Beispiellösung: Flaschenzug

Hinweis:

Der Aufgabentext wird hier nur der Vollständigkeit halber abgedruckt. Die Dokumentation zu einer Aufgabenbearbeitung muss und soll den Aufgabentext nicht enthalten.

Familie Soda beginnt einen 14-tägigen Abenteuerurlaub. Ziel ist die trinkwasserlose, unbewohnte Insel Drøgø, deren Küste ringsherum sehr steil ist. Frühere Urlauber haben bereits einen Flaschenzug installiert, mit dem die vielen benötigten Getränkeflaschen nach oben gezogen werden können. Zum Glück stehen auch viele Behälter mit genügend Platz für alle Flaschen zur Verfügung, damit mehrere Flaschen auf einmal transportiert werden können.

Während die Eltern die mitgebrachten Flaschen auf Behälter verteilen, überlegen ihre Kinder Cora und Linus, wie viele Möglichkeiten es wohl insgesamt gibt, die Flaschen auf die Behälter zu verteilen.

Bei sieben Flaschen und zwei Behältern, von denen in den einen drei und in den anderen fünf Flaschen passen, gibt es genau zwei Möglichkeiten: Der kleinere Behälter ist entweder ganz voll oder enthält genau zwei Flaschen. Auf drei Behälter mit Platz für genau zwei, drei und vier Flaschen lassen sich die sieben Flaschen auf genau sechs Arten verteilen.

Aufgabe

Schreibe ein Programm, das eine Anzahl N von Flaschen, eine Anzahl k von Behältern und die k Fassungsvermögen der Behälter einliest und berechnet, auf wie viele Arten die Flaschen verteilt werden können. Die Flaschen sind nicht unterscheidbar, aber die Behälter sind es, auch wenn sie gleich groß sind.

Wende dein Programm mindestens auf die Beispiele an, die du unter bundeswettbewerb-informatik.de findest, und dokumentiere jeweils das Ergebnis.

Lösungsidee

Für bestimmte Eingabewerte – Anzahl der Flaschen, Anzahl der Behälter und Fassungsvermögen der Behälter – soll eine Zahl berechnet werden: die Anzahl der verschiedenen Möglichkeiten, die Flaschen auf die Behälter zu verteilen. Diese Zahl nennen wir *Verteilungszahl*.

Das Ganze erinnert sehr an eine Funktion in der Mathematik. Für einen Wert x berechnet eine Funktion f einen Wert y: $y = f(x)$. Die Verteilungszahl wird durch eine Funktion (genannt v) berechnet, nur mit mehreren Ausgangswerten, nämlich N (Anzahl der Flaschen), k (Anzahl der Behälter) und F (Liste mit den k Fassungsvermögen): $v(N, k, F)$

Alles aufzählen: Brute-Force

Bei der Berechnung der Funktion $v(N, k, F)$ gibt es zwei einfache Fälle:

> Wenn keine Flaschen vorhanden sind ($N = 0$), gibt es genau eine Möglichkeit, die Flaschen zu verteilen: alle Behälter bleiben leer. Es gilt also $v(0, k, F) = 1$, unabhängig von k und F.

> Wenn keine Behälter vorhanden sind ($k = 0$), gibt es keine Möglichkeit, die Flaschen zu verteilen. Es gilt also $v(N, 0, F) = 0$, unabhängig von N und F.

Was aber, wenn es Flaschen und Behälter gibt? Dann müssen systematisch alle Möglichkeiten zur Flaschenverteilung aufgezählt werden. Wir fangen mit dem ersten Behälter an und legen z. B. eine Flasche hinein. Die Anzahl der Verteilungsmöglichkeiten, bei der im ersten Behälter eine Flasche liegt, ergibt sich dann aus der Anzahl der Möglichkeiten, die anderen $N–1$ Flaschen auf die restlichen $k–1$ Behälter zu verteilen, also: $v(N–1, k–1, F[2..k])$. Dabei bedeutet $F[2..k]$ die Liste der Fassungsvermögen ab Behälter 2. Alle Verteilungsmöglichkeiten insgesamt erhalten wir dann, wenn wir alle Möglichkeiten für den ersten Behälter durchgehen – von 0 bis zum Fassungsvermögen F[1] des ersten Behälters – und die jeweiligen Ergebnisse aufsummieren:

$v(N, k, F) = v(N–0, k–1, F[2..k]) + v(N–1, k–1, F[2..k]) + \ldots + v(N–F[1], k–1, F[2..k])$

Die Ausdrücke auf der rechten Seite können wir rekursiv auf die gleiche Weise berechnen, usw. Insgesamt werden so alle Möglichkeiten aufgezählt, nach dem Prinzip Brute-Force. Dabei wird das Problem rekursiv gelöst, indem auf gleiche Weise Lösungen für immer weiter „verkleinerte" Teilprobleme gesucht werden, solange bis für einfache Fälle Lösungen direkt angegeben werden können.

Zwischenergebnisse merken

Wenden wir die beschriebene Methode einmal auf das Beispiel aus der Aufgabenstellung an:

$$
\begin{aligned}
v(7,2,[3,5]) &= v(7,1,[5]) + v(6,1,[5]) + v(5,1,[5]) + v(4,1,[5]) \\
&= v(7,0,[]) + v(6,0,[]) + \ldots + v(2,0,[]) \\
&\quad + v(6,0,[]) + \ldots + v(1,0,[]) \\
&\quad + v(5,0,[]) + \ldots + v(0,0,[]) \\
&\quad + v(4,0,[]) + \ldots + v(–1,0,[]) \\
&= 0 + 0 + 0 + 0 + 0 + 0 \\
&\quad + 0 + 0 + 0 + 0 + 0 + 0 \\
&\quad + 0 + 0 + 0 + 0 + 0 + 1 \\
&\quad + 0 + 0 + 0 + 0 + 1 + 0 \\
&= 2
\end{aligned}
$$

Wir erhalten das aus der Aufgabenstellung bekannte Ergebnis. Die Rechnung zeigt aber, dass sehr viele Teilprobleme mehrfach berechnet werden. Um das zu vermeiden, können die Lösungen von Teilproblemen gespeichert werden. Nur wenn dann die Lösung für ein Teilproblem noch unbekannt ist, muss sie berechnet (und gespeichert) werden; ansonsten wird sie einfach aus dem Speicher geholt.

