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 $\sum_{i \in I} w_i = G$

> **Exkurs: Teilsummenproblem**
> Das **Teilsummenproblem** (SUBSET SUM oder SSP) ist ein spezielles Rucksackproblem. Gegeben sei eine Menge von ganzen Zahlen $I = \{w_1, w_2, \ldots, w_n\}$. 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.

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 $1 \leq i \leq n$ 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?

Algorithmus 1: 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* $1 \ldots G$ **do** ; // Wenn der Gutscheinbetrag größer als 0 ist und es keine Warenwerte gibt, d.h. $n = 0$, kann der Gutschein nicht eingelöst werden.

  table$[0][k] =$ false
end

for i *in* $0 \ldots n$ **do** ; // Ist der Gutscheinbetrag 0, dann kann er immer eingelöst werden.

  table$[i][0] =$ true
end
for i *in* $1 \ldots n$ **do** ; // Durchlaufe jede Zeile der Warenwerte

  for k *in* $1 \ldots G$ **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 w1,
 ↪  ..., wn von
```

```
12      *          Warenwerten.
13      *
14      * @return Wahr, wenn der Gutscheinbetrag vollständig in Warenwerten
    ↪  eingelöst
15      *         werden kann, falsch wenn der Betrag nicht vollständig
    ↪  eingelöst
16      *         werden kann.
17      */
18    public static boolean gutscheinDP(int G, int W[]) {
19      // Der Eintrag in der Tabelle tabelle[i][k] ist wahr,
20      // wenn es eine Teilsumme der
21      // W[0..i-1] gibt, die gleich k ist.
22      int n = W.length;
23      boolean table[][] = new boolean[n + 1][G + 1];
24
25      // Wenn der Gutschein-Betrag größer als 0 ist und es keine
26      // Warenwerte (n = 0) gibt, kann der Gutschein nicht eingelöst
27      // werden.
28      for (int k = 1; k <= G; k++) {
29        table[0][k] = false;
30      }
31
32      // Ist der Gutscheinbetrag 0, dann kann er immer eingelöst werden.
33      for (int i = 0; i <= n; i++) {
34        table[i][0] = true;
35      }
36
37      for (int i = 1; i <= n; i++) {
38        for (int k = 1; k <= G; k++) {
39          table[i][k] = table[i - 1][k];
40          // Warenwert
41          int w = W[i - 1];
42          if (k >= w && !table[i][k]) {
43            table[i][k] = table[i - 1][k - w];
44          }
45        }
46      }
47      return table[n][G];
48    }
49
50    public static void main(String[] args) {
51      System.out.println(gutscheinDP(50, new int[] { 10, 30, 40, 20, 15
    ↪  }));
52      System.out.println(gutscheinDP(41, new int[] { 10, 30, 40, 20, 15
    ↪  }));
53
54      System.out.println(gutscheinDP(3, new int[] { 1, 2, 3 }));
55      System.out.println(gutscheinDP(5, new int[] { 1, 2, 3 }));
56      System.out.println(gutscheinDP(6, new int[] { 1, 2, 3 }));
57      System.out.println(gutscheinDP(2, new int[] { 1, 2, 3 }));
58      System.out.println(gutscheinDP(1, new int[] { 1, 2, 3 }));
59      System.out.println(gutscheinDP(7, new int[] { 1, 2, 3 }));
60    }
61  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/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.

```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);
  }

  @Test
  public void eingelöst() {
    assertEingelöst(0, warenWerte);
    assertEingelöst(10, warenWerte);
    assertEingelöst(100, warenWerte);
    assertEingelöst(115, warenWerte);
    assertEingelöst(15, warenWerte);
    assertEingelöst(20, warenWerte);
    assertEingelöst(30, warenWerte);
    assertEingelöst(40, warenWerte);
    assertEingelöst(60, warenWerte);
    assertEingelöst(70, warenWerte);
  }

  @Test
  public void nichtEingelöst() {
    assertNichtEingelöst(11, warenWerte);
    assertNichtEingelöst(31, warenWerte);
    assertNichtEingelöst(41, warenWerte);
    assertNichtEingelöst(21, warenWerte);
    assertNichtEingelöst(16, warenWerte);
    assertNichtEingelöst(999, warenWerte);
  }
}
```

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/GutscheinTest.java`

- `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)

```
[   0  1  2  3  4  5  6  7 G
 0 [w, f, f, f, f, f, f, f],
 1 [w, w, f, f, f, f, f, f],
 2 [w, w, w, w, f, f, f, f],
 3 [w, w, w, w, w, w, w, f]
```

```
 W
]
```

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

```
[    0  1  2  3  4  5  6  7  8  9  10 G
 0  [w, f, f, f, f, f, f, f, f, f, f],
 10 [w, f, f, f, f, f, f, f, f, f, w],
 30 [w, f, f, f, f, f, f, f, f, f, w],
 40 [w, f, f, f, f, f, f, f, f, f, w],
 20 [w, f, f, f, f, f, f, f, f, f, w],
 15 [w, f, f, f, f, f, f, f, f, f, w]
 W
]
```