Aufgabe 2: Nummernmerker

Team-ID: 00485

Team-Name: FXC3

Bearbeiter/-innen dieser Aufgabe:
Leonardo Benini

3. November 2019

Inhaltsverzeichnis

Lösungsidee

Die Aufteilung von einer Nummer in mehrere Blöcke mit den in der Aufgabe genannten Kriterien kann algorithmisch gelöst werden. Die Aufteilung besteht aus 4 simplen Schritten. Der erste Schritt ist es die Nummer in Blöcke zu teilen, die immer mit einer 0 enden (die erste Ziffer wird dabei ignoriert). Somit ist garantiert, dass vorerst möglichst wenige Blöcke mit 0 anfangen. Der nächste Schritt ist es herauszufinden welche Blöcke aus nur einer Ziffer bestehen und sich die Indizes zu merken. Blöcke, die nur aus einer Ziffer bestehen, sind immer eine 0 (der letzte Block kann auch eine andere Ziffer in sich tragen, da er der letzte ist). Der vorletzte Schritt ist es die Blöcke, die nur aus einer Ziffer bestehen, dem jeweils vorigen Block anzufügen. Somit sind alle Nullen jetzt immer am Ende eines Blockes aufzufinden. Der letzte Schritt ist es dann die zu großen Blöcke von hinten in Blöcke bestehend aus 4 Ziffern zu teilen. Wenn dies am Ende nicht aufgeht, wird mit einem Block bestehend aus drei oder zwei Ziffern aufgefüllt - manchmal sogar mit einem von beiden.

Beispielnummer	Schritt 1	Schritt 2	Schritt 3	Schritt 4
01365400606	0136540 0 60 6	Index 1 und 3	01365400 606	0136 5400 606

Umsetzung

Die Lösungsidee wird in Python implementiert. Das Programm besteht aus den vier in der Lösungsidee genannten Schritten.

1. Ein for loop iteriert einmal durch die ganze Nummer und merkt sich die Positionen der Nullen und teilt die Nummer dann entsprechend auf. Die einzelnen Blöcke werden als Strings einem Array „blocks“ hinzugefügt.

```
# split number into blocks which always end on a 0 (except for the last one)
for i in range(len(number)):
    if i != 0:
        if number[i] != '0' and i != len(number)-1:
            ie += 1
        else:
            ie += 1
            blocks.append(number[ib: ie])
            ib = ie
            ie = ib
```

2. Ein for loop iteriert durch „blocks“ und checkt ob die Länge des Elements gleich eins ist und wenn ja fügt er den Index dieses Elements dem Array „indexes“ hinzu.

```
# find out the blocks which only consist of one number
indexes = []
for i in range(len(blocks)):
    if len(blocks[i]) == 1:
        indexes.append(i)
```

3. Ein weiterer for loop iteriert durch das eben geschaffene Array „indexes“ und fügt dem block[index – 1] den block[index] hinzu und löscht den block[index].

```
# append these blocks to the blocks before them
for index in indexes[::-1]:
    blocks[index - 1] = blocks[index - 1] + blocks[index]
    del blocks[index]
```

4. Im letzten Schritt iteriert ein for loop durch die Blöcke in „blocks“ und checkt mit einer if Bedingung, ob der Block länger ist als vier Ziffern. Wenn nicht fügt er den Block dem Array „split“ hinzu. Wenn der Block aber zu lang ist, nimmt eine while Schleife so lange vier Ziffern große Blöcke von hinten weg und fügt sie einem Array „p“ hinzu bis der ursprüngliche Block weniger als 6 Ziffern hat. Sobald dies der Fall ist, wird bei der Länge von 5 Ziffern der Block in einen zweier und einen dreier Block unterteilt und bei Länge kleiner 5 wird der übrige Block einfach dem Array „p“ hinzugefügt. Da der Block von hinten angegangen worden ist, ist der Array „p“ falsch herum, und deswegen fügen wir „split“ den invertierten Array „p“ an.

```
# devide the too big blocks into blocks of 4 and when fill up with one block of 2 or 3 or both
split = []
for block in blocks:
    if len(block) <= 4:
        split.append(block)
    else:
        p = []
        while len(block) > 5:
            p.append(block[-4:])
            block = block[:-4]
        if len(block) == 5:
            p.append(block[-3:])
            p.append(block[:2])
        else:
            p.append(block)
        p.reverse()
        for item in p:
            split.append(item)
```

Nach diesen vier Schritten ist die Nummer in Blöcke mit der Länge 2-4 Ziffern unterteilt und möglichst wenige von ihnen fangen mit 0 an.

Beispiele

```
005480000005179734: ['00', '54', '800', '0000', '517', '9734']

03495929533790154412660: ['03', '4959', '2953', '3790', '15', '441', '2660']

5319974879022725607620179: ['531', '9974', '8790', '227', '2560', '7620', '179']

90887610516994827890383312 67: ['90', '88', '7610', '5169', '9482', '7890', '3833',
'1267']

011000000011000100111111101011: ['01', '1000', '0000', '11', '000', '100', '1111',
'1110', '10', '11']
```

Quellcode

Das Projekt ist auf Github einsehbar.

Link: https://github.com/FloxiGH/BWINF2019