

# 46115 Herbst 2015

Theoretische Informatik / Algorithmen / Datenstrukturen (nicht vertieft)

Aufgabenstellungen mit Lösungsvorschlägen



**Die Bschlangaul-Sammlung**

Hermine Bschlangaul and Friends

# Aufgabenübersicht

Thema Nr. 1 . . . . .	3
Aufgabe 3 [Unimodale Zahlenfolge] . . . . .	3
 Aufgabe 3 . . . . .	 3
 Thema Nr. 2 . . . . .	 9
Aufgabe 8 [Hashing mit Modulo 8] . . . . .	9
Aufgabe 4 [Methode function: Formale Verifikation - Induktionsbeweis]	11



## Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.



Diese Materialsammlung unterliegt den Bestimmungen der Creative Commons Namensnennung-Nicht kommerziell-Share Alike 4.0 International-Lizenz.

# Thema Nr. 1

## Aufgabe 3 [Unimodale Zahlenfolge]

### 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  $f(x) = -x^2$ .<sup>a</sup>

<sup>a</sup>[https://de.wikipedia.org/wiki/Unimodale\\_Abbildung](https://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. Grenvergleiche, arithmetische Operationen und Arrayzugriffe knnen wie blich in konstanter Zeit ( $\mathcal{O}(1)$ ) gettigt werden. Hinweise: binre Suche, divide-and-conquer.

Lsungsvorschlag

Wir whlen 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 grer, setzen wir die Suche wie oben beschrieben in der linken Hlfte, sonst in der rechten Hlfte fort.

- (b) Begrnden Sie, dass Ihr Algorithmus tatschlich in Zeit  $\mathcal{O}(\log n)$  luft.

Lsungsvorschlag

Da der beschriebene Algorithmus nach jedem Bearbeitungsschritt nur auf der Hlfte 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 bentigt. Somit hat der Algorithmus zum Finden des Maximums in einer unimodalen Folge in der Landau-Notation ausgedrckt die Zeitkomplexitt  $\mathcal{O}(\log n)$ .

- (c) Schreiben Sie Ihren Algorithmus in Pseudocode oder in einer Programmiersprache Ihrer Wahl, z. B. Java, auf. Sie drfen voraussetzen, dass die Eingabe in Form eines Arrays der Gre  $n$  vorliegt.

## 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/orgs/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/orgs/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.

```
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/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.

Lösungsvorschlag

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

Lösungsvorschlag

## 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)



# Thema Nr. 2

## Aufgabe 8 [Hashing mit Modulo 8]

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:  $8_5$ ).

### 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?

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.

#### Aufgabe 4 [Methode function: Formale Verifikation - Induktionsbeweis]

Gegeben sei die folgende Methode `function`:

```
double function(int n) {
    if (n == 1)
        return 0.5 * n;
    else
        return 1.0 / (n * (n + 1)) + function(n - 1);
}
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bachelor/examen/examen\\_46115/jahr\\_2015/herbst/Induktion.java](https://github.com/orgs/bachelor/examen/examen_46115/jahr_2015/herbst/Induktion.java)

Beweisen Sie folgenden Zusammenhang mittels vollständiger Induktion:

$$\forall n \geq 1: \text{function}(n) = f(n) \text{ mit } f(n) := 1 - \frac{1}{n+1}$$

Hinweis: Eventuelle Rechenungenauigkeiten, wie z. B. in Java, bei der Behandlung von Fließkommazahlen (z. B. `double`) sollen beim Beweis nicht berücksichtigt werden - Sie dürfen also annehmen, Fließkommazahlen würden mathematische Genauigkeit aufweisen.

##### Induktionsanfang

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

$$f(1) := 1 - \frac{1}{1+1} = 1 - \frac{1}{2} = \frac{1}{2}$$

##### Induktionsvoraussetzung

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

$$f(n) := 1 - \frac{1}{n+1}$$

##### Induktionsschritt

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

**zu zeigen:**

$$f(n+1) := 1 - \frac{1}{(n+1)+1} = f(n)$$

**Vorarbeiten (Java in Mathe umwandeln):**

$$\text{function}(n) = \frac{1}{n \cdot (n+1)} + f(n-1)$$

$f(n+1) = \frac{1}{(n+1) \cdot ((n+1)+1)} + f((n+1)-1)$	$n+1$ eingesetzt
$= \frac{1}{(n+1) \cdot (n+2)} + f(n)$	vereinfacht
$= \frac{1}{(n+1) \cdot (n+2)} + 1 - \frac{1}{n+1}$	für $f(n)$ Formel eingesetzt
$= 1 + \frac{1}{(n+1) \cdot (n+2)} - \frac{1}{n+1}$	1. Bruch an 2. Stelle geschrieben
$= 1 + \frac{1}{(n+1) \cdot (n+2)} - \frac{1 \cdot (n+2)}{(n+1) \cdot (n+2)}$	2. Bruch mit $(n+2)$ erweitert
$= 1 + \frac{1 - (n+2)}{(n+1) \cdot (n+2)}$	die 2 Brüche subtrahiert
$= 1 + \frac{1 - n - 2}{(n+1) \cdot (n+2)}$	$- + 2 = -2$
$= 1 + \frac{-1 - n}{(n+1) \cdot (n+2)}$	$1 - 2 = -1$
$= 1 + \frac{-1 \cdot (1+n)}{(n+1) \cdot (n+2)}$	$(n+1)$ ausgeklammert
$= 1 + \left( -1 \cdot \frac{(1+n)}{(n+1) \cdot (n+2)} \right)$	minus vor den Bruch bringen
$= 1 - \frac{(1+n)}{(n+1) \cdot (n+2)}$	plus minus ist minus
$= 1 - \frac{1}{n+2}$	$(n+1)$ gekürzt
$= 1 - \frac{1}{(n+1)+1}$	Umformen zur Verdeutlichung