Aufgabe 1: Wörter aufräumen

Team-ID: 00372

Team-Name: Der Teufel schreibt Java

Bearbeiter/-innen dieser Aufgabe:
Frederico Aberle

23. November 2020

Inhaltsverzeichnis

1 Lösungsidee

Ein Wort kann einer Lücke eindeutig zugeordnet werden, wenn es von allen Wörtern nur ein einziges gibt, das in die Lücke passen kann. Da wir davon ausgehen, dass es nur eine einzige Lösung gibt, besteht die Idee darin, nach und nach eindeutige Wörter in die Lücke zu füllen und sie von unserer Liste an Wörtern zu streichen. Das macht man solange, bis keine Wörter mehr übrig sind.

2 Umsetzung

Die Eingabe wird in eine Liste mit den Lücken und in eine Liste mit den Wörtern konvertiert. Damit man die Wörter nach dem Streichen immer noch den Lücken zuordnen kann, muss eine Kopie von den Wörtern erstellt werden. Erst dann können die eindeutigen Wörter von der Kopie entfernt werden und man sieht welche Wörter noch übrig sind.

Die Funktion, die prüft, ob Wort und Lücke zueinander passen können, sieht wie folgt aus: Man iteriert solange die Unterstriche der Lücke bis man auf Buchstaben der Lücke trifft. Man nimmt die Indexe dieser Buchstaben und vergleicht ihn mit den Buchstaben des Worts auf denselben Indexen. Sind die Buchstaben gleich, können Lücke und Wort zueinander passen. Die Lücken ohne Buchstaben werden dabei außen vor gelassen und spielen erst später eine Rolle, wenn alle Lücken mit einem Buchstaben schon zugeordnet sind.

In einer while-Schleife werden nun alle eindeutigen Lücken nach und nach mit den Wörtern ausgefüllt: Zuerst erstellt man eine Liste mit boolean-Werten, um zu wissen, ob ein Wort schon zugeordnet wurde. Das hat den Vorteil, dass man es einfach überspringen kann (Funktion überprüft den Wert und gibt gegebenenfalls False aus) und die Wörter der Kopie nicht nochmal einer schon ausgefüllten Lücke zugeordnet werden. Dazu muss erst gezählt werden, welche Lücken eindeutig sind. In einer Liste wird gespeichert, wie viele Wörter in die jeweilige Lücke passen können. Dies erreicht man mit zwei for-Schleifen, die durch die Lücken iteriert und durch die übrigen Wörter in der Kopie iteriert. Dabei unterscheidet man zwei Fälle:

1. Fall (Satzanfang): Es handelt sich um einen Satzanfang, wenn das Element vor der Lücke ein „.“ „!“ oder „?“ ist. Wenn dann noch die Länge der Lücke und die Länge des Worts der Kopie gleich sind, Wort und Lücke von den Buchstaben passgenau sind und der Anfangsbuchstabe des Worts ein Großbuchstabe ist, dann wird der Zähler der Lücke um eins erhöht. Man muss jedoch beachten, dass der Zähler nicht erhöht werden darf, wenn das Wort der Kopie beim Iterieren schon vorgekommen ist, um Mehrfachzählungen eines Worts zu verhindern. Dazu iteriert man nochmal durch die Kopie der Wörter. Wenn das übrige Wort schon vorgekommen ist, soll der Zähler nicht erhöht werden.

Hinweis: Normalerweise geht man davon aus, dass die erste Lücke auch ein Satzanfang ist und deshalb dazu gezählt werden muss. Jedoch beginnt das Beispiel raetsel0.txt mit einem Kleinbuchstaben („oh je, was für eine Arbeit!“) und würde daher eine falsche Ausgabe liefern. Wenn man dieses Beispiel aber nicht beachtet wird, muss nur in der Bedingung if i != 0 entfernt werden, um die erste Lücke als Satzanfang dazuzuzählen, (da text[0-1] = text[-1] ist und somit auf das letzte Element von Text zurückgreift, welches in einem deutschen Satz eines der Satzzeichen „.“ „!“ oder „?“ ist).

