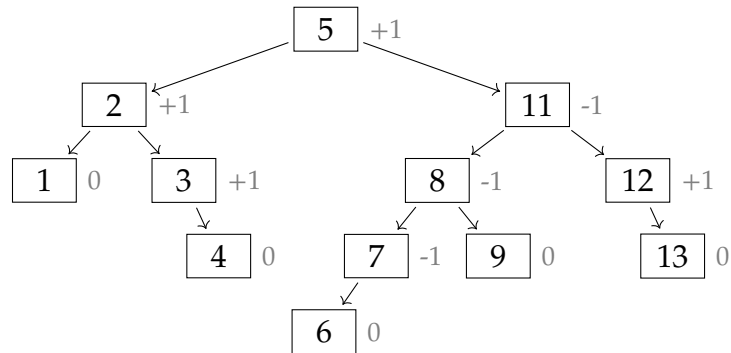
Nach dem Einfügen von „13“:



Nach der Linksrotation:



Nach dem Einfügen von „6“:



- (c) Geben Sie eine möglichst gute untere Schranke (in Ω -Notation) für die Anzahl der Schlüssel in einem 2-3-4-Baum der Höhe h an.

Hinweis: Überlegen Sie sich, wie ein 2-3-4-Baum mit Höhe h und möglichst wenigen Schlüsseln aussieht.

Lösungsvorschlag

Ein 2-3-4-Baum mit möglichst wenigen Schlüsseln sieht aus wie ein Binärbaum:

- Ein Baum der Höhe 1 hat 1 Schlüssel.
- Ein Baum der Höhe 2 hat 3 Schlüssel.
- Ein Baum der Höhe 3 hat 7 Schlüssel.
- ...
- Ein Baum der Höhe h hat $2^h - 1$ Schlüssel.

Also liegt die Untergrenze für die Anzahl der Schlüssel in $\Omega(2^h)$.

- (d) Geben Sie eine möglichst gute obere Schranke (in \mathcal{O} -Notation) für die Anzahl der Schlüssel in einem 2-3-4-Baum der Höhe h an.

Lösungsvorschlag

Ein 2-3-4-Baum mit möglichst vielen Schlüsseln hat in jedem Knoten drei Schlüssel. Und jeder Knoten, der kein Blatt ist, hat vier Kinder:

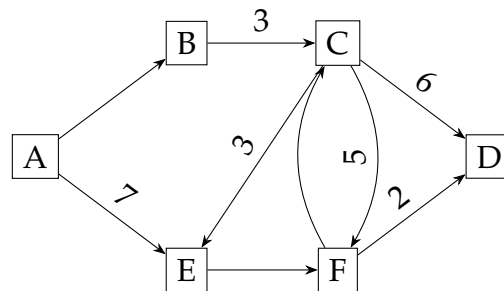
- Ein Baum der Höhe 1 hat 3 Schlüssel.
- Ein Baum der Höhe 2 hat 15 Schlüssel.
- Ein Baum der Höhe 3 hat 63 Schlüssel.
- ...
- Ein Baum der Höhe h hat $4^h - 1$ Schlüssel.

Also liegt die Obergrenze für die Anzahl der Schlüssel in $\mathcal{O}(4^h)$.

46115 / 2012 / Frühjahr / Thema 1 / Aufgabe 6

Aufgabe 6

Gegeben sei der folgende gerichtete Graph $G = (V, E, d)$ mit den angegebenen Kantengewichten.



- (a) Geben Sie eine formale Beschreibung des abgebildeten Graphen G durch Auflistung von V , E und d an.

Lösungsvorschlag

$$G = (V, E, d)$$

mit

$$V = \{A, B, C, D, E, F\}$$

und

$$E = \{(A, B), (A, E), (B, C), (C, D), (C, E), (C, F), (E, F), (F, C), (F, D), \}$$

und

$$d = \{1, 7, 3, 6, 3, 5, 1, 1, 2\}$$

Als Adjazenzliste

$$A: \rightarrow B \quad \xrightarrow{7} E$$

$$B: \xrightarrow{3} C$$

$$C: \xrightarrow{6} D \quad \xrightarrow{3} E \quad \xrightarrow{5} F$$

D:

$$E: \rightarrow F$$

$$F: \rightarrow C \quad \xrightarrow{2} D$$

- (b) Erstellen Sie die Adjazenzmatrix A zum Graphen G .

Lösungsvorschlag

	A	B	C	D	E	F
A	*	1	—	—	7	—
B	—	*	3	—	—	—
C	—	—	*	6	3	5
D	—	—	—	*	—	—
E	—	—	—	—	*	1
F	—	—	1	2	—	*

- (c) Berechnen Sie unter Verwendung des Algorithmus nach Dijkstra - vom Knoten A beginnend - den kürzesten Weg, um alle Knoten zu besuchen. Die Restknoten werden in einer Halde (engl. Heap) gespeichert. Geben Sie zu jedem Arbeitsschritt den Inhalt dieser Halde an.

Nr.	besucht	A	B	C	D	E	F
0		0	∞	∞	∞	∞	∞
1	A	0	1	∞	∞	7	∞
2	B		1	4	∞	7	∞
3	C			4	10	7	9
4	E				10	7	8
5	F				10		8
6	D				10		
nach	Entfernung	Reihenfolge		Pfad			
A → A	0	1					
A → B	1	2		A → B			
A → C	4	3		A → B → C			
A → D	10	6		A → B → C → D			
A → E	7	4		A → E			
A → F	8	5		A → E → F			

46115 / 2013 / Frühjahr / Thema 2 / Aufgabe 4

„Streuspeicherung“

Die Werte 7, 0, 9, 11, 18, 4, 5, 3, 13, 24, 2 sollen in eine Hashtabelle der Größe 11 (Fächer 0 bis 10) eingetragen werden. Die zur Hashfunktion $h(x) = (7 \cdot x) \% 11$ gehörenden Schlüssel sind in der folgenden Tabelle bereits ausgerechnet:

x	7	0	9	11	18	4	5	3	13	24	2
$h(x)$	5	0	8	0	5	6	2	10	3	3	3

- (a) Fügen Sie die oben genannten Schlüssel in der vorgegebenen Reihenfolge in einen Streuspeicher ein, welcher zur Kollisionsauflösung verkettete Listen verwendet, und stellen Sie die endgültige Streutabelle dar.

Lösungsvorschlag

Index	0	1	2	3	4	5	6	7	8	9	10
Schlüssel	0		5	13		7	4		8		3
	11			24		18					
				2							

- (b) Fügen Sie die gleichen Schlüssel mit linearem Sondieren bei Schrittweite +1 zur Kollisionsauflösung in eine neue Hash-Tabelle ein. Geben Sie für jeden Schlüssel an, auf welche Felder beim Einfügen zugegriffen wird und ob Kollisionen auftreten. Geben Sie die gefüllte Streutabelle an.

Lösungsvorschlag

Index	0	1	2	3	4	5	6	7	8	9	10
Schlüssel	0	11 ₁	5	13	24 ₁	7	18 ₁	4 ₁	9	2 ₆	3

- (c) Wie hoch ist der „Load“-Faktor (die Belegung) der Hashtabelle aus a) bzw. b) in Prozent? Können Sie weitere Schlüssel einfügen?

Lösungsvorschlag

Teilaufgabe a)

$\frac{11}{11} = 100\%$: Es können allerdings weitere Elemente eingefügt werden. Die Verkettung lässt einen Loadfaktor über 100% zu. Der Suchaufwand wird dann jedoch größer.

Teilaufgabe b)

$\frac{11}{11} = 100\%$: Es können keine weiteren Elemente eingefügt werden, da alle Buckets belegt sind.

Halde (Heap)
Binärbaum
AVL-Baum
Min-Heap

- (d) Würden Sie sich bei dieser Zahlensequenz für das Hashing-Verfahren nach a) oder nach b) entscheiden? Begründen Sie kurz Ihre Entscheidung.

Lösungsvorschlag

Das Verfahren a) scheint hier sinnvoller, da noch nicht zu viele Suchoperationen notwendig sind (max. 2), während bei Verfahren b) einmal bereits 6-mal sondiert werden muss.

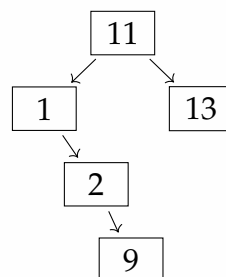
46115 / 2014 / Frühjahr / Thema 1 / Aufgabe 7

Fügen Sie nacheinander die Zahlen 11, 1, 2, 13, 9, 10, 7, 5

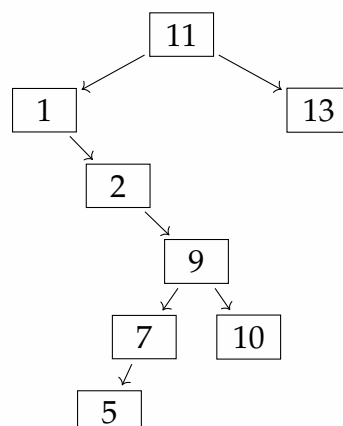
- (a) in einen leeren binären Suchbaum und zeichnen Sie den Suchbaum jeweils nach dem Einfügen von „9“ und „5“

Lösungsvorschlag

Nach dem Einfügen von „9“:

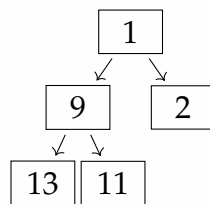


Nach dem Einfügen von „5“:

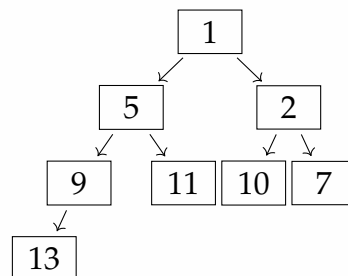


- (b) in einen leeren Min-Heap ein, der bzgl. „ \leq “ angeordnet ist und geben Sie den Heap nach „9“ und nach „5“ an

Nach dem Einfügen von „9“:

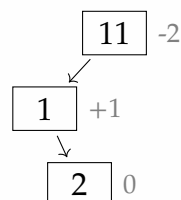


Nach dem Einfügen von „5“:

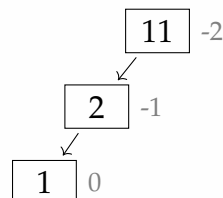


- (c) in einen leeren AVL-Baum ein! Geben Sie den AVL Baum nach „2“ und „5“ an und beschreiben Sie die ggf. notwendigen Rotationen beim Einfügen dieser beiden Elemente!

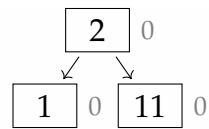
Nach dem Einfügen von „2“:



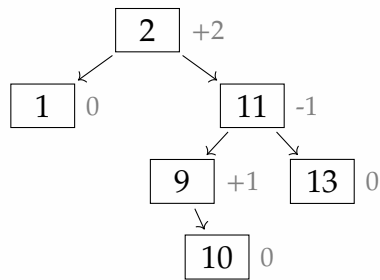
Nach der Linksrotation:



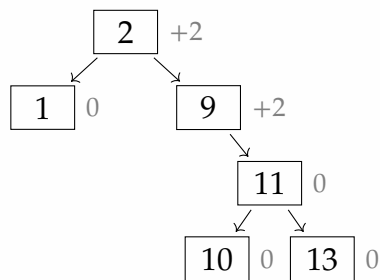
Nach der Rechtsrotation:



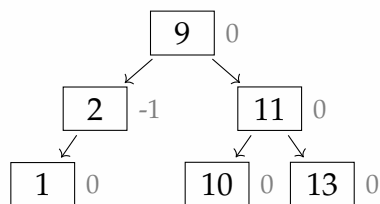
Nach dem Einfügen von „10“:



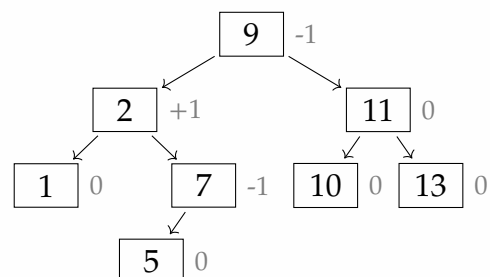
Nach der Rechtsrotation:



Nach der Linksrotation:



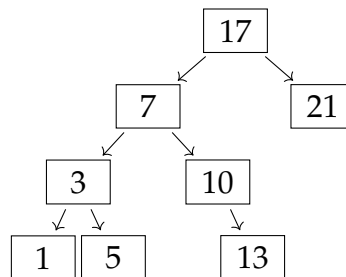
Nach dem Einfügen von „5“:



46115 / 2014 / Frühjahr / Thema 2 / Aufgabe 3

Frühjahr 2014 (46115) - Thema 2 Aufgabe 3

- (a) Fügen Sie die Zahlen 17, 7, 21, 3, 10, 13, 1, 5 nacheinander in der vorgegebenen Reihenfolge in einen binären Suchbaum ein und zeichnen Sie das Ergebnis!



- (b) Implementieren Sie in einer objektorientierten Programmiersprache eine rekursiv festgelegte Datenstruktur, deren Gestaltung sich an folgender Definition eines binären Baumes orientiert!

Ein binärer Baum ist entweder ein leerer Baum oder besteht aus einem Wurzelement, das einen binären Baum als linken und einen als rechten Teilbaum besitzt. Bei dieser Teilaufgabe können Sie auf die Implementierung von Methoden (außer ggf. notwendigen Konstruktoren) verzichten!

Klasse Knoten

```

class Knoten {
    public int wert;
    public Knoten links;
    public Knoten rechts;
    public Knoten elternKnoten;

    Knoten(int wert) {
        this.wert = wert;
        links = null;
        rechts = null;
        elternKnoten = null;
    }

    public Knoten findeMiniumRechterTeilbaum() {
    }

    public void anhängen (Knoten knoten) {
    }
}

public class BinärerSuchbaum {
    public Knoten wurzel;
}
  
```

```
BinärerSuchbaum(Knoten wurzel) {
    this.wurzel = wurzel;
}

BinärerSuchbaum() {
    this.wurzel = null;
}

public void einfügen(Knoten knoten) {
}

public void einfügen(Knoten knoten, Knoten elternKnoten) {
}

public Knoten suchen(int wert) {
}

public Knoten suchen(int wert, Knoten knoten) {
}
}
```

- (c) Beschreiben Sie durch Implementierung in einer gängigen objektorientierten Programmiersprache, wie bei Verwendung der obigen Datenstruktur die Methode `loescheKnoten(w)` gestaltet sein muss, mit der der Knoten mit dem Eintrag `w` aus dem Baum entfernt werden kann, ohne die Suchbaumeigenschaft zu verletzen!

Lösungsvorschlag

```
public void loescheKnoten(int w) {
    Knoten knoten = suchen(w);
    if (knoten == null) return;
    // Der Knoten hat keine Teilbäume.
    if (knoten.links == null && knoten.rechts == null) {
        if (w < knoten.elternKnoten.wert) {
            knoten.elternKnoten.links = null;
        } else {
            knoten.elternKnoten.rechts = null;
        }
    }

    // Der Knoten besitzt einen Teilbaum.
    // links
    else if (knoten.links != null && knoten.rechts == null) {
        knoten.elternKnoten.anhängen(knoten.links);
    }
    // rechts
    else if (knoten.links == null) {
        knoten.elternKnoten.anhängen(knoten.rechts);
    }

    // Der Knoten besitzt zwei Teilbäume.
    else {
        Knoten minimumKnoten = knoten.findeMiniumRechterTeilbaum();
    }
}
```

```
        minimumKnoten.links = knoten.links;  
        minimumKnoten.rechts = knoten.rechts;  
        knoten.elternKnoten.anhängen(minimumKnoten);  
    }  
}
```

Code-Beispiel auf Github ansehen:

[src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/suchbaum/BinaererSuchbaum.java](https://github.com/bschlangaul/examen_exam_46115_jahr_2014_fruehjahr_suchbaum/BinaererSuchbaum.java)

Klasse „BinärerSuchbaum“

```
public class BinaererSuchbaum {
    public Knoten wurzel;

    BinaererSuchbaum(Knoten wurzel) {
        this.wurzel = wurzel;
    }

    BinaererSuchbaum() {
        this.wurzel = null;
    }

    public void einfügen(Knoten knoten) {
        if (wurzel != null) {
            einfügen(knoten, wurzel);
        } else {
            wurzel = knoten;
            knoten.elternKnoten = wurzel;
        }
    }

    public void einfügen(int wert) {
        einfügen(new Knoten(wert));
    }

    public void einfügen(Knoten knoten, Knoten elternKnoten) {
        if (knoten.wert <= elternKnoten.wert) {
            if (elternKnoten.links != null) {
                einfügen(knoten, elternKnoten.links);
            } else {
                elternKnoten.links = knoten;
                knoten.elternKnoten = elternKnoten;
            }
        } else {
            if (elternKnoten.rechts != null) {
                einfügen(knoten, elternKnoten.rechts);
            } else {
                elternKnoten.rechts = knoten;
                knoten.elternKnoten = elternKnoten;
            }
        }
    }
}
```

```
public Knoten suchen(int wert) {
    if (wurzel == null || wurzel.wert == wert) {
        return wurzel;
    } else {
        return suchen(wert, wurzel);
    }
}

public Knoten suchen(int wert, Knoten knoten) {
    if (knoten.wert == wert) {
        return knoten;
    } else if (wert < knoten.wert && knoten.links != null) {
        return suchen(wert, knoten.links);
    } else if (wert > knoten.wert && knoten.rechts != null) {
        return suchen(wert, knoten.rechts);
    }
    return null;
}

public void loescheKnoten(int w) {
    Knoten knoten = suchen(w);
    if (knoten == null) return;
    // Der Knoten hat keine Teilbäume.
    if (knoten.links == null && knoten.rechts == null) {
        if (w < knoten.elternKnoten.wert) {
            knoten.elternKnoten.links = null;
        } else {
            knoten.elternKnoten.rechts = null;
        }
    }

    // Der Knoten besitzt einen Teilbaum.
    // links
    else if (knoten.links != null && knoten.rechts == null) {
        knoten.elternKnoten.anhängen(knoten.links);
    }
    // rechts
    else if (knoten.links == null) {
        knoten.elternKnoten.anhängen(knoten.rechts);
    }

    // Der Knoten besitzt zwei Teilbäume.
    else {
        Knoten minimumKnoten = knoten.findeMiniumRechterTeilbaum();
        minimumKnoten.links = knoten.links;
        minimumKnoten.rechts = knoten.rechts;
        knoten.elternKnoten.anhängen(minimumKnoten);
    }
}

// Der Baum aus dem Foliensatz
public BinaererSuchbaum erzeugeTestBaum() {
    BinaererSuchbaum binärerSuchbaum = new BinaererSuchbaum();
    binärerSuchbaum.einfügen(new Knoten(7));
}
```

```
binärerSuchbaum.einfügen(new Knoten(3));
binärerSuchbaum.einfügen(new Knoten(11));
binärerSuchbaum.einfügen(new Knoten(2));
binärerSuchbaum.einfügen(new Knoten(6));
binärerSuchbaum.einfügen(new Knoten(9));
binärerSuchbaum.einfügen(new Knoten(1));
binärerSuchbaum.einfügen(new Knoten(5));
return binärerSuchbaum;
}

public void ausgebenInOrder() {

}

public static void main(String[] args) {
    BinaererSuchbaum binärerSuchbaum = new BinaererSuchbaum();
    BinaererSuchbaum testBaum = binärerSuchbaum.erzeugeTestBaum();

    // Teste das Einfügen

    System.out.println(testBaum.wurzel.wert); // 7
    System.out.println(testBaum.wurzel.links.wert); // 3
    System.out.println(testBaum.wurzel.links.links.wert); // 2
    System.out.println(testBaum.wurzel.links.rechts.wert); // 6
    System.out.println(testBaum.wurzel.rechts.wert); // 11

    // Teste das Suchen

    System.out.println("Gesucht nach 5 und gefunden: " + testBaum.suchen(5).wert);
    System.out.println("Gesucht nach 9 und gefunden: " + testBaum.suchen(9).wert);
    System.out.println("Gesucht nach 7 und gefunden: " + testBaum.suchen(7).wert);
    System.out.println("Gesucht nach 10 und gefunden: " + testBaum.suchen(10));

    // Teste das Löschen

    // Der Knoten hat keine Teilbäume.
    System.out.println("Noch nicht gelöschter Knoten 9: " +
        ↳ testBaum.suchen(9).wert);
    testBaum.loescheKnoten(9);
    System.out.println("Gelöschter Knoten 9: " + testBaum.suchen(9));

    // Der Knoten hat einen Teilbaum.
    // fristen Testbaum erzeugen.
    testBaum = binärerSuchbaum.erzeugeTestBaum();
    Knoten elternKnoten = testBaum.suchen(3);
    System.out.println("Rechts Kind von 3 vor dem Löschen: " +
        ↳ elternKnoten.rechts.wert);
    testBaum.loescheKnoten(6);
    System.out.println("Rechts Kind von 3Nach dem Löschen: " +
        ↳ elternKnoten.rechts.wert);

    // Der Knoten hat zwei Teilbäume.
    // fristen Testbaum erzeugen.
    testBaum = binärerSuchbaum.erzeugeTestBaum();
```

```

Knoten wurzel = testBaum.wurzel;
System.out.println("Linkes Kind der Wurzel vor dem Löschen: " +
    ↪ wurzel.links.wert); // 5
testBaum.loescheKnoten(3);
System.out.println("Linkes Kind der WurzelNach dem Löschen: " +
    ↪ wurzel.links.wert); // 3
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/suchbaum/BinaererSuchbaum.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/suchbaum/BinaererSuchbaum.java)

Klasse „Knoten“

```

class Knoten {
    public int wert;
    public Knoten links;
    public Knoten rechts;
    public Knoten elternKnoten;

    Knoten(int wert) {
        this.wert = wert;
        links = null;
        rechts = null;
        elternKnoten = null;
    }

    public Knoten findeMiniumRechterTeilbaum() {
        if (rechts != null) {
            Knoten minimumKnoten = rechts;
            while (minimumKnoten.links != null) {
                minimumKnoten = minimumKnoten.links;
            }
            return minimumKnoten;
        }
        return null;
    }

    public void anhängen (Knoten knoten) {
        if (knoten.wert < wert) {
            links = knoten;
        } else {
            rechts = knoten;
        }
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/suchbaum/Knoten.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/suchbaum/Knoten.java)

46115 / 2015 / Herbst / Thema 2 / Aufgabe 1

Fügen Sie die folgenden Werte in der gegebenen Reihenfolge in eine Streutabelle der Größe 8 (mit den Indizes 0 bis 7) und der Streufunktion $h(x) = x \bmod 8$ ein. Verwenden Sie die jeweils angegebene Hash-Variante bzw. Kollisionsauflösung: 15, 3, 9, 23, 1,

8, 17, 4

(a) Offenes Hashing

Zur Kollisionsauflösung wird Verkettung verwendet.

Beispiel

Für die beiden Werte 8 und 16 würde die Lösung wie folgt aussehen:

Bucket	0	1	2	...
Inhalt	8			
	16			

Lösungsvorschlag

$ \begin{aligned} h(15) &= 15 \bmod 8 = 7 \\ h(3) &= 3 \bmod 8 = 3 \\ h(9) &= 9 \bmod 8 = 1 \\ h(23) &= 23 \bmod 8 = 7 \\ h(1) &= 1 \bmod 8 = 1 \\ h(8) &= 8 \bmod 8 = 0 \\ h(17) &= 17 \bmod 8 = 1 \\ h(4) &= 4 \bmod 8 = 4 \end{aligned} $									
Bucket	0	1	2	3	4	5	6	7	
Inhalt	8	9		3	4			15	
		1						23	
		17							

(b) Geschlossenes Hashing

Zur Kollisionsauflösung wird lineares Sondieren (nur hochzählend) mit Schrittweite +5 verwendet.

Treten beim Einfügen Kollisionen auf, dann notieren Sie die Anzahl der Versuche zum Ablegen des Wertes im Subskript (z. B. das Einfügen des Wertes 8 gelingt im 5. Versuch: 8₅).

Beispiel

Für die beiden Werte 8 und 16 würde die Lösung wie folgt aussehen:

Bucket	0	1	2	3	4	5	...
Inhalt	8					16 ₁	

Lösungsvorschlag

$$h'(x) = x \bmod 8$$

$$h(x, i) = (h'(x) + i \cdot 5) \bmod 8$$

17 einfügen

Bucket	0	1	2	3	4	5	6	7
Inhalt	8	9		3	23 ₂		1 ₂	15

1. Versuch: $h(17, 0) = (h'(17) + 0 \cdot 5) \bmod 8 = (1 + 0) \bmod 8 = 1 \bmod 8 = 1$ (belegt von 9)
2. Versuch: $h(17, 1) = (h'(17) + 1 \cdot 5) \bmod 8 = (1 + 5) \bmod 8 = 6 \bmod 8 = 6$ (belegt von 1)
3. Versuch: $h(17, 2) = (h'(17) + 2 \cdot 5) \bmod 8 = (1 + 10) \bmod 8 = 11 \bmod 8 = 3$ (belegt von 3)
4. Versuch: $h(17, 3) = (h'(17) + 3 \cdot 5) \bmod 8 = (1 + 15) \bmod 8 = 16 \bmod 8 = 0$ (belegt von 8)
5. Versuch: $h(17, 4) = (h'(17) + 4 \cdot 5) \bmod 8 = (1 + 20) \bmod 8 = 21 \bmod 8 = 5$

4 einfügen

Bucket	0	1	2	3	4	5	6	7
Inhalt	8	9		3	23 ₂	17 ₅	1 ₂	15

1. Versuch: $h(4, 0) = (h'(4) + 0 \cdot 5) \bmod 8 = (4 + 0) \bmod 8 = 4$ (belegt von 23)
2. Versuch: $h(4, 1) = (h'(4) + 1 \cdot 5) \bmod 8 = (4 + 5) \bmod 8 = 1$ (belegt von 9)
3. Versuch: $h(4, 2) = (h'(4) + 2 \cdot 5) \bmod 8 = (4 + 10) \bmod 8 = 6$ (belegt von 1)
4. Versuch: $h(4, 3) = (h'(4) + 3 \cdot 5) \bmod 8 = (4 + 15) \bmod 8 = 3$ (belegt von 3)
5. Versuch: $h(4, 4) = (h'(4) + 4 \cdot 5) \bmod 8 = (4 + 20) \bmod 8 = 0$ (belegt von 8)
6. Versuch: $h(4, 5) = (h'(4) + 5 \cdot 5) \bmod 8 = (4 + 25) \bmod 8 = 5$ (belegt von 17)
7. Versuch: $h(4, 6) = (h'(4) + 6 \cdot 5) \bmod 8 = (4 + 30) \bmod 8 = 2$

Bucket	0	1	2	3	4	5	6	7
Inhalt	8	9	4 ₇	3	23 ₂	17 ₅	1 ₂	15

- (c) Welches Problem tritt auf, wenn zur Kollisionsauflösung lineares Sondieren mit Schrittweite 4 verwendet wird? Warum ist 5 eine bessere Wahl?

Lösungsvorschlag

Beim linearen Sondieren mit der Schrittweite 4 werden nur zwei verschiedene Buckets erreicht, beispielsweise: 1, 5, 1, 5, etc.

Beim linearen Sondieren mit der Schrittweite 5 werden nacheinander alle möglichen Buckets erreicht, beispielsweise: 1, 6, 3, 0, 5, 2, 7, 4.

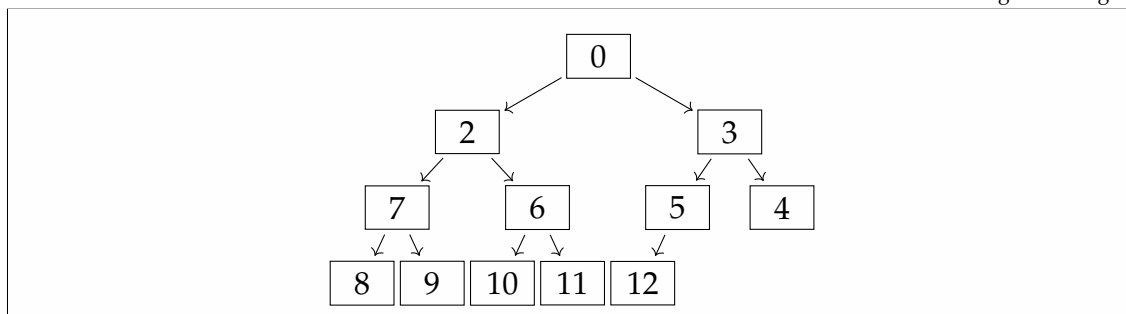
46115 / 2017 / Herbst / Thema 2 / Aufgabe 6

Gegeben sei folgende Feld-Einbettung (Array-Darstellung) einer Min-Halde:

0	1	2	3	4	5	6	7	8	9	10	11
0	2	3	7	6	5	4	8	9	10	11	12

- (a) Stellen Sie die Halde graphisch als (links-vollständigen) Baum dar.

Lösungsvorschlag



- (b) Entfernen Sie das kleinste Element (die Wurzel 0) aus der obigen initialen Halde, stellen Sie die Haldeneigenschaft wieder her und geben Sie nur das Endergebnis in Felddarstellung an.

Lösungsvorschlag

0	1	2	3	4	5	6	7	8	9	10
2	6	3	7	10	5	4	8	9	12	11

- (c) Fügen Sie nun den Wert 1 in die obige initiale Halde ein, stellen Sie die Haldeneigenschaft wieder her und geben Sie nur das Endergebnis in Felddarstellung an.

Lösungsvorschlag

0	1	2	3	4	5	6	7	8	9	10	11	12
0	2	1	7	6	3	4	8	9	10	11	12	5

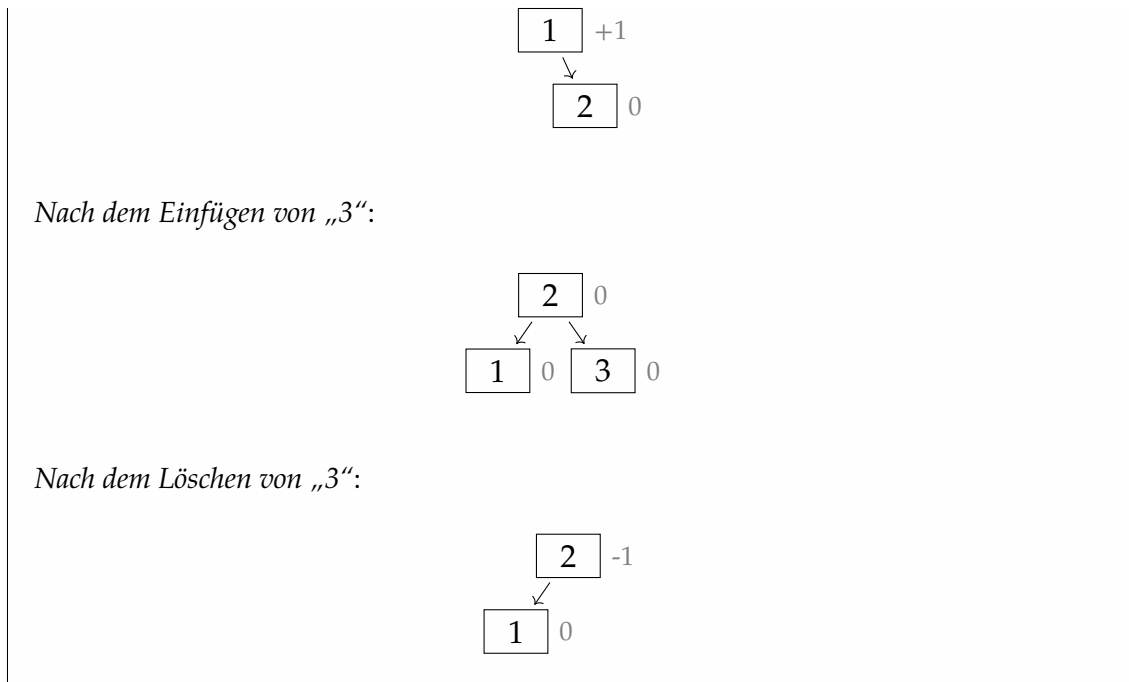
46115 / 2019 / Frühjahr / Thema 2 / Aufgabe 3

- (a) Zeigen oder widerlegen Sie die folgende Aussage: Wird ein Element in einen AVL-Baum eingefügt und unmittelbar danach wieder gelöscht, so befindet sich der AVL-Baum wieder in seinem Ursprungszustand.

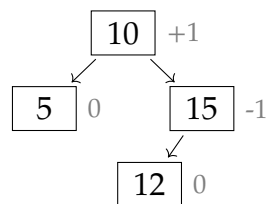
Lösungsvorschlag

Die Aussage ist falsch. Wir widerlegen die Aussage durch ein konkretes Beispiel:

Unser Ausgangs-AVL-Baum:



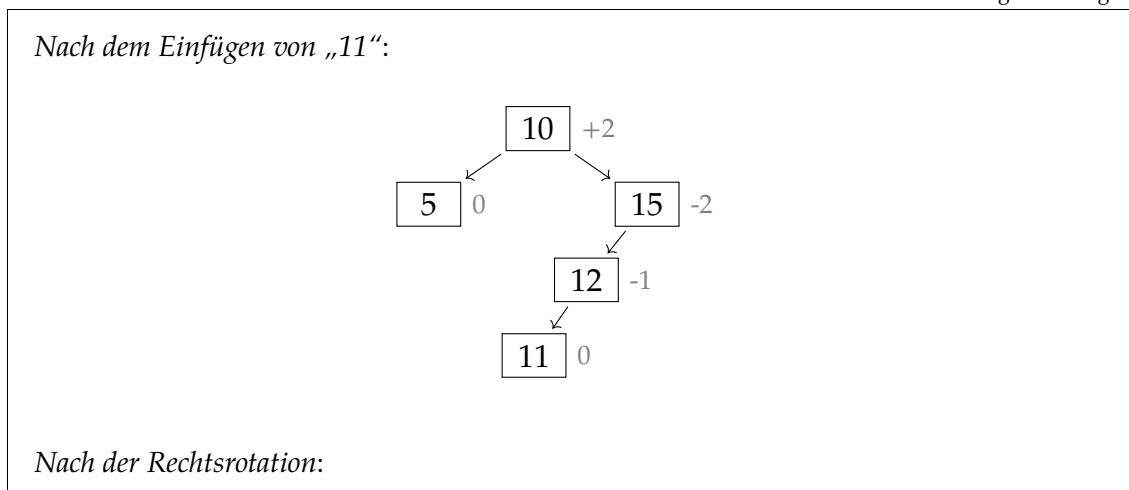
(b) Fügen Sie in den gegebenen Baum den Schlüssel 11 ein.

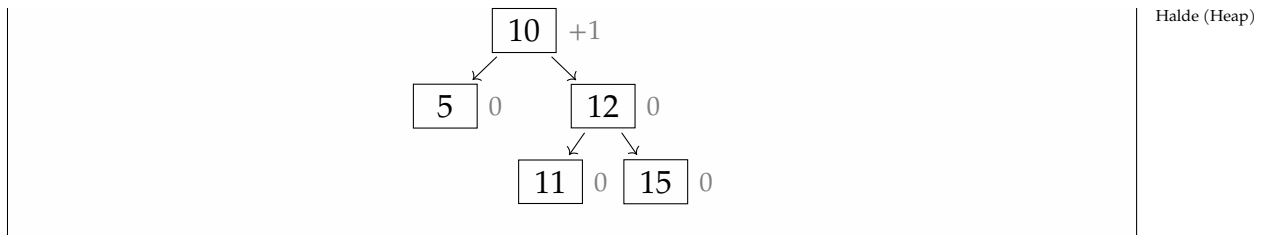


Rebalancieren Sie anschließend den Baum so, dass die AVL-Eigenschaft wieder erreicht wird. Zeichnen Sie den Baum nach jeder Einfach- und Doppelrotation und benennen Sie die Art der Rotation (Links-, Rechts-, Links-Rechts-, oder Rechts-Links-Rotation). Argumentieren Sie jeweils über die Höhenbalancen der Teilbäume.

Tipp: Zeichnen Sie nach jedem Schritt die Höhenbalancen in den Baum ein.

