Aufgabe 2: Nummernmerker

Team-ID: 00041

Team-Name: team@sebastianbrunnert

Bearbeiter/-innen dieser Aufgabe:
Sebastian Brunnert

2. September 2019

Inhaltsverzeichnis

Lösungsidee

Zu jeder beliebigen Zahl, sollen Blöcke einer Länge von mindestens zwei und maximal vier Ziffern gebildet werden. Dabei sollen möglichst wenig Blöcke mit einer Null beginnen. Vorausgesetzt ist also, dass die vom Nutzer angegebene Zahl mindestens zwei Ziffern lang ist, da sonst kein Block gebildet werden kann. Meine **Zusatzfuntkion** ist es, dass immer die Lösung mit der niedrigsten Anzahl an Blöcken gesucht wird

Alle Möglichkeiten finden: Brute-Force

Aufgrund dessen, dass nicht einfach ein Format von Blöcken der Länge zwei bis vier gesucht wird, ist es nötig, alle potenziellen Lösungen dieses Problems auszuprobieren und die Lösung zu finden, welche mit den wenigsten Nullen beginnt.
Nun werden alle Möglichkeiten gesucht, wie man mit Blöcken einer Länge von zwei bis vier Ziffern, die angegebene Zahl bilden kann. Sind alle potenziellen Lösungen gefunden, gilt es nur noch aus diesen diejenige zu ermitteln, bei der die wenigsten Blöcke mit null beginnen.

Umsetzung

Die Umsetzung des Programms geschieht in der Programmiersprache Python. Benötigt wird die Libary sys, damit Kommandozeilenargumente übergeben werden können.

Erst wird geprüft, ob ein Argument beim Start des Programms von dem Nutzer übergeben wurde, und ob dieses Argument eine Zahl mit mindestens einer Länge von zwei Ziffern ist.

Um nun alle potenziellen Lösungen zu finden wird ein Algorithmus verwendet. Zweck des Algorithmus ist es, alle Kombinationen einer Liste von Zahlen zu finden, die zusammen eine festgelegte Summe bilden, wobei es möglich ist, dass eine Zahl mehrmals zur Bildung verwendet wird. Durch diesen Algorithmus wird die Länge der angegebenen Zahl gesendet und ausgelesen.

Nun, wo alle Möglichkeiten gefunden sind, gilt es die Anzahl der einen Block beginnenden Nullen in jeder potenziellen Lösung zu finden. Das Folgende wird also für jede Möglichkeit ausprobiert: Als erstes wird überprüft, ob die erste Ziffer der angegeben Zahl, welche immer den Beginn eines Blocks darstellt, eine Null ist. Ist dies der Fall gilt: $n_{AnzahlNullen} = 1$
Anschließend wird für jeden Block eine der Blocklänge (es muss gelten $5>n_{Blocklänge}<1$) entsprechenden Ziffernanzahl von der durch den Nutzer angegeben Zahl abgezogen, bis kein Block mehr vorhanden ist. Ist nun die erste Ziffer eine Null wird Eins zu $n_{AnzahlNullen}$ ($n_{AnzahlNullen}+1$) addiert. Jetzt werden $n_{AnzahlNullen}$ und die potenzielle Lösung im Key-Value-System (Python-Dictionary) zwischengespeichert.

Schlussendlich muss nur noch der zum niedrigsten Value gehörende Key gefunden werden und in der Konsole ausgesendet werden. Sollten mehrere Keys den selben niedrigsten Value haben, wird der Key mit den wenigsten Blöcken herausgesucht.

Beispiele

Wir rufen das Python-Programm mit den von BWINF vorgesehenen Beispielen auf:

1	$ python3 nummernmerker.py 005480000005179734 005 4800 0000 517 9734 2 Blöcke beginnen mit 0
2	$ python3 nummernmerker.py 03495929533790154412660 034 9592 9533 7901 5441 2660 1 Block beginnt mit 0
3	$ python3 nummernmerker.py 5319974879022725607620179 531 9974 8790 227 2560 7620 179 Kein Block beginnt mit 0
4	$ python3 nummernmerker.py 9088761051699482789038331267 9088 7610 5169 9482 7890 3833 1267 Kein Block beginnt mit 0
5	$ python3 nummernmerker.py 01100000001100010011111110101 1 01 1000 0000 1100 0100 1111 1110 1011 3 Blöcke beinnen mit 0

Quellcode

Der vollständige Quellcode, ist in der Datei nummernmerker.py zu finden.

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

# Libary für die Übergabe von Kommandozeilenargumenten
import sys

# Hilfs-Funktion, um mögliche Kombinationen zu finden die zusammen n ergeben
# @arg1 - Array der Zahlen die benutzt werden dürfen
# @arg2 - Nummer die schlussendlich in der Summe des Resultats herauskommen soll
# @arg3 - Generator Objekt; ist eine Liste dessen Inhalt Listen sind die in der Summen @arg2 ergeben
def possible_combinations(numbers, target, partial=[], partial_sum=0):
    if partial_sum == target:
        yield partial
    if partial_sum >= target:
        return
    for n in numbers:
        yield from possible_combinations(numbers, target, partial + [n], partial_sum + n)

# Rufe eingegebene Zeichenkette auf
number = str(sys.argv[1])

# Definiere mithilfe der Funktion possible_combinations eine Liste mit den erlaubten Blocklängen
allowed_block_lengths = list(possible_combinations([2,3,4], len(number)))

# Definiere ein Verzeichnis
# @key - Liste mit den verschiedenen Blocklängen der Nummer
# @value - Anzahl wieviele Blöcke mit 0 beginnen
number_of_zeros = {}

# Gehe in einer For-Schleife alle erlaubten Blocklängen durch und trage in number_of_zeros ein wieviele Bläcke mit 0 starten
for block_lengths in allowed_block_lengths:
    zeros = 0
    # Kopiere den String, da dieser durch die Benutzung dieses Algorithmus verändert wird
    copy_number = number
```

```
    # Prüfe ob die erste Ziffer eine 0 ist
    if copy_number[0] == "0":
        zeros += 1
    for i in block_lengths:
        # Lösche die ersten n Ziffern
        copy_number = copy_number[i:]
        # Überprüfe ob nun die erste Ziffer existiert bzw. eine 0 ist (wenn er nicht existiert ist das Ende der Zahl erreicht)
        if len(copy_number) > 0 and copy_number[0] == "0":
            # Addiere einen zu der Anzahl der Nullen
            zeros += 1

    # Setze die Anzahl der Ziffern und die Blocklängen in das Verzeichnis number_of_zeros ein
    number_of_zeros[tuple(block_lengths)] = zeros

# Suche nun die minimalste Anzahl an mit null beginnenden Blöcken hat
solution = min(number_of_zeros.items(), key=lambda x: x[1])[0]

# Suche nun die Varianten, die die minimalste Anzahl an mit null beginnenden Blöcken hat (und gleichzeitig die minimalste Anzahl an mit null beginnenden Blöcken)
for possible_solution in number_of_zeros:
    if number_of_zeros[solution] == number_of_zeros[possible_solution]:
        if len(solution) > len(possible_solution):
            solution = possible_solution
```