

Aufgabe 1: Wörter aufräumen

Team-ID: 00981

Team-Name: Die Dullis Reloaded

Bearbeiter/-innen dieser Aufgabe:
Lasse Friedrich

23. November 2020

Analyse	1
Lösungsidee	1
Umsetzung	1
Beispiele.....	2
Quellcode	5
Literaturverzeichnis	5

Analyse

Die Aufgabe besteht darin, anhand einer gegebenen Wörterliste einen Lückentext sinnvoll zu füllen, sodass am Ende ein grammatikalisch korrekter Text entsteht, bei dem alle vorhandenen Wörter benutzt worden sind.

Lösungsidee

Hierzu wird als Kerngedanke ein iteratives Ausschlussverfahren verwendet. Iterativ bezeichnet die schrittweise Annäherung zu einer Lösung durch methodisches Durchprobieren aller möglichen Kombinationen. Ähnlich zu einem Vorhängeschloss, welches durch Ausprobieren aller Kombination geöffnet werden kann, ist es auch möglich, den richtigen Text zu ermitteln, indem für jede Textlücke, anhand vorgegebener Buchstaben und Anzahl der Wortlänge, mögliche Wörter aus der Liste herausgefiltert werden.

Sobald jeder dieser Lücken eine Liste an möglichen Wörtern besitzt, werden zuerst die Lücken besetzt, bei denen es nur ein mögliches Wort gibt. Dadurch, dass diese Wörter schon sicher vergeben sind, kann die Wörterliste um die schon vergebenen Wörter gekürzt werden. Das bietet den Vorteil, dass nun die anderen Lücken insgesamt weniger Wörter haben, aus denen ausgewählt werden muss.

Durch das Wiederholen dieser Methode fallen immer mehr Wörter und Lücken weg. Eine Lücke mit vier möglichen Wörtern, hat nach genügend Iterationen, nur noch ein passendes Wort, dass genau passt.

h, _a__r_e_b_!
arbeit eine für je oh was

↓ mögliche Wörter nach der ersten Iteration,
bei der zweiten Lücke gibt es zwei Möglichkeiten

je
oh oh was für eine arbeit

h, _a__r_e_b_!
arbeit eine für je oh was

↓
Jede Lücke mit nur einem möglichen Wort wurde besetzt.
Die zweite Lücke hat nur noch eine Möglichkeit.

je

oh __, was für eine arbeit!
~~arbeit eine für je oh was~~

Umsetzung

Die oben beschriebene Lösung wurde in der Programmiersprache Python und den dazugehörigen Standard Bibliotheken umgesetzt.

Um die Anzahl der Möglichkeiten zu reduzieren, wird nicht jedes Wort in jede Lücke eingesetzt, sondern im Vorhinein selektiert, welches Wort in welche Lücke passen könnte. Die entscheidenden Kriterien hierfür sind erstens, die wenn vorhandenen, vorgegebenen Buchstaben einer Lücke und zweitens die Größe der Lücke selbst. Somit würde es zum Beispiel bei der Lücke `_h` wenig Sinn machen das Wort „*je*“ auszuprobieren, da es offensichtlich ist, dass „*je*“ an zweiter Stelle kein `h` besitzt. Auch wenn das Wort sonst von der Anzahl der Buchstaben übereinstimmen würde. Andererseits kann davon ausgegangen werden, dass das Wort „*eine*“ für `_h` auch nicht infrage kommt, da die Lücke nur zwei Buchstaben groß ist, während „*eine*“ jedoch vier Buchstaben lang ist.

Wenn also ein Lückentext gelöst werden soll, wird nach dem Lesen der Textdatei und dessen Umwandlung in zwei Listen, damit begonnen, dass für jede Lücke ein Objekt der Klasse `Gap()` erstellt wird. Diese beinhaltet Parameter wie zum Beispiel die Größe der Lücke, ob es ein Satzzeichen gibt und ob es bereits einen Buchstaben, inklusive Position gibt.