Lösungsvorschlag





46115 / 2019 / Herbst / Thema 2 / Aufgabe 7

Schreiben Sie in Pseudocode eine Methode `heapify(int[] a)`, welche im übergebenen Array der Länge n die Heapeigenschaft in $\mathcal{O}(n)$ Schritten herstellt. D. h. als Ergebnis soll in a gelten, dass $a[i] \leq a[2i + 1]$ und $a[i] \leq a[i + 2]$.

```
import org.bschlangaul.helfer.Konsole;

/**
 * Nach Pseudocode nach
 * https://www.oreilly.com/library/view/algorithms-in-a/9780596516246/ch04s06.html
 */
public class Heapify {

    public static void buildHeap(int a[]) {
        int n = a.length;
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(a, i, n);
        }
    }

    public static void heapify(int a[], int index, int max) {
        int left = 2 * index + 1;
        int right = 2 * index + 2;
        int smallest;

        if (left < max && a[left] < a[index]) {
            smallest = left;
        } else {
            smallest = index;
        }

        if (right < max && a[right] < a[smallest]) {
            smallest = right;
        }

        if (smallest != index) {
            int tmp = a[index];
            a[index] = a[smallest];
            a[smallest] = tmp;
            heapify(a, smallest, max);
        }
    }

    public static void main(String[] args) {
```

```

int[] a = new int[] { 5, 3, 16, 2, 10, 14 };
buildHeap(a);
Konsole.zeigeZahlenFeld(a); // 2 3 14 5 10 16
}

}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/Heapify.java](https://github.com/bschlangaul/examen/examen_46115/jahr_2019/herbst/Heapify.java)

Halde (Heap)
Streutabellen (Hashing)

46115 / 2020 / Frühjahr / Thema 2 / Aufgabe 7

Sei H ein Max-Heap, der n Elemente speichert. Für ein Element v in H sei $h(v)$ die Höhe von v , also die Länge eines längsten Pfades von v zu einem Blatt im Teilheap mit Wurzel v .

- Geben Sie eine rekursive Definition von $h(v)$ an, in der Sie sich auf die Höhen der Kinder $v.\text{left}$ und $v.\text{right}$ von v beziehen (falls v Kinder hat).
- Geben Sie eine möglichst niedrige obere asymptotische Schranke für die Summe der Höhen aller Elemente in H an, also für $\sum_{v \in H} h(v)$ und begründen Sie diese.
Tipp: Denken Sie daran, wie man aus einem beliebigen Feld einen Max-Heap macht.
- Sei H' ein Feld der Länge n . Geben Sie einen Algorithmus an, der in Linearzeit testet, ob H ein Max-Heap ist.

46115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 4

Eine Sondierfolge $s(k,i)$ liefert für einen Schlüssel k aus einem Universum U und Versuchsnummern $i = 0, 1, \dots, m-1$ eine Folge von Indizes für eine Hashtabelle $T[0 \dots m-1]$. Mithilfe einer Sondierfolge wird beim Hashing mit offener Adressierung z. B. beim Einfügen eines neuen Schlüssels k nach einem noch nicht benützten Tabelleneintrag gesucht. Seien h und h' zwei verschiedene Hash-funktionen, die U auf $0, 1, \dots, m-1$ abbilden. Beantworten Sie die folgenden Fragen und geben Sie an, um welche Art von Sondieren es sich jeweils handelt.

- Was ist problematisch an der Sondierfolge $s(k,i) = (h(k) + 2i) \bmod m$, wobei $m = 1023$ die Größe der Hashtabelle ist?

Lösungsvorschlag

Art Es handelt sich um lineares Sondieren.

Problematisch Es wird für einen großen Bereich an Sondierfolgen (512 (0-511, 512-1023)) nur in jeden zweiten Bucket (z. B. geradzahlig) sondiert, erst dann wird in den bisher ausgelassenen Buckets (z. B. ungeradzahlig) sondiert.

- Was ist problematisch an der Sondierfolge $s(k,i) = (h(k) + i(i+1)) \bmod m$, wobei $m = 1024$ die Größe der Hashtabelle ist?

Art Es handelt sich um quadratisches Sondieren

Problematisch $i(i+1)$ gibt immer eine gerade Zahl. Eine gerade Zahl Modulo 1024 gibt auch immer eine grade Zahl. Es wird nie in den ungeraden Buckets sondiert.

- (c) Was ist vorteilhaft an der Sondierfolge $s(k, i) = (h(k) + i \cdot h'(k)) \bmod m$, wobei m die Größe der Hashtabelle ist?

Lösungsvorschlag

Auch die Sondierfolge ist abhängig von dem Schlüsselwert. Die Entstehung von Ballungen ist unwahrscheinlicher bei gut gewählten Hashfunktionen, eine gleichmäßige Verteilung wahrscheinlicher.

- (d) Sei $h(k) = k \bmod 6$ und $h(k) = k^2 \bmod 6$

Fügen Sie die Schlüssel 14, 9, 8, 3, 2 in eine Hashtabelle der Größe 7 ein. Verwenden Sie die Sondierfolge $s(k, i) = (h(k) + i \cdot h'(k)) \bmod 7$ und offene Adressierung. Notieren Sie die Indizes der Tabellenfelder und vermerken Sie neben jedem Feld die erfolglosen Sondierungen.

66112 / 2003 / Herbst / Thema 2 / Aufgabe 8

- (a) Implementieren Sie in einer objektorientierten Sprache einen binären Suchbaum für ganze Zahlen! Dazu gehören Methoden zum Setzen und Ausgeben der Attribute `zahl`, `linker_teilbaum` und `rechter_teilbaum`. Design: eine Klasse `Knoten` und eine Klasse `BinBaum`. Ein Knoten hat einen linken und einen rechten Nachfolger. Ein Baum verwaltet die Wurzel. Er hängt neue Knoten an und löscht Knoten.

Lösungsvorschlag

```
public class BinBaum {

    private Knoten wurzel = null;

    public void setzeWurzel(Knoten knoten) {
        wurzel = knoten;
    }
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java](https://github.com/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java)

```
public class Knoten {
    private int zahl;
    private Knoten links = null;
    private Knoten rechts = null;

    public Knoten() {
    }

    public Knoten(int zahl) {
```



```

    this.zahl = zahl;
}

public void setzeZahl(int zahl) {
    this.zahl = zahl;
}

public int gibZahl() {
    return zahl;
}

public void setzeLinks(Knoten k) {
    links = k;
}

public Knoten gibLinks() {
    return links;
}

public void setzeRechts(Knoten k) {
    rechts = k;
}

public Knoten gibRechts() {
    return rechts;
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/Knoten.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/Knoten.java)

(b) Schreiben Sie eine Methode `fügeEin(...)`, die eine Zahl in den Baum einfügt!

Lösungsvorschlag

```

public void fügeEin(int zahl) {
    Knoten aktueller = wurzel;
    Knoten neuerKnoten = new Knoten(zahl);
    if (wurzel == null) {
        wurzel = neuerKnoten;
        return;
    }
    while (aktueller != null) {
        // suche links
        if (zahl <= aktueller.gibZahl() && aktueller.gibLinks() != null) {
            aktueller = aktueller.gibLinks();
            // fuege ein
        } else if (zahl <= aktueller.gibZahl() && aktueller.gibLinks() == null)
            → {
                aktueller.setzeLinks(neuerKnoten);
                break;
            }
        // suche rechts
        if (zahl > aktueller.gibZahl() && aktueller.gibRechts() != null) {
            aktueller = aktueller.gibRechts();
            // fuege ein
        }
    }
}

```

```

    } else if (zahl > aktueller.gibZahl() && aktueller.gibRechts() == null)
    ↪ {
        aktueller.setzeRechts(neuerKnoten);
        break;
    }
}
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java](https://github.com/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java)

- (c) Schreiben Sie eine Methode `void besuchePostOrder(...)`, die die Zahlen in der Reihenfolge postorder ausgibt!

Lösungsvorschlag

```

public static void besuchePostOrder(Knoten knoten) {
    // Sonderfall leerer (Teil-)Baum
    if (knoten == null) {
        System.out.println("Leerer Baum");
    } else {
        // Linker
        if (knoten.gibLinks() != null) {
            besuchePostOrder(knoten.gibLinks());
        }
        // Rechter
        if (knoten.gibRechts() != null) {
            besuchePostOrder(knoten.gibRechts());
        }
        System.out.println(knoten.gibZahl());
    }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java](https://github.com/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java)

- (d) Ergänzen Sie Ihr Programm um die rekursiv implementierte Methode `int berechneSumme(...)`, die die Summe der Zahlen des Unterbaums, dessen Wurzel der Knoten ist, zurückgibt! Falls der Unterbaum leer ist, ist der Rückgabewert 0!

`int summe (Knoten x)...`

Lösungsvorschlag

```

public int berechneSumme(Knoten knoten) {
    int ergebnis = 0;

    // Sonderfall: leerer Unterbaum
    if (knoten == null) {
        return 0;
    }
    // linker
    if (knoten.gibLinks() != null) {
        ergebnis = ergebnis + berechneSumme(knoten.gibLinks());
    }
    // rechter
    if (knoten.gibRechts() != null) {

```

```

        ergebnis = ergebnis + berechneSumme(knoten.gibRechts());
    }
    // Wurzel
    ergebnis = ergebnis + knoten.gibZahl();
    return ergebnis;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java](https://github.com/orgs/bschlangaul/repositories/examen/examen_66112/jahr_2003/herbst/BinBaum.java)

- (e) Schreiben Sie eine Folge von Anweisungen, die einen Baum mit Namen BinBaum erzeugt und nacheinander die Zahlen 5 und 7 einfügt! In den binären Suchbaum werden noch die Zahlen 4, 11, 6 und 2 eingefügt. Zeichnen Sie den Baum, den Sie danach erhalten haben, und schreiben Sie die eingefügten Zahlen in der Reihenfolge der Traversierungsmöglichkeit postorder auf!
- (f) Implementieren Sie eine Operation `isSorted(...)`, die für einen (Teil-)baum feststellt, ob er sortiert ist.

Lösungsvorschlag

```

public boolean istSortiert(Knoten knoten) {
    // Baum leer
    if (knoten == null) {
        return true;
    }

    // linker Nachfolger nicht okay
    if (knoten.gibLinks() != null && knoten.gibLinks().gibZahl() >
        knoten.gibZahl()) {
        return false;
    }

    // rechter Nachfolger nicht okay
    if (knoten.gibRechts() != null && knoten.gibRechts().gibZahl() <=
        knoten.gibZahl()) {
        return false;
    }

    // sonst prüfe Teilbaeume
    return (istSortiert(knoten.gibRechts()) &&
        istSortiert(knoten.gibLinks()));
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java](https://github.com/orgs/bschlangaul/repositories/examen/examen_66112/jahr_2003/herbst/BinBaum.java)

Lösungsvorschlag

```

public class BinBaum {

    private Knoten wurzel = null;

    public void setzeWurzel(Knoten knoten) {
        wurzel = knoten;
    }
}

```

```
public void fügeEin(int zahl) {
    Knoten aktueller = wurzel;
    Knoten neuerKnoten = new Knoten(zahl);
    if (wurzel == null) {
        wurzel = neuerKnoten;
        return;
    }
    while (aktueller != null) {
        // suche links
        if (zahl <= aktueller.gibZahl() && aktueller.gibLinks() != null) {
            aktueller = aktueller.gibLinks();
            // fuege ein
        } else if (zahl <= aktueller.gibZahl() && aktueller.gibLinks() == null) {
            aktueller.setzeLinks(neuerKnoten);
            break;
        }
        // suche rechts
        if (zahl > aktueller.gibZahl() && aktueller.gibRechts() != null) {
            aktueller = aktueller.gibRechts();
            // fuege ein
        } else if (zahl > aktueller.gibZahl() && aktueller.gibRechts() == null) {
            aktueller.setzeRechts(neuerKnoten);
            break;
        }
    }
}

public static void besuchePostOrder(Knoten knoten) {
    // Sonderfall leerer (Teil-)Baum
    if (knoten == null) {
        System.out.println("Leerer Baum");
    } else {
        // Linker
        if (knoten.gibLinks() != null) {
            besuchePostOrder(knoten.gibLinks());
        }
        // Rechter
        if (knoten.gibRechts() != null) {
            besuchePostOrder(knoten.gibRechts());
        }
        System.out.println(knoten.gibZahl());
    }
}

public int berechneSumme(Knoten knoten) {
    int ergebnis = 0;

    // Sonderfall: leerer Unterbaum
    if (knoten == null) {
        return 0;
    }
    // linker
    if (knoten.gibLinks() != null) {
        ergebnis = ergebnis + berechneSumme(knoten.gibLinks());
    }
}
```

```
    }
    // rechter
    if (knoten.gibRechts() != null) {
        ergebnis = ergebnis + berechneSumme(knoten.gibRechts());
    }
    // Wurzel
    ergebnis = ergebnis + knoten.gibZahl();
    return ergebnis;
}

public boolean istSortiert(Knoten knoten) {
    // Baum leer
    if (knoten == null) {
        return true;
    }

    // linker Nachfolger nicht okay
    if (knoten.gibLinks() != null && knoten.gibLinks().gibZahl() >
        ↪ knoten.gibZahl()) {
        return false;
    }

    // rechter Nachfolger nicht okay
    if (knoten.gibRechts() != null && knoten.gibRechts().gibZahl() <=
        ↪ knoten.gibZahl()) {
        return false;
    }

    // sonst prüfe Teilbaeume
    return (istSortiert(knoten.gibRechts()) && istSortiert(knoten.gibLinks()));
}

public static void main(String[] args) {
    BinBaum baum = new BinBaum();

    baum.fügeEin(5);
    baum.fügeEin(7);
    baum.fügeEin(4);
    baum.fügeEin(11);
    baum.fügeEin(6);
    baum.fügeEin(2);

    besuchePostOrder(baum.wurzel);
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java](https://github.com/bschlangaul/examen/examen_66112/jahr_2003/herbst/BinBaum.java)

```
public class Knoten {
    private int zahl;
    private Knoten links = null;
    private Knoten rechts = null;

    public Knoten() {
    }
}
```

```
public Knoten(int zahl) {
    this.zahl = zahl;
}

public void setzeZahl(int zahl) {
    this.zahl = zahl;
}

public int gibZahl() {
    return zahl;
}

public void setzeLinks(Knoten k) {
    links = k;
}

public Knoten gibLinks() {
    return links;
}

public void setzeRechts(Knoten k) {
    rechts = k;
}

public Knoten gibRechts() {
    return rechts;
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66112/jahr_2003/herbst/Knoten.java](https://github.com/orgs/bschlangaul/examen/examen_66112/jahr_2003/herbst/Knoten.java)

66112 / 2005 / Frühjahr / Thema 2 / Aufgabe 8

Gegeben seien die folgenden Zahlen: 7, 4, 3, 5, 0, 1

- (a) Zeichnen Sie eine Hash-Tabelle mit 8 Zellen und tragen Sie diese Zahlen genau in der oben gegebenen Reihenfolge in Ihre Hash-Tabelle ein. Verwenden Sie dabei die Streufunktion $f(n) = n^2 \bmod 7$ und eine Kollisionsauflösung durch lineares Sondieren.

Lösungsvorschlag

```
f(7) = 7^2 mod 7 = 49 mod 7 = 0
f(4) = 4^2 mod 7 = 16 mod 7 = 2
f(3) = 3^2 mod 7 = 9 mod 7 = 2 lineares Sondieren: +1 = 3
f(5) = 5^2 mod 7 = 25 mod 7 = 4
f(0) = 0^2 mod 7 = 0 mod 7 = 0 lineares Sondieren: +1 = 1
f(1) = 1^2 mod 7 = 1 mod 7 = 1 lineares Sondieren: -1 = 0, -1 = 7
```

0	1	2	3	4	5	6	7
7	0	4	3	5			1

- (b) Welcher Belegungsfaktor ist für die Streutabelle und die Streufunktion aus Teilaufgabe a zu erwarten, wenn sehr viele Zahlen eingeordnet werden und eine Kollisionsauflösung durch Verkettung (verzeigerte Listen) verwendet wird? Begründen Sie Ihre Antwort kurz.

Lösungsvorschlag

Der Belegungsfaktor berechnet sich aus der Formel:

$$\text{Belegungsfaktor} = \frac{\text{Anzahl tatsächlich eingetragenen Schlüssel}}{\text{Anzahl Hashwerte}}$$

Der Belegungsfaktor steigt kontinuierlich, je mehr Zahlen in die Streutabelle gespeichert werden.

Die Streufunktion legt die Zahlen nur in die Buckets 0, 1, 2, 4.

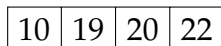
66112 / 2005 / Herbst / Thema 2 / Aufgabe 6

- (a) Erzeugen Sie aus der gegebenen Folge einen B-Baum der Ordnung $m = 2$:

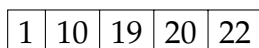
22, 10, 19, 20, 1, 13, 11, 12, 7, 8, 5, 42, 33, 21, 52, 48, 50

Fügen Sie dazu die einzelnen Elemente in gegebener Reihenfolge in einen anfangs leeren B-Baum ein. Stellen Sie für jeden Wert die entsprechenden Zwischenergebnisse und die angewendeten Operationen als Bäume dar!

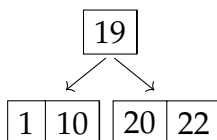
- $\boxed{+22}$ $\boxed{+10}$ $\boxed{+19}$ $\boxed{+20}$ Einfügen der ersten Zahlen bis zur kompletten Füllung der Wurzel:



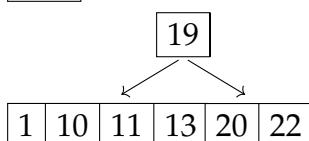
- $\boxed{+1}$ Einfügen der 1 führt zum Überlauf, deshalb Aufspaltung:



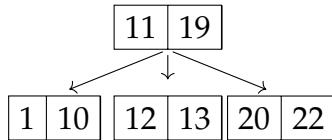
- Übernahme des mittleren Elements (19) in die Wurzel:



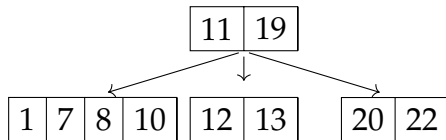
- $\boxed{+13}$ Einfügen der 13:



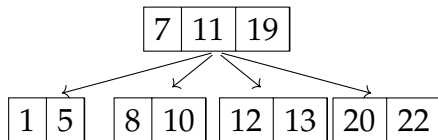
- **+12** Einfügen der 12 nicht möglich, also wieder Aufspaltung. 11 als mittleres Element wird nach oben geschrieben:



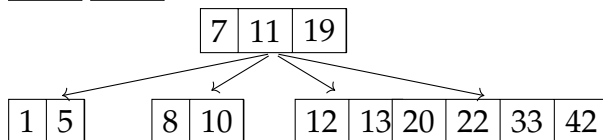
- **+7** **+8** Einfügen von 7 und 8:



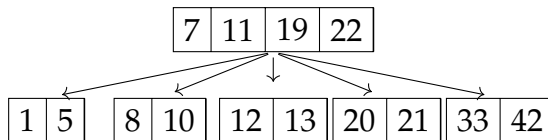
- **+5** Einfügen von 5 nicht möglich, deshalb Aufspaltung, 7 als mittleres Element wird nach oben geschrieben:



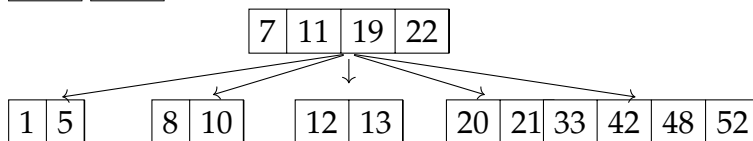
- **+42** **+33** Einfügen von 42 und 33:



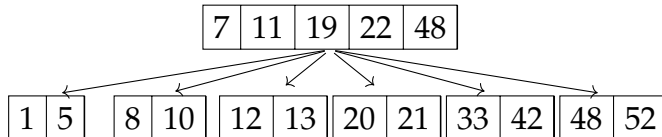
- **+21** Einfügen von 21 nicht möglich, also Aufspaltung, 22 nach oben schieben



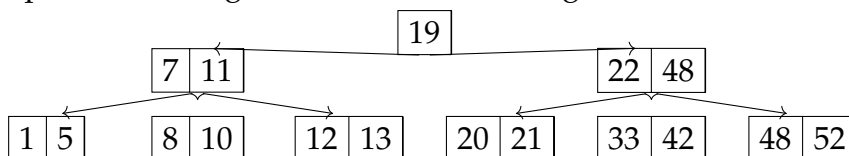
- **+52** **+48** Einfügen von 52 und 48



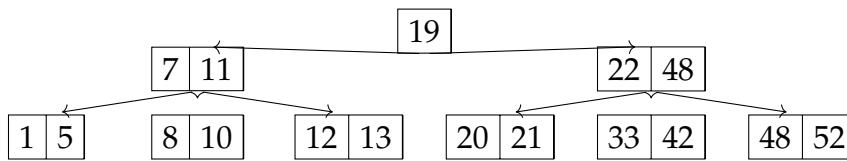
- **+50** Einfügen von 50 nicht möglich, daher splitten und 48 eine Ebene nach oben schieben



- Einfügen von 48 oben nicht möglich, da Knoten ebenfalls voll! -> weiterer Splitt notwendig, der neue Ebene erzeugt!



- (b) In dem Ergebnisbaum suchen wir nun den Wert 17. Stellen Sie den Ablauf des Suchalgorithmus an einer Zeichnung graphisch dar!



66114 / 2016 / Herbst / Thema 2 / Aufgabe 5

- (a) Erläutern Sie die wesentliche Eigenschaft eines Tupel-Identifikators (TID) in ein bis zwei Sätzen.

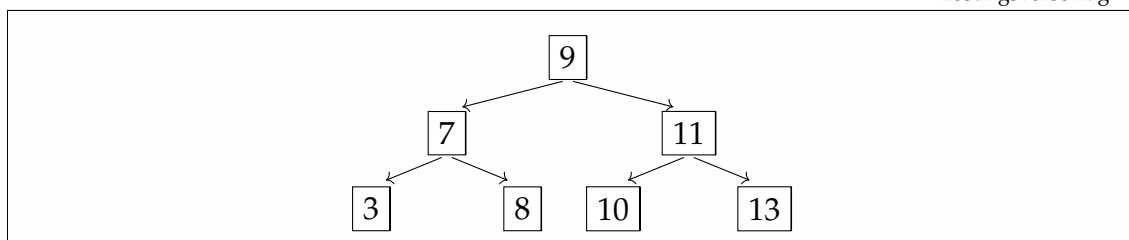
Lösungsvorschlag

- Daten werden in Form von *Sätzen* auf der Festplatte abgelegt, um auf Sätze zugreifen zu können, verfügt jeder Satz über eine *eindeutige, unveränderliche Satzadresse*
- TID = Tupel Identifier: dient zur Adressierung von Sätzen in einem Segment und besteht aus zwei Komponenten:
 - Seitennummer (Seiten bzw. Blöcke sind größere Speichereinheiten auf der Platte)
 - Relative Indexposition innerhalb der Seite
- Satzverschiebung innerhalb einer Seite bleibt ohne Auswirkungen auf den TID. Wird ein Satz auf eine andere Seite migriert, wird eine „Stellvertreter-TID“ zum Verweis auf den neuen Speicherort verwendet. Die eigentliche TID-Adresse bleibt stabil.

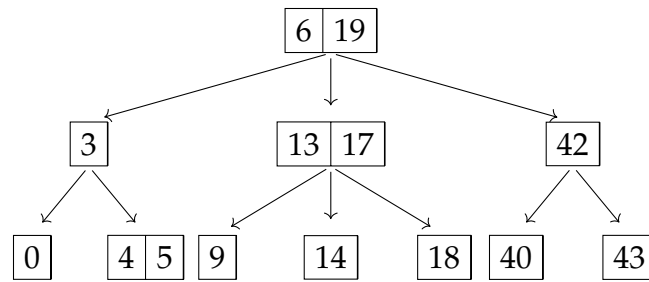
- (b) Fügen Sie in einen anfangs leeren B-Baum mit $k = 1$ (maximal 2 Schlüsselwerte pro Knoten) die im Folgenden gegebenen Schlüsselwerte der Reihe nach ein. Zeichnen Sie den Endzustand des Baums nach jedem Einfügevorgang. Falls Sie Zwischenschritte zeichnen, kennzeichnen Sie die sieben Endzustände deutlich.

3, 7, 13, 11, 9, 10, 8

Lösungsvorschlag



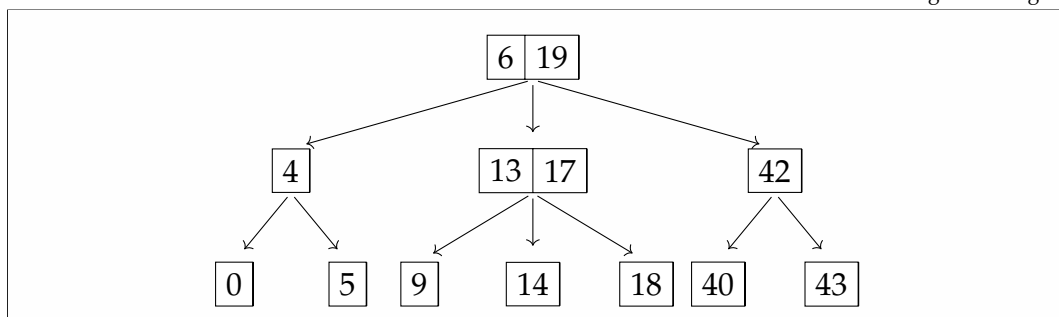
- (c) Gegeben ist der folgende B-Baum:



Die folgenden Teilaufgaben sind voneinander unabhängig.

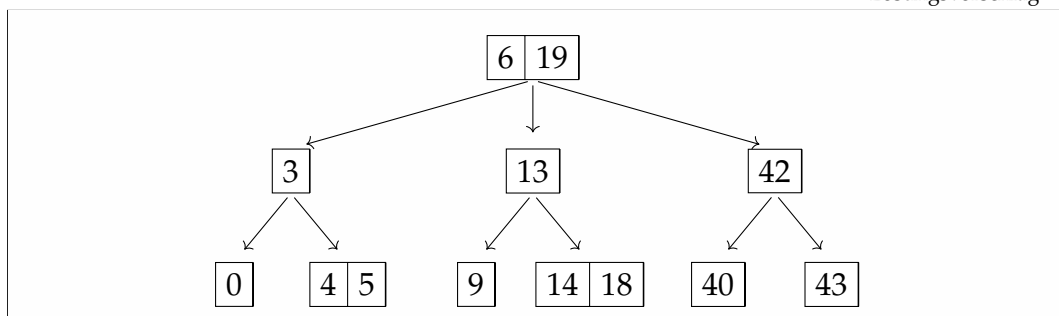
- (i) Löschen Sie aus dem gegebenen B-Baum den Schlüssel 3 und zeichnen Sie den Endzustand des Baums nach dem Löschvorgang. Falls Sie Zwischenschritte zeichnen, kennzeichnen Sie den Endzustand deutlich.

Lösungsvorschlag



- (ii) Löschen Sie aus dem (originalen) gegebenen B-Baum den Schlüssel 17 und zeichnen Sie den Endzustand des Baums nach dem Löschvorgang. Falls Sie Zwischenschritte zeichnen, kennzeichnen Sie den Endzustand deutlich.

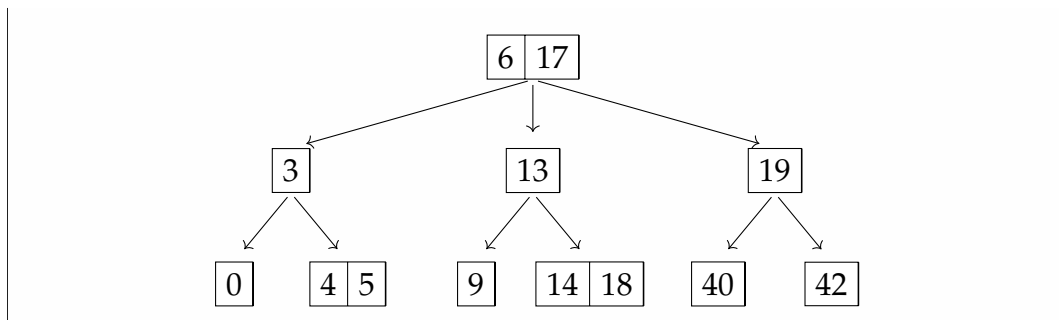
Lösungsvorschlag



- (iii) Löschen Sie aus dem (originalen) gegebenen B-Baum den Schlüssel 43 und zeichnen Sie den Endzustand des Baums nach dem Löschvorgang. Falls Sie Zwischenschritte zeichnen, kennzeichnen Sie den Endzustand deutlich.

Lösungsvorschlag





66115 / 2010 / Herbst / Thema 2 / Aufgabe 3

Gegeben sei ein Array der Größe 10, z. B. `int[] hashfeld = new int [10]`. Die Hashfunktion sei der Wert modulo 10, $h(x) = x \% 10$. Kollisionen werden mit linearer Verschiebung um 1 (modulo 10) gelöst.

`in(x)` bedeutet, dass die Zahl x eingefügt wird, `search(x)`, dass nach x gesucht wird mit den Antworten „ja“ bzw. „nein“ und `out(x)`, dass x gelöscht wird, sofern x gespeichert ist.

Es wird folgende Sequenz von Operationen auf ein anfangs leeres Array ausgeführt:

`in(19)`, `in(29)`, `in(39)`, `in(10)`, `out(29)`, `out(39)`, `search(29)`, `in(11)`, `in(17)`, `out(10)`, `in(2)`, `in(22)`

Geben Sie den Inhalt von `hashfeld` an

nach `search(29)`

nach `out(10)`

und nach `in(22)`.

66115 / 2012 / Herbst / Thema 2 / Aufgabe 7

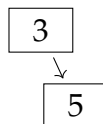
(a) Fügen Sie nacheinander die Zahlen 3, 5, 1, 2, 4

(i) in einen leeren binären Suchbaum ein

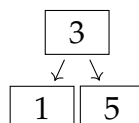
Nach dem Einfügen von „3“:



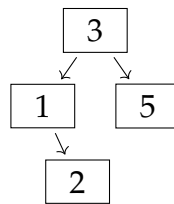
Nach dem Einfügen von „5“:



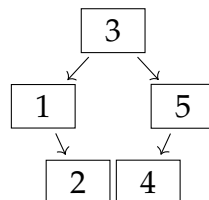
Nach dem Einfügen von „1“:



Nach dem Einfügen von „2“:

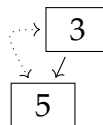


Nach dem Einfügen von „4“:

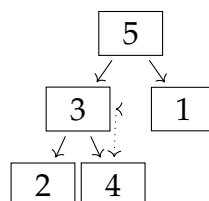


(ii) in einen leeren Heap ein

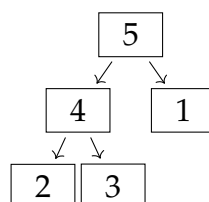
Erstellen einer Max.-Halde, einfügen von 3 und 5, Versickern notwendig:



Einfügen von 1 und 2 ohne Änderungen, Einfügen von 4, versickern notwendig:

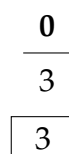


Fertiger Heap:

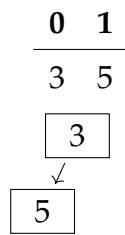


Ausführlicher als Max-Halde

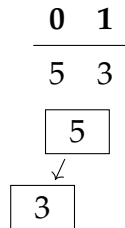
Nach dem Einfügen von „3“:



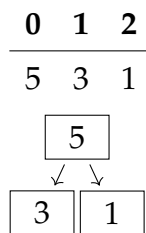
Nach dem Einfügen von „5“:



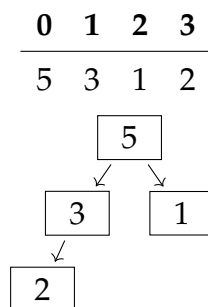
Nach dem Vertauschen von „5“ und „3“:



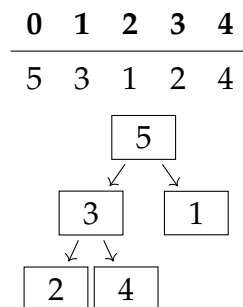
Nach dem Einfügen von „1“:



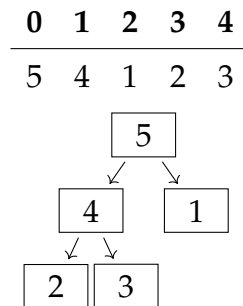
Nach dem Einfügen von „2“:



Nach dem Einfügen von „4“:

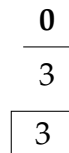


Nach dem Vertauschen von „4“ und „3“:

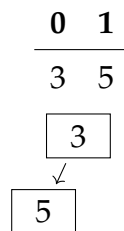


Ausführlicher als Min-Halbe

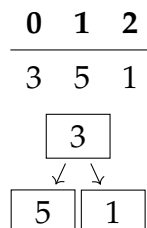
Nach dem Einfügen von „3“:



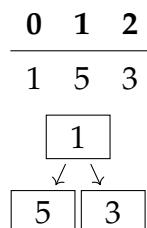
Nach dem Einfügen von „5“:



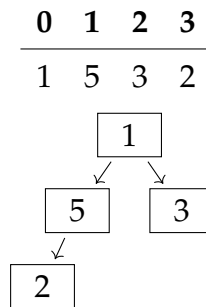
Nach dem Einfügen von „1“:



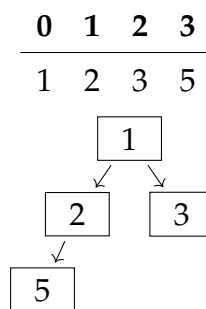
Nach dem Vertauschen von „1“ und „3“:



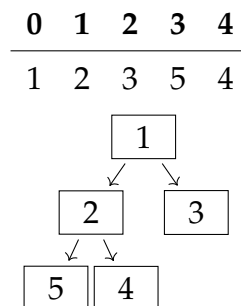
Nach dem Einfügen von „2“:



Nach dem Vertauschen von „2“ und „5“:



Nach dem Einfügen von „4“:



Geben Sie die Ergebnisse an (Zeichnung)

- (b) Geben Sie zwei Merkmale an, bei denen sich Heaps und binäre Suchbäume wesentlich unterscheiden. Ein wesentlicher Unterschied zwischen Bubblesort und Mergesort ist z. B. die *worst case* Laufzeit mit $\mathcal{O}(n^2)$ für Bubblesort und $\mathcal{O}(n \log n)$ für Mergesort.

Lösungsvorschlag

	Binärer Suchbaum	Heap
Suchen beliebiger Wert (worst case)	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$
Suchen Min-Max (average case)	$\mathcal{O}(\log(n))$	$\mathcal{O}(1)$

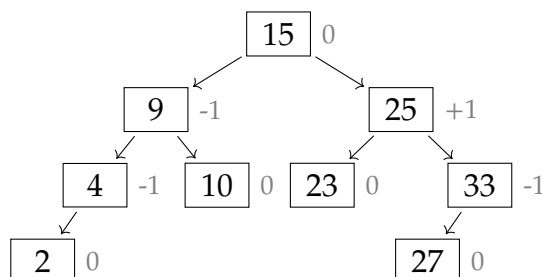
a

^a<https://cs.stackexchange.com/q/27860>

AVL-Baum

66115 / 2012 / Herbst / Thema 2 / Aufgabe 8

Gegeben sei der folgende AVL-Baum T . Führen Sie auf T folgende Operationen durch.



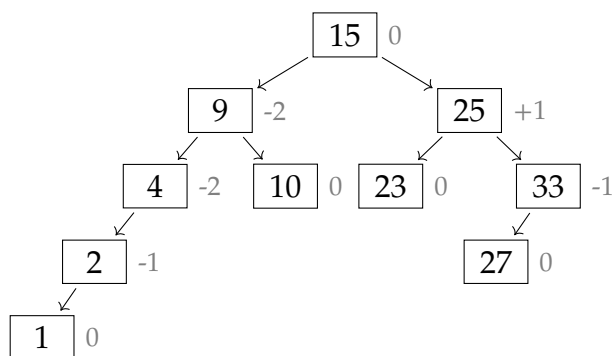
Lösungsvorschlag

Wir führen alle Operationen am Ursprungsbaum T durch und nicht am veränderten Baum.

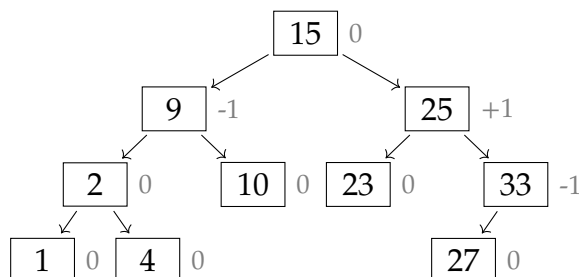
(a) Einfüge-Operationen:

- (i) Fügen Sie den Wert 1 in T ein. Balancieren Sie falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.

Nach dem Einfügen von „1“:

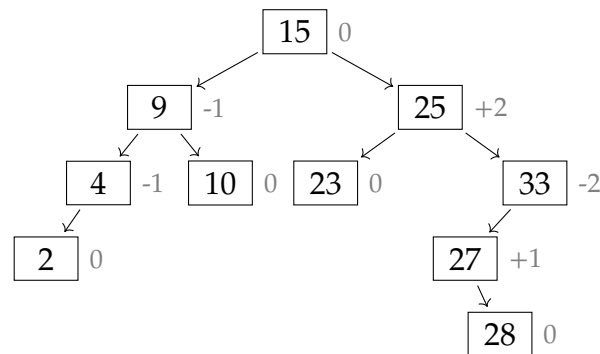


Nach der Rechtsrotation:

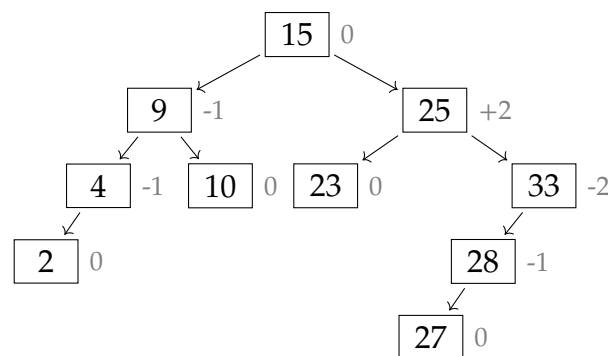


- (ii) Fügen Sie nun den Wert 28 in T ein. Balancieren Sie falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.

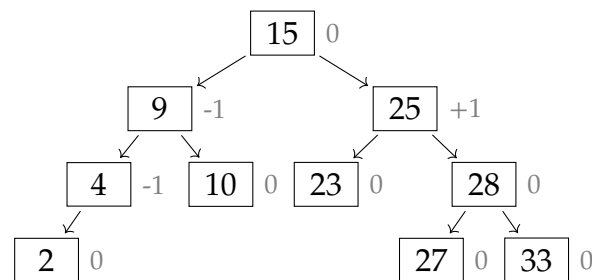
Nach dem Einfügen von „28“:



Nach der Linksrotation:

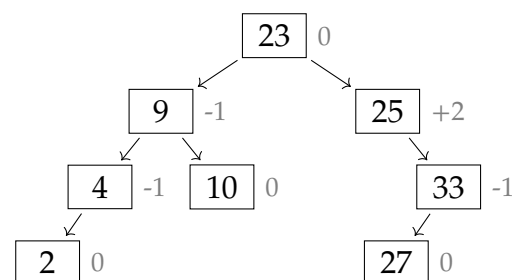


Nach der Rechtsrotation:

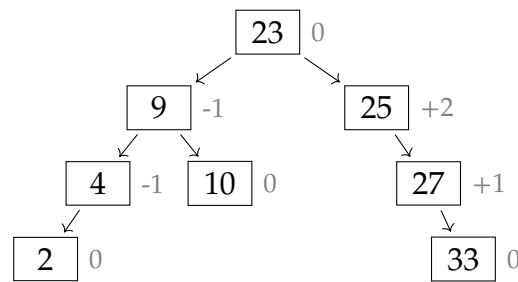


- (b) Löschen Sie aus T den Wert 15. Balancieren Sie falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.

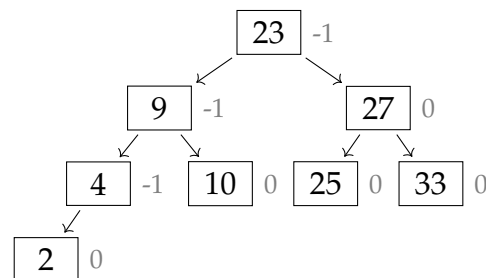
Nach dem Löschen von „15“:



Nach der Rechtsrotation:

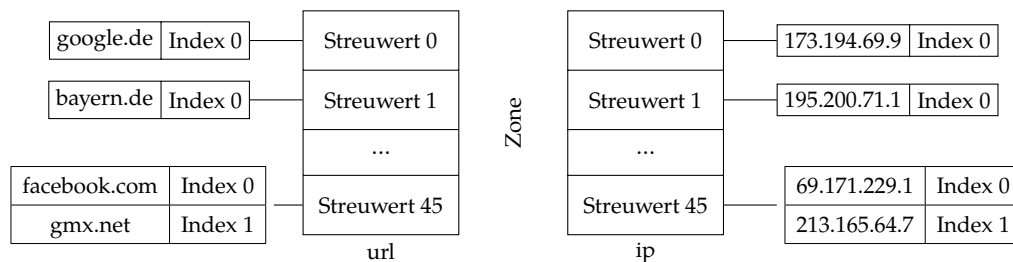


Nach der Linksrotation:



66115 / 2013 / Frühjahr / Thema 1 / Aufgabe 6

Um die URL (zum Beispiel google.de) und die zugehörige IP des Servers (hier 173.194.69.9) zu verwalten, werden Streutabellen verwendet, die eine bestimmte Zone von Adressen abbilden. Die Streutabellen werden als zwei dynamische Arrays (in Java: ArrayLists) realisiert. Kollisionen innerhalb einer Zone werden ebenfalls in dynamischen Arrays verwaltet.



Um zu einer URL die IP zu finden, berechnet man zunächst mittels der Funktion `hash()` den entsprechenden Streuwert, entnimmt dann den Index der Tabelle URL und sucht schließlich an entsprechender Stelle in der Tabelle IP die IP-Adresse.

- Erläutern Sie am vorgestellten Beispiel, wie ein Hash-Verfahren zum Speichern großer Datenmengen prinzipiell funktioniert und welche Voraussetzungen und Bedingungen daran geknüpft sind.
- Nun implementieren Sie Teile dieser IP- und URL-Verwaltung in einer objektorientierten Sprache Ihrer Wahl. Verwenden Sie dabei die folgende Klasse (die Vorgaben sind in der Sprache Java gehalten):

```
class Zone {
    private ArrayList<ArrayList<String>> urlList =
        new ArrayList<ArrayList<String>>();
    private ArrayList<ArrayList<String>> ipList =
        new ArrayList<ArrayList<String>>();
    public int hash(String url) { /* calculates hash-value h, >=0 */}
}
```

- (i) Prüfen Sie in einer Methode `boolean exists(int h)` der Klasse `Zone`, ob bereits mindestens ein Eintrag für einen gegebenen Streuwert vorhanden ist. Falls `h` größer ist als die derzeitige Größe der Streutabelle, existiert der Eintrag nicht.

Lösungsvorschlag

```
boolean exists(int h) {
    if (urlList.size() - 1 < h || ipList.size() - 1 < h)
        return false;

    ArrayList<String> urlCollisionList = urlList.get(h);
    ArrayList<String> ipCollisionList = ipList.get(h);
    if (urlCollisionList.size() == 0 || ipCollisionList.size() == 0)
        return false;

    return true;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2013/fruehjahr/Zone.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2013/fruehjahr/Zone.java)

- (ii) Die Methode `int getIndex (String url, ArrayList<String> urlList)` soll den Index einer URL in der Kollisionsliste berechnen. Ist die URL in der Kollisionsliste nicht vorhanden, soll `-1` zurückgeliefert werden.

