Aufgabe 2: Nummernmerker

Team-ID: 00062

Team-Name: CubeFlo

Bearbeiter dieser Aufgabe:
Florian Rädiker

21. Oktober 2019

Inhalt

Lösungsidee

Das Problem wird rekursiv gelöst. Von einer Nummer werden links testweise zwei, drei und vier Ziffern abgeschnitten und der Rest wird weiter nach demselben Schema zerlegt. Aus den drei Zerlegungen wird dann die ausgewählt, bei der es am wenigsten Blöcke gibt, die mit einer Null beginnen („Nullblöcke“). Ob tatsächlich zwei, drei oder vier Ziffern abgeschnitten werden können, hängt von der Länge der Nummer ab:

- Ist die Länge größer als 5, ist es möglich, sowohl zwei, drei als auch vier Ziffern abzuschneiden.
- Ist die Länge gleich 5, können nur zwei oder drei Ziffern abgeschnitten werden. Zudem ist eine weitere rekursive Zerlegung nicht nötig: Der restliche Teil hat entweder eine Länge von 3 oder eine Länge von 2 Ziffern. In beiden Fällen ist eine Zerlegung nicht möglich.
- Ist die Länge gleich 4, können nur zwei Ziffern abgeschnitten werden. Dies ist aber nicht sinnvoll. Es ist besser, eine Nummer der Länge 4 nicht zu zerlegen, denn durch eine Zerlegung kann ein zusätzlicher Nullblock entstehen (z.B. `4003`). Also wird eine Nummer der Länge 4 so betrachtet, als könnte sie nicht weiter zerlegt werden und zählt damit zum nächsten Fall.
- Ist die Länge kleiner als 4, kann die Nummer nicht weiter zerlegt werden, ohne Blöcke der Länge eins zu erhalten.

Die Funktion, die diesen Algorithmus implementiert, gibt außer der zerlegten Nummer (also dem ersten, abgeschnittenen Teil und dem restlichen Teil, der rekursiv weiter zerlegt wurde) auch die Anzahl der Nullblöcke zurück. Dafür wird zur Anzahl der Nullblöcke aus dem restlichen Teil eins addiert, wenn der abgeschnittene Block mit einer 0 beginnt. Falls die Länge kleiner als 4 ist und der Block mit einer Null beginnt, wird eine 1 zurückgegeben, ansonsten 0.

Es kann passieren, dass gleiche Nummern der Funktion mehrfach übergeben werden. Die Nummer `4000440023` kann beispielsweise zunächst zu `4000-440023` oder `40-00-440023` zerlegt werden. In beiden Fällen wird die Funktion mit der gleichen Nummer, `440023`, zweimal aufgerufen. Um dies zu verhindern, wird das Prinzip der dynamischen Programmierung angewendet: Zwischenergebnisse wie hier die Zerlegung der Nummer `440023` werden gespeichert, sodass bei einem weiteren Aufruf der Funktion das Ergebnis aus dem Speicher abgerufen und nicht nochmals berechnet wird.

Die Geschwindigkeit kann weiter verbessert werden, indem für jede der bis zu drei Möglichkeiten vor dem Berechnen der verbleibenden Möglichkeiten überprüft wird, ob die Anzahl der Nullblöcke bereits 0 beträgt. Ist das der Fall, ist das Ergebnis nämlich bereits optimal und die gefundene Zerlegung kann direkt zurückgegeben werden.[1] Dabei muss allerdings berücksichtigt werden, dass das Überprüfen der Anzahl natürlich auch Zeit kostet. Diese Variante wurde im Programm umgesetzt.

Umsetzung

Das Programm wurde mit Python 3 geschrieben.
Die Funktion `split_number` implementiert den beschriebenen Algorithmus. Durch mehrere if-Bedingungen wird zunächst die Länge der Nummer überprüft. Ist die Länge kleiner als 5, wird wie in

[1] Falls die Nummer mit einer Null beginnt, so wird dies ignoriert, da es sowieso einen Nullblock am Anfang geben wird. Die Anzahl der Nullblöcke, die hier zurückgegeben wird, beträgt also 0, wenn die Nummer nicht mit einer Null beginnt, ansonsten 1.

der Lösungsidee beschrieben die Nummer zurückgegeben sowie eine 1, wenn die Nummer mit einer Null beginnt. Falls die Länge größer oder gleich 5 ist, werden in einer Liste alle Zerlegungsoptionen – also die Zerlegung zusammen mit der Anzahl der Nullblöcke – gespeichert. Die Möglichkeit mit der kleinsten Anzahl an Nullblöcken wird zurückgegeben.
Um Zwischenergebnisse zu speichern, wird der Decorator `lru_cache` aus dem `functools`-Modul verwendet.

