Aufgabe 4

Das GUTSCHEIN-Problem ist gegeben durch eine Folge $w_1, \ldots, w_n$ von Warenwerten (wobei $w \in N_0$ für $i = 1, \ldots, n$) 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 $I \subseteq \{1, \ldots, n\}$ gibt, sodass $\sum_{i \in I} w_i = G$

(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.

50
$I = \{1, 3\}$

(ii) Geben Sie einen Gutscheinbetrag G mit $40 < G < 115$ an, sodass die GUTSCHEIN-Instanz keine Lösung hat.

51

(b) Sei *table* eine $(n \times (G + 1))$-Tabelle mit Einträgen table$[i, k]$, für $1 < i < n$ und $0 \leq k \leq G$, sodass

$$\text{table}[i,k] = \begin{cases} \textbf{true} & \text{falls es } I \subseteq \{1,\ldots,n\} \text{ mit } \sum_{i \in I} w_i = G \text{ gibt} \\ \textbf{false} & \text{sonst} \end{cases}$$

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?

```
3  // import java.util.Arrays;
4
5  /**
6   * https://www.geeksforgeeks.org/subset-gutscheinBetrag-problem-dp-25/
7   */
8  public class Gutschein {
9    /**
10    * @param gutscheinBetrag Das GUTSCHEIN-Betrag von 0, 1, ...
11    *
12    * @param warenWerte      Das GUTSCHEIN-Problem ist gegeben durch
      ↪  eine Folge w1,
13    *                        ..., wn von Warenwerten.
14    *
15    * @return Wahr, wenn der Gutscheinbetrag vollständig in Warenwerten
      ↪  eingelöst
16    *         werden kann, falsch wenn der Betrag nicht vollständig
      ↪  eingelöst
17    *         werden kann.
```

```java
    */
    public static boolean gutscheinDP(int gutscheinBetrag, int warenWerte[]) {
      // Der Eintrag in der Tabelle tabelle[i][k] ist wahr,
      // wenn es eine Teilsumme der
      // warenWerte[0..i-1] gibt, die gleich k ist.
      int n = warenWerte.length;
      boolean tabelle[][] = new boolean[n + 1][gutscheinBetrag + 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 <= gutscheinBetrag; k++)
        tabelle[0][k] = false;

      // Ist der Gutscheinbetrag 0, dann kann er immer eingelöst werden.
      for (int i = 0; i <= n; i++)
        tabelle[i][0] = true;

      for (int i = 1; i <= n; i++) {
        for (int k = 1; k <= gutscheinBetrag; k++) {
          tabelle[i][k] = tabelle[i - 1][k];
          if (k >= warenWerte[i - 1])
            tabelle[i][k] = tabelle[i][k] || tabelle[i - 1][k - warenWerte[i - 1]];
        }
      }
      // System.out.println(Arrays.deepToString(tabelle));
      return tabelle[n][gutscheinBetrag];
    }

    public static void main(String[] args) {
      System.out.println(gutscheinDP(10, new int[] { 10, 30, 40, 20, 15 }));
      System.out.println(gutscheinDP(3, new int[] { 1, 2, 3 }));
    }
  }
```

github: raw

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: `tabelle[warenWerte.length][gutscheinBetrag]`.

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

public class GutscheinTest {

  int[] warenWerte = new int[] { 10, 30, 40, 20, 15 };

  private void assertEingelöst(int gutscheinBetrag, int[] warenWerte) {
    assertEquals(Gutschein.gutscheinDP(gutscheinBetrag, warenWerte), true);
  }

  private void assertNichtEingelöst(int gutscheinBetrag, int[] warenWerte) {
    assertEquals(Gutschein.gutscheinDP(gutscheinBetrag, warenWerte), false);
  }

```

```
18    @Test
19    public void eingelöst() {
20      assertEingelöst(0, warenWerte);
21      assertEingelöst(10, warenWerte);
22      assertEingelöst(100, warenWerte);
23      assertEingelöst(115, warenWerte);
24      assertEingelöst(15, warenWerte);
25      assertEingelöst(20, warenWerte);
26      assertEingelöst(30, warenWerte);
27      assertEingelöst(40, warenWerte);
28      assertEingelöst(60, warenWerte);
29      assertEingelöst(70, warenWerte);
30    }
31
32    @Test
33    public void nichtEingelöst() {
34      assertNichtEingelöst(11, warenWerte);
35      assertNichtEingelöst(31, warenWerte);
36      assertNichtEingelöst(41, warenWerte);
37      assertNichtEingelöst(21, warenWerte);
38      assertNichtEingelöst(16, warenWerte);
39      assertNichtEingelöst(999, warenWerte);
40    }
41  }
```

- `gutscheinDP(3, new int[] { 1, 2, 3 }));`: wahr (w)

```
[
  [w, f, f, f],
  [w, w, f, f],
  [w, w, w, w],
  [w, w, w, w]
]
```

- `gutscheinDP(7, new int[] { 1, 2, 3 });`: falsch (f)

```
[
  [w, f, f, f, f, f, f, f],
  [w, w, f, f, f, f, f, f],
  [w, w, w, w, f, f, f, f],
  [w, w, w, w, w, w, w, f]
]
```

- `gutscheinDP(10, new int[] { 10, 30, 40, 20, 15 });`: wahr (w)

```
[
  [w, f, f, f, f, f, f, f, f, f, f],
  [w, f, f, f, f, f, f, f, f, f, w],
  [w, f, f, f, f, f, f, f, f, f, w],
  [w, f, f, f, f, f, f, f, f, f, w],
  [w, f, f, f, f, f, f, f, f, f, w],
  [w, f, f, f, f, f, f, f, f, f, w]
]
```