Lösungsvorschlag

```
int getIndex(String url, ArrayList<String> urlList) {
    for (int i = 0; i < urlList.size(); i++) {
        if (urlList.get(i).equals(url))
            return i;
    }
    return -1;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2013/fruehjahr/Zone.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2013/fruehjahr/Zone.java)

- (iii) Ergänzen Sie die Klasse `Zone` um eine Methode `String lookup (String url)`, die in der Streutabelle die IP-Adresse zur `url` zurückgibt. Wird eine nicht vorhandene Adresse abgerufen, wird eine Fehlermeldung zurückgegeben.

Lösungsvorschlag

```
String lookup(String url) {
    int h = hash(url);
    int collisionIndex = getIndex(url, urlList.get(h));
    if (collisionIndex == -1)
        return "Die URL konnte nicht in der Tabelle gefunden werden";
}
```

```
return ipList.get(h).get(collisionIndex);  
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2013/fruehjahr/Zone.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2013/fruehjahr/Zone.java)

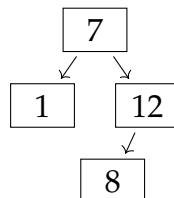
Binärbaum
Halde (Heap)

66115 / 2013 / Herbst / Thema 2 / Aufgabe 7

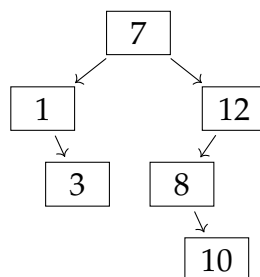
(a)

- (i) Fügen Sie nacheinander die Zahlen 7, 1, 12, 8, 10, 3, 5 in einen leeren binären Suchbaum ein und zeichnen Sie den Suchbaum nach „8“ und nach „3“.

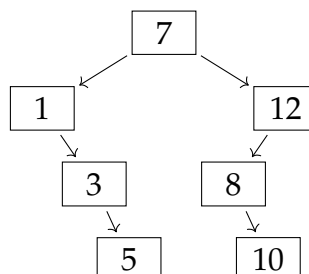
Nach dem Einfügen von „8“:



Nach dem Einfügen von „3“:

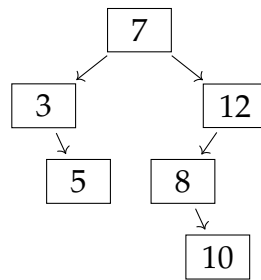


Nach dem Einfügen von „5“:



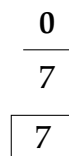
- (ii) Löschen Sie die „1“ aus dem in (i) erstellten Suchbaum und zeichnen Sie den Suchbaum.

Nach dem Löschen von „1“:

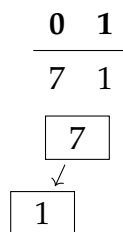


- (iii) Fügen Sie 7, 1, 12, 8, 10, 3, 5 in einen leeren MIN-Heap ein, der bzgl. „ \leq “ angeordnet ist. Geben Sie den Heap nach jedem Element an.

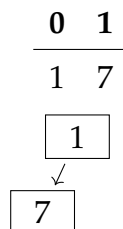
Nach dem Einfügen von „7“:



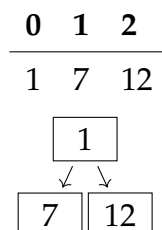
Nach dem Einfügen von „1“:



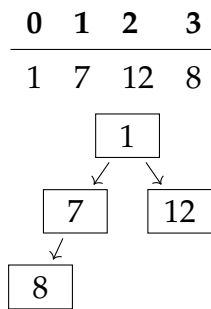
Nach dem Vertauschen von „1“ und „7“:



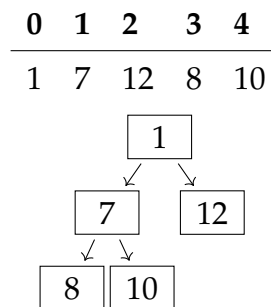
Nach dem Einfügen von „12“:



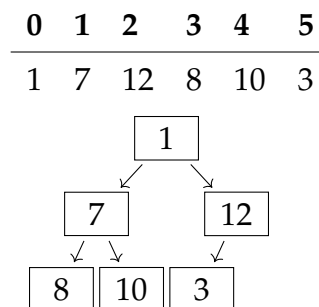
Nach dem Einfügen von „8“:



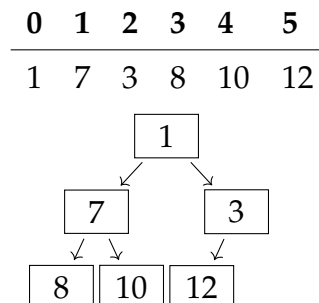
Nach dem Einfügen von „10“:



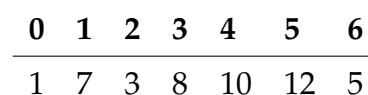
Nach dem Einfügen von „3“:

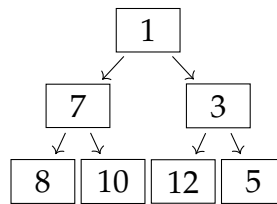


Nach dem Vertauschen von „3“ und „12“:



Nach dem Einfügen von „5“:





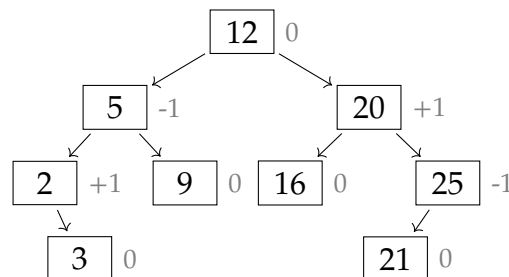
- (b) Was ist die worst-case Laufzeit in O-Notation für das Einfügen eines Elements in einen Heap der Größe n ? Begründen Sie ihre Antwort.

Lösungsvorschlag

Die worst-case Laufzeit berechnet sich aus dem Aufwand für das Durchsickern eines eingefügten Elementes. Da das Durchsickern entlang eines Pfades im Baum erfolgt, entspricht der Aufwand im ungünstigsten Fall der Höhe des Baumes, $\approx \log_2 n$. Insgesamt ergibt sich somit eine worst-case Laufzeit von $\mathcal{O}(\log n)$.

66115 / 2013 / Herbst / Thema 2 / Aufgabe 8

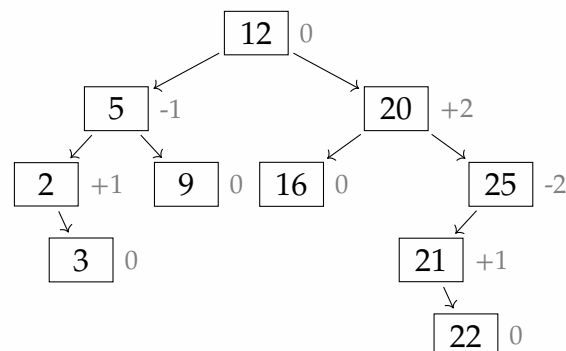
Gegeben sei der folgende AVL-Baum T . Führen Sie auf T folgende Operationen durch.



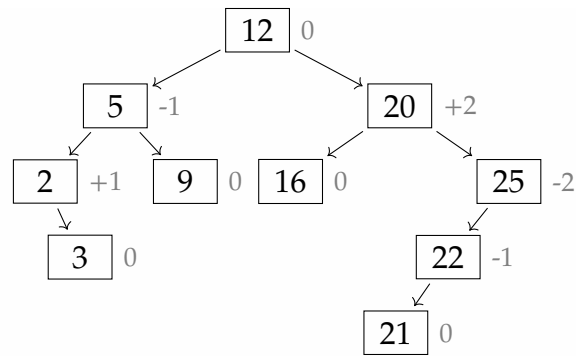
- (a) Fügen Sie den Wert 22 in T ein. Balancieren Sie falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.

Lösungsvorschlag

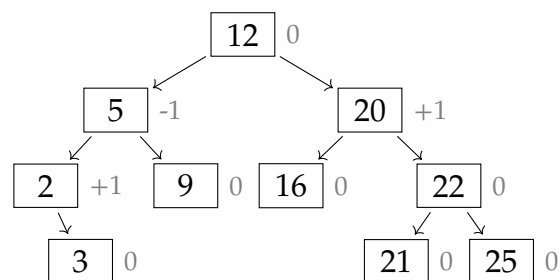
Nach dem Einfügen von „22“:



Nach der Linksrotation:



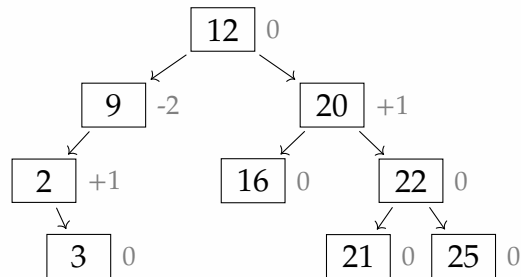
Nach der Rechtsrotation:



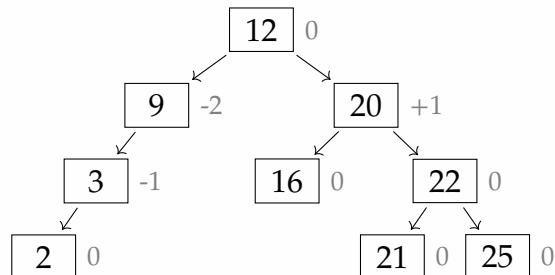
- (b) Löschen Sie danach die 5. Balancieren Sie T falls nötig und geben Sie den entstandenen Baum (als Zeichnung) an.

Lösungsvorschlag

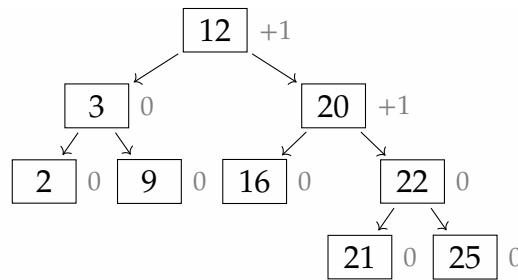
Nach dem Löschen von „5“:



Nach der Linksrotation:



Nach der Rechtsrotation:



66115 / 2014 / Frühjahr / Thema 1 / Aufgabe 2

Implementieren Sie in einer objekt-orientierten Sprache Ihrer Wahl eine Klasse namens `BinBaum`, deren Instanzen binäre Bäume mit ganzzahligen Datenknoten darstellen, nach folgender Spezifikation:

(a) Beginnen Sie zunächst mit der Datenstruktur selbst:

- Mit Hilfe des Konstruktors soll ein neuer Baum erstellt werden, der aus einem einzelnen Knoten besteht, in dem der dem Konstruktor als Parameter übergebene Wert (ganzzahlig, in Java z.B. `int`) gespeichert ist.

Lösungsvorschlag

```

class Knoten {
    int value;

    Knoten left;
    Knoten right;

    public Knoten(int value) {
        this.value = value;
    }
  
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java)

- Die Methoden `setLeft(int value)` bzw. `setRight(int value)` sollen den linken bzw. rechten Teilbaum des aktuellen Knotens durch einen jeweils neuen Teilbaum ersetzen, der seinerseits aus einem einzelnen Knoten besteht, in dem der übergebene Wert `value` gespeichert ist. Sie haben keinen Rückgabewert.

Lösungsvorschlag

```

public void setLeft(Knoten left) {
    this.left = left;
}

public void setRight(Knoten right) {
    this.right = right;
}
  
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java)

- Die Methoden `getLeft()` bzw. `getRight()` sollen den linken bzw. rechten Teilbaum zurückgeben (bzw. `null`, wenn keiner vorhanden ist)

Lösungsvorschlag

```
public Knoten getLeft() {
    return left;
}

public Knoten getRight() {
    return right;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java)

- Die Methode `int getValue()` soll den Wert zurückgeben, der im aktuellen Wurzelknoten gespeichert ist.

Lösungsvorschlag

```
public int getValue() {
    return value;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java)

- (b) Implementieren Sie nun die Methoden `preOrder()` bzw. `postOrder()`. Sie sollen die Knoten des Baumes mit Tiefensuche traversieren, die Werte dabei in pre-order bzw. post-order Reihenfolge in eine Liste (z.B. `List<Integer>`) speichern und diese Ergebnisliste zurückgeben. Die Tiefensuche soll dabei zuerst in den linken und dann in den rechten Teilbaum absteigen.

Lösungsvorschlag

```
void preOrder(Knoten knoten, List<Integer> list) {
    if (knoten != null) {
        list.add(knoten.getValue());
        preOrder(knoten.getLeft(), list);
        preOrder(knoten.getRight(), list);
    }
}

List<Integer> preOrder() {
    List<Integer> list = new ArrayList<>();
    preOrder(head, list);
    return list;
}

void postOrder(Knoten knoten, List<Integer> list) {
    if (knoten != null) {
        postOrder(knoten.getLeft(), list);
        postOrder(knoten.getRight(), list);
        list.add(knoten.getValue());
    }
}

List<Integer> postOrder() {
```

```

List<Integer> list = new ArrayList<>();
postOrder(head, list);
return list;
}

```

AVL-Baum

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java)

- (c) Ergänzen Sie schließlich die Methode `isSearchTree()`. Sie soll überprüfen, ob der Binärbaum die Eigenschaften eines binären Suchbaums erfüllt. Beachten Sie, dass die Laufzeit-Komplexität Ihrer Implementierung linear zur Anzahl der Knoten im Baum sein muss.

Lösungsvorschlag

```

boolean isSearchTree(Knoten knoten) {
    if (knoten == null) {
        return true;
    }

    if (knoten.getLeft() != null && knoten.getValue() <
        ↪ knoten.getLeft().getValue()) {
        return false;
    }

    if (knoten.getRight() != null && knoten.getValue() >
        ↪ knoten.getRight().getValue()) {
        return false;
    }

    return isSearchTree(knoten.getLeft()) && isSearchTree(knoten.getRight());
}

```

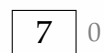
Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/BinBaum.java)

66115 / 2014 / Frühjahr / Thema 1 / Aufgabe 3

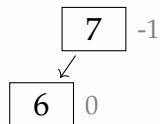
- (a) Fügen Sie die Zahlen (7, 6, 2, 1, 5, 3, 8, 4) in dieser Reihenfolge in einen anfangs leeren AVL Baum ein. Stellen Sie die AVL Eigenschaft ggf. nach jedem Einfügen mit geeigneten Rotationen wieder her. Zeichnen Sie den AVL Baum einmal vor und einmal nach jeder einzelnen Rotation.

Lösungsvorschlag

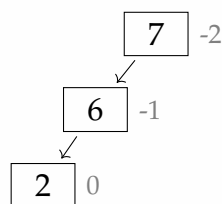
Nach dem Einfügen von „7“:



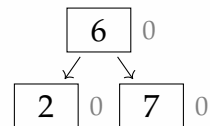
Nach dem Einfügen von „6“:



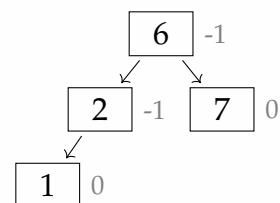
Nach dem Einfügen von „2“:



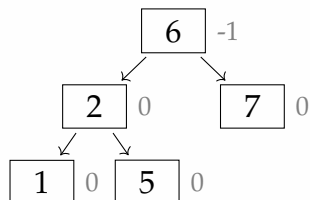
Nach der Rechtsrotation:



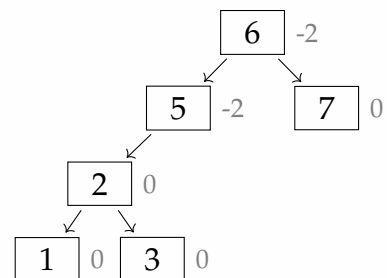
Nach dem Einfügen von „1“:



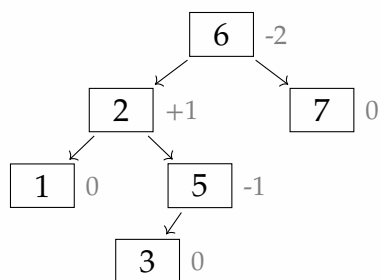
Nach dem Einfügen von „5“:



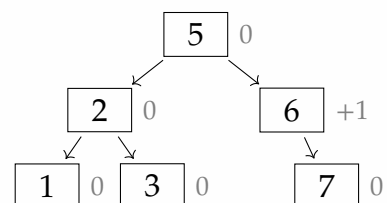
Nach der Linksrotation:



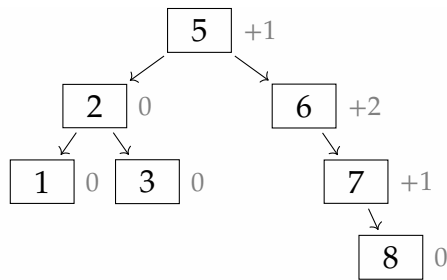
Nach dem Einfügen von „3“:



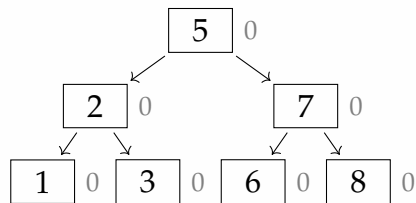
Nach der Rechtsrotation:



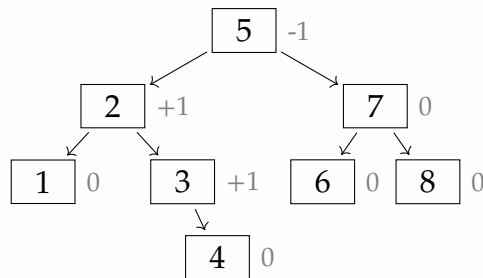
Nach dem Einfügen von „8“:



Nach der Linksrotation:

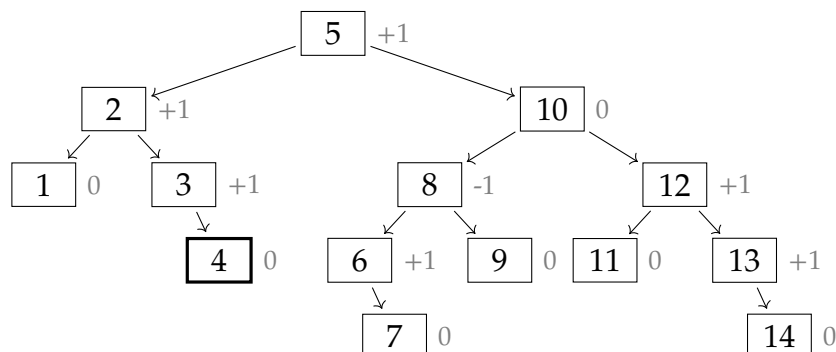


Nach dem Einfügen von „4“:



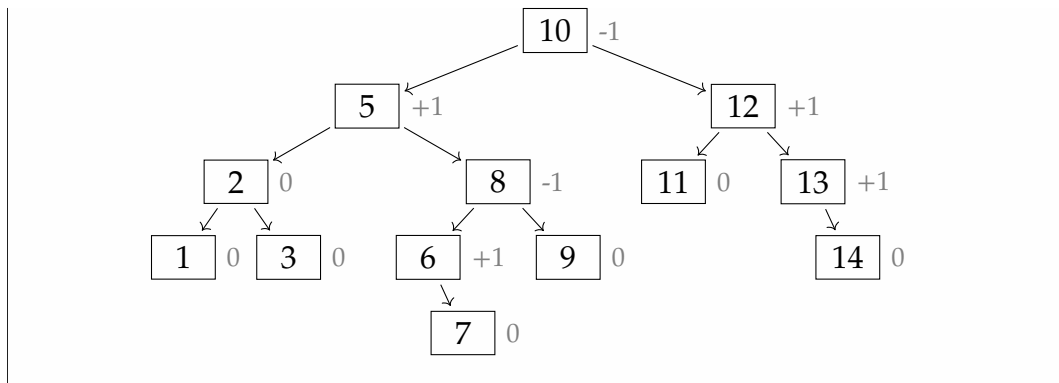
- (b) Entfernen Sie den jeweils markierten Knoten aus den folgenden AVL-Bäumen. Stellen Sie die AVL-Eigenschaft ggf. durch geeignete Rotationen wieder her. Zeichnen Sie nur den resultierenden Baum.

(i) Baum 1:

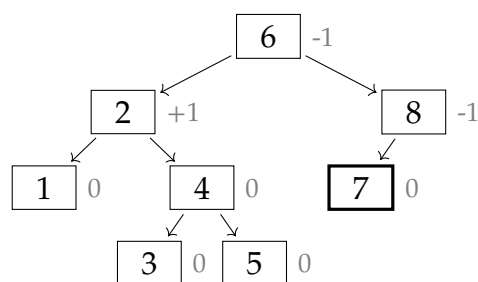


Lösungsvorschlag

Nach dem Löschen von „4“:

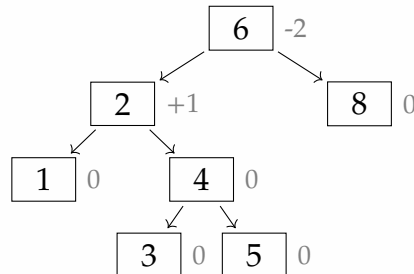


(ii) Baum 2:

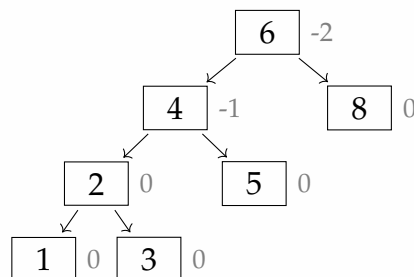


Lösungsvorschlag

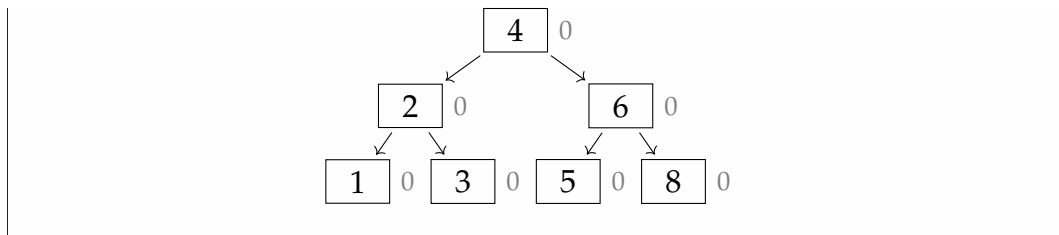
Nach dem Löschen von „7“:



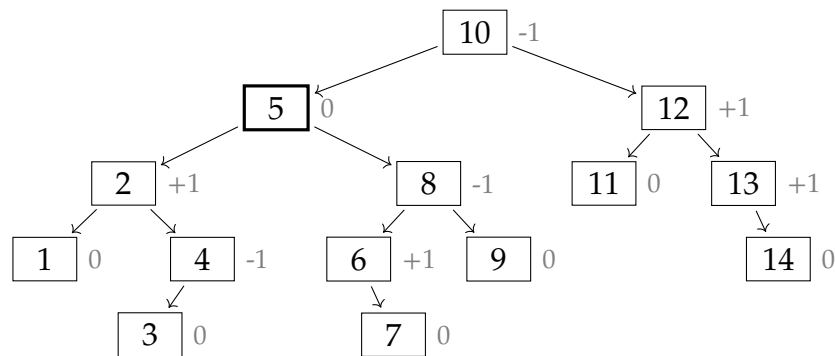
Nach der Linksrotation:



Nach der Rechtsrotation:

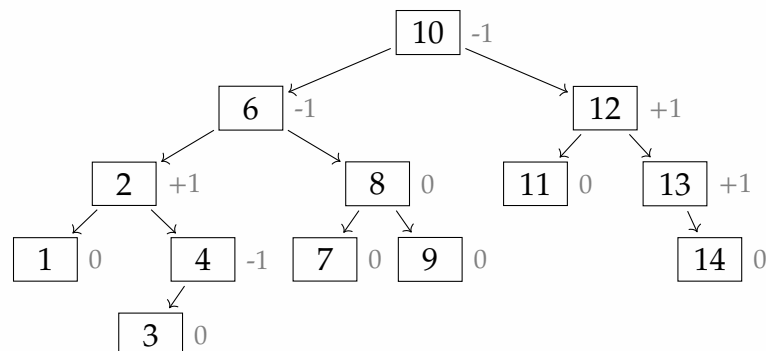


(iii) Baum 3:



Lösungsvorschlag

Nach dem Löschen von „5“:

**66115 / 2016 / Frühjahr / Thema 2 / Aufgabe 4**Betrachte eine Hashtabelle der Größe $m = 10$.

(a) Welche der folgenden Hashfunktionen ist für Hashing mit verketteten Listen am besten geeignet? Begründen Sie Ihre Wahl!

(i) $h_1(x) = (4x + 3) \bmod m$

Lösungsvorschlag

- 1 $h_1(1) = (4 \cdot 1 + 3) \bmod 10 = 7$
- 2 $h_1(2) = (4 \cdot 2 + 3) \bmod 10 = 1$
- 3 $h_1(3) = (4 \cdot 3 + 3) \bmod 10 = 5$
- 4 $h_1(4) = (4 \cdot 4 + 3) \bmod 10 = 9$
- 5 $h_1(5) = (4 \cdot 5 + 3) \bmod 10 = 3$

$$\begin{aligned}
6 \quad & h_1(6) = (4 \cdot 6 + 3) \bmod 10 = 7 \\
7 \quad & h_1(7) = (4 \cdot 7 + 3) \bmod 10 = 1 \\
8 \quad & h_1(8) = (4 \cdot 8 + 3) \bmod 10 = 5 \\
9 \quad & h_1(9) = (4 \cdot 9 + 3) \bmod 10 = 9 \\
10 \quad & h_1(10) = (4 \cdot 10 + 3) \bmod 10 = 3
\end{aligned}$$

(ii) $h_2(x) = (3x + 3) \bmod m$

Lösungsvorschlag

$$\begin{aligned}
1 \quad & h_2(1) = (3 \cdot 1 + 3) \bmod 10 = 6 \\
2 \quad & h_2(2) = (3 \cdot 2 + 3) \bmod 10 = 9 \\
3 \quad & h_2(3) = (3 \cdot 3 + 3) \bmod 10 = 2 \\
4 \quad & h_2(4) = (3 \cdot 4 + 3) \bmod 10 = 5 \\
5 \quad & h_2(5) = (3 \cdot 5 + 3) \bmod 10 = 8 \\
6 \quad & h_2(6) = (3 \cdot 6 + 3) \bmod 10 = 1 \\
7 \quad & h_2(7) = (3 \cdot 7 + 3) \bmod 10 = 4 \\
8 \quad & h_2(8) = (3 \cdot 8 + 3) \bmod 10 = 7 \\
9 \quad & h_2(9) = (3 \cdot 9 + 3) \bmod 10 = 0 \\
10 \quad & h_2(10) = (3 \cdot 10 + 3) \bmod 10 = 3
\end{aligned}$$

Lösungsvorschlag

Damit die verketteten Listen möglichst klein bleiben, ist eine möglichst gleichmäßige Verteilung der Schlüssel in die Buckets anzustreben. h_2 ist dafür besser geeignet als h_1 , da h_2 in alle Buckets Schlüssel ablegt, h_1 jedoch nur in Buckets mit ungerader Zahl.

(b) Welche der folgenden Hashfunktionen ist für Hashing mit offener Adressierung am besten geeignet? Begründen Sie Ihre Wahl!

(i) $h_1(x, i) = (7 \cdot x + i \cdot m) \bmod m$

(ii) $h_2(x, i) = (7 \cdot x + i \cdot (m - 1)) \bmod m$

Lösungsvorschlag

$h_2(x, i)$ ist besser geeignet. h_1 sondiert immer im selben Bucket, $(i \cdot m) \bmod m$ heben sich gegenseitig auf, zum Beispiel ergibt:

$$\begin{aligned}
- \quad & h_1(3, 0) = (7 \cdot 3 + 0 \cdot 10) \bmod 10 = 1 \\
- \quad & h_1(3, 1) = (7 \cdot 3 + 1 \cdot 10) \bmod 10 = 1 \\
- \quad & h_1(3, 2) = (7 \cdot 3 + 2 \cdot 10) \bmod 10 = 1
\end{aligned}$$

Während hingegen h_2 verschiedene Buckets belegt.

$$- \quad h_2(3, 0) = (7 \cdot 3 + 0 \cdot 9) \bmod 10 = 1$$

$$\begin{aligned}
 - h_2(3,1) &= (7 \cdot 3 + 1 \cdot 9) \bmod 10 = 0 \\
 - h_2(3,2) &= (7 \cdot 3 + 2 \cdot 9) \bmod 10 = 9
 \end{aligned}$$

66115 / 2016 / Frühjahr / Thema 2 / Aufgabe 7

Wofür eignen sich die folgenden Baum-Datenstrukturen im Vergleich zu den anderen angeführten Baumstrukturen am besten, und warum. Sprechen Sie auch die Komplexität der wesentlichen Operationen und die Art der Speicherung an.

(a) Rot-Schwarz-Baum

Lösungsvorschlag

Einfügen (Zeitkomplexität)

$\mathcal{O}(\log n)$ (im Durchschnitt)

$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)

$\mathcal{O}(\log n)$ (im Durchschnitt)

$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)

$\mathcal{O}(\log n)$ (im Durchschnitt)

$\mathcal{O}(\log n)$ (im schlechtesten Fall) ^a

^atutorialspoint.com

(b) AVL-Baum

Lösungsvorschlag

Einfügen (Zeitkomplexität)

$\mathcal{O}(\log_2 n)$ (im Durchschnitt)

$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)

$\mathcal{O}(\log_2 n)$ (im Durchschnitt)

$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)

$\mathcal{O}(\log_2 n)$ (im Durchschnitt)

$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall) ^a

^atutorialspoint.com

(c) Binärer-Heap

Lösungsvorschlag

Verwendungszweck zum effizienten Sortieren von Elementen. ^a

Einfügen (Zeitkomplexität)

$\mathcal{O}(1)$ (im Durchschnitt)

$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität) $\mathcal{O}(\log n)$ (im Durchschnitt) $\mathcal{O}(\log n)$ (im schlechtesten Fall)**Suchen (Zeitkomplexität)** $\mathcal{O}(n)$ (im Durchschnitt) $\mathcal{O}(n)$ (im schlechtesten Fall) ^b^adeut. Wikipedia^bengl. WikipediaB-Baum
R-Baum
AVL-Baum
Halde (Heap)
Binärbaum

(d) B-Baum

Lösungsvorschlag

Einfügen (Zeitkomplexität) $\mathcal{O}(\log n)$ (im Durchschnitt) $\mathcal{O}(\log n)$ (im schlechtesten Fall)**Löschen (Zeitkomplexität)** $\mathcal{O}(\log n)$ (im Durchschnitt) $\mathcal{O}(\log n)$ (im schlechtesten Fall)**Suchen (Zeitkomplexität)** $\mathcal{O}(\log n)$ (im Durchschnitt) $\mathcal{O}(\log n)$ (im schlechtesten Fall) ^a^atutorialspoint.com

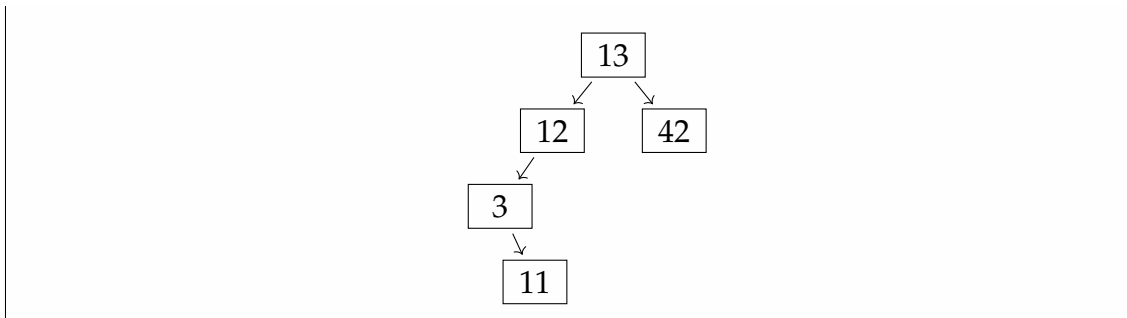
(e) R-Baum

Lösungsvorschlag

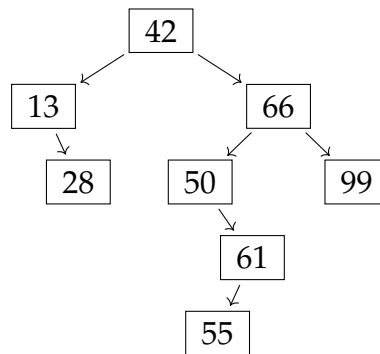
Verwendungszweck Ein R-Baum erlaubt die schnelle Suche in mehrdimensionalen ausgedehnten Objekten. ^a**Suchen (Zeitkomplexität)** $\mathcal{O}(\log_M n)$ (im Durchschnitt) ^b $\mathcal{O}(n)$ (im schlechtesten Fall) ^c^adeut. Wikipedia^beng. Wikipedia^cSimon Fraser University, Burnaby, Kanada**66115 / 2017 / Herbst / Thema 2 / Aufgabe 8**

- (a) Fügen Sie die Zahlen 13, 12, 42, 3, 11 in der gegebenen Reihenfolge in einen zunächst leeren binären Suchbaum mit aufsteigender Sortierung ein. Stellen Sie nur das Endergebnis dar.

