

## 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: Teilsummenproblem

Das **Teilsummenproblem** (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.

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

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

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?

Pseudo-Code stimmt nicht mit dem Java-Code überein. Möglicherweise stimmt der Pseudo-Code nicht.

### Algorithmus 1: Gutschein-Problem

```
table = boolean array  $n \times (G + 1)$  ; // Initialisiere ein boolsches
Feld mit n Zeilen nummeriert von 1 bis n und G + 1 Spalten nummeriert
von 0 bis G
for  $j$  in  $2 \dots n$  do ;           // Durchlaufe alle weiteren Zeilen

    for  $k$  in  $0 \dots G$  do ;       // Durchlaufe jede Spalte in dieser Zeile

        if  $table[j - 1][k] == true$  then ; // Wenn in der vorherigen
        Zeile in dieser Spalte true steht

             $table[j - 1][k] = true$  ; // schreibe hier auch true
            if  $k + w_j \leq G$  then ; // Wenn der neue Warenwert addiert
            das Gutscheinvolumen nicht übersteigt..

                 $table[j][k + w_j] = true$  ; // markiere diesen Zustand,
                da er auch „erreichbar ist“
            end
        end
    end
end
```

```
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
13      ↪ eine Folge w1,
14      *                        ..., wn von Warenwerten.
15      *
16      * @return Wahr, wenn der Gutscheinbetrag vollständig in Warenwerten
17      ↪ eingelöst
18      *      werden kann, falsch wenn der Betrag nicht vollständig
19      ↪ eingelöst
20      *      werden kann.
21      */
22     public static boolean gutscheinDP(int gutscheinBetrag, int
23     ↪ warenWerte[]) {
24         // Der Eintrag in der Tabelle tabelle[i][k] ist wahr,
25         // wenn es eine Teilsumme der
26         // warenWerte[0..i-1] gibt, die gleich k ist.
27         int n = warenWerte.length;
28         boolean tabelle[][] = new boolean[n + 1][gutscheinBetrag + 1];
29
30         // Wenn der Gutschein-Betrag größer als 0 ist und es keine
31         // Warenwerte (n = 0) gibt, kann der Gutschein nicht eingelöst
32         // werden.
33         for (int k = 1; k <= gutscheinBetrag; k++)
34             tabelle[0][k] = false;
```

```

31
32 // Ist der Gutscheinbetrag 0, dann kann er immer eingelöst werden.
33 for (int i = 0; i <= n; i++)
34     tabelle[i][0] = true;
35
36 for (int i = 1; i <= n; i++) {
37     for (int k = 1; k <= gutscheinBetrag; k++) {
38         tabelle[i][k] = tabelle[i - 1][k];
39         if (k >= warenWerte[i - 1])
40             tabelle[i][k] = tabelle[i][k] || tabelle[i - 1][k -
41                 ↪ warenWerte[i - 1]];
42     }
43 }
44 // System.out.println(Arrays.deepToString(tabelle));
45 return tabelle[n][gutscheinBetrag];
46 }
47
48 public static void main(String[] args) {
49     System.out.println(gutscheinDP(10, new int[] { 10, 30, 40, 20, 15
50         ↪ }));
51     System.out.println(gutscheinDP(3, new int[] { 1, 2, 3 }));
52 }

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen\\_66115/jahr\\_2020/herbst/Gutschein.java](https://github.com/orgs/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:  
`tabelle[warenWerte.length][gutscheinBetrag]`.

```

3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class GutscheinTest {
7
8     int[] warenWerte = new int[] { 10, 30, 40, 20, 15 };
9
10    private void assertEingelöst(int gutscheinBetrag, int[] warenWerte) {
11        assertEquals(Gutschein.gutscheinDP(gutscheinBetrag, warenWerte), true);
12    }
13
14    private void assertNichtEingelöst(int gutscheinBetrag, int[] warenWerte)
15        ↪ {
16        assertEquals(Gutschein.gutscheinDP(gutscheinBetrag, warenWerte),
17            ↪ false);
18    }
19
20    @Test
21    public void eingelöst() {
22        assertEingelöst(0, warenWerte);
23        assertEingelöst(10, warenWerte);
24        assertEingelöst(100, warenWerte);
25        assertEingelöst(115, warenWerte);
26        assertEingelöst(15, warenWerte);
27        assertEingelöst(20, warenWerte);
28        assertEingelöst(30, warenWerte);
29        assertEingelöst(40, warenWerte);
30        assertEingelöst(60, warenWerte);
31        assertEingelöst(70, warenWerte);
32    }
33 }

```

```

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  }

```

Code-Beispiel auf Github ansehen: [src/test/java/org/beschlangaul/examen/examen\\_66115/jahr\\_2020/herbst/GutscheinTest.java](src/test/java/org/beschlangaul/examen/examen_66115/jahr_2020/herbst/GutscheinTest.java)

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

```

1  [
2      [w, f, f, f],
3      [w, w, f, f],
4      [w, w, w, w],
5      [w, w, w, w]
6  ]

```

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

```

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

```

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

```

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

```