Dabei ist wichtig, dass die Fassungsvermögen für das Speichern von Teilproblem-Lösungen keine Rolle spielen. Auch wenn in der Rekursion nicht immer alle Behälter betrachtet werden, die Liste der Fassungsvermögen bleibt insgesamt gleich. Es genügt also ein Speicher S für alle möglichen Paarungen von N und k: $S(N, k)$

Umsetzung

Die Lösungsidee wird in Python implementiert. Als **Speicher** für Teillösungen wird ein Array verwendet. Weil Python keine „richtigen" Arrays mit mehreren Dimensionen kennt, wird das Bibliotheksmodul **numpy** verwendet, das Array-Funktionen realisiert.

Die Funktion v wird als Python-Funktion **verteilungszahl** implementiert. Sie prüft zunächst die einfachen Fälle ab und ruft sich für alle anderen Teilprobleme rekursiv auf – es sei denn, das Ergebnis für das Teilproblem ist schon gespeichert. Außerdem muss das Einlesen der Eingabedaten aus einer Datei realisiert werden. Mit Hilfe des Moduls **sys** wird schließlich ermöglicht, dass das Programm mit der einzulesenden Datei als Parameter auf der Kommandozeile aufgerufen werden kann.

Tests zeigen, dass die Ergebnisse und damit auch die gespeicherten Teilergebnisse sehr große Zahlen sein können. Python kann mit solchen großen Zahlen ohne Weiteres umgehen. In anderen Programmiersprachen müssen dafür passende Bibliotheken verwendet oder das Rechnen mit großen Zahlen (nur die Addition wird benötigt) selbst implementiert werden.

Beispiele

Wir rufen das Python-Programm mit den verschiedenen BWINF-Eingabedateien auf. Diese Dateien liegen im gleichen Ordner wie die Programmdatei. Alle Ergebnisse werden innerhalb weniger Sekunden berechnet. Ohne den Teillösungs-Speicher wäre das nicht möglich.

```
$ ./flaschenzug.py flaschenzug0.txt
2
$ ./flaschenzug.py flaschenzug1.txt
13
$ ./flaschenzug.py flaschenzug2.txt
48
$ ./flaschenzug.py flaschenzug3.txt
6209623185136
$ ./flaschenzug.py flaschenzug4.txt
743587168174197919278525
$ ./flaschenzug.py flaschenzug5.txt
42376183321681306437343953352208634086 28
```

Quelltext

```
#!/usr/bin/env python3

# Für Übergabe von Argumenten in der Kommandozeile:
import sys

# Einlesen der Eingabe: Eingabedatei öffnen …
with open(sys.argv[1]) as f:
    # … und mit next(f) je eine Zeile als String einlesen.
    N = int(next(f))
    k = int(next(f))
    F = []
    for i in next(f).split():
        # split() macht aus Zeile 3 eine Liste.
        F.append(int(i))

# Für den Ergebnisspeicher nehmen wir ein Array aus
# der Bibliothek numpy.Es soll beliebige Werte
# (Typ "object") enthalten dürfen,
# also auch beliebig große Zahlen.
import numpy
# Leeres Array erzeugen …
Speicher = numpy.empty([N+1, k+1],
                       dtype = numpy.object)
# … und auf allen Positionen mit -1 initialisieren:
Speicher.fill(-1)

def verteilungszahl(AnzFlaschen, AnzBehaelter, FassungsListe):
    # Ohne Flaschen gibt es eine Möglichkeit:
    # Alle Behälter bleiben leer.
    if AnzFlaschen == 0: return(1)
    # Weniger als 0 Flaschen geht gar nicht.
    if AnzFlaschen < 0: return(0)
    # Ohne Behälter gibt es keine Möglichkeit.
    if AnzBehaelter == 0: return(0)
    # Steht für die gegebenen Zahlen schon ein Wert
    # im Speicher, nehmen wir den.
    if Speicher[AnzFlaschen, AnzBehaelter] != -1:
        return(Speicher[AnzFlaschen, AnzBehaelter])

    # Das waren die einfachen Fälle;
    # jetzt muss rekursiv gerechnet werden:
    Speicher[AnzFlaschen, AnzBehaelter] = 0
    # Wir legen alle möglichen Anzahlen von Flaschen
    # in den ersten Behälter …
    for AnzFlaschenB1 in range(FassungsListe[0]+1):
        if (AnzFlaschen - AnzFlaschenB1) < 0 :
            # (alle Flaschen verbraucht)
            break
        # … und addieren dann jeweils die Verteilungszahl
        # für die restlichen Flaschen und Behälter.
        Speicher[AnzFlaschen, AnzBehaelter] = \
            Speicher[AnzFlaschen, AnzBehaelter] + \
                verteilungszahl(AnzFlaschen - AnzFlaschenB1,
                                AnzBehaelter - 1,
                                FassungsListe[1:])
    return(Speicher[AnzFlaschen, AnzBehaelter])

print(verteilungszahl(N, k, F))
```