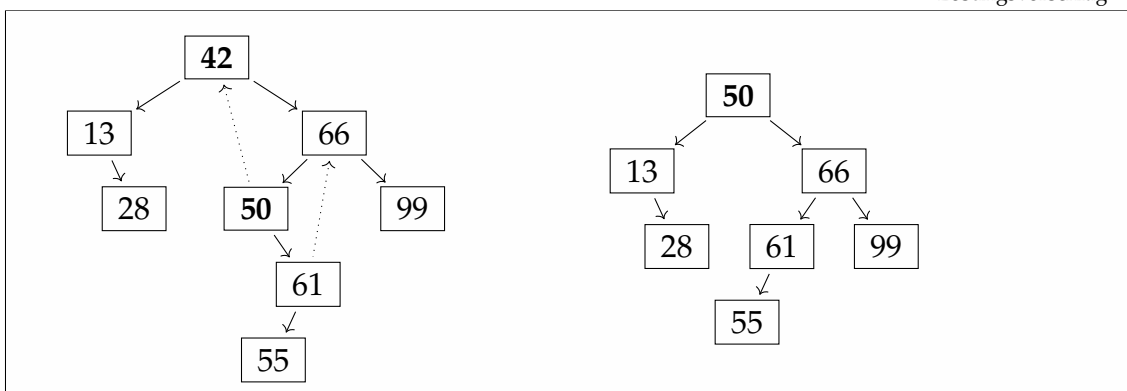
Lösungsvorschlag



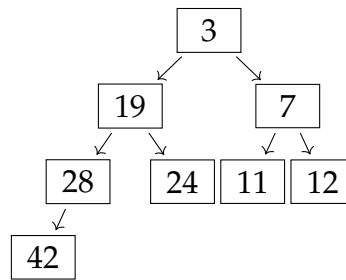
- (b) Löschen Sie den Wurzelknoten mit Wert 42 aus dem folgenden *binären* Suchbaum mit aufsteigender Sortierung und ersetzen Sie ihn dabei durch einen geeigneten Wert aus dem *rechten* Teilbaum. Lassen Sie möglichst viele Teilbäume unverändert und erhalten Sie die Suchbaumeigenschaft.



Lösungsvorschlag

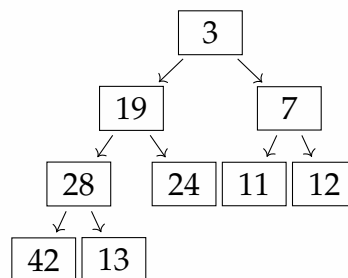


- (c) Fügen Sie einen neuen Knoten mit dem Wert 13 in die folgende Min-Halde ein und stellen Sie anschließend die Halden-Eigenschaft vom neuen Blatt aus beginnend wieder her, wobei möglichst viele Knoten der Halde unverändert bleiben und die Halde zu jedem Zeitpunkt links-vollständig sein soll. Geben Sie nur das Endergebnis an.

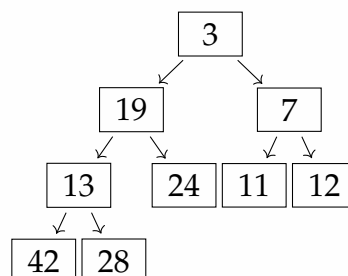


Lösungsvorschlag

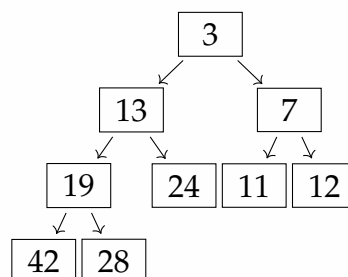
Nach dem Einfügen von „13“:



Nach dem Vertauschen von „13“ und „28“:



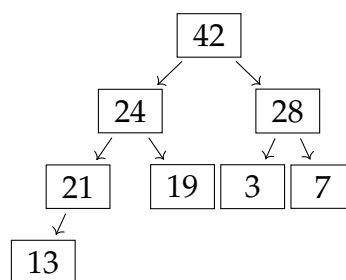
Nach dem Vertauschen von „13“ und „19“:



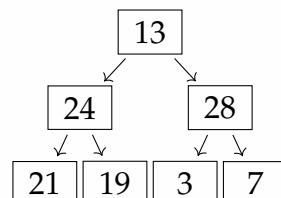
- (d) Geben Sie für die ursprüngliche Min-Halde aus Teilaufgabe c) (ööhne den neu eingefügten Knoten mit dem Wert 13) die Feld-Einbettung (Array-Darstellung) an.

0	1	2	3	4	5	6	7
3	19	7	28	24	11	12	42

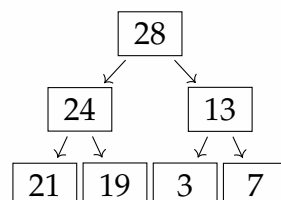
- (e) Löschen Sie den Wurzelknoten mit Wert 42 aus der folgenden Max-Halde und stellen Sie anschließend die Halden-Eigenschaft ausgehend von einer neuen Wurzel wieder her, wobei möglichst viele Knoten der Halde unverändert bleiben und die Halde zu jedem Zeitpunkt links-vollständig sein soll. Geben Sie nur das Endergebnis an.



Nach dem Ersetzen von „42“ mit „13“:

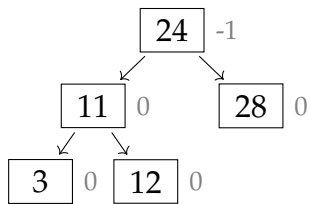


Nach dem Vertauschen von „13“ und „28“:



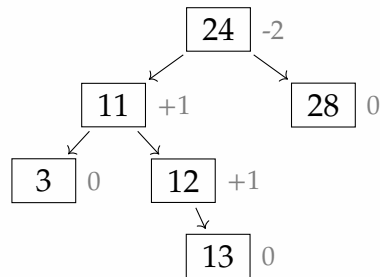
- (f) Fügen Sie in jeden der folgenden AVL-Bäume mit aufsteigender Sortierung jeweils einen neuen Knoten mit dem Wert 13 ein und führen Sie anschließend bei Bedarf die erforderliche(n) Rotation(en) durch. Zeichnen Sie den Baum vor und nach den Rotationen.

(i) AVL-Baum A

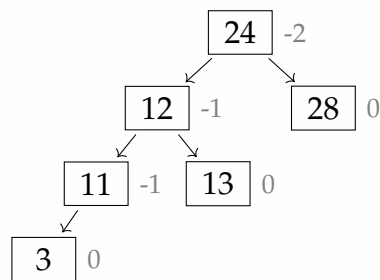


Lösungsvorschlag

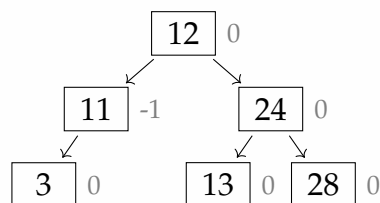
Nach dem Einfügen von „13“:



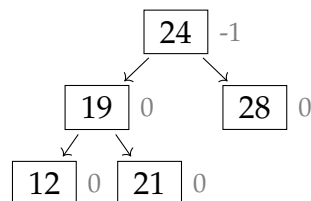
Nach der Linksrotation:



Nach der Rechtsrotation:

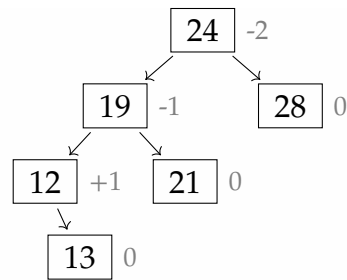


(ii) AVL-Baum B

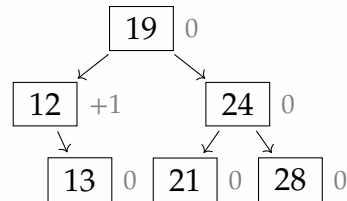


Lösungsvorschlag

Nach dem Einfügen von „13“:



Nach der Rechtsrotation:

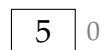


66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 8

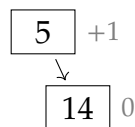
Bearbeiten Sie folgende Aufgaben zu AVL-Suchbäumen. Geben Sie jeweils bei jeder einzelnen Operation zum Einfügen, Löschen, sowie jeder elementaren Operation zum Wiederherstellen der AVL-Baumeigenschaften den entstehenden Baum als Baumzeichnung an. Geben Sie zur Darstellung der elementaren Operation auch vorübergehend ungültige AVL-Bäume an und stellen Sie Doppelrotationen in zwei Schritten dar. Dabei sollen die durchgeführten Operationen klar gekennzeichnet sein und die Baumknoten immer mit aktuellen Balancewerten versehen sein.

- (a) Fügen Sie (manuell) nacheinander die Zahlen 5, 14, 28, 10, 3, 12, 13 in einen anfangs leeren AVL-Baum ein.

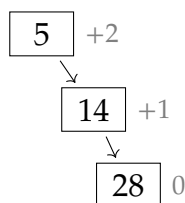
Einfügen von „5“:



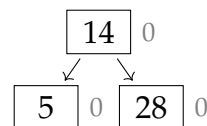
Einfügen von „14“:



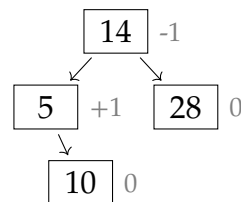
Einfügen von „28“:



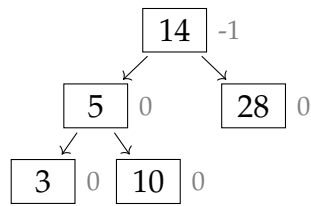
Linksrotation:



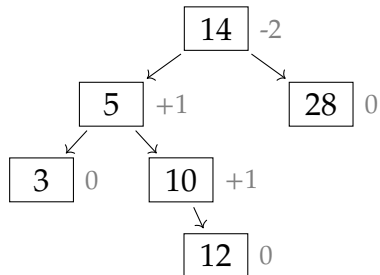
Einfügen von „10“:



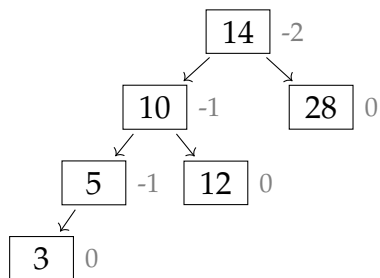
Einfügen von „3“:



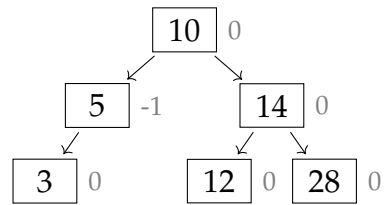
Einfügen von „12“:



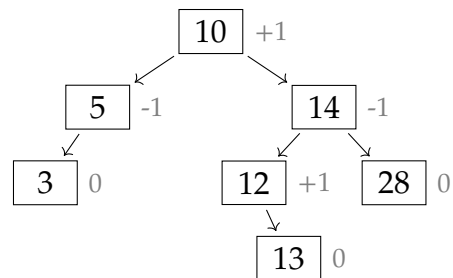
Linksrotation:



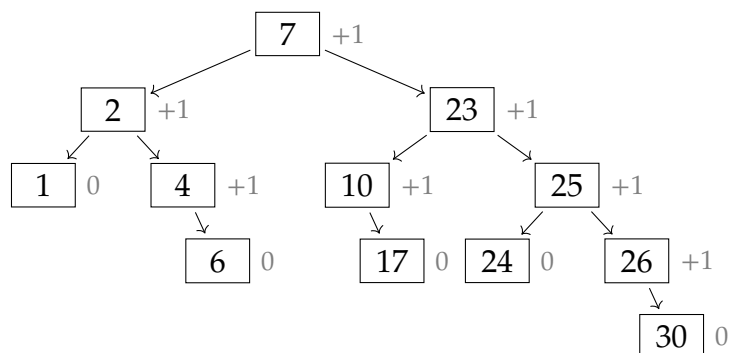
Rechtsrotation:



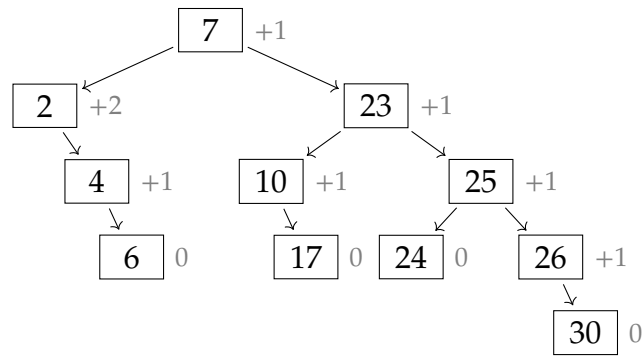
Einfügen von „13“:



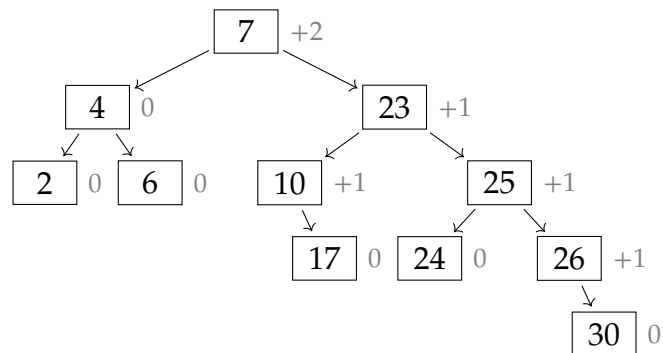
- (b) Gegeben sei folgender AVL-Baum. Löschen Sie nacheinander die Knoten 1 und 23. Bei Wahlmöglichkeiten nehmen Sie jeweils den kleineren Wert anstatt eines größeren.



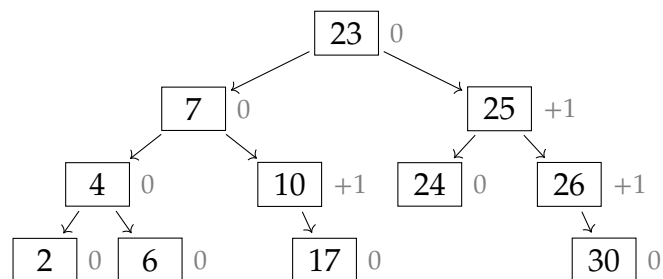
Löschen von „1“:



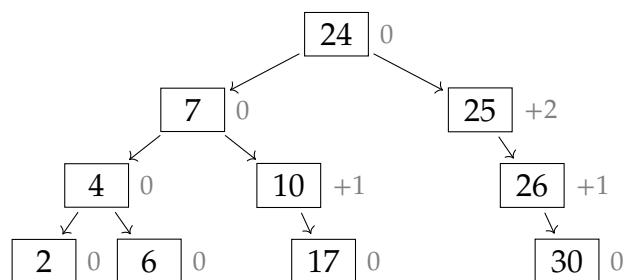
Linksrotation:



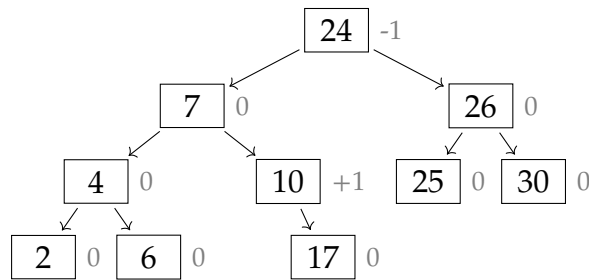
Linksrotation:



Löschen von „23“:



Linksrotation:



66115 / 2018 / Herbst / Thema 1 / Aufgabe 5

Hinweis: Wir betrachten in dieser Aufgabe binäre Suchbäume, bei denen jeder innere Knoten genau zwei Kinder hat. Schlüssel werden nur in den inneren Knoten gespeichert - die Blätter speichern keinerlei Informationen.

- (a) Welche Eigenschaften muss ein binärer Suchbaum haben, damit er ein AVL-Baum ist?

Lösungsvorschlag

Er muss die zusätzliche Eigenschaft haben, dass sich an jedem Knoten die Höhe der beiden Teilbäume um höchstens eins unterscheidet

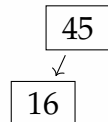
- (b) Mit $n(h)$ bezeichnen wir die minimale Anzahl innerer Knoten eines AVL-Baums der Höhe h .
- Begründen Sie, dass $n(1) = 1$ und $n(2) = 2$.
 - Begründen Sie, dass $n(h) = 1 + n(h-1) + n(h-2)$.
 - Folgern Sie, dass $n(h) > 2^{\frac{h}{2}-1}$.
- (c) Warum ist die Höhe jedes AVL-Baums mit n inneren Knoten $O(\log n)$?
- (d) Fügen Sie die Elemente (45, 16, 79, 31, 51, 87, 49, 61) in der angegebenen Reihenfolge in einen anfangs leeren binären Suchbaum ein (ohne Rebalancierungen). Zeichnen Sie den resultierenden Suchbaum nach jeder Einfügeoperation.

Lösungsvorschlag

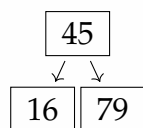
Einfügen von „45“:



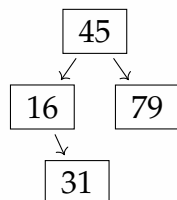
Einfügen von „16“:



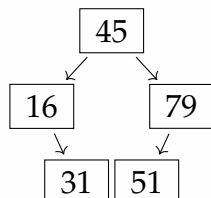
Einfügen von „79“:



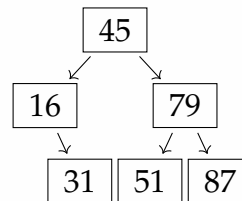
Einfügen von „31“:



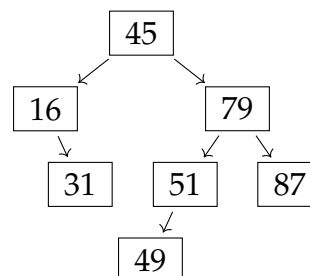
Einfügen von „51“:



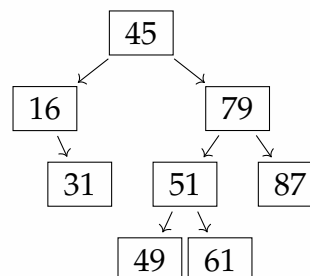
Einfügen von „87“:



Einfügen von „49“:



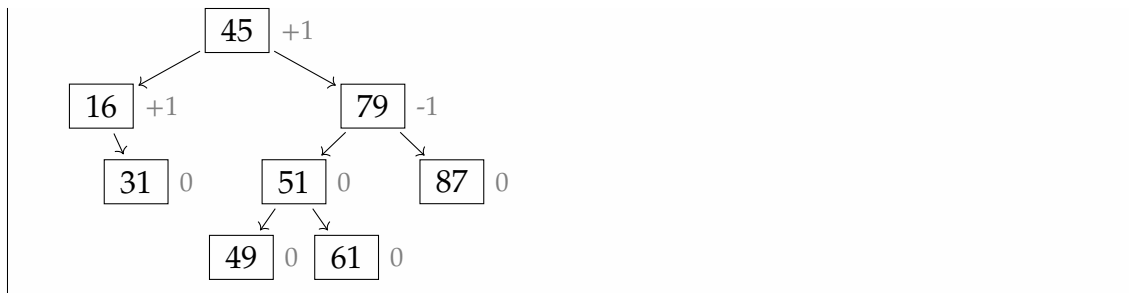
Einfügen von „61“:



- (e) Ist der resultierende Suchbaum aus Teilaufgabe 5.4 ein AVL-Baum? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Ja, wie in der untenstehenden Grafik zu sehen ist, unterscheiden sich die Höhe der Teilbäume von allen Knoten nur um höchstens eins.



- (f) Das Einfügen in einen AVL-Baum funktioniert (zunächst) wie beim binären Suchbaum durch Erweitern eines äußeren Knotens w:

vor dem Einfügen von 54 nach dem Einfügen von 54

Anschließend wird die AVL-Baum Eigenschaft (falls notwendig) durch eine (Doppel-)Rotation wiederhergestellt: Wenn z der erste Knoten auf dem Pfad P von w zur Wurzel ist, der nicht balanciert ist, y das Kind von z auf P und x das Kind von y auf P , und wenn (a, b, c) die Inorder-Reihenfolge von x, y, z ist, dann führen wir die Rotation aus, die benötigt wird, um b zum obersten Knoten der drei zu machen.

Die folgende Illustration zeigt den Fall, dass $\text{key}(y) < \text{key}(x) < \text{key}(z)$, $\delta(a, b, c) = (y, x, z)$, wobei w ein Knoten in T_y ist.

Sei h die Höhe des Teilbaums T_z . Für $i = 0, 1, 2$ sei h_i die Höhe des Teilbaums T_i und für $v = 2, y, z$ sei h_v die Höhe des Teilbaums mit der Wurzel v vor der Restrukturierung. Begründen Sie, dass

- (i) $h_0 = h$
 - (ii) $h_1 = h - 1$
 - (iii) $h_2 = h$
 - (iv) $h_x = h + 1$
 - (v) $h_y = h + 2$
 - (vi) $h_z = h + 3$
- (g) Welche Höhe haben die Teilbäume mit den Wurzeln x, y, z nach der Restrukturierung? Begründen Sie Ihre Antworten.
- (h) Begründen Sie, dass die oben gezeigte Doppelrotation die AVL-Baum-Eigenschaft wiederherstellt.
- (i) Beschreiben Sie, wie ein binärer Baum der Höhe h in einem Array repräsentiert werden kann. Wie viel Speicherplatz ist für so eine Darstellung erforderlich?
- (j) Warum verwendet man bei der Implementierung von AVL-Bäumen eine verzeigte Struktur und nicht eine Array-basierte Repräsentation?

66115 / 2019 / Frühjahr / Thema 2 / Aufgabe 1

Gegeben sei eine unsortierte Liste von n verschiedenen natürlichen Zahlen. Das k -kleinste Element ist das Element, das größer als genau $k - 1$ Elemente der Liste ist.

- (a) Geben Sie einen Algorithmus mit Laufzeit $\mathcal{O}(n \cdot \log n)$ an, um das k -kleinste Element zu berechnen.

Lösungsvorschlag

a

^a<https://en.wikipedia.org/wiki/Quickselect>

- (b) Gegeben sei nun ein Algorithmus A , der den Median einer unsortierten Liste von n Zahlen in $\mathcal{O}(n)$ Schritten berechnet. Nutzen Sie Algorithmus A um einen Algorithmus B anzugeben, welcher das k -kleinste Element in $\mathcal{O}(n)$ Schritten berechnet. Argumentieren Sie auch, dass der Algorithmus die gewünschte Laufzeit besitzt.

Lösungsvorschlag

a

^ahttps://en.wikipedia.org/wiki/Median_of_medians

- (c) Geben Sie einen Algorithmus an, der für alle $i = 1 \dots, \lfloor n/k \rfloor$ das $i \cdot k$ -kleinste Element berechnet. Die Laufzeit Ihres Algorithmus sollte $\mathcal{O}(n \cdot \log(n/k))$ sein. Sie dürfen weiterhin Algorithmus A , wie in Teilaufgabe (b) beschrieben, nutzen.

66115 / 2019 / Herbst / Thema 1 / Aufgabe 7

Gegeben sei die folgende Realisierung von binären Bäumen (in einer an Java angelehnten Notation):

```
class Node {
    Node left, right;
    int value;
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/herbst/Node.java

- (a) Beschreiben Sie in möglichst wenigen Worten, was die folgende Methode `foo` auf einem nicht-leeren binären Baum berechnet.

```
int foo(Node node) {
    int b = node.value;
    if (b < 0) {
        b = -1 * b;
    }
    if (node.left != null) {
        b = b + foo(node.left);
    }
    if (node.right != null) {
        b = b + foo(node.right);
    }
    return b;
}
```

Lösungsvorschlag

Die Methode `foo(Node node)` berechnet die Summe aller Knoten des Unterbaums des als Parameter übergebenen Knotens `node`. Der Schlüsselwert des Knotens `node` selbst wird in die Summenberechnung mit einbezogen.

- (b) Die Laufzeit der Methode `foo(tree)` ist linear in n , der Anzahl von Knoten im übergebenen Baum `tree`. Begründen Sie kurz, warum `foo(tree)` eine lineare Laufzeit hat.

Lösungsvorschlag

Die Methode `foo(Node node)` wird pro Knoten des Baums `tree` genau einmal aufgerufen. Es handelt sich um eine rekursive Methode, die auf den linken und rechten Kindknoten aufgerufen wird. Hat ein Knoten keine Kinder mehr, dann wird die Methode nicht aufgerufen.

- (c) Betrachten Sie den folgenden Algorithmus für nicht-leere, binäre Bäume. Beschreiben Sie in möglichst wenigen Worten die Wirkung der Methode `magic(tree)`. Welche Rolle spielt dabei die Methode `max`?

```
void magic(Node node) {
    Node m = max(node);
    if (m.value > node.value) {
        // Werte von m und node vertauschen
        int tmp = m.value;
        m.value = node.value;
        node.value = tmp;
    }
    if (node.left != null)
        magic(node.left);
    if (node.right != null)
        magic(node.right);
}

Node max(Node node) {
    Node max = node;
    if (node.left != null) {
        Node tmp = max(node.left);
        if (tmp.value > max.value)
            max = tmp;
    }
    if (node.right != null) {
        Node tmp = max(node.right);
        if (tmp.value > max.value)
            max = tmp;
    }
    return max;
}
```

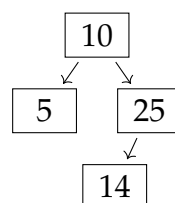
Methode `magic(tree)` vertauscht für jeden Knoten des Unterbaums den Wert mit dem Maximal-Knoten. Die Methode `max` liefert dabei den Knoten mit der größten Schlüsselwert im Unterbaum des übergebenen Knoten (sich selbst eingeschlossen).

- (d) Geben Sie in Abhängigkeit von n , der Anzahl von Knoten im übergebenen Baum `tree`, jeweils eine Rekursionsgleichung für die asymptotische Best-Case-Laufzeit ($B(n)$) und Worst-Case-Laufzeit ($W(n)$) des Aufrufs `magic(tree)` sowie die entsprechende Komplexitätsklasse (Θ) an. Begründen Sie Ihre Antwort.

Hinweis: Überlegen Sie, ob die Struktur des übergebenen Baumes Einfluss auf die Laufzeit hat. Die lineare Laufzeit von `max(t)` in der Anzahl der Knoten des Baumes `t` darf vorausgesetzt werden.

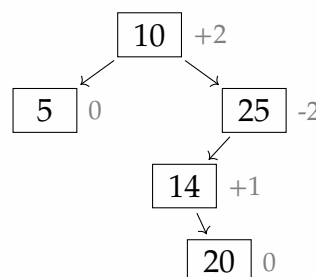
66115 / 2019 / Herbst / Thema 2 / Aufgabe 7

Fügen Sie (manuell) nacheinander die Zahlen 20, 31, 2, 17, 7 in folgenden AVL-Baum ein. Löschen Sie anschließend aus dem entstandenen Baum nacheinander 14 und 25.

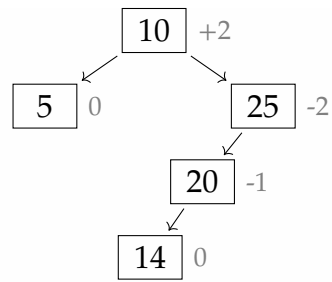


Zeichnen Sie jeweils direkt nach jeder einzelnen Operation zum Einfügen oder Löschen eines Knotens, sowie nach jeder elementaren Rotation den entstehenden Baum. Insbesondere sind evtl. anfallende Doppelrotationen in zwei Schritten darzustellen. Geben Sie zudem an jedem Knoten die Balancewerte an.

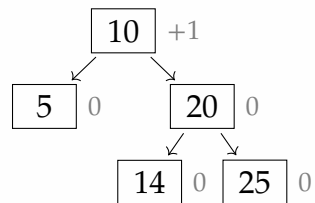
Nach dem Einfügen von „20“:



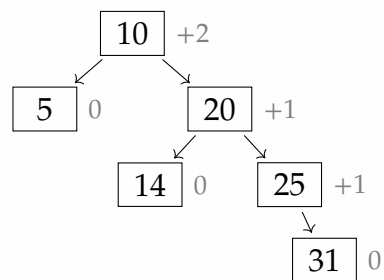
Nach der Linksrotation:



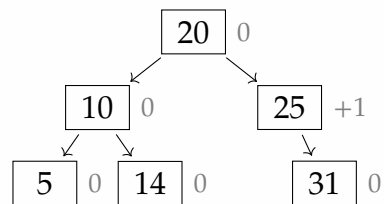
Nach der Rechtsrotation:



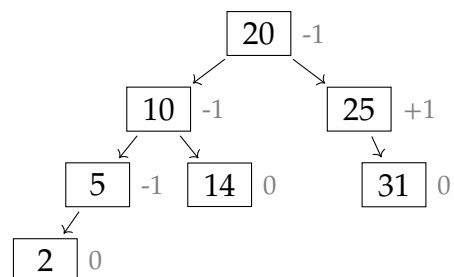
Nach dem Einfügen von „31“:



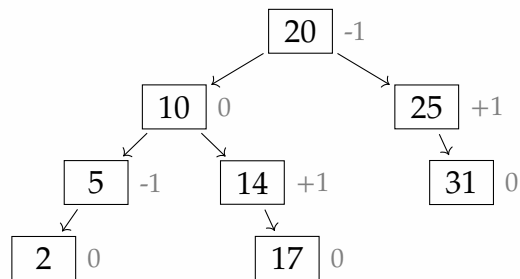
Nach der Linksrotation:



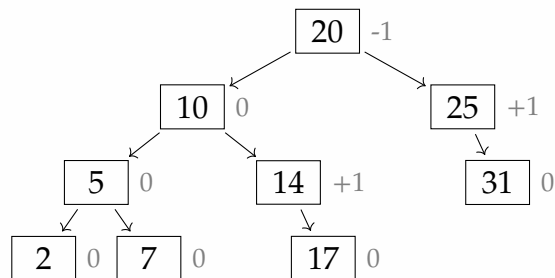
Nach dem Einfügen von „2“:



Nach dem Einfügen von „17“:

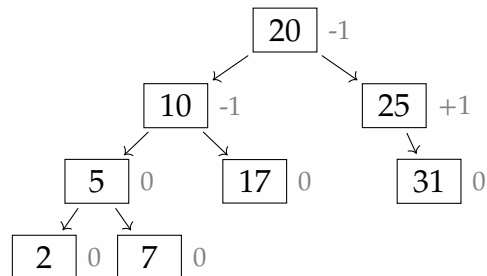


Nach dem Einfügen von „7“:

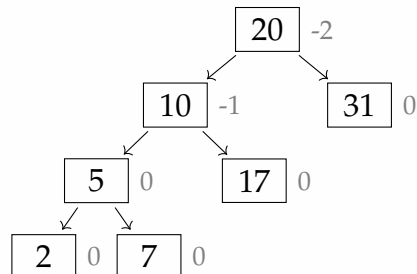


Löschen

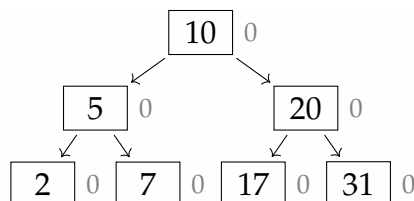
Nach dem Löschen von „14“:



Nach dem Löschen von „25“:



Nach der Rechtsrotation:



66115 / 2019 / Herbst / Thema 2 / Aufgabe 9

Verwenden Sie die Hashfunktion $h(k, i) = (h'(k) + i^2) \bmod 11$ mit $h'(k) = k \bmod 13$, um die Werte 12, 29 und 17 in die folgende Hashtabelle einzufügen. Geben Sie zudem jeweils an, auf welche Zellen der Hashtabelle zugegriffen wird.

0	1	2	3	4	5	6	7	8	9	10
			16		5				22	

Lösungsvorschlag

Einfügen des Wertes 12

$$h'(12) = 12 \bmod 13 = 12$$

$$h(12, 0) = 12 + 0^2 \bmod 11 = 1$$

0	1	2	3	4	5	6	7	8	9	10
	12		16		5				22	

Einfügen des Wertes 29

$$h'(29) = 29 \bmod 13 = 3$$

$$h(29, 0) = 3 + 0^2 \bmod 11 = 3 \text{ (belegt von 16)}$$

$$h(29, 1) = 3 + 1^2 \bmod 11 = 4$$

0	1	2	3	4	5	6	7	8	9	10
	12		16	29	5				22	

Einfügen des Wertes 17

$$h'(17) = 17 \bmod 13 = 4$$

$$h(17, 0) = 4 + 0^2 \bmod 11 = 4 \text{ (belegt von 29)}$$

$$h(17, 1) = 4 + 1^2 \bmod 11 = 5 \text{ (belegt von 5)}$$

$$h(17, 2) = 4 + 2^2 \bmod 11 = 8$$

0	1	2	3	4	5	6	7	8	9	10
	12		16	29	5		17	22		

66115 / 2020 / Frühjahr / Thema 2 / Aufgabe 10

Sei B ein binärer Suchbaum. In jedem Knoten v von B wird ein Schlüssel $v.\text{key} \in \mathbb{N}$ gespeichert sowie Zeiger $v.\text{left}$, $v.\text{right}$ und $v.\text{parent}$ auf sein linkes Kind, auf sein rechtes Kind und auf seinen Elternknoten. Die Zeiger sind nil , wenn der entsprechende Nachbar nicht existiert. Für zwei Knoten u und v ist wie üblich der *Abstand* die Anzahl der Kanten auf dem kürzesten Pfad von u nach v .

Für einen Knoten w von B sei $B(w)$ der Teilbaum von B mit Wurzel w . Für zwei Knoten u und v von B ist w ein *gemeinsamer Vorfahre*, wenn u und v in $B(w)$ liegen. Wir suchen den niedrigsten gemeinsamen Vorfahren $\text{ngV}(u, v)$ von u und v , also einen gemeinsamen Vorfahren w , so dass für jeden Vorfahren w von u und v gilt, dass w in $B(w)$ liegt. Wir betrachten verschiedene Szenarien, in denen Sie jeweils den niedrigsten gemeinsamen Vorfahren von u und v berechnen sollen.

Exkurs: Lowest Common Ancestor

Als Lowest Common Ancestor (LCA) oder „letzter gemeinsamer Vorfahre“ wird in der Informatik und Graphentheorie ein Ermittlungskonzept bezeichnet, das einen gegebenen gewurzelten Baum von Datenstrukturen effizient vorverarbeitet, sodass anschließend Anfragen nach dem letzten gemeinsamen Vorfahren für beliebige Knotenpaare in konstanter Zeit beantwortet werden können.

^a

^ahttps://de.wikipedia.org/wiki/Lowest_Common_Ancestor

- (a) Wir bekommen u und v als Zeiger auf die entsprechenden Knoten in B geliefert. Beschreiben Sie in Worten und in Pseudocode einen Algorithmus, der den niedrigsten gemeinsamen Vorfahren von u und v berechnet. Analysieren Sie die Laufzeit Ihres Algorithmus.

```
/**
 * NBV = Niedrigster gemeinsamer Vorfahre.
 *
 * https://afteracademy.com/blog/lowest-common-ancestor-of-a-binary-tree
 */
public class NGV {
    static class Knoten {
        int schlüssel;
        Knoten links;
        Knoten rechts;

        public Knoten(int schlüssel) {
            this.schlüssel = schlüssel;
        }
    }

    /**
     * ngV = niedrigster gemeinsamer Vorfahre
     *
     * @param wurzel Der Wurzelknoten des Binärbaums.
     * @param knoten1 Der erste Knoten, dessen niedrigster gemeinsamer Vorfahre
     *                gesucht werden soll.
     * @param knoten2 Der zweite Knoten, dessen niedrigster gemeinsamer Vorfahre
```

```
*          gesucht werden soll.
*
* @return Der niedrigste gemeinsame Vorfahre der Knoten 1 und 2.
*/
public static Knoten ngVRekursiv(Knoten wurzel, Knoten knoten1, Knoten
↪ knoten2) {
    if (wurzel == null)
        return null;
    if (wurzel.equals(knoten1) || wurzel.equals(knoten2))
        return wurzel;
    Knoten links = ngVRekursiv(wurzel.links, knoten1, knoten2);
    Knoten rechts = ngVRekursiv(wurzel.rechts, knoten1, knoten2);
    if (links == null)
        return rechts;
    else if (rechts == null)
        return links;
    else
        return wurzel;
}

/**
 * <pre>
 * {@code
 *      20
 *     / \
 *    8   22
 *   / \
 *  4   12
 *   / \
 *  10  14
 * }
 * </pre>
 *
 * Beispiele von
 * https://www.geeksforgeeks.org/lowest-common-ancestor-in-a-binary-search-
↪ tree/
 *
 * @param args Kommandozeilen-Argumente
 */
public static void main(String[] args) {
    Knoten wurzel = new Knoten(20);

    Knoten knoten8 = new Knoten(8);
    Knoten knoten22 = new Knoten(22);
    Knoten knoten4 = new Knoten(4);
    Knoten knoten12 = new Knoten(12);
    Knoten knoten10 = new Knoten(10);
    Knoten knoten14 = new Knoten(14);

    wurzel.links = knoten8;
    wurzel.rechts = knoten22;
    wurzel.links.links = knoten4;
    wurzel.links.rechts = knoten12;
    wurzel.links.rechts.links = knoten10;
    wurzel.links.rechts.rechts = knoten14;
```

```

    System.out.println(ngVRekursiv(wurzel, knoten10, knoten14).schlüssel); // 12
    System.out.println(ngVRekursiv(wurzel, knoten14, knoten8).schlüssel); // 8
    System.out.println(ngVRekursiv(wurzel, knoten10, knoten22).schlüssel); // 20
  }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/fruehjahr/NGV.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/fruehjahr/NGV.java)

- (b) Wir bekommen u und v wieder als Zeiger auf die entsprechenden Knoten in B geliefert. Seien d_u , und d_v , die Abstände von u bzw. v zum niedrigsten gemeinsamen Vorfahren von u und v . Die Laufzeit Ihres Algorithmus soll $O(\max\{d_u, d_v\})$ sein. Dabei kann Ihr Algorithmus in jedem Knoten v eine Information $v.info$ speichern. Skizzieren Sie Ihren Algorithmus in Worten.
- (c) Wir bekommen die Schlüssel $u.key$ und $v.key$. Die Laufzeit Ihres Algorithmus soll proportional zum Abstand der Wurzel von B zum niedrigsten gemeinsamen Vorfahren von u und v sein. Skizzieren Sie Ihren Algorithmus in Worten.