Danach ruft die `main()` Methode in jedem `Gap()` Objekt die Methode `evaluate()` auf. Welche dafür verantwortlich ist, anhand einer übergebenen Wörterliste herauszufinden, welche Wörter aus dieser Liste in diese Lücke passen könnten. Sollte es nur ein mögliches Wort geben, wird dieses als *finalword* in diese Lücke gesetzt. Wenn das Eintritt, wird dieses *finalword* als *return* Wert an die `main()` Funktion zurückgegeben und aus der Liste aller noch verfügbaren Wörter entfernt. Dadurch verkleinert sich die Wörterliste, welche in der nächsten Iteration wieder an alle Lücken- Objekte übergeben wird.

Sobald alle Lücken ein *finalword* besitzen, werden diese nacheinander ausgegeben. Zudem werden die zuvor gespeicherten Satzzeichen wieder hinzugefügt. Diese wurden zuvor entfernt, um die Lücken leichter mit den Wörtern vergleichen zu können.

Ein Problem entsteht, wenn es ein Wort mehrfach gibt. Wenn es zum Beispiel zweimal das Wort „*was*“ und es zwei identische Lücken dafür gibt, hat jede der beiden Lücken immer mindestens zwei Möglichkeiten zur Besetzung. Da eine Lücke jedoch nur dann besetzt wird, wenn sie nur noch eine Möglichkeit hat, führt solch ein Duplikat zu einer Endlosschleife. Zur Vermeidung wird jede Wörterliste einer Lücke auf Duplikate geprüft. Bleiben nur noch Duplikate übrig, wird das erste Element als *finalword* verwendet.

Beispiele

Das Programm wird mit „`python Aufgabe1.py |Pfad|`“ aufgerufen. Im Folgenden werden die gegebenen Beispiele behandelt.

1. python Aufgabel.py Test0.txt

_h __, _a_ __r __e __b__!

arbeit eine für je oh was

---Found Solution!---

oh je, was für eine arbeit!

>>> Dieses Beispiel wurde oben schon angebracht, ohne Probleme lösbar.

2. python Aufgabel.py Test1.txt

_m __a__ __e __s __e__ ___. D__ _a__ __i__ __u__ __
 __e__ _n_ _u__ __l_ _h__ __h__
 __e__.

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

---Found Solution!---

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

>>> In diesem Beispiel war es wichtig, die Datei im UTF-8 Standard[1] einzulesen, da es sonst Probleme mit
 den Umlauten gibt.

3. python Aufgabel.py Test2.txt

_s __e__ _a__ e__ __n_ _u__ __m__ ___, _a__ _r s__
 _n__ m__ t__ u__ m__ __e__ __e__.

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

---Found Solution!---

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

>>> Hier gab es keine Probleme. Durch die obige Methode konnte es gelöst werden.

4. python Aufgabel.py Test3.txt

_____ _t d_____ _e_____ _n _e_____ _g,
 _e_____, _a_____ _n_____ _r_____ _n_____ _o_____, _____ _e_
 _____ _t_____ _a_____ _i_____.

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

---Found Solution!---

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

>>> Hier kam das erste Mal das Problem mit den Duplikaten auf.

5. python Aufgabel.py Test4.txt

_a_____ _n_____ _t_____ _n_____ _i_____ _e_____ _s d_____ _e_____ _d_____ _i_____ _e_
 _t____. _s_____ _s_____ _n_____ _e_____ _n_____ _n_____ _e_____, _i_____ _i_____ _i_____ _h_____ _e_
 _b_____ _d_____ _s_____, _o_____ _s_____ _s_____ _i_____ _t_____ _e_____ _r_____.
 _____ _n_____ _n_____ _a_____ _s_____ _e_____ _b_____ _d_____ _n_____ _e_____.

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

---Found Solution!---

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.

>>> obwohl das hier das letzte Beispiel und somit das schwierigste sein sollte, kam es nach den Anpassungen im
 oberen Teil, hier zu keinen Komplikationen.