2. Fall (kein Satzanfang): Sonst wenn die Länge der Lücke und die Länge des Worts der Kopie gleich sind und Wort und Lücke von den Buchstaben passgenau sind, dann wird der Zähler der Lücke um eins erhöht. Man muss jedoch beachten, dass der Zähler nicht erhöht werden darf, wenn das Wort der Kopie beim Iterieren schon vorgekommen ist, um Mehrfachzählungen eines Worts zu verhindern. Dazu iteriert man nochmal durch die Kopie der Wörter. Wenn das übrige Wort schon vorgekommen ist, soll der Zähler nicht erhöht werden.

Im nächsten Schritt iteriert man diesmal in einer while-Schleife und in einer for-Schleife durch die Lücken und die übrigen Wörter der Kopie. Wenn bei beiden Längen und Buchstaben passen und nun der Zähler bei dieser Lücke auf eins gesetzt ist, kann der Lücke das Wort zugeordnet werden. Man sucht den Index des Wortes von der Kopie in der Liste von Wörtern und ordnet das Wort von der Liste der Lücke zu. Zum Schluss muss man das übrige Wort noch von der Kopie entfernen und setzt den boolean-Wert auf False. Der Nachteil einer for-Schleife ist, dass man nach Entfernen der Kopie der Index bei allen nachkommenden Elementen sich um eins verkleinert. Deswegen erhöht man den Index bei der while-Schleife nur, wenn man kein Wort der Kopie zugeordnet hat. Bei einer Zuordnung, gelangt man durch Verkleinerung der Indexe automatisch zum nächsten Wort. Hier muss man auch wieder zwischen den beiden Fällen Satzanfang und kein Satzanfang unterscheiden. Der Fall des Satzanfang ist gleich dem kein Satzanfang. In der Bedingung muss aber nur geändert werden, dass das Element vor der Lücke ein „.“ „!“ oder „?“ ist und das erste Element des Wortes mit einem Großbuchstaben beginnt. Ansonsten sind gleiche Länge, Passgenauigkeit der Buchstaben und Zähler auf Eins wie beim zweiten Fall klar. Das Wort wird auch hier der Lücke zugeordnet. Die gesamte while-Schleife wird durch die Bedingung verlassen, wenn der Zähler bei allen Lücken null ist.

Nach Verlassen der while-Schleife sind nun alle Lücken mit Buchstaben gefüllt. Übrig bleiben Lücken ohne Buchstaben. Da es nur eine Lösung gibt, muss nur geprüft werden, ob die Länge der Lücke mit der Länge der übrigen Wörter übereinstimmt. Wenn ja, wird wieder der Index von dem Wort der Kopie in der ursprünglichen Liste der Wörter gesucht und dieses Wort dann der Lücke zugeordnet. Satzanfänge werden wieder gesondert behandelt. Zum Schluss wird der ausgefüllte Lückentext in Form einer Liste in einen normalen Text konvertiert.

3 Beispiele

Die eigenen Beispiele beginnen ab raetsel5.txt

raetsel0.txt

Ausgabe: oh je, was für eine arbeit!

raetsel1.txt

Ausgabe: Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr wütend und wurde allenthalben als Schritt in die falsche Richtung angesehen.

raetsel2.txt

Ausgabe: Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt.

raetsel3.txt

Ausgabe: Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, besonders der automatischen Verarbeitung mit Digitalrechnern.

raetsel4.txt

Ausgabe: Opa Jürgen blättert in einer Zeitschrift aus der Apotheke und findet ein Rätsel. Es ist eine Liste von Wörtern gegeben, die in die richtige Reihenfolge gebracht werden sollen, so dass sie eine lustige Geschichte ergeben. Leerzeichen und Satzzeichen sowie einige Buchstaben sind schon vorgegeben.

raetsel5.txt

Eingabe:

__us __u_

laut haus

Ausgabe: haus laut

raetsel6.txt

Eingabe:

____h _i__ n____ __s_?

nich mich mach nass

Ausgabe: mach mich nich nass?