66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 2

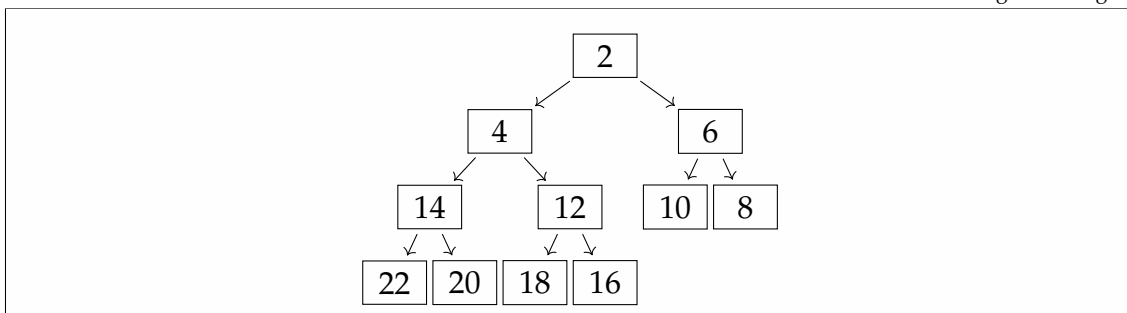
Wir betrachten ein Feld A von ganzen Zahlen mit n Elementen, die über die Indizes $A[0]$ bis $A[n-1]$ angesprochen werden können. In dieses Feld ist ein binärer Baum nach den folgenden Regeln eingebettet: Für das Feldelement mit Index i befindet sich

- der Elternknoten im Feldelement mit Index $\lfloor \frac{i-1}{2} \rfloor$,
- der linke Kindknoten im Feldelement mit Index $2 \cdot i + 1$, und
- der rechte Kindknoten im Feldelement mit Index $2 \cdot i + 2$.

- (a) Zeichnen Sie den durch das folgende Feld repräsentierten binären Baum.

i	0	1	2	3	4	5	6	7	8	9	10
A[i]	2	4	6	14	12	10	8	22	20	18	16

Lösungsvorschlag



(b) Der folgende rekursive Algorithmus sei gegeben:

Pseudo-Code / Pascal

```

procedure magic(i, n : integer) : boolean
begin
  if (i > (n - 2) / 2) then
    return true;
  endif
  if (A[i] <= A[2 * i + 1] and A[i] <= A[2 * i + 2] and
    magic(2 * i + 1, n) and magic(2 * i + 2, n)) then
    return true;
  endif
  return false;
end

```

Java-Implementation

```

public static boolean magic(int i, int n) {
  System.out.println(String.format("i: %s n: %s", i, n));
  if (i > (n - 2) / 2) {
    return true;
  }
  if (A[i] <= A[2 * i + 1] && A[i] <= A[2 * i + 2] && magic(2 * i + 1, n) &&
    ↪ magic(2 * i + 2, n)) {
    return true;
  }
  return false;
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java)

Gegeben sei folgendes Feld:

i	0	1	2	3
A[i]	2	4	6	14

Führen Sie `magic(0,3)` auf dem Feld aus. Welches Resultat liefert der Algorithmus zurück?

Lösungsvorschlag

true

(c) Wie nennt man die Eigenschaft, die der Algorithmus `magic` auf dem Feld A prüft? Wie lässt sich diese Eigenschaft formal beschreiben?

Lösungsvorschlag

Die sogenannte „Haldeneigenschaft“ bzw. „Heap-Eigenschaft“ einer Min-Halde. Der Schlüssel eines jeden Knotens ist kleiner (oder gleich) als die Schlüssel seiner Kinder.

Ein Baum erfüllt die Heap-Eigenschaft bezüglich einer Vergleichsrelation „>“ auf den Schlüsselwerten genau dann, wenn für jeden Knoten u des Baums gilt,

dass $u_{\text{wert}} > v_{\text{wert}}$ für alle Knoten v aus den Unterbäumen von u .

- (d) Welche Ausgaben sind durch den Algorithmus `magic` möglich, wenn das Eingabefeld aufsteigend sortiert ist? Begründen Sie Ihre Antwort.

Lösungsvorschlag

`true`. Eine sortierte aufsteigende Zahlenfolge entspricht den Haldeneigenschaften einer Min-Heap.

- (e) Geben Sie zwei dreielementige Zahlenfolgen (bzw. Felder) an, eine für die `magic(0,2)` den Wert `true` liefert und eine, für die `magic(0,2)` den Wert `false` liefert.

Lösungsvorschlag

```
A = new int[] { 1, 2, 3 };
System.out.println(magic(0, 2)); // true

A = new int[] { 2, 1, 3 };
System.out.println(magic(0, 2)); // false
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java)

- (f) Betrachten Sie folgende Variante `almostmagic` der oben bereits erwähnten Prozedur `magic`, bei der die Anweisungen in Zeilen 3 bis 5 entfernt wurden:

Pseudo-Code / Pascal

```
procedure almostmagic(i, n : integer) : boolean
begin
  // leer
  // leer
  // leer
  if (A[i] <= A[2 * i + 1] and A[i] <= A[2 * i + 2] and
      magic(2 * i + 1, n) and magic(2 * i + 2, n)) then
    return true;
  endif
  return false;
end
```

Java-Implementation

```
public static boolean almostmagic(int i, int n) {
  System.out.println(String.format("i: %s n: %s", i, n));
  if (A[i] <= A[2 * i + 1] && A[i] <= A[2 * i + 2] && magic(2 * i + 1, n) &&
      magic(2 * i + 2, n)) {
    return true;
  }
  return false;
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java)

Beschreiben Sie die Umstände, die auftreten können, wenn `almostmagic` auf einem Feld der Größe `n` aufgerufen wird. Welchen Zweck erfüllt die entfernte bedingte Anweisung?

Lösungsvorschlag

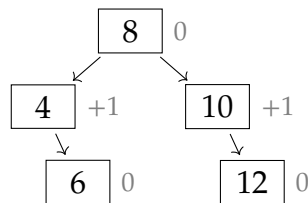
Wird die Prozedur zum Beispiel mit `almostmagic(0, n + 1)` aufgerufen, kommst es zu einem sogenannten „Array-Index-Out-of-Bounds“ Fehler, d. h. die Prozedur will auf Index des Feldes zugreifen, der im Feld gar nicht existiert. Die drei zusätzlichen Zeilen in der Methode `magic` bieten dafür einen Schutz, indem sie vor den Index-Zugriffen auf das Feld `true` zurückgeben.

```
// A = new int[] { 1, 2, 3 };
// System.out.println(almostmagic(0, 4)); // Exception in thread "main"
↪ java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for
↪ length 3
```

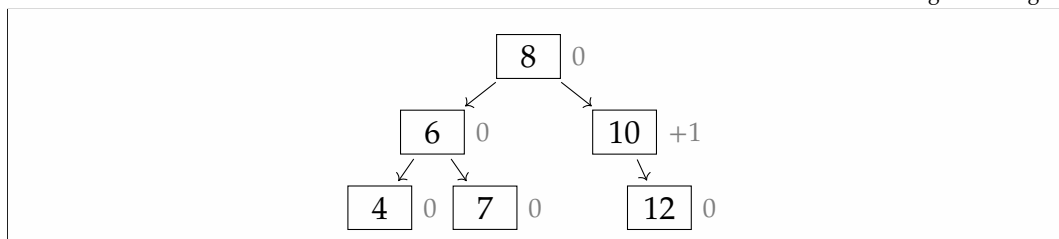
Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java)

- (g) Fügen Sie jeweils den angegebenen Wert in den jeweils angegebenen AVL-Baum mit aufsteigender Sortierung ein und zeichnen Sie den resultierenden Baum vor möglicherweise erforderlichen Rotationen. Führen Sie danach bei Bedarf die erforderliche(n) Rotation(en) aus und zeichnen Sie dann den resultierenden Baum. Sollten keine Rotationen erforderlich sein, so geben Sie dies durch einen Text wie „keine Rotationen nötig“ an.

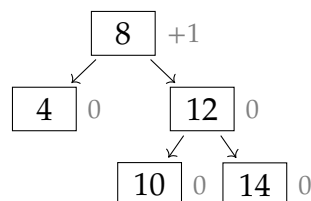
- (i) Wert 7 einfügen

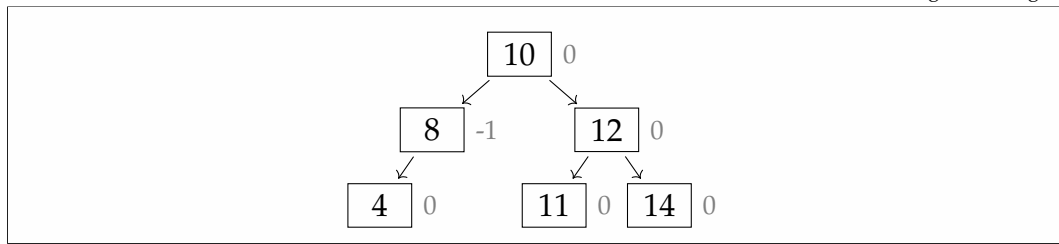


Lösungsvorschlag

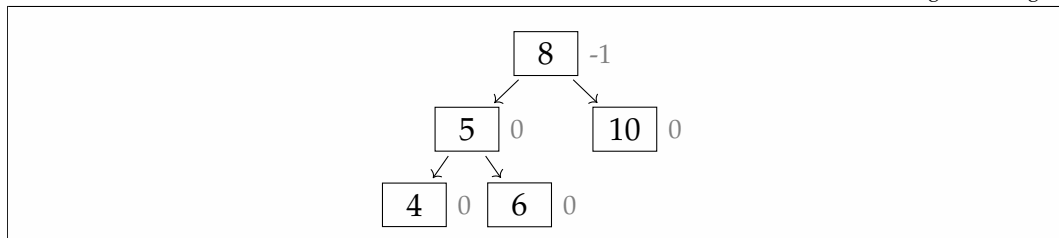
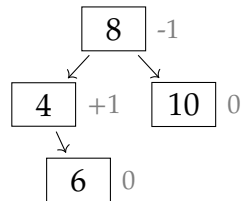


- (ii) Wert 11 einfügen





(iii) Wert 5 einfügen



66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 5

Gegeben seien die folgenden Schlüssel k zusammen mit ihren Streuwerten $h(k)$:

k	B	Y	E	!	A	U	D	?
$h(k)$	5	4	0	4	4	0	7	2

- (a) Fügen Sie die Schlüssel in der angegebenen Reihenfolge (von links nach rechts) in eine Streutabelle der Größe 8 ein und lösen Sie Kollisionen durch verkettete Listen auf.

Stellen Sie die Streutabelle in folgender Art und Weise dar:

Fach	Schlüssel k (verkettete Liste, zuletzt eingetragener Schlüssel rechts)
0	E, U,
1	
2	?
3	
4	Y, !, A
5	B,
6	
7	D

- (b) Fügen Sie die gleichen Schlüssel in der gleichen Reihenfolge und mit der gleichen Streufunktion in eine neue Streutabelle der Größe 8 ein. Lösen Sie Kollisionen diesmal aber durch lineares Sondieren mit Schrittweite +1 auf.

Geben Sie für jeden Schlüssel jeweils an, welche Fächer Sie in welcher Reihenfolge sondiert haben und wo der Schlüssel schlussendlich gespeichert wird.

Lösungsvorschlag

Fach	Schlüssel k
0	E
1	U
2	D
3	?
4	Y
5	B
6	!
7	A

Schlüssel	Sondierung	Speicherung
B		5
Y		4
E		0
!	4, 5	6
A	4, 5, 6	7
U	0	1
D	7, 0, 1	2
?	2	3

- (c) Bei der doppelten Streuadressierung verwendet man eine Funktionsschar h_i , die sich aus einer primären Streufunktion h_0 und einer Folge von sekundären Streufunktionen h_1, h_2, \dots zusammensetzt. Die folgenden Werte der Streufunktionen sind gegeben:

k	B	Y	E	!	A	U	D	?
$h_0(k)$	5	4	0	4	4	0	7	2
$h_1(k)$	6	3	3	3	1	2	6	0
$h_2(k)$	7	2	6	2	6	4	5	6
$h_3(k)$	0	1	1	1	3	6	4	4

Fügen Sie die Schlüssel in der angegebenen Reihenfolge (von links nach rechts) in eine Streutabelle der Größe 8 ein und geben Sie für jeden Schlüssel jeweils an, welche Streufunktion h_i zur letztendlichen Einsortierung verwendet wurde.

Lösungsvorschlag

Fach	Schlüssel k
0	E
1	A
2	U
3	!
4	Y
5	B
6	?
7	D

Schlüssel	Streufunction
B	$h_0(k)$
Y	$h_0(k)$
E	$h_0(k)$
!	$h_1(k)$
A	$h_1(k)$
U	$h_1(k)$
D	$h_0(k)$
?	$h_2(k)$

66115 / 2020 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 1

Eine Hashfunktion h wird verwendet, um Vornamen auf die Buchstaben $\{A, \dots, Z\}$ abzubilden. Dabei bildet h auf den ersten Buchstaben des Vornamens als Hashwert ab.

Sei H die folgende Hashtabelle (Ausgangszustand):

Schlüssel	Inhalt	Schlüssel	Inhalt
A		N	
B		O	
C		P	
D	Dirk	Q	
E		R	
F		S	
G		T	
H		U	
I	Inge	V	
J		W	
K	Kurt	X	
L		Y	
M		Z	

(a) Fügen Sie der Hashtabelle H die Vornamen

Martin, Michael, Norbert, Matthias, Alfons, Bert, Christel, Adalbert, Edith, Emil

in dieser Reihenfolge hinzu, wobei Sie Kollisionen durch lineares Sondieren (mit Inkrementieren zum nächsten Buchstaben) behandeln.

Lösungsvorschlag

Schlüssel	Inhalt	Schlüssel	Inhalt
A	Alfons	N	Michael
B	Bert	O	Norbert
C	Christel	P	Matthias
D	Dirk	Q	
E	Adalbert	R	
F	Edith	S	
G	Emil	T	
H		U	
I	Inge	V	
J		W	
K	Kurt	X	
L		Y	
M	Martin	Z	

(b) Fügen Sie der Hashtabelle H die Vornamen

Brigitte, Elmar, Thomas, Katrin, Diana, Nathan, Emanuel, Sebastian, Torsten,
Karolin

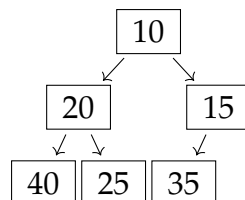
in dieser Reihenfolge hinzu, wobei Sie Kollisionen durch Verkettung der Überläufer behandeln. (Hinweis: Verwenden Sie die Hashtabelle im Ausgangszustand.)

Lösungsvorschlag

Schlüssel	Inhalt	Schlüssel	Inhalt
A		N	Nathan
B	Brigitte	O	
C		P	
D	Dirk, Diana	Q	
E	Elmar, Emanuel	R	
F		S	Sebastian
G		T	Thomas, Torsten
H		U	
I	Inge	V	
J		W	
K	Kurt, Katrin, Karolin	X	
L		Y	
M		Z	

66115 / 2020 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 3

Es sei der folgende Min-Heap gegeben:



- (a) Geben Sie obigen Min-Heap in der Darstellung eines Feldes an, wobei die Knoten in Level-Order abgelegt sind.

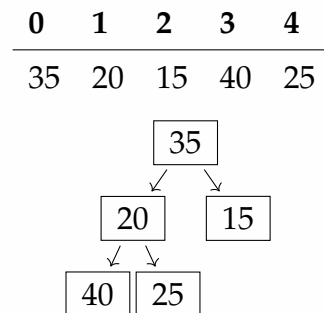
Lösungsvorschlag

0	1	2	3	4	5
10	20	15	40	25	35

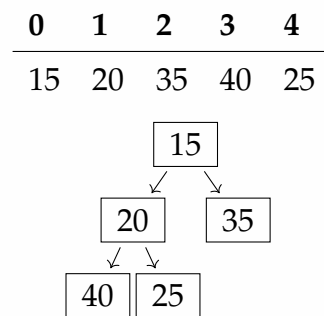
- (b) Führen Sie wiederholt DeleteMin-Operationen auf dem gegebenen Heap aus, bis der Heap leer ist. Zeichnen Sie dafür den aktuellen Zustand des Heaps als Baum und als Feld nach jeder Änderung des Heaps, wobei Sie nur gültige Bäume zeichnen (d. h. solche die keine Lücken haben). Dokumentieren Sie, was in den einzelnen Schritten geschieht.

Löschen von 10

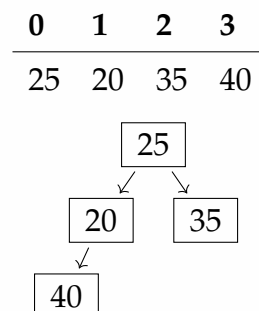
Nach dem Ersetzen von „10“ durch „35“:



Nach dem Vertauschen von „35“ und „15“:

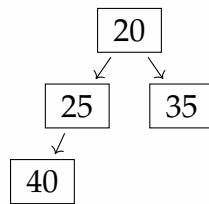
**Löschen von 15**

Nach dem Ersetzen von „15“ mit „25“:



Nach dem Vertauschen von „25“ und „20“:

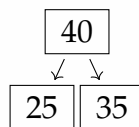
0	1	2	3
20	25	35	40



Löschen von 20

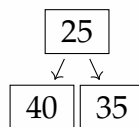
Nach dem Vertauschen von „20“ mit „40“:

0	1	2
40	25	35



Nach dem Vertauschen von „40“ und „25“:

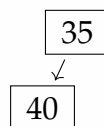
0	1	2
25	40	35



Löschen von 25

Nach dem Ersetzen von „25“ durch „35“:

0	1
35	40



Löschen von 35

Nach dem Ersetzen von „35“ mit „40“:

0
40
40

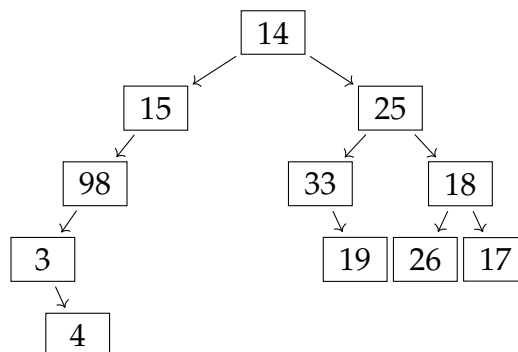
Löschen von 40

Nach dem Löschen von „40“:

66115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 3

(a) Betrachten Sie folgenden Binärbaum T.

Geben Sie die Schlüssel der Knoten in der Reihenfolge an, wie sie von einem Preorder-Durchlauf (= TreeWalk) von T ausgegeben werden.

**Exkurs: Preorder-Traversierung eines Baum**

besuche die Wurzel, dann den linken Unterbaum, dann den rechten Unterbaum; auch: WLR

```

private void besuchePreorder(BaumKnoten knoten, ArrayList<Comparable>
    ↪ schlüssel) {
    if (knoten != null) {
        schlüssel.add((Comparable) knoten.gibSchlüssel());
        besuchePreorder(knoten.gibLinks(), schlüssel);
        besuchePreorder(knoten.gibRechts(), schlüssel);
    }
}
  
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/baum/BinaerBaum.java](https://github.com/bschlangaul/baum/BinaerBaum.java)

Lösungsvorschlag

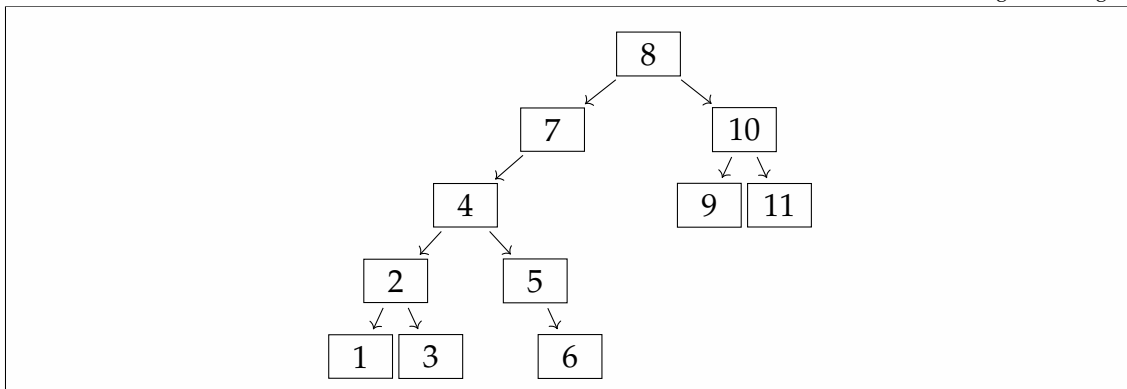
14, 15, 98, 3, 4, 25, 33, 19, 18, 26, 17

- (b) Betrachten Sie folgende Sequenz als Ergebnis eines Preorder-Durchlaufs eines binären Suchbaumes T . Zeichnen Sie T und erklären Sie, wie Sie zu Ihrer Schlussfolgerung gelangen.

[8,7,4,2,1,3,5,6,10,9,11]

Hinweis: Welcher Schlüssel ist die Wurzel von T ? Welche Knoten sind in seinem linken/rechten Teilbaum gespeichert? Welche Schlüssel sind die Wurzeln der jeweiligen Teilbäume?

Lösungsvorschlag



- (c) Anstelle von sortierten Zahlen soll ein Baum nun verwendet werden, um relative Positionsangaben zu speichern. Jeder Baumknoten enthält eine Beschriftung und einen Wert (vgl. Abb. 1), der die ganzzahlige relative Verschiebung in horizontaler Richtung gegenüber seinem Elternknoten angibt. Die zu berechnenden Koordinaten für einen Knoten ergeben sich aus seiner Tiefe im Baum als y -Wert und aus der Summe aller Verschiebungen auf dem Pfad zur Wurzel als x -Wert. Das Ergebnis der Berechnung ist in Abb. 2 visualisiert. Geben Sie einen Algorithmus mit linearer Laufzeit in Pseudo-Code oder einer objektorientierten Programmiersprache Ihrer Wahl an. Der Algorithmus erhält den Zeiger auf die Wurzel eines Baumes als Eingabe und soll Tupel mit den berechneten Koordination aller Knoten des Baums in der Form (Beschriftung, x , y) zurück- oder ausgeben.

Lösungsvorschlag

```

public class Knoten {
    public Knoten links;
    public Knoten rechts;
    public String name;

    /**
     * Bewegung bezüglich des Vorknotens. Relative Lage.
     */
    public int xVerschiebung;
  