Quellcode

```
import sys

import operator

from itertools import count

#wandelt den Input in eine Lücken Liste und eine Wörter Liste um

def ProcessInput(path):

    #UTF-8 wird benutzt um Umlaute anzeigen zu können

    file = open(path, "r", 1, encoding="utf-8")

    if file.mode == "r":

        contents = file.readlines()

        available_gaps = contents[0].split()

        words = contents[1].split()

        gaps = []

        gaps = sorted(gaps, key=operator.attrgetter('length','has_letter'))

        evaluated_gaps = 0

        #Schleife, die jede Lücke dazu aufruft zu überprüfen, ob sie mit den zurzeitigen Wörter

        #ein eindeutiges Wort herausfiltern können

        while evaluated_gaps is not len(gaps):

            evaluated_gaps = 0

            #entfernt alle Wörter die schon sicher vergeben sind aus der Wörter Liste

            for gap in gaps:

                evaluated_word = gap._evaluate_(words)

                if evaluated_word is not None:

                    words.remove(evaluated_word)

            for gap in gaps:

                if gap.has_finalword:

                    evaluated_gaps += 1

            print("Gaps filled: ", evaluated_gaps, " / ", len(gaps))

        #Ausgabe, wenn das Programm eine Lösung gefunden hat

        result = sorted(gaps, key=operator.attrgetter('position'))
```

#eine Lückeklasse die für jede zu besetzende Lücke erschaffen wird

```
class Gap():
```

```
    counter = count(0)
```

```
    def __init__(self, word, has_punctuation, punctuation):
```

```
        self.finalword = word
```

```
        self.length = len(self.finalword)
```

```
        self.position = next(self.counter)
```

```
        self.has_finalword = False
```

```
        self.letter = None
```

```
        self.letterIndex = None
```

```
        self.possible_words = []
```

```
#Methode die potentiellen Wörter für eine Lücke durchsucht
```

```
def _evaluate_(self, current_allwords):
```

```
    #wenn die Lücke schon besetzt ist
```

```
    if self.has_finalword:
```

```
        return None
```

```
    #Pool an möglichen Wörtern
```

```
    self.possible_words = []
```

```
    #Wenn es ein Wort gibt mit einem Buchstaben an derselben Stelle wie die Lücke
```

```
    #fügt man es, wenn es auch dieselbe Länge hat zu einem Pool hinzu
```

```
    for word in current_allwords:
```

```
        if self.length is len(word):
```

```
            if self.letter is not None:
```

```
                if self.letter is word[self.letterIndex]:
```

```
                    self.possible_words.append(word)
```

```
            else:
```

```
                self.possible_words.append(word)
```

```
    #Wenn es nur ein mögliches Wort gibt, ist dass, das finale Wort für diese Lücke
```

```
    if len(self.possible_words) is 1:
```

```
        self.finalword = self.possible_words[0]
```

```
        self.has_finalword = True
```

```
        return self.finalword

    self.counter = 1

    self.duplicate = ""

    #überprüft, ob es duplikate gibt, die sich an mehreren Stellen einsetzen lassen

    self.dupes = [x for n, x in enumerate(self.possible_words) if x in self.possible_words[:n]]

    if len(self.dupes) is not 0:

        self.duplicate = self.dupes[0]

    for pos in self.possible_words:

        if pos is self.duplicate:

            self.counter += 1

    #Wenn der Pool nun noch Duplikate eines Wortes enthält und es sonst keine möglichen Wörter gibt

    #gibt man der Lücke dieses Duplikat als finales Wort

    if len(self.dupes) is 1 and self.counter is len(self.possible_words):

        self.finalword = self.dupes[0]

        self.has_finalword = True

        return self.finalword

    return None
```

Literaturverzeichnis

[1]Python Docs
<https://docs.python.org/3/howto/unicode.html>
(abgerufen am 22.11.2020)