Hinweis: In diesen Beispiel muss in Zeile 58 d!=0, in Zeile 94 h!=0 und in Zeile 123 i!=0 entfernt werden

raetsel7.txt

Eingabe:

__s H___, __s ___r __t.

das Haus Das leer ist

Ausgabe: Das Haus, das leer ist.

raetsel8.txt

Eingabe:

__s H___, __s ___r __t. W__ i__ d__?

das Haus Das leer ist das ist Was

Ausgabe: Das Haus, das leer ist. Was ist das?

raetsel9.txt

Eingabe:

__s H___, __s ___r __t. _i_ i__ __e?

das Haus Das leer ist wie ist Wie

Ausgabe: Das Haus, das leer ist. Wie ist wie?

raetsel10.txt

Eingabe:

___ H___ ___ l___.

ist leer Das Haus

Ausgabe: Das Haus ist leer.

4 Quellcode

```
1
"""ueberprueft, ob das Wort aus 'wordsRemain' mit der Luecke in 'text' uebereinstimmt"""
3 def isPair(firstPair,secondPair,n):
    if wordsToCheck[n]:
5       for c in range(0,len(firstPair)):
           if firstPair[c] != "_":
7             if firstPair[c] != secondPair[c]:
                 return False
9       return True
    return False
11

13 """zaehlt die moeglichen Woerter, die in die Luecke passen"""
counter = [0]*len(text)
15 for d in range(0,len(text)):
   for e in range(0,len(wordsRemain)):
17       if d != 0 and (text[d-1] == "." or text[d-1] == "!" or text[d-1] == "?"):
           if len(text[d]) == len(wordsRemain[e]) and isPair(text[d],wordsRemain[e],d)
19             if ord(wordsRemain[e][0]) >= 65 and ord(wordsRemain[e][0]) <= 90:
                  for f in range(0,e):
21                   if wordsRemain[e] == wordsRemain[f]:
                        break
23                for else:
                     counter[d] += 1
```

```
        elif len(text[d]) == len(wordsRemain[e]) and isPair(text[d],wordsRemain[e],d):
           for f in range(0,e):
              if wordsRemain[e] == wordsRemain[f]:
                 break
           else:
              counter[d] += 1


"""fuellt die passenden Luecken aus und entfernt sie von 'wordsRemain'"""
g = 0
while g < len(wordsRemain):
    for h in range(0,len(text)):
        if h != 0 and (text[h-1] == "." or text[h-1] == "!" or text[h-1] == "?"):
           if len(wordsRemain[g]) == len(text[h]) and isPair(text[h],wordsRemain[g],h)
               if ord(wordsRemain[g][0]) >= 65 and ord(wordsRemain[g][0]) <= 90
                   ifcounter[h] == 1:
                      text[h] = words[words.index(wordsRemain[g])]
                      wordsRemain.remove(wordsRemain[g])
                      wordsToCheck[h] = False
                      break
        elif len(wordsRemain[g]) == len(text[h]) and isPair(text[h],wordsRemain[g],h)
            if counter[h] == 1:
               text[h] = words[words.index(wordsRemain[g])]
               wordsRemain.remove(wordsRemain[g])
               wordsToCheck[h] = False
               break
        elif h == len(text)-1:
           g += 1


"""fuellt die restlichen Luecken mit den noch uebrigen Woertern aus"""
for i in range(0,len(text)):
    if text[i][0] == "_":
       if i != 0 and (text[i-1] == "." or text[i-1] == "!" or text[i-1] == "?"):
          for j in range(0,len(wordsRemain)):
             if len(text[i]) == len(wordsRemain[j])
                 if ord(wordsRemain[j][0]) >= 65 and ord(wordsRemain[j][0]) <= 90:
                    text[i] = words[words.index(wordsRemain[j])]
       else:
          for j in range(0,len(wordsRemain)):
             if len(text[i]) == len(wordsRemain[j]):
                text[i] = words[words.index(wordsRemain[j])]
```