```

```
public Knoten(String name, int xVerschiebung) {
    this.name = name;
    this.xVerschiebung = xVerschiebung;
}

public void durchlaufen() {
    durchlaufe(this, 0 + xVerschiebung, 0);
}

private void durchlaufe(Knoten knoten, int x, int y) {
    System.out.println("Beschriftung: " + knoten.name + " x: " + x + " y: " +
        ↪ y);

    if (links != null) {
        links.durchlaufe(links, x + links.xVerschiebung, y + 1);
    }
    if (rechts != null) {
        rechts.durchlaufe(rechts, x + rechts.xVerschiebung, y + 1);
    }
}

public static void main(String[] args) {
    Knoten a = new Knoten("a", 1);
    Knoten b = new Knoten("b", 1);
    Knoten c = new Knoten("c", -2);
    Knoten d = new Knoten("d", 2);
    Knoten e = new Knoten("e", 0);

    a.links = b;
    a.rechts = c;
    c.links = d;
    c.rechts = e;

    a.durchlaufen();
}
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/Knoten.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/Knoten.java)

66115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 5

(a) Nennen Sie zwei wünschenswerte Eigenschaften von Hashfunktionen.

Lösungsvorschlag

surjektiv Die Abbildung soll surjektiv sein, öjeder Index soll berechnet werden können.

gleichverteilt Durch die Hashfunktion soll möglichst eine Gleichverteilung auf die Buckets (Indexliste) erfolgen.

effizient Zudem sollte die Verteilung mittels Hashfunktion möglichst effizient gewählt werden.

- (b) Wie viele Elemente können bei Verkettung und wie viele Elemente können bei offener Adressierung in einer Hashtabelle mit m Zeilen gespeichert werden?

Lösungsvorschlag

Verkettung Es darf mehr als ein Element pro Bucket enthalten sein, deswegen können beliebig viele Element gespeichert werden.

offene Adressierung (normalerweise) ein Element pro Bucket, deshalb ist die Anzahl der speicherbaren Elemente höchstens m . Können in einem Bucket k Elemente gespeichert werden, dann beträgt die Anzahl der speicherbaren Elemente $k \cdot m$.

- (c) Angenommen, in einer Hashtabelle der Größe m sind alle Einträge (mit mindestens einem Wert) belegt und insgesamt n Werte abgespeichert.

Geben Sie in Abhängigkeit von m und n an, wie viele Elemente bei der Suche nach einem nicht enthaltenen Wert besucht werden müssen. Sie dürfen annehmen, dass jeder Wert mit gleicher Wahrscheinlichkeit und unabhängig von anderen Werten auf jeden der m Plätze abgebildet wird (einfaches gleichmäßiges Hashing).

Lösungsvorschlag

$\frac{n}{m}$ Beispiel: 10 Buckets, 30 Elemente: $\frac{30}{10} = 3$ Elemente im Bucket, die man durchsuchen muss.

- (d) Betrachten Sie die folgende Hashtabelle mit der Hashfunktion $h(x) = x \bmod 11$. Hierbei steht \emptyset für eine Zelle, in der kein Wert hinterlegt ist.

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	\emptyset	\emptyset	3	\emptyset	16	28	18	\emptyset	\emptyset	32

Führen Sie nun die folgenden Operationen mit offener Adressierung mit linearem Sondieren aus und geben Sie den Zustand der Datenstruktur nach jedem Schritt an. Werden für eine Operation mehrere Zellen betrachtet, aber nicht modifiziert, so geben Sie deren Indizes in der betrachteten Reihenfolge an.

- (i) Insert 7

Lösungsvorschlag

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	\emptyset	\emptyset	3	\emptyset	16	28	18	7 ₂	\emptyset	32

- (ii) Insert 20

Lösungsvorschlag

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	\emptyset	\emptyset	3	\emptyset	16	28	18	7 ₂	20 ₁	32

(iii) Delete 18

Lösungsvorschlag

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	\emptyset	\emptyset	3	\emptyset	16	28	del	7_2	20_1	32

del ist eine Marke, die anzeigt, dass gelöscht wurde und der Bucket nicht leer ist.

(iv) Search 7

Lösungsvorschlag

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	\emptyset	\emptyset	3	\emptyset	16	28	del	7_2	20_1	32

$h(7) = 7 \bmod 11 = 7$
 7 (Index) \rightarrow del lineares sondieren \rightarrow 8 (Index) \rightarrow gefunden

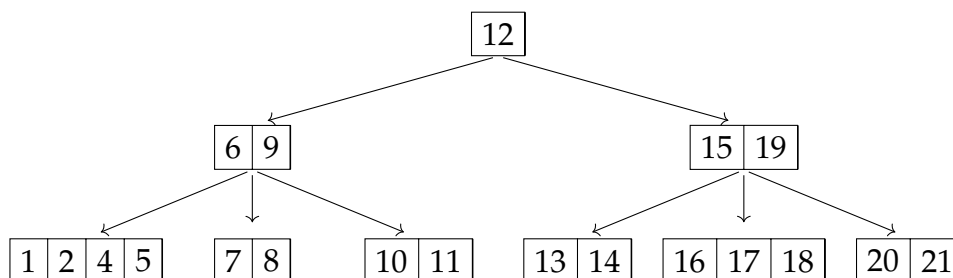
(v) Insert 5

Lösungsvorschlag

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	\emptyset	\emptyset	3	\emptyset	16	28	5_3	7_2	20_1	32

66116 / 2013 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 3

Gegeben sei der folgende B-Baum:

(a) Was bedeutet k bei einem B-Baum mit Grad k ? Geben Sie k für den obigen B-Baum an.

Lösungsvorschlag

Jeder Knoten außer der Wurzel hat mindestens k und höchstens $2k$ Einträge. Die Wurzel hat zwischen einem und $2k$ Einträgen. Die Einträge werden in allen Knoten sortiert gehalten. Alle Knoten mit n Einträgen, außer den Blättern, haben $n + 1$ Kinder.

Für den gegebenen Baum kann die Ordnung $k = 2$ angegeben werden.

- (b) Was sind die Vorteile von B-Bäumen im Vergleich zu binären Baumen?

Lösungsvorschlag

B-Bäume sind immer höhenbalanciert. B-Bäume haben eine geringere Höhe, wodurch eine schnellere Suche möglich wird, da weniger Aufrufe nötig sind.^a

^a<http://www.bayer.in.tum.de/lehre/WS2001/HSEM-bayer/BTreesAusarbeitung.pdf>

- (c) Wozu werden B-Bäume in der Regel verwendet und wieso?

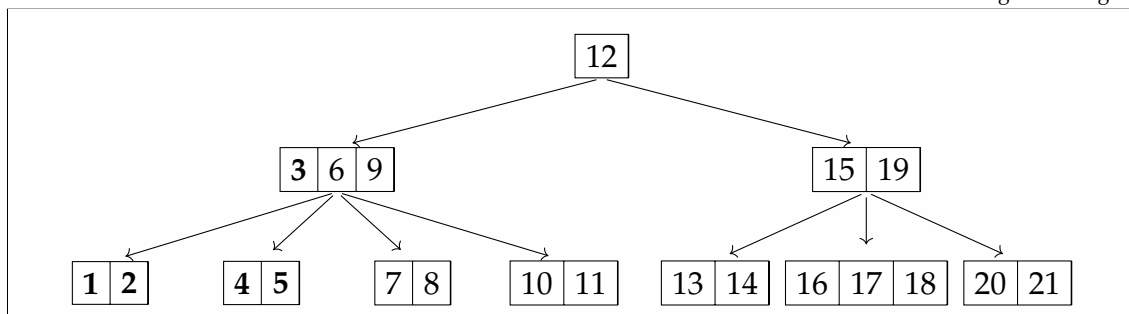
Lösungsvorschlag

B-Bäume werden für Hintergrundspeicherung (z. B. von Datenbanksystemen, Dateisystem) verwendet. Die Knotengrößen werden auf die Seitenkapazitäten abgestimmt.

B-Bäume sind eine daten- und Indexstruktur, die häufig in Datenbanken und Dateisystemen eingesetzt werden. Da ein B-Baum immer vollständig balanciert ist und die Schlüssel sortiert gespeichert werden, ist ein schnelles Auffinden von Inhalten gegeben.

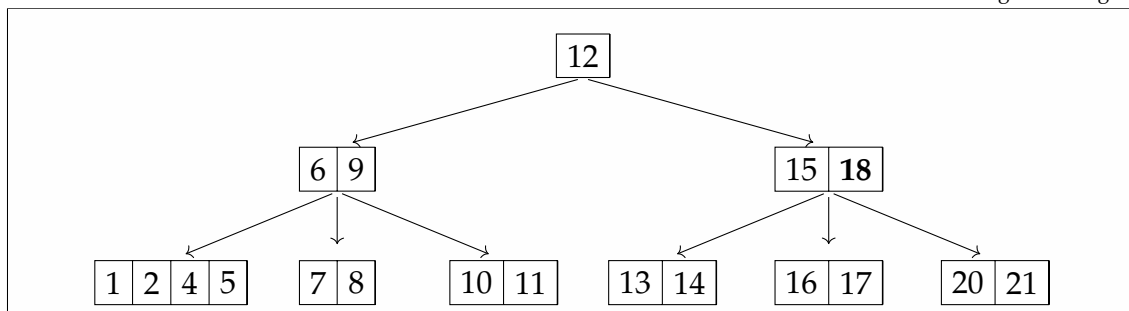
- (d) Fügen Sie den Wert 3 in den B-Baum ein, und zeichnen Sie den vollständigen B-Baum nach dem Einfügen und möglichen darauf folgenden Operationen.

Lösungsvorschlag



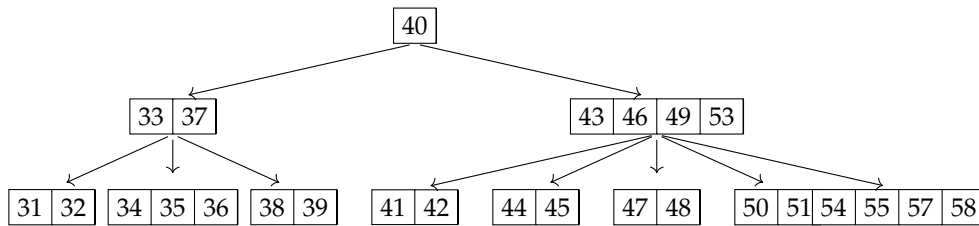
- (e) Entfernen Sie aus dem ursprünglichen B-Baum den Wert 19. Zeichnen Sie das vollständige Ergebnis nach dem Löschen und möglichen darauf folgenden Operationen. Sollte es mehrere richtige Lösungen geben, reicht es eine Lösung zu zeichnen.

Lösungsvorschlag



66116 / 2015 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 3

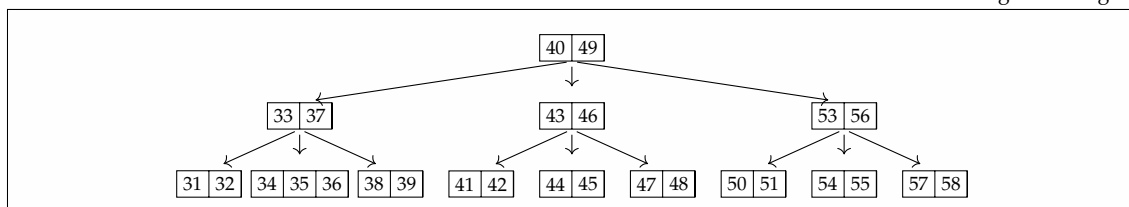
Als Indexstruktur einer Datenbank sei folgender B-Baum ($k = 2$) gegeben:



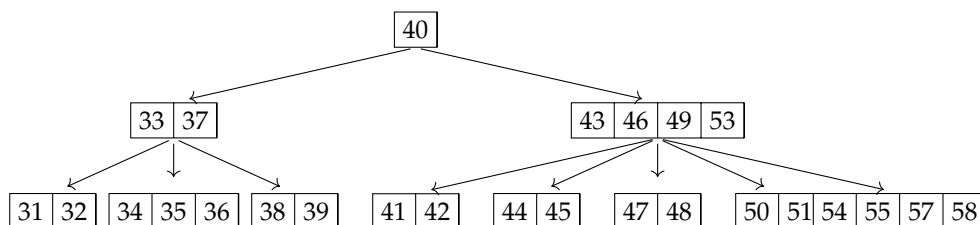
Führen Sie nacheinander die folgenden Operationen aus. Geben Sie die auftretenden Zwischenergebnisse an. Teilbäume, die sich in einem Schritt nicht verändern, müssen nicht erneut gezeichnet werden. Sollten Wahlmöglichkeiten auftreten, so sind größere Schlüsselwerte bzw. weiter rechts liegende Knoten zu bevorzugen.

(a) Einfügen des Wertes 56

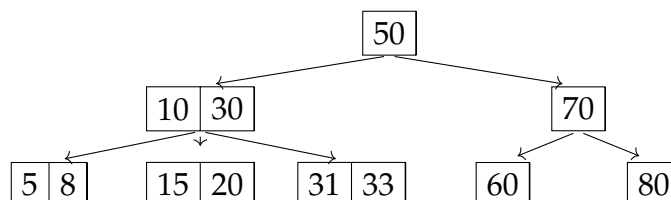
Lösungsvorschlag



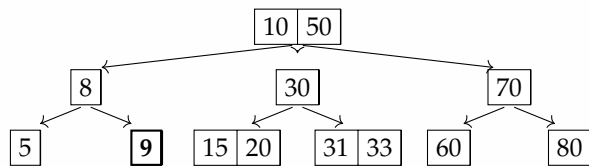
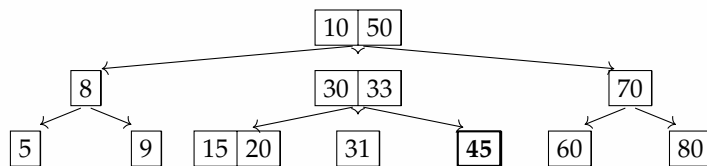
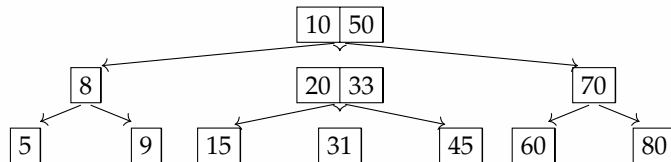
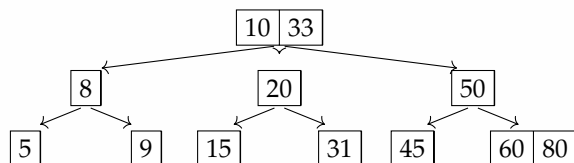
(b) Löschen des Wertes 37

**66116 / 2015 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 3**

Gegeben ist der folgende B-Baum der Ordnung 3 (max. drei Kindknoten, max. zwei Schlüssel pro Knoten):



Fügen Sie die Werte 9 und 45 ein. Löschen Sie anschließend die Werte 30 und 70. Zeichnen Sie den Baum nach jeder Einfüge- bzw. Lösch-Operation.

9 einfügen**45 einfügen****30 löschen****70 löschen****66116 / 2016 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 5**

- (a) Erläutern Sie die wesentliche Eigenschaft eines Tupel-Identifikators (TID) in ein bis zwei Sätzen.

- Daten werden in Form von *Sätzen* auf der Festplatte abgelegt, um auf Sätze zugreifen zu können, verfügt jeder Satz über eine *eindeutige, unveränderliche Satzadresse*
- TID = Tupel Identifier: dient zur Adressierung von Sätzen in einem Segment und besteht aus zwei Komponenten:
 - Seitennummer (Seiten bzw. Blöcke sind größere Speichereinheiten auf der Platte)
 - Relative Indexposition innerhalb der Seite
- Satzverschiebung innerhalb einer Seite bleibt ohne Auswirkungen auf den TID. Wird ein Satz auf eine andere Seite migriert, wird eine „Stellvertreter-TID“ zum Verweis auf den neuen Speicherort verwendet. Die eigentliche

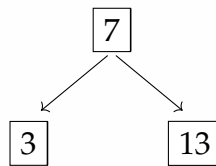
TID-Adresse bleibt stabil.

- (b) Fügen Sie in einen anfangs leeren B-Baum mit $k = 1$ (maximal 2 Schlüsselwerte pro Knoten) die im Folgenden gegebenen Schlüsselwerte der Reihe nach ein. Zeichnen Sie den Endzustand des Baums nach jedem Einfügevorgang. Falls Sie Zwischenschritte zeichnen, kennzeichnen Sie die sieben Endzustände deutlich.

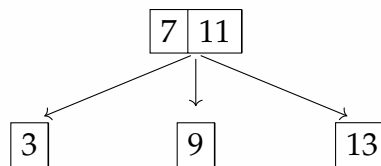
3, 7, 13, 11, 9, 10, 8

Lösungsvorschlag

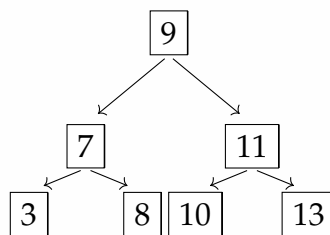
- 3 (einfaches Einfügen)
- 7 (einfaches Einfügen)
- 13 (Split)



- 11 (einfaches Einfügen)
- 9 (Split)



- 10 (einfaches Einfügen)
- 8 (Doppel-Split)



- (c) Gegeben ist der folgende B-Baum:

Die folgenden Teilaufgaben sind voneinander unabhängig.

- (i) Löschen Sie aus dem gegebenen B-Baum den Schlüssel 3 und zeichnen Sie den Endzustand des Baums nach dem Löschvorgang. Falls Sie Zwischenschritte zeichnen, kennzeichnen Sie den Endzustand deutlich.
- (ii) Löschen Sie aus dem (originalen) gegebenen B-Baum den Schlüssel 17 und zeichnen Sie den Endzustand des Baums nach dem Löschvorgang. Falls Sie Zwischenschritte zeichnen, kennzeichnen Sie den Endzustand deutlich.

- (iii) Löschen Sie aus dem (originalen) gegebenen B-Baum den Schlüssel 43 und zeichnen Sie den Endzustand des Baums nach dem Löschvorgang. Falls Sie Zwischenschritte zeichnen, kennzeichnen Sie den Endzustand deutlich.

66116 / 2017 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 2

Konstruieren Sie einen B-Baum, dessen Knoten maximal 4 Einträge enthalten können, indem Sie der Reihe nach diese Suchschlüssel einfügen:

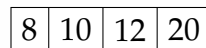
8, 10, 12, 20, 5, 30, 25, 11

Anschließend löschen Sie den Eintrag mit dem Suchschlüssel 8.

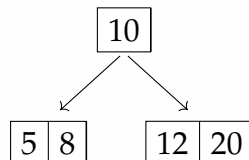
Zeigen Sie jeweils graphisch den entstehenden Baum nach relevanten Zwischenschritten; insbesondere nach Einfügen der 5 sowie nach dem Einfügen der 11 und nach dem Löschen der 8.

Lösungsvorschlag

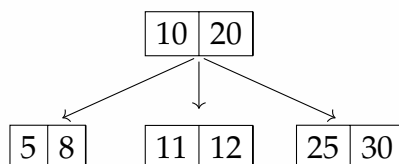
- Schlüsselwert 8 (einfaches Einfügen)
- Schlüsselwert 10 (einfaches Einfügen)
- Schlüsselwert 12 (einfaches Einfügen)
- Schlüsselwert 20 (einfaches Einfügen)



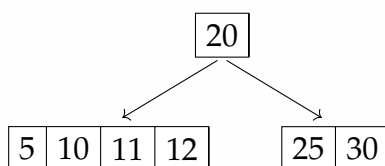
- Schlüsselwert 5 (Split)



- Schlüsselwert 30 (einfaches Einfügen)
- Schlüsselwert 25 (einfaches Einfügen)
- Schlüsselwert 11 (Split)

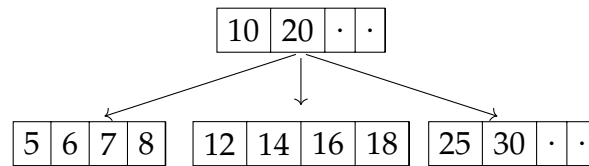


- Löschen des Schlüsselwerts 8 (Mischen/Verschmelzen)



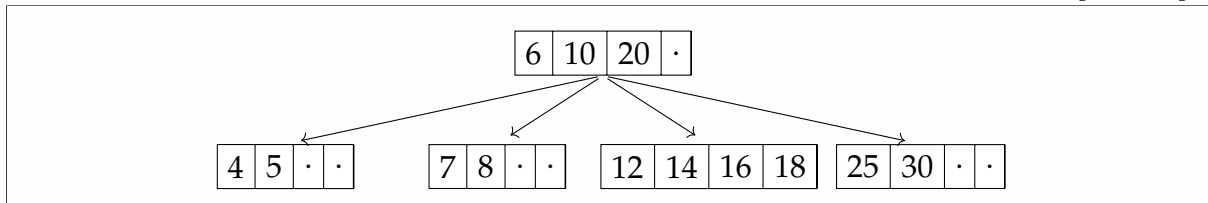
66116 / 2020 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 6

Fügen Sie die Zahl 4 in den folgenden B-Baum ein.



Zeichnen Sie den vollständigen, resultierenden Baum.

Lösungsvorschlag

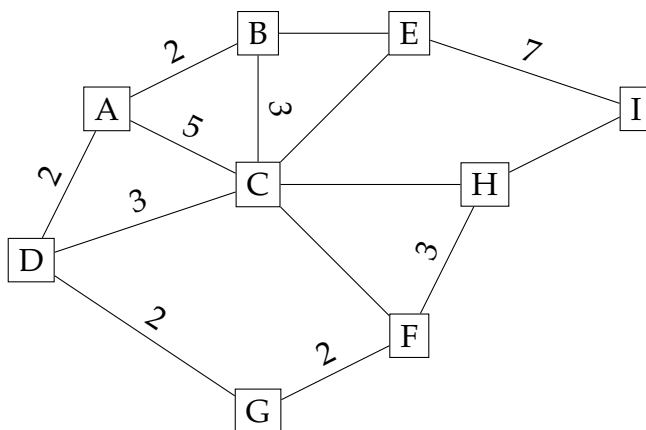


Graphen

66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6

Dijkstra-Algorithmus

Führen Sie auf dem gegebenen Graphen die Suche nach der kürzesten Distanz aller Knoten zum Startknoten A mit dem Algorithmus von Dijkstra durch. Tragen Sie die Abarbeitungsreihenfolge, den unmittelbaren Vorgängerknoten, sowie die ermittelte kürzeste Distanz für jeden Knoten ein! Bei gleichen Distanzen arbeiten Sie die Knoten in lexikalischer Reihenfolge ab.



Lösungsvorschlag

Nr.	besucht	A	B	C	D	E	F	G	H	I
0		0	∞	∞	∞	∞	∞	∞	∞	∞
1	A	0	2	5	2	∞	∞	∞	∞	∞
2	B		2	5	2	3	∞	∞	∞	∞
3	D			5	2	3	∞	4	∞	∞
4	E			4		3	∞	4	∞	10
5	C			4			5	4	5	10
6	G						5	4	5	10
7	F						5		5	10
8	H								5	6
9	I									6

nach	Entfernung	Reihenfolge	Pfad
$A \rightarrow A$	0	0	
$A \rightarrow B$	2	2	$A \rightarrow B$
$A \rightarrow C$	4	5	$A \rightarrow B \rightarrow E \rightarrow C$
$A \rightarrow D$	2	3	$A \rightarrow D$
$A \rightarrow E$	3	4	$A \rightarrow B \rightarrow E$
$A \rightarrow F$	5	7	$A \rightarrow B \rightarrow E \rightarrow C \rightarrow F$
$A \rightarrow G$	4	6	$A \rightarrow D \rightarrow G$
$A \rightarrow H$	5	8	$A \rightarrow B \rightarrow E \rightarrow C \rightarrow H$
$A \rightarrow I$	6	9	$A \rightarrow B \rightarrow E \rightarrow C \rightarrow H \rightarrow I$

66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6

Flugverbindung zwischen sieben Städten

Nehmen Sie an, es gibt sieben Städte A, B, C, D, E, F und G. Sie wohnen in der Stadt A und möchten zu jeder der anderen Städte die preiswerteste Flugverbindung finden (einfach ohne Rückflug). Sie sind dazu bereit, beliebig oft umzusteigen. Folgende Direktflüge stehen Ihnen zur Verfügung:

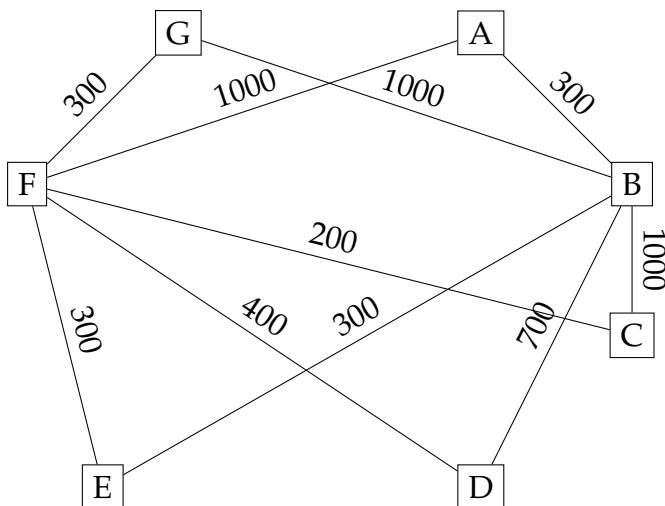
Städte	Preis
$A \leftrightarrow B$	300 €
$A \leftrightarrow F$	1000 €
$B \leftrightarrow C$	1000 €
$B \leftrightarrow D$	700 €
$B \leftrightarrow E$	300 €
$B \leftrightarrow G$	1000 €
$C \leftrightarrow F$	200 €
$D \leftrightarrow F$	400 €
$E \leftrightarrow F$	300 €
$F \leftrightarrow G$	300 €

Der Preis p in einer Zeile

Städte	Preis
$x \leftrightarrow y$	p

gilt dabei sowohl für einen einfachen Flug von x nach y als auch für einen einfachen Flug von y nach x. Bestimmen Sie mit dem Algorithmus von Dijkstra (führen Sie den Algorithmus händisch durch!) die Routen und die Preise für die preiswertesten Flugverbindungen von der Stadt A zu jeder der anderen Städte.

Als TikZ-Umgebung

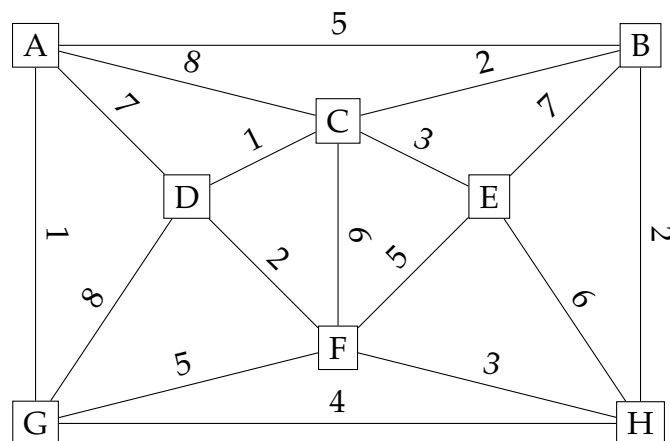


Schritt	besuchte Knoten	A	B	C	D	E	F	G
Init		0	∞	∞	∞	∞	∞	∞
1	A	0	300,A	∞	∞	∞	1000,F	∞
2	A,B	0		1300,B	1000,B	600,B	1000,F	1300,B
3	A,B,E	0		1300,B	1000,B		900,E	1300,B
4	A,B,E,F	0		1100,F	1000,B			1200,F
5	A,B,E,F,D	0		1100,F				1200,F
6	A,B,E,F,D,C	0						1200,F
7	A,B,E,F,D,C,G	0						

Städte	Preis
A → B	300
A → B → E → F → C	1100
A → B → D	1000
A → B → E	600
A → B → E → F	900
A → B → E → F → G	1200

Spannbaum

Ermitteln Sie einen minimalen Spannbaum des vorliegenden Graphen. Nutzen Sie den *Knoten A als Startknoten* in ihrem Algorithmus.



(a) Welches Gewicht hat der Spannbaum insgesamt?

Das Kantengewicht des minimalen Spannbaums beträgt 15.

Wir setzen den Algorithmus von Prim ein. Der Algorithmus läuft folgendermaßen ab:

Besuche Knoten „A“

Füge Kante (A, B, 5) hinzu

Füge Kante (A, C, 8) hinzu

Füge Kante (A, D, 7) hinzu

Füge Kante (A, G, 1) hinzu

Besuche Knoten „G“

Füge Kante (G, F, 5) hinzu

Füge Kante (G, H, 4) hinzu

Besuche Knoten „H“

Aktualisiere Kante (H, B, 2)

Füge Kante (H, E, 6) hinzu

Aktualisiere Kante (H, F, 3)

Besuche Knoten „B“

Aktualisiere Kante (B, C, 2)

Besuche Knoten „C“

Aktualisiere Kante (C, D, 1)

Aktualisiere Kante (C, E, 3)

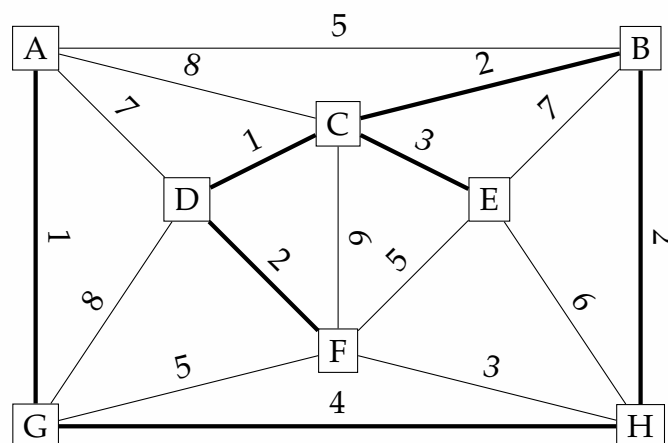
Besuche Knoten „D“

Aktualisiere Kante (D, F, 2)

Besuche Knoten „F“

Besuche Knoten „E“

schwarze	graue
(A, null, 0)	(B, A, 5); (C, A, 8); (D, A, 7); (G, A, 1);
(G, A, 1)	(B, A, 5); (C, A, 8); (D, A, 7); (F, G, 5); (H, G, 4);
(H, G, 4)	(B, H, 2); (C, A, 8); (D, A, 7); (E, H, 6); (F, H, 3);
(B, H, 2)	(C, B, 2); (D, A, 7); (E, H, 6); (F, H, 3);
(C, B, 2)	(D, C, 1); (E, C, 3); (F, H, 3);
(D, C, 1)	(E, C, 3); (F, D, 2);
(F, D, 2)	(E, C, 3);
(E, C, 3)	



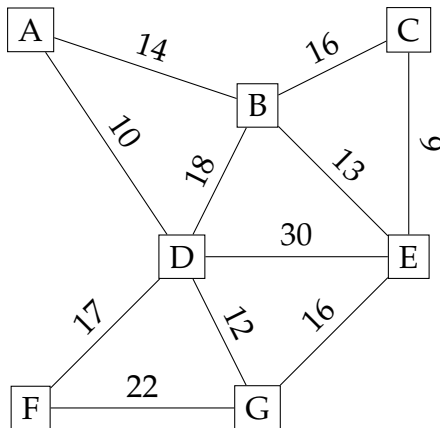
(b) Welchen Algorithmus haben Sie zur Ermittlung eingesetzt?

Lösungsvorschlag

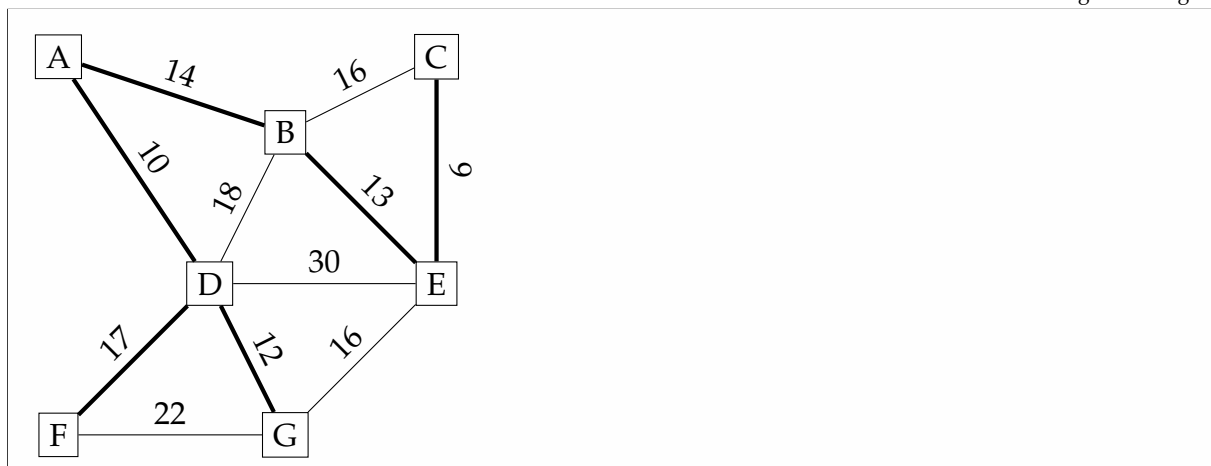
Algorithmus von Prim. Dieser Algorithmus benötigt einen Startknoten.

Algorithmus von Prim

1

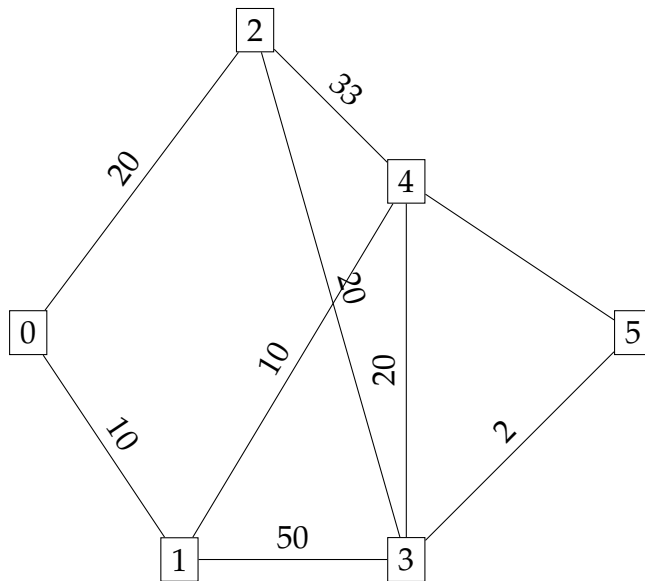


Lösungsvorschlag



Standardbeispiel TUM

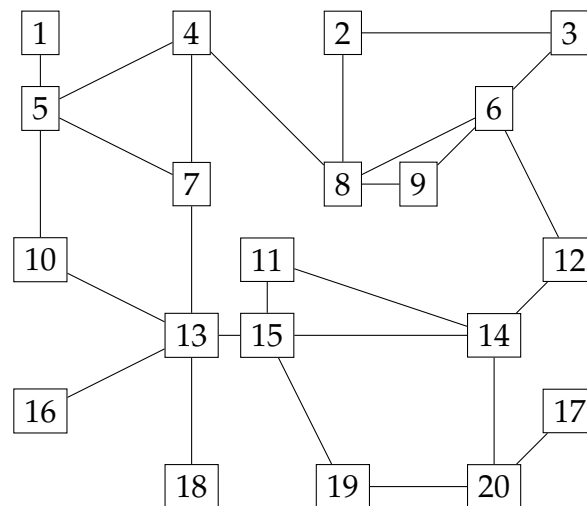
Standardbeispiel ²¹<https://studyflix.de/informatik/prim-algorithmus-1293>²https://algorithms.discrete.ma.tum.de/graph-algorithms/mst-prim/index_en.html



66116 / 2020 / Frühjahr / Thema 1 / Aufgabe 6

Breiten- & Tiefensuche

- (a) Geben Sie die Reihenfolge an, in der die Knoten besucht werden, wenn auf dem folgenden Graphen *Breitensuche* ausgehend von Knoten 1 ausgeführt wird. Wenn mehrere Knoten zur Wahl stehen, wählen Sie den Knoten mit dem kleinsten Schlüssel.



Lösungsvorschlag

```

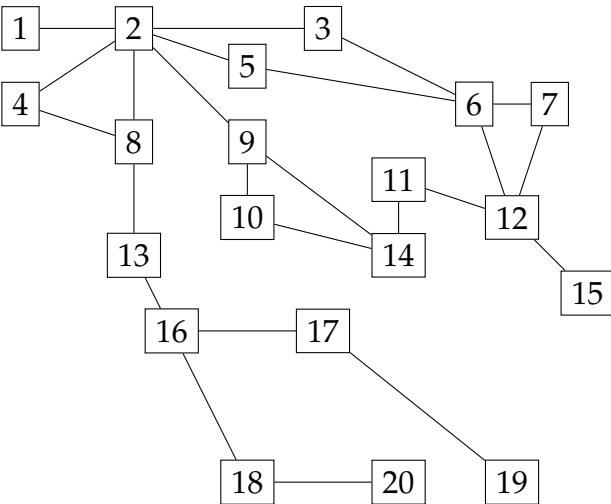
    add 1  [1]
del 1
    add 5  [5]
del 5
    add 4  [4]
    add 7  [4, 7]
    add 10 [4, 7, 10]
```

```

del 4
    add 8 [7, 10, 8]
del 7
    add 13 [10, 8, 13]
del 10
del 8
    add 2 [13, 2]
    add 6 [13, 2, 6]
    add 9 [13, 2, 6, 9]
del 13
    add 15 [2, 6, 9, 15]
    add 16 [2, 6, 9, 15, 16]
    add 18 [2, 6, 9, 15, 16, 18]
del 2
    add 3 [6, 9, 15, 16, 18, 3]
del 6
    add 12 [9, 15, 16, 18, 3, 12]
del 9
del 15
    add 11 [16, 18, 3, 12, 11]
    add 14 [16, 18, 3, 12, 11, 14]
    add 19 [16, 18, 3, 12, 11, 14, 19]
del 16
del 18
del 3
del 12
del 11
del 14
    add 20 [19, 20]
del 19
del 20
    add 17 [17]
del 17
Reihenfolge: 1,5,4,7,10,8,13,2,6,9,15,16,18,3,12,11,14,19,20,17

```

- (b) Geben Sie die Reihenfolge an, in der die Knoten besucht werden, wenn auf dem folgenden Graphen *Tiefensuche* ausgehend vom Knoten 1 ausgeführt wird. Wenn mehrere Knoten zur Wahl stehen, wählen Sie den Knoten mit dem kleinsten Schlüssel.



Rekursive Tiefensuche:

```
add 1
add 2
add 3
add 6
add 5
    exit 5
add 7
add 12
add 11
add 14
add 9
add 10
    exit 10
    exit 9
    exit 14
    exit 11
add 15
    exit 15
    exit 12
    exit 7
    exit 6
    exit 3
add 4
add 8
add 13
add 16
add 17
add 19
    exit 19
    exit 17
add 18
add 20
    exit 20
    exit 18
    exit 16
```

```
exit 13
exit 8
exit 4
exit 2
exit 1
```

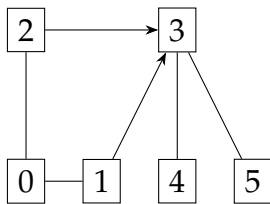
Reihenfolge: 1,2,3,6,5,7,12,11,14,9,10,15,4,8,13,16,17,19,18,20

Mit Stapel

