Aufgabe 1: Störung

Team-ID: 01103

Team: Skadi Baumgarte

Bearbeiter/-innen dieser Aufgabe:
Skadi Baumgarte

17. November 2022

Inhaltsverzeichnis

Lösungsidee

Um diese Aufgabe zu lösen, habe ich das Problem in kleinere Probleme aufgeteilt, damit nicht immer mit dem ganzen Text gearbeitet werden muss. Hierfür wird im Text erst mal nur das erste Wort des Lückentextes gesucht und dann alle Möglichkeiten als Phrasen mit der selben Anzahl an Wörtern wie der Lückentextes gespeichert. Mit diesen Phrasen kann dann weiter gearbeitet werden, ohne dass der komplette Buchtext benutzt werden muss. Um dann zu überprüfen, ob die restlichen durch den Lückentext gegebenen Wörter auch stimmen, wird jeweils ein Wort mit dem Wort oder

Unterstrich des Lückentexts verglichen. Ist beim Lückentext ein Unterstrich gegeben wird mit dem nächsten Wort weiter gemacht. Wenn jedoch ein Wort gegeben ist, kann es entweder das selbe sein, auch dann wird das nächste Wort überprüft, oder es ist eben ein anderes Wort dann wird diese Möglichkeit gelöscht.

Umsetzung

Der gesamte Code wurde in 2 Klassen umgesetzt da man so in der Theorie das Ganze für mehre Buchtexte oder Lückentexte anwenden könnte.

Book

Die Klasse Book wird benutzt für alle Methoden die etwas mit dem Buchtext zu tun haben.

get_start

In der man eine Methode get_start werden alle Orte raus gesucht, an denen das Startwort steht. Dafür wird die Methode find() der Klasse str benutzt, da diese jedoch ohne einen Start Parameter immer nur das erste mal ausgeben würde, wo das Wort steht, muss man immer hinter dem letzten gefundenen Wert also Möglichkeit für das Startwort weiter suchen. So wird die Rückgabe des letzten Aufrufs plus 1 gerechnet und dann als neuer Startwert benutzt, außer der Wert wurde -1, was heißt, dass kein weiteres Wort mehr gefunden wurde. In diesem Fall wird eine Liste mit allen gespeicherten Startwerten zurückgegeben

get_end

In der Methode get_end wird für jedes Startwort der Ort rausgesucht, an der das letzte Wort einer Phrase endet in Abhängigkeit der Anzahl der Wörter des Lückentextes. Hier für wird der Text ab der Stelle, wo das Startwort beginnt, nach Leerzeichen durchsucht, wobei jeder Buchstabe überprüft wird und sobald die Anzahl der gefundenen Leerzeichen, der Anzahl der Wörter entspricht, wird die aktuelle Position im Text als Endwert zu dem Startwert des Startworts gespeichert.

words

In der Methode words wird für einen eingegebenen Text das Ende jedes in diesem Text enthaltenen Worts in einer Liste zurückgegeben. Hierfür wird einfach für jedes Zeichen hochgezählt und, falls das Zeichen ein Leerzeichen sein sollte, dieses als neuer Eintrag in einer Liste gespeichert.

found

Diese Methode gibt für alle möglichen Startpositionen, die Phrase und wo die Wörter enden, aus. Dafür werden erst die Methoden get_start und get_end aufgerufen und dann für jeden Eintrag in der Liste start, der String zwischen dem Startwert und dem Endwert gespeichert, sowie für diesen String die Methode words aufgerufen und ebenfalls gespeichert. Am Ende wird eine Liste mit allen möglichen Phrasen und eine Liste der Ende der Wörter dieser Phrasen zurückgegeben.

Cloze

Die Klasse Cloze wird benutzt, um jegliche Bearbeitung des Lückentextes und Vergleiche durchzuführen.

get_length

Die Methode bestimmst die Anzahl der Wörter im Lückentext, indem die Leerzeichen gezählt werden und gibt diese zurück.

start_word

Die Methode sucht das erste Leerzeichen und gibt alles vor diesem als Startwort zurück.

final

In der Methode wird erst die Methode words aus book für den Lückentext ausgeführt, um auch dort die Position der Leerzeichen zu erhalten. Danach wird die Methode found aus book aufgerufen und die Rüchgabe in die Phrasen und die Angaben der Leerzeichen unterteilt. Daraufhin wird für jede der Phrasen überprüft ob ...

1. alles bis dahin funktioniert hat, also ob die Phrase wirklich genau so viele Wörter hat wie der Lückentext.

2. Jedes Wort entweder gleich ist oder bei dem Lückentext ein Unterstich steht.

Sollte eine der beiden Bedingungen falsch sein wird diese Möglichkeit gelöscht, bevor alle übrigen Möglichkeiten ausgegeben werden.

Zusatz

Da die Öffnung einer txt-Datei außerhalb der Methoden geschieht, damit das ganze variabel vom Datei typ anpassbar wäre, werden, bevor die Objecte überhaupt erstellt werden, noch ein paar Dinge bearbeitet. Bei dem Buch wurden die ganzen Sonderzeichen entfernt, ebenso überschüssige Leerzeichen, da man dann einfacher damit arbeiten kann. Natürlich wurden auch beide txt-Dateien in Strings umgewandelt, um überhaupt mit ihnen arbeiten zu können. Außerdem wird alles klein geschrieben zur besseren Vergleichbarkeit.

