Aufgabe 4: Auto-Scrabble

1. Stimmt es, dass TIMO nicht auf einem Kennzeichen stehen kann?

Kennzeichen = Kürzel aus Kürzelliste + [A-Z]{1,2}[3] *# Zahl wird ignoriert*

(Kürzelliste ist die offizielle Liste aller Unterscheidungszeichen für deutsche Nummernschilder)

1. Möglichkeit

"TIMO" = Kürzel + "MO" → Kürzel müsste sein: "TI"

2. Möglichkeit

"TIMO" = Kürzel + "O" → Kürzel müsste sein: "TIM"

Kürzelliste enthält weder "TIM" noch "TI"

→ Daher kann "TIMO" nicht auf einem Kennzeichen stehen

2. Gib ein weiteres Wort mit vier Buchstaben an, das nicht auf einem Kennzeichen stehen kann. Findest du sogar ein solches Wort mit drei Buchstaben? Mit zwei?

Ein weiteres Wort mit 4 Buchstaben, das nicht auf einem Kennzeichen stehen kann ist zum Beispiel: „JAZZ“

Um solche Wörter mit 2-5 Buchstaben zu finden habe ich folgendes Script geschrieben:

```
# !usr/bin/env python3
# encoding: utf-8

import sys

def open_file(name):
    with open(name, 'r', encoding='UTF-8') as file:
        return sorted({line.strip() for line in
file.readlines()})
```

[3] Regulärer Ausdruck. Bedeutung: Aneinanderreihung der Großbuchstaben (A-Z) mit min. Länge 1 und max. Länge 2

```
def is_possible(s, kuerzel):
    if len(s) > 5: return False
    umlaute = set('ÄÖÜ')
    s = s.upper().strip()
    for i in range(1, min(len(s), 4)):
        # Teilt String in 2 Teile
        pre, post = s[:i], s[i:]
        # Zweiter Teil darf laut Aufgabestellung nicht >2 sein
        if len(post) > 2: continue
        if not any((c in umlaute) for c in post) and pre in
kuerzel:
            return True
    return False

def find_words(wordlist, kuerzel):
    w_list, k_list = open_file(wordlist), open_file(kuerzel)
    return sorted(set([word for word in w_list if not
is_possible(word, k_list)]))


if __name__ == '__main__':
    nr_letters = 4
    if len(sys.argv) == 2:
        nr_letters = sys.argv[1]
    wordlist = "wordlist.txt"
    kuerzel = "../txt/kuerzelliste.txt"
    words = find_words(wordlist, kuerzel)
    [print(w) for w in words if len(w) == nr_letters]
```

Die Funktion *is_possible(s, kuerzel)* spaltet *s* am Index $i \in [1, \min(4, len(s)) - 1]$ in zwei Substrings s_1 und s_2 auf und überprüft ob sich s_1 in $kuerzel$ befindet und s_2 maximal zwei Zeichen lang ist und keine Sonderzeichen enthält.

Das Script kann mit

./4b.py [len]

ausgeführt werden und gibt alle Wörter der Länge *len* aus der Wortliste *wordlist.txt* aus, die nicht auf einem Nummernschild dargestellt werden können.

Beispielwörter:

Länge 4:	Länge 3:	Länge 2:
ADAC	ICH	IM
ADAM	IHM	OB
ADEL	IHR	OK
ADER	IST	TU
AFFE	ORT	UM
ALEX	TAG	
ALGE	TAL	
ALLE	TAT	
ALSO	TAU	
ALTE	TUN	
ARIE	TYP	
ARME	UFO	
AXEL	UKW	
BREI	USA	
BROT	ICH	

3. Schreibe ein Programm, das ein Wort einliest und überprüft, ob es mit mehreren Kennzeichen geschrieben werden kann und eine Folge von Kennzeichen ausgibt, mit denen das Wort gebildet werden kann

Lösungsidee

Mithilfe der Funktion *is_possible()* aus c) können wir nun überprüfen, ob eine Zeichenkette der Länge 2-5 auf einem einzigen Nummernschild darstellbar ist. Nun müssen wir nur noch überprüfen, ob für ein gegebenes beliebig langes Wort eine Aneinanderreihung auf einem Kennzeichen darstellbarer Zeichenketten existiert.

Umsetzung

Die Funktion *is_possible()* kann beinahe unverändert übernommen werden. Da die Funktion in diesem Programm sehr oft aufgerufen wird und durch die Zeile

```
if not any((c in umlaute) for c in post) and pre in kuerzel:
```

genauer gesagt durch den Ausdruck

```
pre in kuerzel
```

eine Laufzeit von $O(n)$ hat, habe ich sie durch folgende Zeile ersetzt.

```
if not any((c in umlaute) for c in post) and \
binary_search(KUERZEL, pre):
```

Somit hat *is_possible()* eine Laufzeit von O(log n). Die binäre Suche bietet sich auch deswegen sehr gut an, da „kuerzelliste.txt“ bereits alphabetisch sortiert vorliegt.

Im Durchschnitt terminiert jetzt das Programm nach 0,006 s (davor 0,8 s), wenn es alle Wörter aus der Liste „autoscrabble.txt“ überprüft.

Um nun zu überprüfen, ob ein Wort mit mehreren Kennzeichen geschrieben werden kann, wird *check(s)* ausgeführt.

Diese Funktion funktioniert folgendermaßen:

Es gibt einen Stack *stack* der in jedem Eintrag den Startindex des betrachteten Bereichs von s und die Länge dieses Bereichs speichert.

Es gilt:

I. $stack[0][0] = 0$

II. $stack[n+1][0] = stack[n][0] + stack[n][1]$

III. $2 \leq stack[n][1] \leq 5$

Das heißt, der Stack speichert direkt aneinanderhängende Bereiche der Länge 2–5 in *s*.