Beispiele

Die Beispiele werden folgendermaßen zerlegt. Ausgegeben wird auch jeweils die benötigte Zeit in Sekunden.

```
############
005480000005179734
0054-8000-0005-1797-34

Nullblöcke: 2
Time: 0.0000950

############
03495929533790154412660
0349-5929-5337-9015-4412-660

Nullblöcke: 1
Time: 0.0000429

############
5319974879022725607620179
5319-9748-7902-2725-6076-20-179

Nullblöcke: 0
Time: 0.0000146
```

```
############
9088761051699482789038331267
9088-7610-5169-9482-7890-3833-1267

Nullblöcke: 0
Time: 0.0000174

############
01100000001100010011111111101011
0110-0000-001-1000-1001-1111-110-1011

Nullblöcke: 3
Time: 0.0001422

############
4000440023
4000-4400-23

Nullblöcke: 0
Time: 0.0000087
```

Es ist gut zu sehen, dass die Zerlegungen hauptsächlich aus Blöcken der Länge vier bestehen. Das ist damit zu erklären, dass das Abschneiden von vier Ziffern als erstes getestet wird. Außerdem ist es eine gute Idee, Viererblöcke als erstes zu testen, da mit diesen am meisten Ziffern abgeschnitten werden und durch die geringere Anzahl an verbleibenden Ziffern die Laufzeit verbessert wird.

Quellcode

Die split_number-Funktion

Die Funktion ist im Skript aufgabe2.py zu finden.
Der Funktion wird ein Parameter, die zu zerlegende Nummer, übergeben.

```python
@lru_cache()
def split_number(number):
    # bool wird in Python durch int dargestellt (True ≙ 1, False ≙ 0), deshalb ist der
    # folgende Ausdruck gleichbedeutend mit entweder 0 oder 1
    number_startswith_0 = number[0] == "0"  # nur einmal berechnen, ob die Nummer mit einer 0
                                            # beginnt
    length = len(number)
    if length < 6:
        if length == 5:
            # Mögliche Zerlegungen: 2-3 oder 3-2
```

```
            # 2-3
            possibilities = []
            second_block = number[2:]
            second_block_0 = second_block[0] == "0"
            if second_block_0 == 0:
                # Zerlegung ist bereits optimal
                return number[:2] + "-" + second_block, number_startswith_0
            possibilities.append((number[:2] + "-" + second_block, number_startswith_0 +
                                                                    second_block_0))

            # 3-2
            second_block = number[3:]
            second_block_0 = second_block[0] == "0"
            if second_block_0 == 0:
                return number[:3] + second_block, number_startswith_0
            possibilities.append((number[:3] + "-" + second_block, number_startswith_0 +
                                                                    second_block_0))
        else:
            # Länge ist vier oder kleiner, kann (bzw. sollte, falls es vier Ziffern sind) nicht
            # weiter zerlegt werden
            return number, number_startswith_0
    else:
        possibilities = []

        # vier Ziffern abschneiden
        blocks, count = split_number(number[4:])
        if count == 0:
            return number[:4] + "-" + blocks, number_startswith_0
        count += number_startswith_0
        possibilities.append((number[:4] + "-" + blocks, count))

        # drei Ziffern abschneiden
        blocks, count = split_number(number[3:])
        if count == 0:
            return number[:3] + "-" + blocks, number_startswith_0
        count += number_startswith_0
        possibilities.append((number[:3] + "-" + blocks, count))

        # zwei Ziffern abschneiden
        blocks, count = split_number(number[2:])
        if count == 0:
            return number[:2] + "-" + blocks, number_startswith_0
        count += number_startswith_0
        possibilities.append((number[:2] + "-" + blocks, count))
    # Zerlegung mit der kleinsten Anzahl von Nullblöcken auswählen
    return min(possibilities, key=lambda x: x[1])
```