Beispiele

Die Beispiele von der offiziellen Webseite:

stoerung0.txt

das kommt mir gar nicht richtig vor

stoerung1.txt

ich muß in clara verwandelt

ich muß doch clara sein

stoerung2.txt

fressen katzen gern spatzen

fressen katzen gern spatzen

fressen spatzen gern katzen

die Wiederholung ist beabsichtigt, da der eine Text zweimal im Buch enthalten ist

stoerung3.txt

das spiel fing an

das publikum fing an

stoerung4.txt

ein sehr schöner tag

stoerung5.txt

wollen sie so gut sein

Quellcode

```
# Definition eine Klasse "Book"
class Book(object):
    content = None
    start = None
    end = None
    prospects = None

    # Konstruktor der Klasse "Book"
    def __init__(self,book_input):
        self.content = book_input
        self.start = []
        self.end = []
        self.prospects= []

    # Suche aller Möglichkeiten, wo das Startwort vorkommt
    def get_start(self,word):
        count = 0
```

```
        while count < len(self.content):
            case = str.find(self.content, word, count)
            if count == 0:
                count = 1
            if case == -1:
                break
            else:
                count = case+1
            self.start = self.start + [case]

    # gibt mir das letze Wort für jedes Startwort auf Basis der Länge des zu suchenden Texts
    def get_end(self,ways,length):
        for i in ways:
            count = 0
            length_word = 0
            for char in self.content[i:]:
                if count < length+1:
                    length_word = length_word+1
                    if char == ' ':
                        count = count + 1
                else :
                    break
            self.end = self.end +[(i+length_word)]

    # bestimmt die Länge eines Worts
    def words(self,text):
        words_end = []
        count = 0
        for char in text:
            if  char == " ":
                words_end = words_end + [count]
            count = count + 1
        words_end = words_end + [count]
        return words_end

    # gibt für alle möglichen Startpositionen die Phrase und wo die Wörter enden aus
    def found(self,word,length):
        self.get_start(word)
        self.get_end(self.start,length)
        for i in range (0,len(self.start)):
            if self.content[self.end[i]-1] == " ":
                prospect = self.content[self.start[i]:self.end[i]-1]
            else:
                prospect= self.content[self.start[i]:self.end[i]]
            words_length = self.words(prospect)
            self.prospects = self.prospects + [(prospect, words_length)]
        return self.prospects
```

```
# Definition der Klasse "Cloze" (Lückentext)
class Cloze(object):
    text = None
    book = None

    # Konstruktor der Klasse "Cloze"
    def __init__(self,cloze_input,book_input):
        self.text = cloze_input
        self.book = Book(book_input)

    # bestimmt, wie viele Wörter in dem Lückentext sind
    def get_length (self):
        count = 0
        for char in self.text:
            if char == " ":
                count = count + 1
        return count

    # setzt das erste Wort des Lückentexts als Startwort
    def start_word (self):
        length = str.find(self.text," ")
        start_word = self.text[:length]
        return start_word

    # vergleicht für jede Möglichkeit, ob alle gegebenen Wörter übereinstimmen - das ist die
eigentliche Lösung der Aufgabe
    def final (self):
        cloze_words = self.book.words(self.text)
        ways_text_words = self.book.found(self.start_word(),self.get_length())
        ways_text = []
        ways_words = []

        for i in ways_text_words:
            ways_text = ways_text + [i[0]]
            ways_words = ways_words + [i[1]]
        for j in range (0,len(ways_text)):

            way_text = ways_text[j]
            way_words = ways_words[j]
            if len(cloze_words) != len(way_words):
                ways_text[j]= 0
            else:
                for i in range (0,len(cloze_words)-1):
                    if self.text[cloze_words[i]+1:cloze_words[i+1]] != "_" and
self.text[cloze_words[i]:cloze_words[i+1]] != way_text[way_words[i]:way_words[i+1]]:
                        ways_text[j]= 0
                        break
```

```
        for i in ways_text:
            if i != 0:
                print(i)

# Öffnen der Buchdatei mit dem Text
book = open("Alice_im_Wunderland.txt")
book_str = ""

# umwandeln der geöffneten Datei in einen String
for line in book :
    if line != "\n":
        book_str = book_str + line
        book_str = book_str.strip("\n")
        if book_str[len(book_str)-1:len(book_str)] != " ":
            book_str = book_str + " "

# Entfernen der Sonderzeichen und überschüssige Leerzeichen und alles in Kleinschreibung
umwandeln
for i in ["(",")","-","[","]","»","«",",",".","!","?"]:
    book_str = book_str.replace(i,"")
book_str= ' '.join(book_str.split())
book_str = book_str.lower()

# Öffnen der Lückentextdatei und Umwandlung in einen String
cloze = open("stoerung0.txt")
cloze_str = ""
for line in cloze :
    cloze_str = cloze_str + line
```