Solange der im letzten Element des Stacks gespeicherte Index kleiner als die Länge von *s* ist, wird

$$i, n = stack[len(stack) - 1]$$

bestimmt und entweder (a) oder (b) ausgeführt.

a) wenn $word[i: i + n]$ auf einem einzigen Nummernschild darstellbar ist, wird ein neues Element welches nach Regel II als Index den Wert $i + 1$ und für die Länge des nächsten Abschnittes den Wert 5 speichert.

b) sonst (wenn $word[i: i + n]$ nicht auf einem einzigen Nummernschild darstellbar ist)

a. ist $n > 2$ wird n um 1 dekrementiert

b. sonst (wenn $n \leq 2$) wird das oberste Element im Stack entfernt.

a. Ist der Stack jetzt leer, bedeutet dies, dass das Wort nicht mit mehreren Nummernschildern darstellbar ist und *False* zurückgegeben werden kann.

b. Sonst wird nochmal das Element oben auf dem Stack entfernt und beim Vorgänger der Wert der Länge um 1 dekrementiert

Nach Durchlauf dieser Schleife wird das oberste Element entfernt.

Mit folgender List-Comprehension werden die Elemente aus *stack* in Substrings aus *word* „übersetzt“: `comb = [word[i: i + n] for i, n in stack]`

Beispiele

Am Beispielwort *s* = „BIBER“ wird nun der Ablauf der Funktion $check(s)$ nochmal erläutert.

Stack: [0, 5]

```
„BIBER“ = s[0:5]      -> nicht darstellbar
```

Stack: [0, 4]

```
„BIBE“  = s[0:4]      -> nicht darstellbar
```

Stack: [0, 3]

```
„BIB“   = s[0:3]      -> darstellbar
```

Stack:

[3, 5]
[0, 3]

```
„BIB“  = s[0:3]
„ER“   = s[3:3+5]      -> nicht darstellbar
```

…

```
„BIB“ = s[0:3]
„ER“  = s[3:3+2]       -> nicht darstellbar
```

Stack:

[0, 2]

```
„BI“  = s[0:2]         -> darstellbar
```

Stack:

[2, 5]
[0, 2]

```
„BI“  = s[0:2]
„BER“ = s[2:2+5]       -> darstellbar
```

→ BIBER darstellbar: BI + BER

./autoscrabble.py autoscrabble.txt kuerzelliste.txt

```
BIBER
[[0, 5]]
[[0, 4]]
[[0, 3]]
[[0, 3], [3, 5]]
[[0, 3], [3, 4]]
[[0, 3], [3, 3]]
[[0, 3], [3, 2]]
[[0, 2]]
[[0, 2], [2, 5]]
[[0, 2], [2, 5], [7, 5]]
Darstellbar!
['BI', 'BER']

BUNDESWETTBEWERB
[[0, 5]]
[[0, 4]]
[[0, 3]]
[[0, 3], [3, 5]]
[[0, 3], [3, 4]]
[[0, 3], [3, 4], [7, 5]]
[[0, 3], [3, 4], [7, 4]]
[[0, 3], [3, 4], [7, 3]]
[[0, 3], [3, 4], [7, 2]]
[[0, 3], [3, 3]]
[[0, 3], [3, 3], [6, 5]]
[[0, 3], [3, 3], [6, 4]]
[[0, 3], [3, 3], [6, 4], [10, 5]]
[[0, 3], [3, 3], [6, 4], [10, 4]]
[[0, 3], [3, 3], [6, 4], [10, 3]]
```

```
[[0, 3], [3, 3], [6, 4], [10, 3], [13, 5]]
[[0, 3], [3, 3], [6, 4], [10, 3], [13, 4]]
[[0, 3], [3, 3], [6, 4], [10, 3], [13, 3]]
[[0, 3], [3, 3], [6, 4], [10, 3], [13, 2]]
[[0, 3], [3, 3], [6, 4], [10, 2]]
[[0, 3], [3, 3], [6, 4], [10, 2], [12, 5]]
[[0, 3], [3, 3], [6, 4], [10, 2], [12, 5], [17, 5]]
Darstellbar!
['BUN', 'DES', 'WETT', 'BE', 'WERB']

(Ab jetzt ohne Ausgabe des Stacks)

CLINTON
Darstellbar!
['CLI', 'NTON']

DONAUDAMPFSCHIFFFAHRTSKAPITÄNSMÜTZE
Nicht darstellbar!

ETHERNET
Nicht darstellbar!

INFORMATIK
Darstellbar!
['INFO', 'RM', 'ATIK']

LLANFAIRPWLLGWYNGYLLGOGERYCHWYRNDROBWLLLLANTYSILIOGOGOGOCH
Darstellbar!
['LLA', 'NFAI', 'RPW', 'LLG', 'WYN', 'GYL', 'LGO', 'GE', 'RY',
'CH', 'WYR', 'ND', 'ROB', 'WLLL', 'LA', 'NTY', 'SILI', 'OGOG',
'OGO', 'CH']

RINDFLEISCHETIKETTIERUNGSÜBERWACHUNGSAUFGABENÜBERTRAGUNGSGESETZ
Nicht darstellbar!

SOFTWARE
Darstellbar!
['SOFT', 'WARE']

TRUMP
Nicht darstellbar!

TSCHÜSS
Nicht darstellbar!

VERKEHRSWEGEPLANUNGSBESCHLEUNIGUNGSGESETZ
Darstellbar!
['VERKE', 'HRS', 'WEGE', 'PLAN', 'UNGS', 'BES', 'CH', 'LEU',
'NIG', 'UNGS', 'GE', 'SETZ']
```