```

    add 1  [1]
del 1
    add 2  [2]
del 2
    add 3  [3]
    add 4  [4, 3]
    add 5  [5, 4, 3]
    add 8  [8, 5, 4, 3]
    add 9  [9, 8, 5, 4, 3]
del 9
    add 10 [10, 8, 5, 4, 3]
    add 14 [14, 10, 8, 5, 4, 3]
del 14
    add 11 [11, 10, 8, 5, 4, 3]
del 11
    add 12 [12, 10, 8, 5, 4, 3]
del 12
    add 6  [6, 10, 8, 5, 4, 3]
    add 7  [7, 6, 10, 8, 5, 4, 3]
    add 15 [15, 7, 6, 10, 8, 5, 4, 3]
del 15
del 7
del 6
del 10
del 8
    add 13 [13, 5, 4, 3]
del 13
    add 16 [16, 5, 4, 3]
del 16
    add 17 [17, 5, 4, 3]
    add 18 [18, 17, 5, 4, 3]
del 18
    add 20 [20, 17, 5, 4, 3]
del 20
del 17
    add 19 [19, 5, 4, 3]
del 19
del 5
del 4
del 3
```

Reihenfolge: 1,2,3,4,5,8,9,10,14,11,12,6,7,15,13,16,17,18,20,19

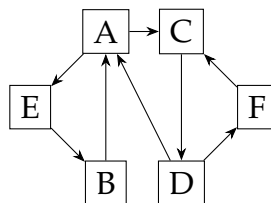
Aufgabe 8³

Following is Depth First Traversal:

0 1 3 4 5 2

46114 / 2006 / Frühjahr / Thema 2 / Aufgabe 6**Aufgabe 6 (Graphrepräsentation)**

Repräsentieren Sie den folgenden Graphen sowohl mit einer Adjazenzmatrix als auch mit einer Adjazenzliste.



Lösungsvorschlag

$$\begin{array}{c}
 A \quad B \quad C \quad D \quad E \quad F \\
 \begin{array}{l}
 A \\
 B \\
 C \\
 D \\
 E \\
 F
 \end{array}
 \begin{pmatrix}
 * & - & 1 & - & 1 & - \\
 1 & * & - & - & - & - \\
 - & - & * & 1 & - & - \\
 1 & - & - & * & - & 1 \\
 - & 1 & - & - & * & - \\
 - & - & 1 & - & - & *
 \end{pmatrix}
 \end{array}$$

³<https://favtutor.com/blogs/depth-first-search-java>

$A \rightarrow C \rightarrow E$
 $B \rightarrow A$
 $C \rightarrow D$
 $D \rightarrow A \rightarrow F$
 $E \rightarrow B$
 $F \rightarrow C$

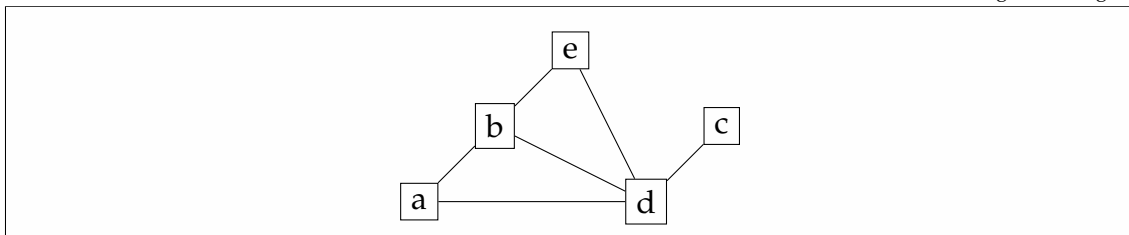
46114 / 2008 / Herbst / Thema 1 / Aufgabe 2

Gegeben sei folgender Graph:

$V: \{a, b, c, d, e\}$
 $E: a \rightarrow a, b$
 $b \rightarrow b, d, e$
 $c \rightarrow c, d$
 $d \rightarrow a, e$

(a) Stellen Sie den Graphen grafisch dar!

Lösungsvorschlag



(b) Berechnen Sie mit dem Algorithmus von Dijkstra schrittweise die Länge der kürzesten Pfade ab dem Knoten a! Nehmen Sie dazu an, dass alle Kantengewichte 1 sind. Erstellen Sie eine Tabelle gemäß folgendem Muster:

ausgewählt | a | b | c | d | e

Ergebnis:

Hinweis: Nur mit Angabe der jeweiligen Zwischenschritte gibt es Punkte. Es reicht also nicht, nur das Endergebnis hinzuschreiben.

Lösungsvorschlag

Nr.	ausgewählt	a	b	c	d	e
1	a	0	1	∞	1	∞
2	b		1	∞	1	2
3	d			2	1	2
4	c			2		2
5	e					2

(c) Welchen Aufwand hat der Algorithmus von Dijkstra bei Graphen mit $|V|$ Knoten und $|E|$ Kanten,

- wenn die Kantengewichte alle 1 sind? Mit welcher Datenstruktur und welchem Vorgehen lässt sich der Aufwand in diesem Fall reduzieren (mit kurzer Begründung)?
- wenn die Kantengewichte beliebig sind und als Datenstruktur eine Halde verwendet wird (mit kurzer Begründung)?

Aufgabe 10: Graphen I

Gegeben seien folgende ungerichtete Graphen in textueller Notation, wobei die erste Menge die Menge der Knoten und die zweite Menge die Menge der Kanten ist:

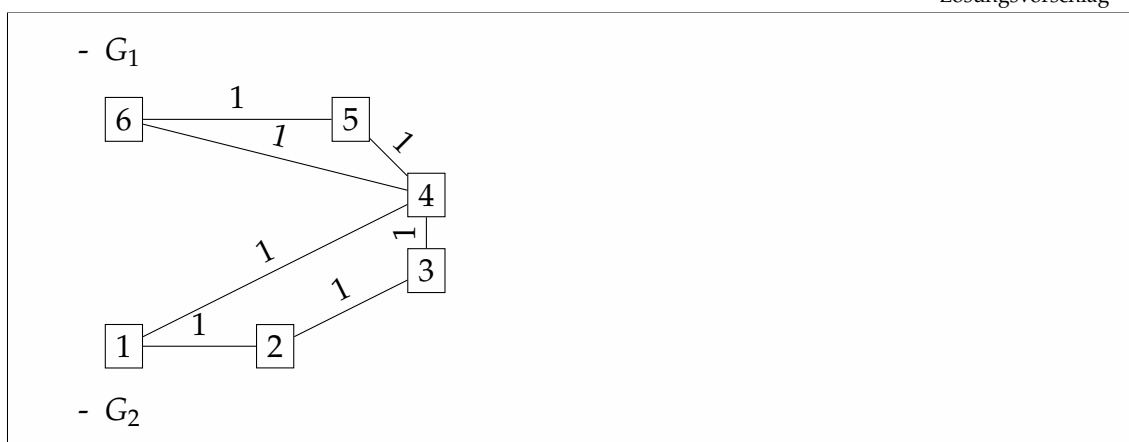
$$G_1 = (\{1, 2, 3, 4, 5, 6\}, \{[1, 2], [1, 4], [2, 3], [3, 4], [4, 5], [4, 6], [5, 6]\})$$

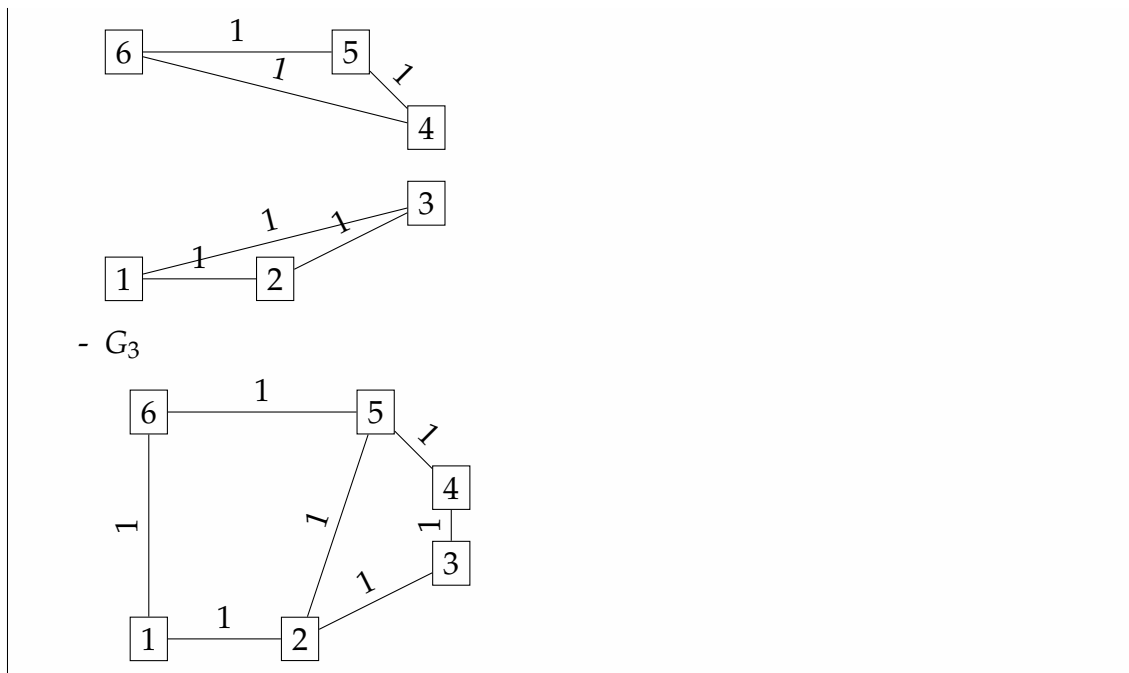
$$G_2 = (\{1, 2, 3, 4, 5, 6\}, \{[1, 2], [1, 3], [2, 3], [4, 5], [4, 6], [5, 6]\})$$

$$G_3 = (\{1, 2, 3, 4, 5, 6\}, \{[1, 2], [1, 6], [2, 3], [2, 5], [3, 4], [4, 5], [5, 6]\})$$

(a) Zeichnen Sie die obigen Graphen.

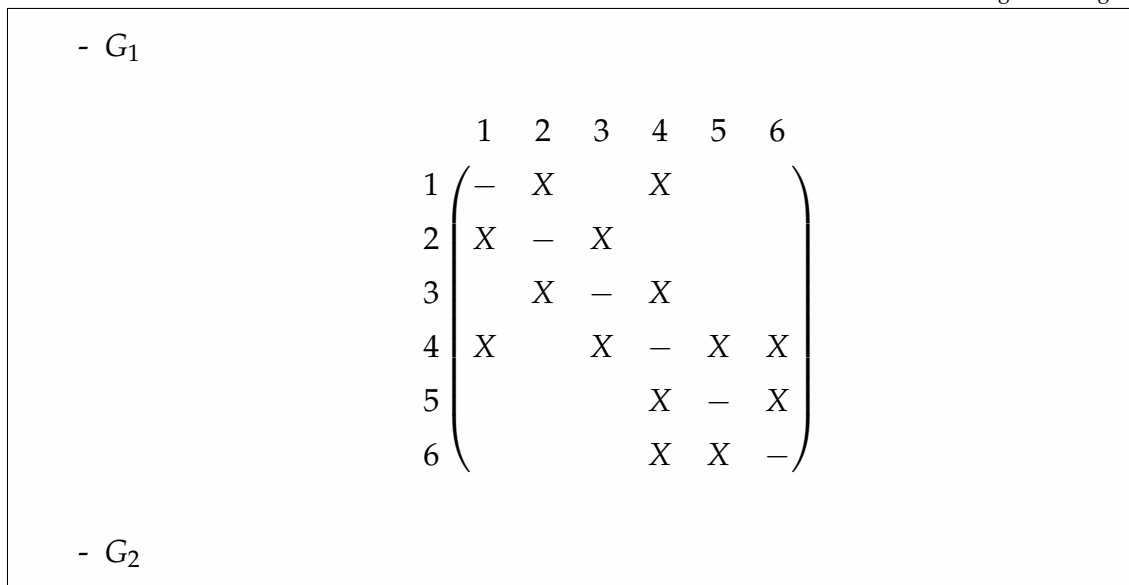
Lösungsvorschlag





- (b) Erstellen Sie zu jedem Graphen die zugehörige Adjazenzmatrix mit X als Symbol für eine eingetragene Kante.

Lösungsvorschlag

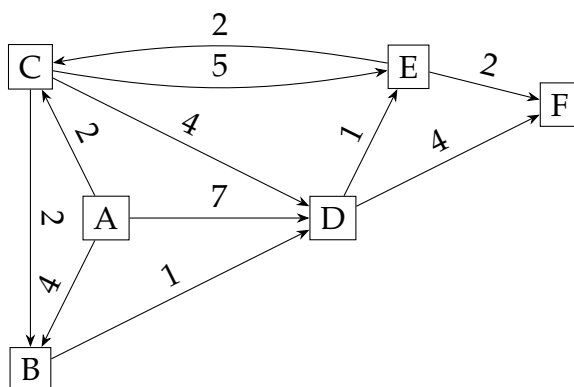


	1	2	3	4	5	6
1	—	X	X			
2	X	—	X			
3	X	X	—			
4				—	X	X
5				X	—	X
6				X	X	—

- G_3

	1	2	3	4	5	6
1	—	X			X	X
2	X	—	X			
3		X	—	X		
4			X	—	X	
5		X		X	—	X
6	X				X	—

(c) Betrachten Sie nun folgenden gerichteten Graphen G_4 :



Bestimmen Sie die kürzeste Entfernung von Knoten A zu jedem anderen Knoten des Graphen. Verwenden Sie dazu den Algorithmus von Dijkstra und tragen Sie Ihre einzelnen Rechenschritte in eine Tabelle folgender Form ein (schreiben Sie neben jede Zeile die Prioritätswarteschlange der noch zu bearbeitenden Knoten, priorisiert nach ihren Wegkosten):

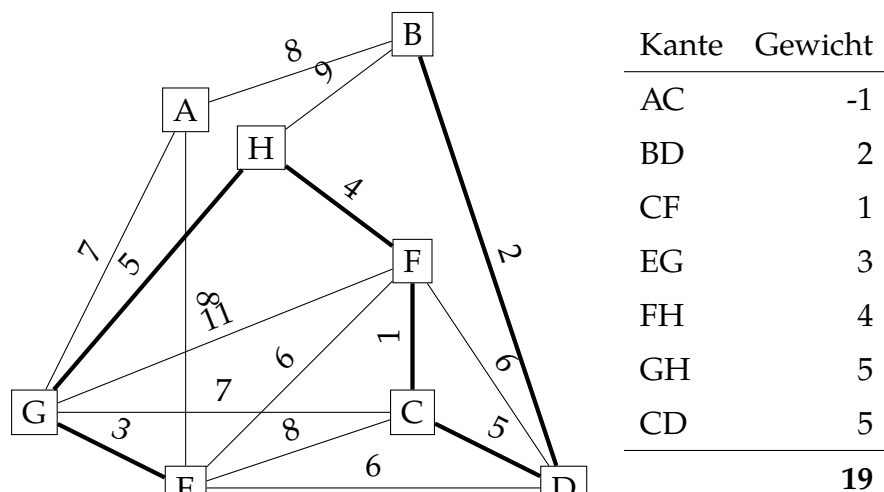
Hinweis: Mit den „Wegkosten“ eines Knotens ist die gegenwärtige Entfernung dieses Knotens vom Startknoten gemeint.

A	B	C	D	E	F	Warteschlange
0	∞	∞	∞	∞	∞	A
	4 (A)	2 (A)	7 (A)	∞	∞	C, B, D
			6 (C)	7 (C)	∞	B, D, E
			5 (B)	7 (C)	∞	D, E
				6 (D)	9 (D)	E, F
					8 (E)	F

Frühjahr 2014 (46115) - Thema 1 Aufgabe 8

Bestimmen Sie einen minimalen Spannbaum für einen ungerichteten Graphen, der durch die nachfolgende Entfernungsmatrix gegeben ist! Die Matrix ist symmetrisch und ∞ bedeutet, dass es keine Kante gibt. Zeichnen Sie den Graphen und geben Sie die Spannbaumkanten ein !

$$\begin{array}{c}
 \begin{array}{cccccccc}
 & A & B & C & D & E & F & G & H \\
 \begin{array}{l} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} & \left(\begin{array}{cccccccc}
 0 & 8 & -1 & \infty & 8 & \infty & 7 & \infty \\
 8 & 0 & \infty & 2 & \infty & \infty & \infty & 9 \\
 -1 & \infty & 0 & 5 & 8 & 1 & 7 & \infty \\
 \infty & 2 & 5 & 0 & 6 & 6 & \infty & \infty \\
 8 & \infty & 8 & 6 & 0 & 6 & 3 & \infty \\
 \infty & \infty & 1 & 6 & 6 & 0 & 11 & 4 \\
 7 & \infty & 7 & \infty & 3 & 11 & 0 & 5 \\
 \infty & 9 & \infty & \infty & \infty & 4 & 5 & 0
 \end{array} \right)
 \end{array}
 \end{array}$$



Nach dem Algorithmus von Kruskal wählt man aus den noch nicht gewählten Kanten immer die kürzeste, die keinen Kreis mit den bisher gewählten Kanten bildet.

46115 / 2018 / Frühjahr / Thema 1 / Aufgabe 8

Berechnen Sie mithilfe des Algorithmus von Prim ausgehend vom Knoten s einen minimalen Spannbaum des ungerichteten Graphen G , der durch folgende Adjazenzmatrix gegeben ist:

	s	a	b	c	d	e	f	g	h
s	*	—	3	—	—	7	—	—	—
a	—	*	—	0	8	—	11	—	—
b	3	—	*	—	5	—	10	—	—
c	—	0	—	*	—	—	1	—	—
d	—	8	5	—	*	2	3	—	6
e	7	—	—	—	2	*	—	—	11
f	—	11	10	1	3	—	*	7	—
g	—	—	—	—	—	—	7	*	4
h	—	—	—	—	6	11	—	4	*

- (a) Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte denjenigen Knoten v , der vom Algorithmus als nächstes in den Ergebnisbaum aufgenommen wird (dieser sog. schwarze Knoten ist damit fertiggestellt) als Tripel (v, p, δ) mit v als Knotenname, p als aktueller Vorgängerknoten und δ als aktuelle Distanz von v zu p an. Führen Sie in der zweiten Spalte alle anderen vom aktuellen Spannbaum direkt erreichbaren Knoten v (sog. graue Randknoten) ebenfalls als Tripel (v, p, δ) auf. Zeichnen Sie anschließend den entstandenen Spannbaum und geben Sie sein Gewicht an.

Der Graph muss nicht gezeichnet werden. Der Algorithmus kann auch nur mit der Adjazenzmatrix durchgeführt werden. Möglicherweise geht das Lösen der Aufgabe schneller mit der Matrix von der Hand.

Kompletter Graph

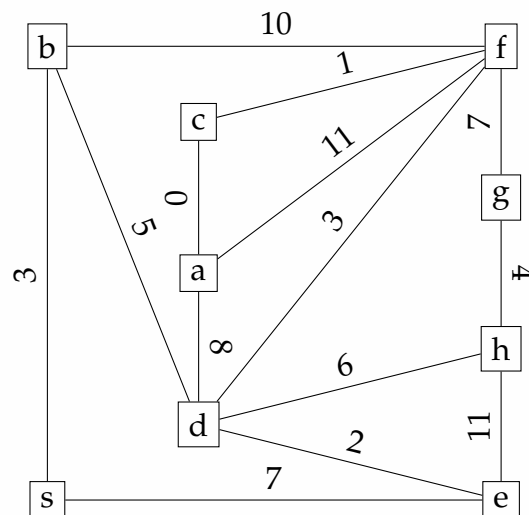
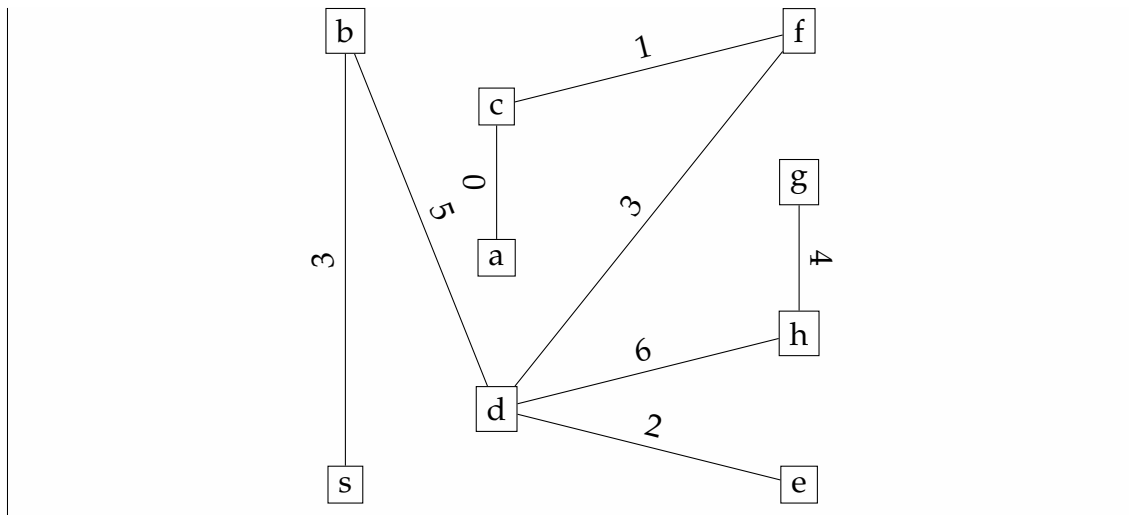


Tabelle schwarz-graue-Knoten

schwarze	graue
(s, null, -)	(b, s, 3); (e, s, 7);
(b, s, 3)	(d, b, 5); (e, s, 7); (f, b, 10);
(d, b, 5)	(a, d, 8); (e, d, 2); (f, d, 3); (h, d, 6);
(e, d, 2)	(a, d, 8); (f, d, 3); (h, d, 6);
(f, d, 3)	(a, d, 8); (c, f, 1); (g, f, 7); (h, d, 6);
(c, f, 1)	(a, c, 0); (g, f, 7); (h, d, 6);
(a, c, 0)	(g, f, 7); (h, d, 6);
(h, d, 6)	(g, h, 4);
(g, h, 4)	

Gewicht des minimalen Spannbaums: 24

Minimaler Spannbaum



Algorithmus von Kruskal
Minimaler Spannbaum
Algorithmus von Prim

- (b) Welche Worst-Case-Laufzeitkomplexität hat der Algorithmus von Prim, wenn die grauen Knoten in einem Heap nach Distanz verwaltet werden? Sei dabei n die Anzahl an Knoten und m die Anzahl an Kanten des Graphen. Eine Begründung ist nicht erforderlich.

Lösungsvorschlag

$$\mathcal{O}(m + n \log n)$$

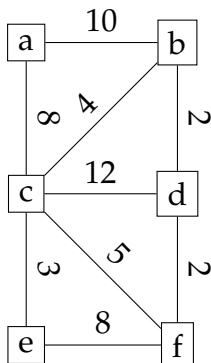
- (c) Beschreiben Sie kurz die Idee des alternativen Ansatzes zur Berechnung eines minimalen Spannbaumes von Kruskal.

Lösungsvorschlag

Kruskal wählt nicht die kürzeste an einen Teilgraphen anschließende Kante, sondern global die kürzeste verbliebene aller Kanten, die keinen Zyklus bildet, ohne dass diese mit dem Teilgraph verbunden sein muss.

46115 / 2018 / Frühjahr / Thema 2 / Aufgabe 4

Sei G der folgende Graph.



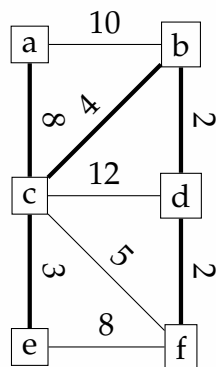
- (a) Der Algorithmus von Prim ist ein Algorithmus zur Bestimmung des minimalen Spannbaums in einem Graphen. Geben Sie einen anderen Algorithmus zur Bestimmung des minimalen Spannbaums an.

Zum Beispiel der Algorithmus von Kruskal

- (b) Führen Sie den Algorithmus von Prim schrittweise auf G aus. Ausgangsknoten soll der Knoten a sein. Ihre Tabelle sollte wie folgt beginnen:

a	b	c	d	e	f	Warteschlange
---	---	---	---	---	---	---------------

Die Einträge der Tabelle geben an, wie weit der angegebene Knoten vom aktuellen Baum entfernt ist.



a	b	c	d	e	f	Warteschlange
0	∞	∞	∞	∞	∞	a
0	10	8	∞	∞	∞	c, b
0	4	0	12	3	5	e, b, f, d
0	4	0	12	0	5	b, f, d
0	0	0	2	0	5	d, f
0	0	0	0	0	2	f
0	0	0	0	0	0	

- (c) Erklären Sie, warum der Kürzeste-Wege-Baum (also das gezeichnete Ergebnis des Dijkstra-Algorithmus) und der minimale Spannbaum nicht notwendigerweise identisch sind.

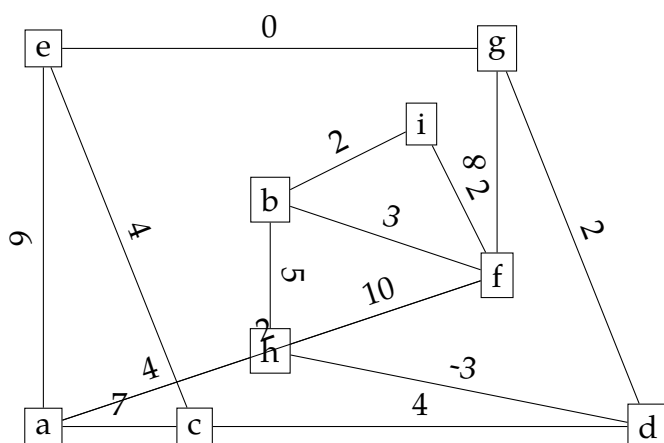
Die Wahl der nächsten Kante erfolgt nach völlig verschiedenen Kriterien:

- Beim Kürzeste-Wege-Baum orientiert sie sich an der Entfernung der einzelnen Knoten vom Startknoten.
- Beim Spannbaum orientiert sie sich an der Entfernung der einzelnen Knoten vom bereits erschlossenen Teil des Spannbaums.

46115 / 2019 / Herbst / Thema 2 / Aufgabe 8

- (a) Durch folgende Adjazenzmatrix sei ein ungerichteter Graph G mit Kantenlängen gegeben.

	a	b	c	d	e	f	g	h	i
a	0	7	0	9	2	0	4	0	
b	0	0	0	0	3	0	5	2	
c	7	0	0	4	4	0	0	0	
d	0	0	4	0	0	0	2	-3	
e	9	0	4	0	0	0	0	0	
f	2	3	0	0	0	0	8	10	
g	0	0	0	2	0	8	0	0	
h	4	5	0	-3	0	10	0	0	
i	0	2	0	0	0	2	0	0	



Wenden Sie den Algorithmus von Jarník/Prim auf G ausgehend von Knoten d an, um einen Spannbaum T mit *maximalem* Gewicht zu berechnen. Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte denjenigen Knoten v an, der vom Algorithmus als nächstes in T aufgenommen wird (dieser sog. „schwarze“ Knoten ist damit fertiggestellt). Führen Sie in der zweiten Spalte alle anderen vom aktuellen Baum T direkt erreichbaren Knoten v (sog. „graue Randknoten“) auf.

Geben Sie in der Tabelle Knoten stets als Tripel $(v, \delta, v.\pi)$ an, mit v als Knotenname, $v.\pi$ als aktueller Vorgängerknoten (anderer Knoten der Kante) und δ als Länge der Kante $\{v, v.\pi\}$.

- (b) Sei $G = (V, E, w)$ ein Graph mit Kantenlängen $w : E \rightarrow \mathbb{N}$ und T ein Spannbaum

von G mit maximalem Gewicht. Beweisen oder widerlegen Sie die folgende Aussage:

Längste (einfache) Wege zwischen zwei Knoten $u, v \in V$ enthalten nur Kanten aus T .

46115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 3

Wir betrachten eine Variante der Breitensuche (BFS), bei der die Knoten markiert werden, wenn sie das erste Mal besucht werden. Außerdem wird die Suche einmal bei jedem unmarkierten Knoten gestartet, bis alle Knoten markiert sind. Wir betrachten gerichtete Graphen. Ein gerichteter Graph G ist *schwach zusammenhängend*, wenn der ungerichtete Graph (der sich daraus ergibt, dass man die Kantenrichtungen von G ignoriert) zusammenhängend ist.

Exkurs: Schwach zusammenhängend gerichteter Graph

Beim gerichteten Graphen musst du auf die Kantenrichtung achten. Würde man die Richtungen der Kanten ignorieren wäre aber trotzdem jeder Knoten erreichbar. Einen solchen Graphen nennt man schwach zusammenhängend.^a

Ein gerichteter Graph heißt (schwach) zusammenhängend, falls der zugehörige ungerichtete Graph (also der Graph, der entsteht, wenn man jede gerichtete Kante durch eine ungerichtete Kante ersetzt) zusammenhängend ist.^b

^a<https://studyflix.de/informatik/grundbegriffe-der-graphentheorie-1285>

^b[https://de.wikipedia.org/wiki/Zusammenhang_\(Graphentheorie\)](https://de.wikipedia.org/wiki/Zusammenhang_(Graphentheorie))

- (a) Beschreiben Sie für ein allgemeines $n \in \mathbb{N}$ mit $n \geq 2$ den Aufbau eines schwach zusammenhängenden Graphen G_n , mit n Knoten, bei dem die Breitensuche $\Theta(n)$ mal gestartet werden muss, bis alle Knoten markiert sind.

Lösungsvorschlag

?

Die Breitensuche benötigt einen Startknoten. Die unten aufgeführten Graphen finden immer nur einen Knoten nämlich den Startknoten.

Oder so:

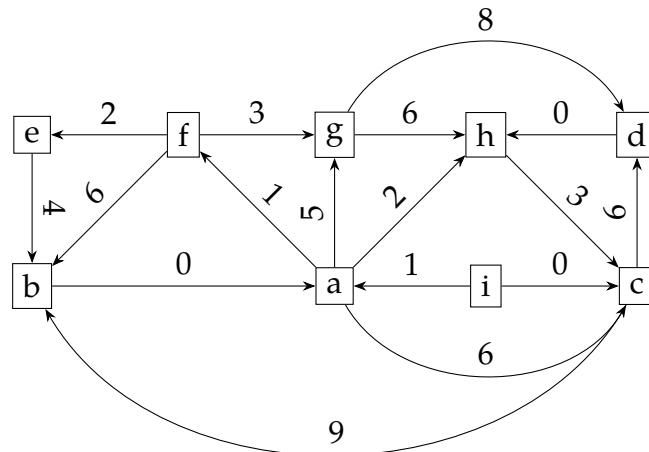
```

    graph TD
      B[B] --> A[A]
      D[D] --> A
      C[C] --> A
  
```

- (b) Welche asymptotische Laufzeit in Abhängigkeit von der Anzahl der Knoten (n) und von der Anzahl der Kanten (m) hat die Breitensuche über alle Neustarts zusammen? Beachten Sie, dass die Markierungen nicht gelöscht werden. Geben Sie die Laufzeit in Θ -Notation an. Begründen Sie Ihre Antwort.

46115 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 4

- (a) Berechnen Sie im gegebenen gerichteten und gewichteten Graph $G = (V, E, w)$ mit Kantenlängen $w : E \rightarrow \mathbb{R}_0^+$ mittels des Dijkstra-Algorithmus die kürzesten (gerichteten) Pfade ausgehend vom Startknoten a .



Knoten, deren Entfernung von a bereits feststeht, seien als schwarz bezeichnet und Knoten, bei denen lediglich eine obere Schranke $\neq \infty$ für ihre Entfernung von a bekannt ist, seien als *grau* bezeichnet.

- (i) Geben Sie als Lösung eine Tabelle an. Fügen Sie jedes mal, wenn der Algorithmus einen Knoten schwarz färbt, eine Zeile zu der Tabelle hinzu. Die Tabelle soll dabei zwei Spalten beinhalten: die linke Spalte zur Angabe des aktuell schwarz gewordenen Knotens und die rechte Spalte mit der bereits aktualisierten Menge grauer Knoten. Jeder Tabelleneintrag soll anstelle des nackten Knotennamens v ein Tripel $(v, v.d, v.\pi)$ sein. Dabei steht $v.d$ für die aktuell bekannte kürzeste Distanz zwischen a und v . $v.\pi$ ist der direkte Vorgänger von v auf dem zugehörigen kürzesten Weg von a .

Lösungsvorschlag

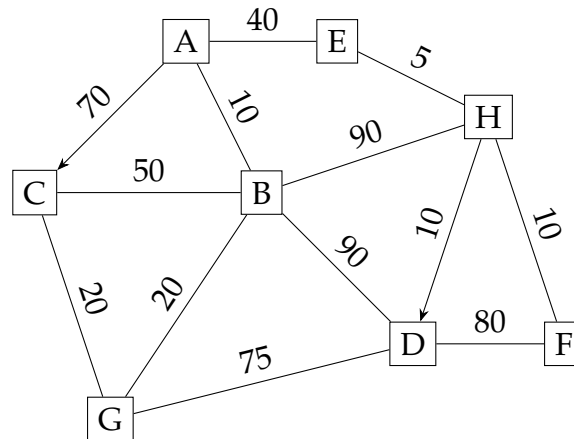
Nr.	besucht	a	b	c	d	e	f	g	h	i
0		0	∞	∞	∞	∞	∞	∞	∞	∞
1	a	0	∞	6	∞	∞	1	5	2	∞
2	f		7	6	∞	3	1	4	2	∞
3	h		7	5	∞	3		4	2	∞
4	e		7	5	∞	3		4		∞
5	g		7	5	12			4		∞
6	c		7	5	11					∞
7	b		7		11					∞
8	d				11					∞
9	i									∞
nach	Entfernung	Reihenfolge		Pfad						
a \rightarrow a	0	1								
a \rightarrow b	7	7		a \rightarrow f \rightarrow b						
a \rightarrow c	5	6		a \rightarrow h \rightarrow c						
a \rightarrow d	11	8		a \rightarrow h \rightarrow c \rightarrow d						
a \rightarrow e	3	4		a \rightarrow f \rightarrow e						
a \rightarrow f	1	2		a \rightarrow f						
a \rightarrow g	4	5		a \rightarrow f \rightarrow g						
a \rightarrow h	2	3		a \rightarrow h						
a \rightarrow i	2.147483647E9	9		a \rightarrow i						

- (ii) Zeichnen Sie zudem den entstandenen Kürzeste-Pfade-Baum.
- (b) Warum berechnet der Dijkstra-Algorithmus auf einem gerichteten Eingabegraphen mit potentiell auch negativen Kantengewichten $w : E \rightarrow \mathbb{R}$ nicht immer einen korrekten Kürzesten-Wege-Baum von einem gewählten Startknoten aus? Geben Sie ein Beispiel an, für das der Algorithmus die falsche Antwort liefert.
- (c) Begründen Sie, warum das Problem nicht gelöst werden kann, indem der Betrag des niedrigsten (also des betragsmäßig größten negativen) Kantengewichts im Graphen zu allen Kanten addiert wird.

66112 / 2004 / Frühjahr / Thema 1 / Aufgabe 5

Ein wichtiges Problem im Bereich der Graphalgorithmen ist die Berechnung kürzester Wege. Gegeben sei der folgende Graph, in dem Städte durch Kanten verbunden

sind. Die Kantengewichte geben Fahrzeiten an. Außer den durch Pfeile als nur in eine Richtung befahrbar gekennzeichneten Straßen sind alle Straßen in beiden Richtungen befahrbar.



- (a) Geben Sie zu dem obigen Graphen zunächst eine Darstellung als Adjazenzmatrix an.

Lösungsvorschlag

$$\begin{array}{c}
 \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array}
 \begin{pmatrix}
 & A & B & C & D & E & F & G & H \\
 * & 10 & 70 & - & 40 & - & - & - \\
 10 & * & 50 & 90 & - & - & 20 & 90 \\
 - & 50 & * & - & - & - & 20 & - \\
 - & 90 & - & * & - & 80 & 75 & - \\
 40 & - & - & - & * & - & - & 5 \\
 - & - & - & 80 & - & * & - & 10 \\
 - & 20 & 20 & 75 & - & - & * & - \\
 - & 90 & - & 10 & 5 & 10 & - & *
 \end{pmatrix}
 \end{array}$$

- (b) Berechnen Sie nun mit Hilfe des Algorithmus von Dijkstra die kürzesten Wege vom Knoten A zu allen anderen Knoten.

Lösungsvorschlag

Nr.	besucht	A	B	C	D	E	F	G	H
0		0	∞	∞	∞	∞	∞	∞	∞
1	A	0	10	70	∞	40	∞	∞	∞
2	B		10	60	100	40	∞	30	100
3	G			50	100	40	∞	30	100
4	E			50	100	40	∞		45
5	H			50	55		55		45
6	C			50	55		55		
7	D				55		55		
8	F						55		

nach	Entfernung	Reihenfolge	Pfad
A \rightarrow A	0	1	
A \rightarrow B	10	2	A \rightarrow B
A \rightarrow C	50	6	A \rightarrow B \rightarrow G \rightarrow C
A \rightarrow D	55	7	A \rightarrow E \rightarrow H \rightarrow D
A \rightarrow E	40	4	A \rightarrow E
A \rightarrow F	55	8	A \rightarrow E \rightarrow H \rightarrow F
A \rightarrow G	30	3	A \rightarrow B \rightarrow G
A \rightarrow H	45	5	A \rightarrow E \rightarrow H

66115 / 2012 / Frühjahr / Thema 1 / Aufgabe 7

Mit der Länge eines Pfads oder eines Kreises bezeichnen wir die Anzahl der Kanten, aus denen der Pfad bzw. der Kreis besteht. Bekanntlich kann man Breitensuche verwenden, um für zwei gegebene Knoten s und t die Länge eines kürzesten s - t -Wegs zu berechnen. Im folgenden geht es um die Berechnung kürzester Kreise.

- (a) Für einen Graphen G und einen Knoten v von G berechnet $KK(G,v)$ (siehe Abbildung 1) die Länge des kürzesten Kreises in G , der durch v geht.

Analysieren Sie die Laufzeit von KK in Abhängigkeit von der Anzahl n der Knoten von G , von der Anzahl m der Kanten von G und vom Grad $\deg(v)$ des übergebenen Knotens v .

- (b) Wenn man den Algorithmus KK für jeden Knoten eines Graphen G aufruft, kann man die Länge eines kürzesten Kreises in G berechnen. Welche Laufzeit hat der resultierende Algorithmus in in Abhängigkeit von n und m ?

- (c) Geben Sie einen Algorithmus $\text{KKschnell}(G, v)$ an, der in $O(n + m)$ Zeit die Länge des kürzesten Kreises in G berechnet, der durch v geht. Argumentieren Sie, warum ihr Algorithmus korrekt ist.

Abbildung 1

vum pam aba ee aan mn a sr lee

KK(ungerichteter UNBEWICHLELEN rap1

1L= 2 Adj): =weV | v,w € E 8 foreach w € Adjlvj do

4 | Sei G' der Graph G ohne die Kante v, w .

5 Sei L die Länge eines kürzesten v - w -Wegs in G' . 6 if $2 < L$ then

7 | Lex

s return L

66115 / 2013 / Frühjahr / Thema 2 / Aufgabe 5

Drei Missionare und drei Kannibalen befinden sich am Ufer eines Flusses und möchten diesen überqueren. Dazu steht ihnen ein Boot zur Verfügung, das ein oder zwei Personen befördern kann. Verbleiben an einem Ufer mehr Kannibalen als Missionare, so werden die Missionare verspeist. Die Frage besteht nun darin, wie alle Personen unverseht auf die andere Seite des Flusses gelangen.

Im Anfangszustand befinden sich alle Personen und das Boot auf einer Seite des Flusses. Nehmen Sie an, es sei die linke Seite des Flusses. Im Zielzustand befinden sich alle Personen und das Boot auf der anderen Seite des Flusses. Jeden Zustand kann man durch folgendes Fünftupel beschreiben:

Missionareyngs X Kannibalen;;,,xs X Missionareyeens X Kannibalenyechis X Boolposition

Dabei werden die Elemente für Missionare und Kannibalen durch natürliche Zahlen und die Bootsposition durch einen der Strings „links“ oder „rechts“ modelliert. Der Anfangszustand wäre somit also $(3, 3, 0, 0, \text{» links"})$ und der Zielzustand $(0, 0, 3, 3, \text{» rechts"})$.

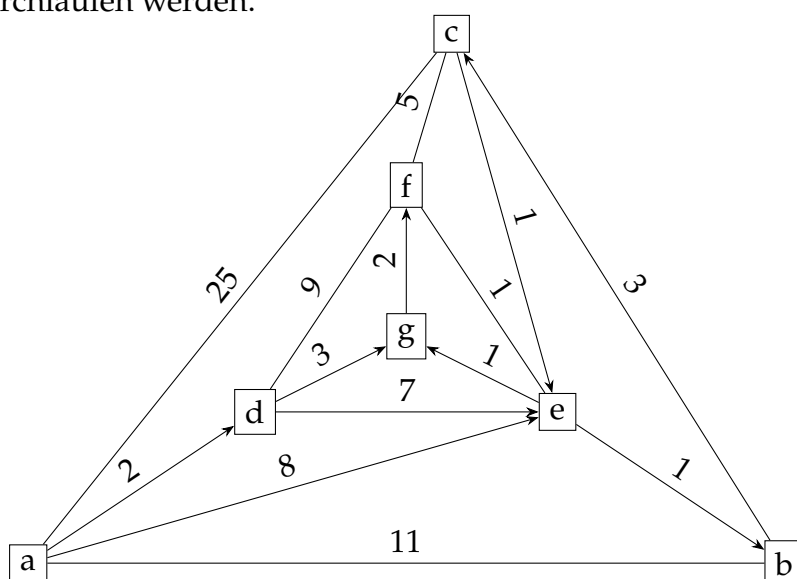
Schreiben Sie Algorithmen zur Lösung dieses Suchproblems und retten Sie die Missionare! Es ist empfehlenswert (aber nicht zwingend), hierzu eine funktionale Programmiersprache zu verwenden. Gehen Sie im Einzelnen vor, wie folgt:

- (a) a.) Schreiben Sie eine Funktion, die alle möglichen Überfahrten von links nach rechts oder umgekehrt modelliert, d. h. die zu einem gegebenen Zustand die Liste der möglichen Folgezustände im Sinne der o. g. Regeln berechnet. Gehen Sie dazu von allen möglichen Überfahrten aus und überprüfen Sie für jede konkrete Überfahrt mittels einer geeigneter Funktion, ob diese zu einem zulässigen Zustand führt. Zustände, die die Missionare nicht überleben, gelten im Sinne des Rettungsvorhabens ebenfalls als nicht zulässig. Nicht zulässige Zustände werden nicht in die Liste der möglichen Folgezustände eingefügt.
- (b) b.) Geben Sie eine Funktion an, die feststellt, ob ein Zyklus vorliegt, d. h. ob ein Zustand in einer Liste bereits besuchter Zustände schon enthalten ist.

- (c) c.) Verwenden Sie Ihre Ergebnisse aus a.) und b.), um eine Funktion anzugeben, die dieses Suchproblem mittels Breitensuche löst. (Sie können die Funktionen aus a.) und b.) hier auch dann verwenden, wenn Sie diese Teilaufgaben nicht vollständig gelöst haben.) Die Funktion erhält als Eingabe einen Start- und einen Zielzustand und liefert als Ergebnis die erste gefundene Liste von Zuständen, die das Problem löst.

66115 / 2013 / Herbst / Thema 2 / Aufgabe 9

Gegeben sei der unten stehende gerichtete Graph $G = (V, E)$ mit positiven Kantenlängen $l(e)$ für jede Kante $e \in E$. Kanten mit Doppelspitzen können in beide Richtungen durchlaufen werden.



- (a) In welcher Reihenfolge werden die Knoten von G ab dem Knoten a durch den Dijkstra-Algorithmus bei der Berechnung der kürzesten Wege endgültig bearbeitet?

Lösungsvorschlag

Nr.	besucht	a	b	c	d	e	f	g
0		0	∞	∞	∞	∞	∞	∞
1	a	0	11	25	2	8	∞	∞
2	d		11	25	2	8	11	5
3	g		11	25		8	7	5
4	f		11	12		8	7	
5	e		9	12		8		
6	b		9	12				
7	c			12				

(b) Berechnen Sie die Länge des kürzesten Weges von a zu jedem Knoten.

Lösungsvorschlag

siehe oben

(c) Geben Sie einen kürzesten Weg von a nach c an.

Lösungsvorschlag

$a \rightarrow d \rightarrow g \rightarrow f \rightarrow c$

66115 / 2015 / Frühjahr / Thema 2 / Aufgabe 7

Auf folgendem ungerichteten, gewichteten Graphen wurde der Dijkstra-Algorithmus (wie auf der nächsten Seite beschrieben) ausgeführt, doch wir wissen lediglich, welcher Knoten als letztes schwarz (black) wurde (Nr. 8) und was seine Distanz zum Startknoten (Nr. 1) ist. Die Gewichte der Kanten sind angegeben.

Finden Sie zunächst den Startknoten, nummerieren Sie anschließend die Knoten in der Reihenfolge, in der sie schwarz wurden, und geben Sie in jedem Knoten die Distanz zum Startknoten an.

Hinweis: Der Startknoten ist eindeutig.

Dijkstra(WeightedGraph G , Vertex s)

```
Initialize( $G$ ,  $s$ );
 $S = \emptyset$ ;
 $Q = \text{new PriorityQueue}(V, d)$ ;
while not  $Q.\text{Empty}()$  do
     $u = Q.\text{ExtractMin}()$ ;
     $S = S \cup \{u\}$ ;
    foreach  $v \in \text{Adj}[u]$  do
        Relax( $u$ ,  $v$ ;  $w$ );

     $u.\text{color} = \text{black}$ ;
```

Initialize(Graph G , Vertex s)

```
foreach  $u \in V$  do
     $u.\text{color} = \text{white}$ ;
     $u.d = \infty$ ;
 $s.\text{color} = \text{gray}$ ;
 $s.d = 0$ ;
```

Relax(u , v ; w)

```
if  $v.d > u.d + w(u, v)$  then
     $v.\text{color} = \text{gray}$ ;
     $v.d = u.d + w(u, v)$ ;
     $Q.\text{DecreaseKey}(v, v.d)$ ;
```

66115 / 2016 / Frühjahr / Thema 2 / Aufgabe 6

- (a) Berechnen Sie für folgenden Graphen den kürzesten Weg von Karlsruhe nach Kassel und dokumentieren Sie den Berechnungsweg:

Verwendete Abkürzungen:

A Augsburg

EF Erfurt

F Frankfurt

KA Karlsruhe

KS Kassel

M München

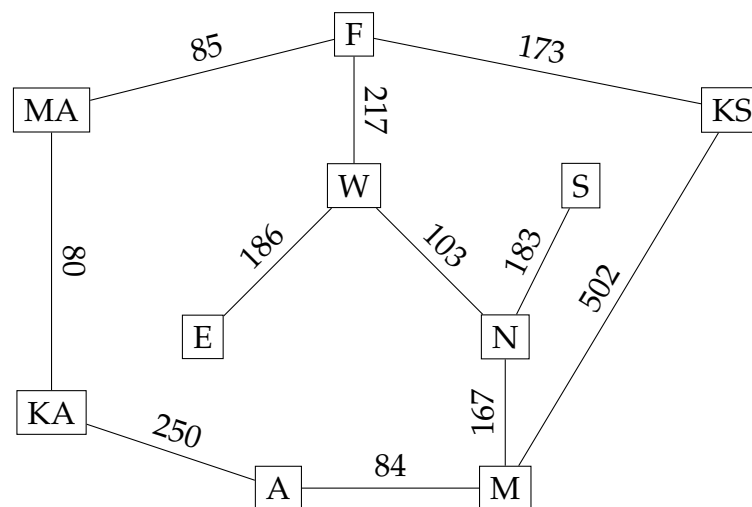
MA Mannheim

N Nürnberg

S Stuttgart

WÜ Würzburg

Zahl = Zahl in Kilometern



Lösungsvorschlag

Nr.	besucht	A	E	F	KA	KS	M	MA	N	S	W
0		∞	∞	∞	0	∞	∞	∞	∞	∞	∞
1	KA	250	∞	∞	0	∞	∞	80	∞	∞	∞
2	MA		∞	165		∞	∞	80	∞	∞	∞
3	F		∞	165		338	∞		∞	∞	382
4	A		∞			338	334		∞	∞	382
5	M		∞			338	334		501	∞	382
6	KS		∞			338			501	∞	382
7	W		568						485	∞	382
8	N		568						485	668	
9	E		568							668	
10	S									668	
nach	Entfernung	Reihenfolge		Pfad							
KA \rightarrow A	250	0		KA \rightarrow A							
KA \rightarrow E	568	9		KA \rightarrow MA \rightarrow F \rightarrow W \rightarrow E							
KA \rightarrow F	165	3		KA \rightarrow MA \rightarrow F							
KA \rightarrow KA	0	1									
KA \rightarrow KS	338	6		KA \rightarrow MA \rightarrow F \rightarrow KS							
KA \rightarrow M	334	5		KA \rightarrow A \rightarrow M							
KA \rightarrow MA	80	2		KA \rightarrow MA							
KA \rightarrow N	485	8		KA \rightarrow MA \rightarrow F \rightarrow W \rightarrow N							
KA \rightarrow S	668	10		KA \rightarrow MA \rightarrow F \rightarrow W \rightarrow N \rightarrow S							
KA \rightarrow W	382	7		KA \rightarrow MA \rightarrow F \rightarrow W							

- (b) Könnte man den Dijkstra Algorithmus auch benutzen, um das Travelling-Salesman Problem zu lösen?

66115 / 2017 / Frühjahr / Thema 1 / Aufgabe 1

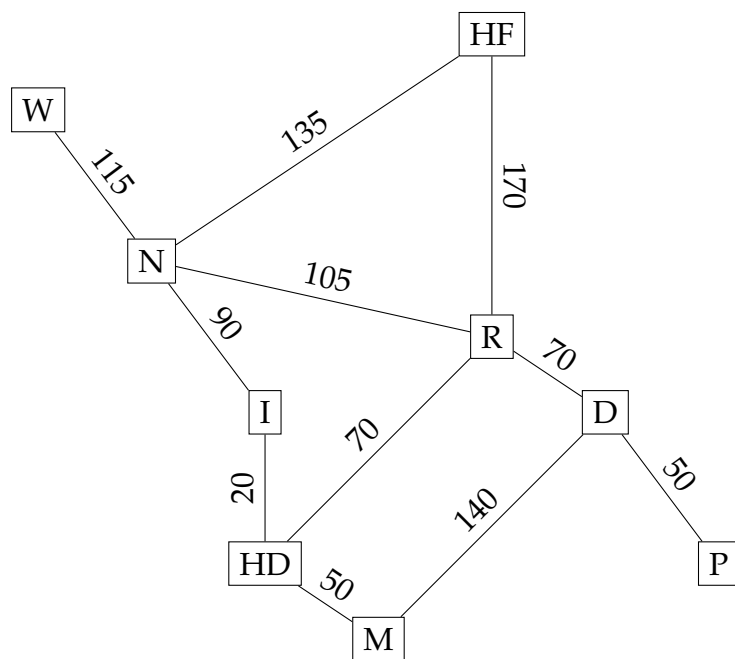
Die folgende Abbildung zeigt die wichtigsten bayerischen Autobahnen zusammen mit einigen anliegenden Orten und die Entfernungen zwischen diesen.

Entfernungstabelle

von	nach	km
Würzburg	Nürnberg	115
Nürnberg	Regensburg	105
Regensburg	AK Deggendorf	70
AK Deggendorf	Passau	50
Hof	Nürnberg	135
Nürnberg	Ingolstadt	90
Ingolstadt	AD Holledau	20
AD Holledau	München	50
München	AK Deggendorf	140
Hof	Regensburg	170
Regensburg	AD Holledau	70

Abkürzungen

D	Deggendorf
HF	Hof
HD	Holledau
I	Ingolstadt
M	München
N	Nürnberg
P	Passau
R	Regensburg
W	Würzburg



- (a) Bestimmen Sie mit dem Algorithmus von *Dijkstra* den kürzesten Weg von Ingolstadt zu allen anderen Orten. Verwenden Sie zur Lösung eine Tabelle gemäß folgendem Muster und markieren Sie in jeder Zeile den jeweils als nächstes zu betrachtenden Ort. Setzen Sie für die noch zu bearbeitenden Orte eine Prioritätswarteschlange ein, wobei gleicher Entfernung wird der ältere Knoten gewählt.

Lösungsvorschlag

Nr.	besucht	D	HD	HF	I	M	N	P	R	W
0		∞	∞	∞	0	∞	∞	∞	∞	∞
1	I	∞	20	∞	0	∞	90	∞	∞	∞
2	HD	∞	20	∞		70	90	∞	90	∞
3	M	210		∞		70	90	∞	90	∞
4	N	210		225			90	∞	90	205
5	R	160		225				∞	90	205
6	D	160		225				210		205
7	W			225				210		205
8	P			225				210		
9	HF			225						

- (b) Die bayerische Landesregierung hat beschlossen, die eben betrachteten Orte mit einem breitbandigen Glasfaser-Backbone entlang der Autobahnen zu verbinden. Dabei soll aus Kostengründen so wenig Glasfaser wie möglich verlegt werden. Identifizieren Sie mit dem Algorithmus von Kruskal diejenigen Strecken, entlang

welcher Glasfaser verlegt werden muss. Geben Sie die Ortspaare (Autobahnsegmente) in der Reihenfolge an, in der Sie sie in Ihre Verkabelungsliste aufnehmen.

Lösungsvorschlag

- (c) Um Touristen den Besuch aller Orte so zu ermöglichen, dass sie dabei jeden Autobahnabschnitt genau einmal befahren müssen, bedarf es zumindest eines sogenannten offenen Eulerzugs. Zwischen welchen zwei Orten würden Sie eine Autobahn bauen, damit das bayerische Autobahnnetz mindestens einen Euler-Pfad enthält?

Exkurs: offener Eulerzug

Ein offener Eulerzug ist gegeben, wenn Start- und Endknoten nicht gleich sein müssen, wenn also statt eines Zyklus lediglich eine Kantenfolge verlangt wird, welche jede Kante des Graphen genau einmal enthält. Ein bekanntes Beispiel ist das „Haus vom Nikolaus“.

Lösungsvorschlag

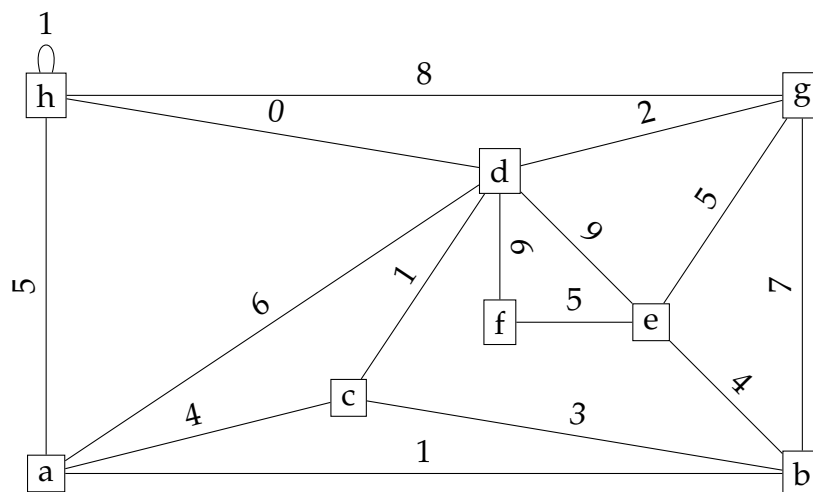
Zwischen Deggendorf und Würzburg

$P \rightarrow D \rightarrow R \rightarrow N \rightarrow \mathbf{W} \rightarrow \mathbf{D} \rightarrow M \rightarrow HD \rightarrow R \rightarrow HF \rightarrow N \rightarrow I \rightarrow HD$

66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 10

- (a) Berechnen Sie mithilfe des Algorithmus von Prim ausgehend vom Knoten a einen minimalen Spannbaum des ungerichteten Graphen G , der durch folgende Adjazenzmatrix gegeben ist:

$$\begin{array}{c}
 \begin{array}{cccccccc}
 & a & b & c & d & e & f & g & h \\
 a & * & 1 & 4 & 6 & - & - & - & 5 \\
 b & 1 & * & 3 & - & 4 & - & 7 & - \\
 c & 4 & 3 & * & 1 & - & - & - & - \\
 d & 6 & - & 1 & * & 9 & 6 & 2 & 0 \\
 e & - & 4 & - & 9 & * & 5 & 5 & - \\
 f & - & - & - & 6 & 5 & * & - & - \\
 g & - & 7 & - & 2 & 5 & - & * & 8 \\
 h & 5 & - & - & 0 & - & - & 8 & 1
 \end{array}
 \end{array}$$

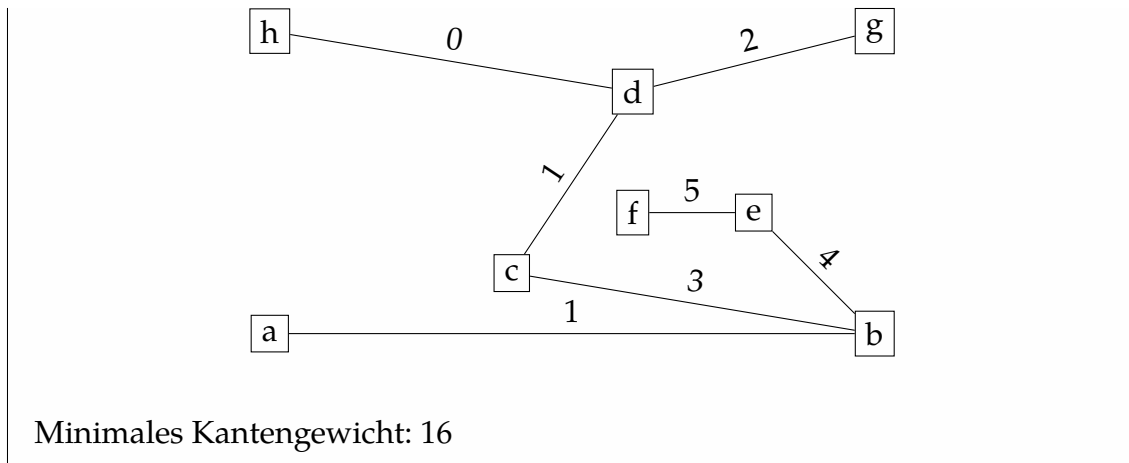


Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte denjenigen Knoten v , der vom Algorithmus als nächstes in den Ergebnisbaum aufgenommen wird (dieser sog. „schwarze“ Knoten ist damit fertiggestellt), als Tripel (v, p, δ) mit v als Knotenname, p als aktueller Vorgängerknoten und δ als aktuelle Distanz von v zu p an. Führen Sie in der zweiten Spalte alle anderen vom aktuellen Spannbaum direkt erreichbaren Knoten v (sog. „graue Randknoten“) ebenfalls als Tripel (v, p, δ) auf.

Zeichnen Sie anschließend den entstandenen Spannbaum und geben sein Gewicht an.

Lösungsvorschlag

„schwarze“	„graue“ Randknoten
(a, NULL, ∞)	$(b, a, 1) (c, a, 4) (h, a, 5) (d, a, 6)$
$(b, a, 1)$	$(c, b, 3) (e, b, 4) (h, a, 5) (d, a, 6) (g, b, 7)$
$(c, b, 3)$	$(d, c, 1) (e, b, 4) (h, a, 5) (g, b, 7)$
$(d, c, 1)$	$(h, d, 0) (g, d, 2) (e, b, 4) (f, d, 6)$
$(h, d, 0)$	$(g, d, 2) (e, b, 4) (f, d, 6)$
$(g, d, 2)$	$(e, b, 4) (f, d, 6)$
$(e, b, 4)$	$(f, e, 5)$
$(f, e, 5)$	



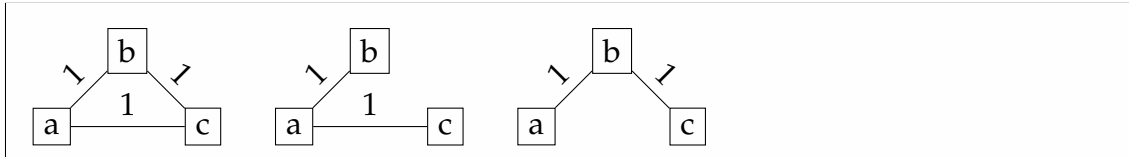
- (b) Welche Worst-Case-Laufzeitkomplexität hat der Algorithmus von Prim, wenn die grauen Knoten in einem Heap (= Halde) nach Distanz verwaltet werden? Sei dabei n die Anzahl an Knoten und m die Anzahl an Kanten des Graphen. Eine Begründung ist nicht erforderlich.

Lösungsvorschlag

$$\mathcal{O}(n \cdot \log(n) + m)$$

- (c) Zeigen Sie durch ein kleines Beispiel, dass ein minimaler Spannbaum eines ungerichteten Graphen nicht immer eindeutig ist.

Lösungsvorschlag



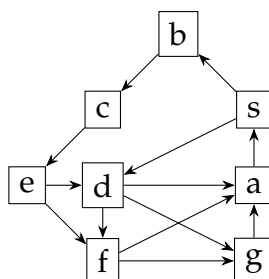
- (d) Skizzieren Sie eine Methode, mit der ein maximaler Spannbaum mit einem beliebigen Algorithmus für minimale Spannbäume berechnet werden kann. In welcher Laufzeitkomplexität kann ein maximaler Spannbaum berechnet werden?

Lösungsvorschlag

Alle Kantengewichte negieren. In $\mathcal{O}(n \cdot \log(n) + m)$ wie der Algorithmus von Prim.

66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 11

Gegeben sei der folgende gerichtete Graph G :



Traversieren Sie G ausgehend vom Knoten s mittels

- (a) Tiefensuche (DFS),

Lösungsvorschlag

Rekursiv ohne Keller:

0	1	2	3	4	5	6	7
s	b	c	e	d	a	f	g

- (b) Breitensuche (BFS)

Lösungsvorschlag

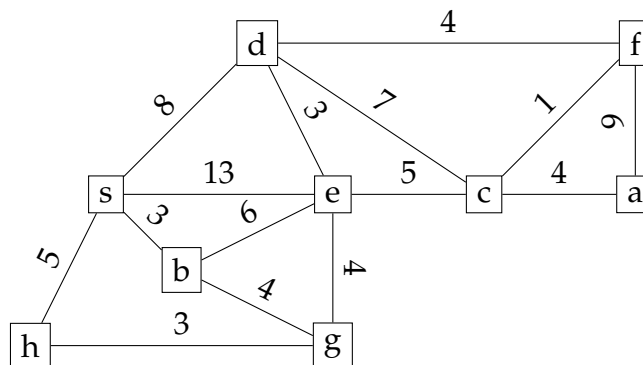
mit Warteschlange:

0	1	2	3	4	5	6	7
s	b	d	c	a	f	g	e

und geben Sie jeweils die erhaltene Nummerierung der Knoten an. Besuchen Sie die Nachbarn eines Knotens bei Wahlmöglichkeiten immer in alphabetisch aufsteigender Reihenfolge.

66115 / 2018 / Frühjahr / Thema 2 / Aufgabe 9

Gegeben sei folgender Graph G .



- (a) Berechnen Sie mithilfe des Algorithmus von Dijkstra die kürzesten Wege vom Knoten s zu allen anderen Knoten im Graphen G . Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte den jeweils als nächstes fertigzustellenden Knoten v (wird sog. „schwarz“) als Tripel (v, p, δ) mit v als Knotenname, p als aktueller Vorgängerknoten und δ als aktuelle Distanz von s zu v über p an. Führen Sie in der zweiten Spalten alle anderen bisher erreichten Knoten v ebenfalls als Tripel (v, p, δ) auf, wobei diese sog. „grauen Randknoten“ in folgenden Durchgängen erneut betrachtet werden müssen. Zeichnen Sie anschließend den entstandenen Wegebaum, öden Graphen G , in dem nur noch diejenigen Kanten vorkommen, die Teil der kürzesten Wege von s zu allen anderen Knoten sind.

Nr	„schwarze“ Knoten	„graue“ Randknoten
1	(s, -, 0)	[(b, s, 3)] (d, s, 8) (e, s, 13) (h, s, 5)
2	(b, s, 3)	(d, s, 8) (e, b, 9) (g, b, 7) [(h, s, 5)]
3	(h, s, 5)	(d, s, 8) (e, b, 9) [(g, b, 7)]
4	(g, b, 7)	[(d, s, 8)] (e, b, 9)
5	(d, s, 8)	(c, d, 15) [(e, b, 9)] (f, d, 12)
6	(e, b, 9)	(c, e, 14) [(f, d, 12)]
7	(f, d, 12)	(a, f, 21) [(c, f, 13)]
8	(c, f, 13)	[(a, c, 17)]
9	(a, c, 17)	

Alternativer Lösungsweg

Nr.	besucht	a	b	c	d	e	f	g	h	s
0		∞	∞	∞	∞	∞	∞	∞	∞	0
1	s	∞	3	∞	8	13	∞	∞	5	0
2	b	∞	3	∞	8	9	∞	7	5	
3	h	∞		∞	8	9	∞	7	5	
4	g	∞		∞	8	9	∞	7		
5	d	∞		15	8	9	12			
6	e	∞		14		9	12			
7	f	21		13			12			
8	c	17		13						
9	a	17								

nach	Entfernung	Reihenfolge	Pfad
$s \rightarrow a$	17	9	$s \rightarrow d \rightarrow f \rightarrow c \rightarrow a$
$s \rightarrow b$	3	2	$s \rightarrow b$
$s \rightarrow c$	13	8	$s \rightarrow d \rightarrow f \rightarrow c$
$s \rightarrow d$	8	5	$s \rightarrow d$
$s \rightarrow e$	9	6	$s \rightarrow b \rightarrow e$
$s \rightarrow f$	12	7	$s \rightarrow d \rightarrow f$
$s \rightarrow g$	7	4	$s \rightarrow b \rightarrow g$
$s \rightarrow h$	5	3	$s \rightarrow h$
$s \rightarrow s$	0	1	

(b) Der Dijkstra-Algorithmus liefert bekanntlich auf Graphen mit negativen Kantengewichten unter Umständen ein falsches Ergebnis.

- (i) Geben Sie einen Graphen mit negativen Kantengewichten an, sodass der Dijkstra-Algorithmus ausgehend von einem von Ihnen ausgezeichneten Startknoten ein korrektes Ergebnis liefert.

Lösungsvorschlag

Startknoten a

```

graph TD
    a[a] ---|10| b[b]
    b[b] ---|-1| c[c]
    a[a] ---|2| c[c]
  
```

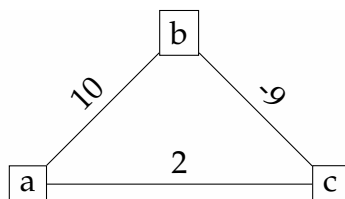
Nr.	besucht	a	b	c
0		0	∞	∞
1	a	0	10	2
2	c		10	2
3	b		10	

Richtig: a - c: 2

- (ii) Geben Sie einen Graphen mit negativen Kantengewichten an, sodass der Dijkstra-Algorithmus ausgehend von einem von Ihnen ausgezeichneten Startknoten ein falsches Ergebnis liefert.

Lösungsvorschlag

Startknoten a



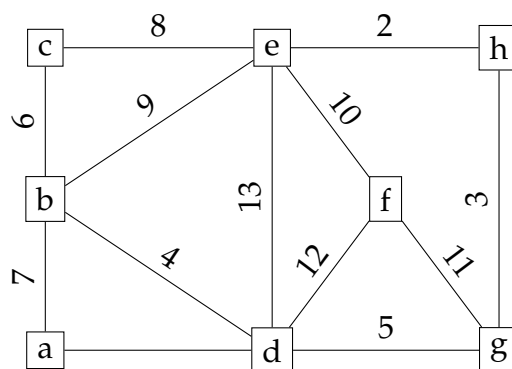
Nr.	besucht	a	b	c
0		0	∞	∞
1	a	0	10	2
2	c		10	2
3	b		10	

falsch: a - c: müsste 1 ($10 - 9$) sein.

Ein Beweis oder eine Begründung ist jeweils nicht erforderlich.

66115 / 2019 / Herbst / Thema 2 / Aufgabe 8

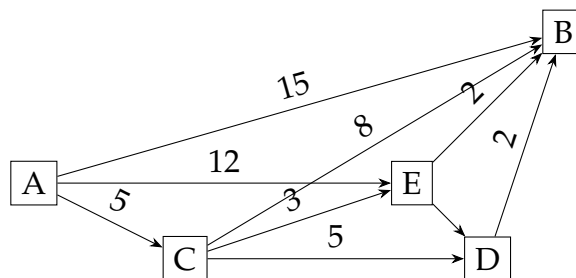
Gegeben Sei der folgende ungerichtete Graph mit Kantengewichten.



- (a) Zeichnen Sie den (hier eindeutigen) minimalen Spannbaum.
- (b) Geben Sie sowohl für den Algorithmus von Jarník-Prim als auch für den Algorithmus von Kruskal die Reihenfolge an, in der die Kanten hinzugefügt werden. Starten Sie für den Algorithmus von Jarník-Prim beim Knoten a .

Übernehmen Sie den Graph auf Ihre Bearbeitung und füllen Sie hierzu das Tupel jeder Kante - aus dem MST in der Form (n, m) aus, wobei die Kante e vom Algorithmus von Jarník-Prim als n 'te Kante und vom Algorithmus von Kruskal als m 'te Kante hinzugefügt wird. Lassen Sie andere Tupel unausgefüllt.

66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 3



- (a) Ermitteln Sie mit dem Algorithmus von Dijkstra den kürzesten Weg vom Knoten A zu allen erreichbaren Knoten in G. Verwenden Sie zur Lösung eine Tabelle der folgenden Form. Markieren Sie in jeder Zeile den jeweils als nächstes zu betrachtenden Knoten und führen Sie die Prioritätswarteschlange der noch zu betrachtenden Knoten (aufsteigend sortiert).

Nr.	besucht	A	B	C	D	E
0		0	∞	∞	∞	∞

Lösungsvorschlag

Nr.	besucht	A	B	C	D	E
0		0	∞	∞	∞	∞
1	A	0	15	5	∞	12
2	C		13	5	10	8
3	E		10		9	8
4	D		10		9	
5	B		10			

- (b) Geben Sie den kürzesten Pfad vom Knoten A zum Knoten B an.

Lösungsvorschlag

A \rightarrow C \rightarrow E \rightarrow B: 10				
nach	Entfernung	Reihenfolge	Pfad	
A \rightarrow A	0	0		
A \rightarrow B	10	5	A \rightarrow C \rightarrow E \rightarrow B	
A \rightarrow C	5	2	A \rightarrow C	
A \rightarrow D	9	4	A \rightarrow C \rightarrow E \rightarrow D	
A \rightarrow E	8	3	A \rightarrow C \rightarrow E	

66115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 3

Wir betrachten eine Variante der Breitensuche (BFS), bei der die Knoten markiert werden, wenn sie das erste Mal besucht werden. Außerdem wird die Suche einmal bei jedem unmarkierten Knoten gestartet, bis alle Knoten markiert sind. Wir betrachten gerichtete Graphen. Ein gerichteter Graph G ist *schwach zusammenhängend*, wenn der ungerichtete Graph (der sich daraus ergibt, dass man die Kantenrichtungen von G ignoriert) zusammenhängend ist.

Exkurs: Schwach zusammenhängend gerichteter Graph

Beim gerichteten Graphen musst du auf die Kantenrichtung achten. Würde man die Richtungen der Kanten ignorieren wäre aber trotzdem jeder Knoten erreichbar. Einen solchen Graphen nennt man schwach zusammenhängend.^a

Ein gerichteter Graph heißt (schwach) zusammenhängend, falls der zugehörige ungerichtete Graph (also der Graph, der entsteht, wenn man jede gerichtete Kante durch eine ungerichtete Kante ersetzt) zusammenhängend ist.^b

^a<https://studyflix.de/informatik/grundbegriffe-der-graphentheorie-1285>

^b[https://de.wikipedia.org/wiki/Zusammenhang_\(Graphentheorie\)](https://de.wikipedia.org/wiki/Zusammenhang_(Graphentheorie))

- (a) Beschreiben Sie für ein allgemeines $n \in \mathbb{N}$ mit $n \geq 2$ den Aufbau eines schwach zusammenhängenden Graphen G_n , mit n Knoten, bei dem die Breitensuche $\Theta(n)$ mal gestartet werden muss, bis alle Knoten markiert sind.

Lösungsvorschlag

?

Die Breitensuche benötigt einen Startknoten. Die unten aufgeführten Graphen finden immer nur einen Knoten nämlich den Startknoten.

Oder so:

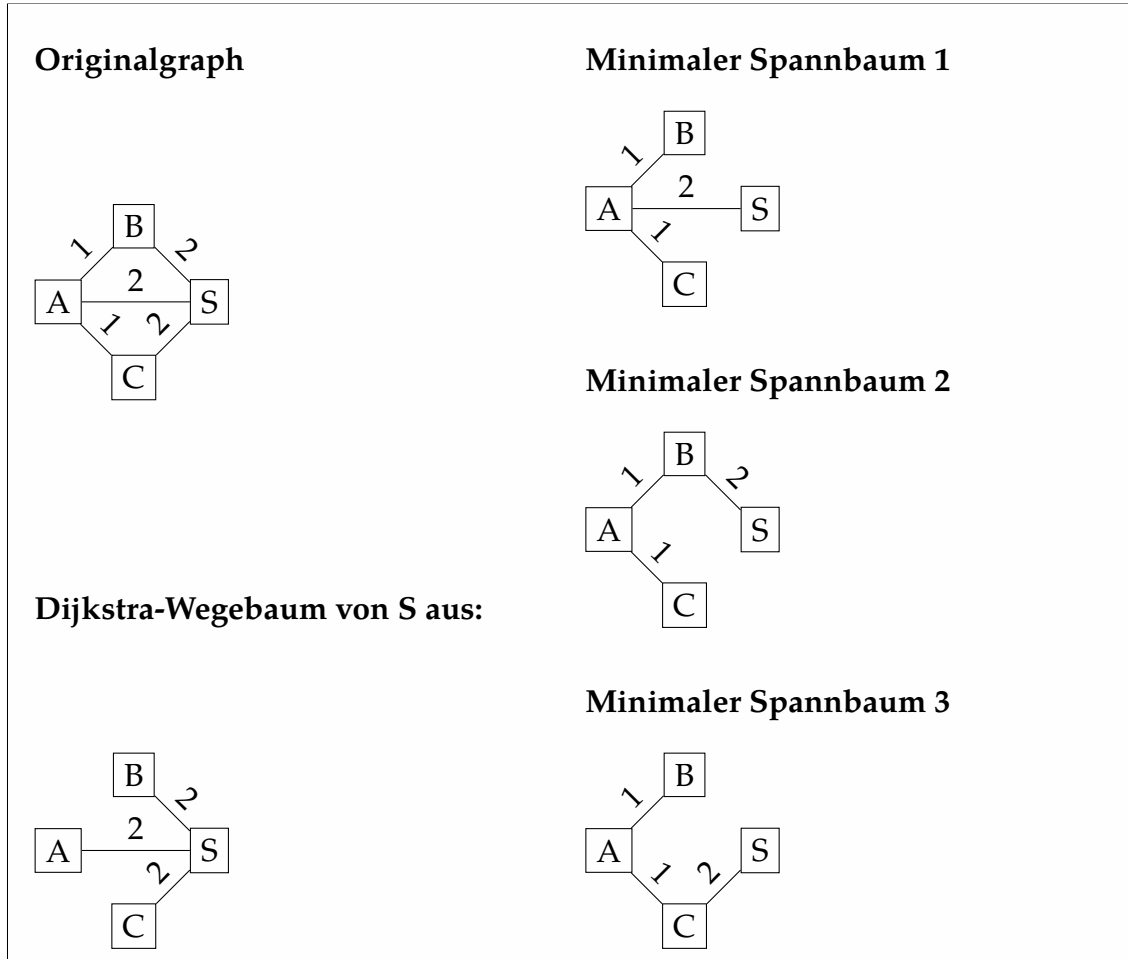
- (b) Welche asymptotische Laufzeit in Abhängigkeit von der Anzahl der Knoten (n) und von der Anzahl der Kanten (m) hat die Breitensuche über alle Neustarts zusammen? Beachten Sie, dass die Markierungen nicht gelöscht werden. Geben Sie die Laufzeit in Θ -Notation an. Begründen Sie Ihre Antwort.

66115 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 4

Die Algorithmen von Dijkstra und Jarník-Prim gehen ähnlich vor. Beide berechnen, ausgehend von einem Startknoten, einen Baum. Allerdings berechnet der Algorithmus von Dijkstra einen Kürzesten-Wege-Baum, während der Algorithmus von Jarník-Prim einen minimalen Spannbaum berechnet.

- (a) Geben Sie einen ungerichteten gewichteten Graphen G mit höchstens fünf Knoten und einen Startknoten s von G an, so dass **Dijkstra** (G, s) und **Jarník-Prim** (G, s) ausgehend von s verschiedene Bäume in G liefern. Geben Sie beide Bäume an.

Lösungsvorschlag



- (b) Geben Sie eine unendlich große Menge von Graphen an, auf denen der Algorithmus von Jarník-Prim asymptotisch schneller ist als der Algorithmus von Kruskal, der ebenfalls minimale Spannbäume berechnet.

Hinweis: Für einen Graphen mit n Knoten und m Kanten benötigt Jarník-Prim $\mathcal{O}(m + n \log n)$ Zeit, Kruskal $\mathcal{O}(m \log m)$ Zeit.

- (c) Sei Z die Menge der zusammenhängenden Graphen und $G \in Z$. Sei n die Anzahl der Knoten von G und m die Anzahl der Kanten von G . Entscheiden Sie mit Begründung, ob $\log m \in \Theta(\log n)$ gilt.

Sonstige

66115 / 2020 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 1

Betrachten Sie die folgende Prozedur `countup`, die aus zwei ganzzahligen Eingabewerten n und m einen ganzzahligen Ausgabewert berechnet:

```

procedure countup(n, m : integer): integer
var x, y : integer;
begin
  x := n;
  y := 0;
  while (y < m) do
    x := x - 1;
    y := y + 1;
  end while
  return x;
end

```

- (a) Führen Sie `countup(3,2)` aus. Geben Sie für jeden Schleifendurchlauf jeweils den Wert der Variablen n , m , x und y zu Beginn der `while`-Schleife und den Rückgabewert der Prozedur an.

Lösungsvorschlag

n	m	x	y	ausgeführter Code, der Änderung bewirkte
3	2	3	0	
3	2	2	1	<code>x := x - 1; y := y + 1;</code>

Rückgabewert: 1

Java-Implementation der Prozedur

```

public class CountUp {

  public static int countup(int n, int m) {
    int x = n;
    int y = 0;
    while (y < m) {
      System.out.println(String.format("%s %s %s %s", n, m, x, y));
      x = x - 1;
      y = y + 1;
    }
    return x;
  }

  public static void main(String[] args) {
    System.out.println(countup(3, 2));
  }
}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/counter/CountUp.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/counter/CountUp.java)

- (b) Gibt es Eingabewerte von n und m , für die die Prozedur `countup` nicht terminiert? Begründen Sie Ihre Antwort.

Nein. Mit jedem Schleifendurchlauf wird der Wert der Variablen y um eins hochgezählt. Die Werte, die y annimmt, sind streng monoton steigend. y nähert sich m an, bis y nicht mehr kleiner ist als m und die Prozedur terminiert.

An diesem Sachverh lt  ndern auch sehr gro e Zahlen, die  ber die Variable m der Prozedur  bergeben werden, nichts.

- (c) Geben Sie die asymptotische worst-case Laufzeit der Prozedur `countup` in der Θ -Notation in Abh ngigkeit von den Eingabewerten n und/oder m an. Begr nden Sie Ihre Antwort.

L sungsvorschlag

Die Laufzeit der Prozedur ist immer $\Theta(m)$. Die Laufzeit h ngt nur von m ab. Es kann nicht zwischen best-, average and worst-case unterschieden werden.

- (d) Betrachten Sie nun die folgende Prozedur `countdown`, die aus zwei ganzzahligen Eingabewerten n und m einen ganzzahligen Ausgabewert berechnet:

```

procedure countdown(n, m : integer) : integer
var x, y : integer;
begin
  x := n;
  y := 0;
  while (n > 0) do
    if (y < m) then
      x := x - 1;
      y := y + 1;
    else
      y := 0;
      n := n / 2; /* Ganzzahldivision */
    end if
  end while
  return x;
end

```

F hren Sie `countdown(3, 2)` aus. Geben Sie f r jeden Schleifendurchlauf jeweils den Wert der Variablen n , m , x und y zu Beginn der `while`-Schleife und den R ckgabewert der Prozedur an.

L sungsvorschlag

n	m	x	y	ausgef�hrter Code, der �nderung bewirkte
3	2	3	0	
3	2	2	1	<code>x := x - 1; y := y + 1;</code>
3	2	1	2	<code>x := x - 1; y := y + 1;</code>
1	2	1	0	<code>y := 0; n := n / 2;</code>
1	2	0	1	<code>x := x - 1; y := y + 1;</code>
1	2	-1	2	<code>x := x - 1; y := y + 1;</code>

R ckgabewert: -1

Java-Implementation der Prozedur

```

public class Countdown {

    public static int countdown(int n, int m) {
        int x = n;
        int y = 0;
        while (n > 0) {
            System.out.println(String.format("%s %s %s %s", n, m, x, y));
            if (y < m) {
                x = x - 1;
                y = y + 1;
            } else {
                y = 0;
                n = n / 2; /* Ganzzahldivision */
            }
        }
        return x;
    }

    public static void main(String[] args) {
        System.out.println(countdown(3, 2));
    }

}

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/counter/CountDown.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2020/herbst/counter/CountDown.java)

- (e) Gibt es Eingabewerte von n und m , für die die Prozedur `countdown` nicht terminiert? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Nein.

$n \leq 0$ terminiert sofort

$m \leq 0$ Der Falsch-Block der Wenn-Dann-Bedingung erniedrigt n $n := n / 2$; bis 0 erreicht ist. Dann terminiert die Prozedur.

$m > 0$ Der erste Wahr-Block der Wenn-Dann-Bedingung erhöht y streng monoton bis $y \geq m$. 2. Falsch-Block der Wenn-Dann-Bedingung halbiert n bis 0. 1. und 2. solange bis $n = 0$

- (f) Geben Sie die asymptotische Laufzeit der Prozedur `countdown` in der Θ -Notation in Abhängigkeit von den Eingabewerten n und/oder m an unter der Annahme, dass $m \geq 0$ und $n > 0$. Begründen Sie Ihre Antwort.

Lösungsvorschlag

Anzahl der Wiederholungen der while-Schleife: $m + 1$:

- m oft: bis $y < m$
- +1 Halbierung von n und y auf 0 setzen

wegen dem $n/2$ ist die Laufzeit logarithmisch, ähnlich wie der worst case bei

der Binären Suche.

n	m	x	y	ausgeführter Code, der Änderung bewirkte
16	3	16	0	
16	3	15	1	
16	3	14	2	
16	3	13	3	
8	3	13	0	<code>y := 0; n := n / 2;</code>
8	3	12	1	
8	3	11	2	
8	3	10	3	
4	3	10	0	<code>y := 0; n := n / 2;</code>
4	3	9	1	
4	3	8	2	
4	3	7	3	
2	3	7	0	<code>y := 0; n := n / 2;</code>
2	3	6	1	
2	3	5	2	
2	3	4	3	
1	3	4	0	<code>y := 0; n := n / 2;</code>
1	3	3	1	
1	3	2	2	
1	3	1	3	

$$\Theta((m+1) \log_2 n)$$

Wegkürzen der Konstanten:

$$\Rightarrow \Theta(m \log n)$$

Index

- Adjazenzliste, 252
- Adjazenzmatrix, 252, 361, 375
- Algorithmen und Datenstrukturen, 394
- Algorithmische Komplexität (O-Notation), 50, 92, 104, 122–127, 129, 133, 141, 146, 148, 151, 153
- Algorithmus von Dijkstra, 91, 239, 252, 349, 350, 362, 363, 373, 374, 378–381, 387, 392, 393
- Algorithmus von Kruskal, 366, 369, 384
- Algorithmus von Prim, 354, 355, 367, 369, 371, 384, 393
- all uses, 119
- Anforderungsüberdeckung, 88
- AVL-Baum, 89, 229, 241, 246, 247, 255, 267, 288, 295, 299, 305, 306, 311, 314, 319, 328

- B-Baum, 231, 232, 246, 279, 281, 306, 341, 343, 344, 347, 348
- Backtracking, 13, 161, 162, 172, 181
- Binärbaum, 223, 241, 255, 258, 272, 292, 297, 306, 323, 325, 337
- Binäre Suche, 44, 46, 55, 56, 62
- Black-Box-Testing, 88
- Breitensuche, 356, 372, 376, 378, 387, 393
- Bubblesort, 69, 74, 77, 78, 87, 103, 118
- Bucketsort, 101
- Bäume, 305, 317

- C1-Test Zweigüberdeckung (Branch Coverage), 119
- C2a Vollständige Pfadüberdeckung (Full Path Coverage), 88

- Datenfluss-annotierter Kontrollflussgraph, 118
- Datenflussorientiertes Testen, 118
- Design by Contract, 63
- Doppelt-verkettete Liste, 22, 212
- Dynamische Programmierung, 32, 160, 167, 169, 185

- Einfach-verkettete Liste, 20, 190, 207, 215, 221

- Entwurfsmuster, 38
- Feld (Array), 36
- Graphen, 361, 377, 386, 393
- Greedy-Algorithmus, 157, 158, 176, 180, 184
- Grenzwertanalyse, 88
- Halde (Heap), 94, 240, 253, 255, 267, 269, 271, 284, 292, 305, 306, 317, 334
- Hashfunktion, 233
- Heapsort, 77, 94

- Implementierung in Java, 8, 20, 29, 32, 35, 38, 46, 80, 91, 161, 202, 205, 209, 215, 223, 272
- Insertionsort, 80, 111
- Iterative Realisation, 12, 29

- Klassendiagramm, 38, 190, 205, 223
- Kompositum (Composite), 38, 190
- Kontrollflussgraph, 66, 88

- Lineare Suche, 51, 52, 59
- Lineares Sondieren, 235

- Master-Theorem, 68, 129, 146, 149, 154
- Mergesort, 67, 69, 72, 75, 81, 101, 118
- Min-Heap, 255
- Minimaler Spannbaum, 352, 366, 369, 391

- Objektdiagramm, 207
- Objektorientierung, 215
- Offene Adressierung, 304

- Physische Datenorganisation, 281

- Quadratisches Sondieren, 236
- Quicksort, 69, 72, 78, 91, 95, 97, 102, 103, 106, 113

- R-Baum, 306
- Radixsort, 101
- Rekursion, 8, 10–13, 23, 24, 28, 31, 37, 42
- Rot-Schwarz-Baum, 305

- Schreibtischlauf (Sortierung), 80
- Selectionsort, 12, 74, 82, 91, 103

Separate Verkettung, 235, 303
Sortieralgorithmen, 77, 86, 100, 103, 111
Stapel (Stack), 24, 134, 141, 196, 197, 199,
202
Streutabellen (Hashing), 233, 237, 238, 254,
264, 271, 278, 283, 290, 303, 322, 329,
332, 339

Teile-und-Herrsche (Divide-and-Conquer),
24, 45, 159, 166, 176, 178
Tiefensuche, 91, 197, 357, 361, 387
Tupel-Identifikator, 281, 344

Vollständige Anweisungsüberdeckung, 66
Vollständige Induktion, 31

Warteschlange (Queue), 193, 196, 204, 209

Zyklomatische Komplexität nach Mc-Cabe,
119

Äquivalenzklassenzerlegung, 88