Aufgabe 1: Wörter aufräumen

Team-ID: 00301

Team-Name: Violent Violets

Bearbeiter/-innen dieser Aufgabe:
Christopher Besch

22. November 2020

Inhaltsverzeichnis

1 Lösungsidee

Der Eingangstext besteht aus unvollständigen Wörtern. Diese Wörter müssen mithilfe von Ersatzwörtern aus der Wörterbank ersetzt werden.

Ein Ersatzwort kann ein Wort ersetzten, wenn

1. die Länge beider Wörter gleich ist und
2. jeder Buchstabe aus dem Ersatzwort mit dem gegenüberliegenden Buchstaben, der Buchstabe an gleicher Stelle aus dem Wort, übereinstimmt, oder der gegenüberliegende Buchstabe eine freie Stelle („_“) ist. Eine freie Stelle stimmt quasi mit jedem Buchstaben überein (=Joker).

Wenn ein Ersatzwort verwendet wurde, kann es nicht erneut für ein folgendes Wort benutzt werden.

Die Aufgabe des Programmes ist es, zu erkennen, ob eine Lösung möglich ist und welches Wort mit welchem Ersatzwort zu ersetzten ist.

Dafür werden für jedes Wort alle passenden Ersatzwörter gesucht. Wenn zu manchen Wörtern mehrere Ersatzwörter passen, gibt es mehrere Möglichkeiten, diese passenden Ersatzwörter auszuwählen. Einige dieser Möglichkeiten sind unmöglich, da sie ein und dasselbe Ersatzwort mehrmals verwenden. Es muss also eine Möglichkeit gefunden werden, die diese Regel nicht verletzt.

2 Umsetzung

Die Lösungsidee wird in Python implementiert. Zuerst wird das Rätsel aus der Datei gelesen. Hierbei werden die einzelnen Wörter im Text, der ersten Zeile in der Datei, voneinander getrennt (**cut_words**). Die Nicht-Wörter vor den Wörtern, nach den Wörtern und zwischen den Wörtern werden ebenfalls für die finale Ausgabe gespeichert. In einer while-Schleife wird in dem Text nach dem ersten Wort mithilfe einer Regular Expression gesucht. Nach jedem Treffer wird der Text gekürzt, sodass das nächste Wort gefunden werden kann.

Nun wird mit der rekursiven Funktion **replace_incomplete_words** eine Lösung gesucht, sie nimmt einen Text und eine Wörterbank entgegen.

Bei jeder Ausführung dieser Funktion werden alle möglichen Ersatzwörter für das erste Wort im Text gesucht. Für jedes mögliche Ersatzwort wird die Funktion erneut aufgerufen, hier wird die Rekursion deutlich. Dieser Ausführung wird allerdings nicht der ganze Text und die ganze Wörterbank gegeben, das erste Wort wird aus dem Text und das verwendete Ersatzwort aus der Wörterbank weggelassen.

Wenn der Funktion ein leerer Text, ein Text ohne Wörter, gegeben wird, wird der Rekursionsanker getroffen, eine Lösung wurde gefunden. Da aus der Aufgabenstellung hervorgeht, dass es immer nur eine Lösung geben kann, wird die Suche abgebrochen, wenn die erste funktionierende Möglichkeit gefunden wurde. Die Funktion gibt den Text mit allen Ersatzwörtern in korrekter Reihenfolge zurück.

Zum Schluss wird der finale Text mit allen Nicht-Wörtern ausgegeben.

3 Beispiele

Nun wird das Programm mit allen Beispieldateien ausgeführt.

raetsel0.txt Es wird das Ergebnis „oh je, was für eine arbeit!“ ausgegeben.

Die Wörter wurden korrekt ersetzt und unter Verwendung der Nicht-Wörtern zu einem sinnvollem Satz zusammengefügt.

raetsel1.txt Das Ergebnis ist „Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr wütend und wurde allenthalben als Schritt in die falsche Richtung angesehen.“

Hier erkennt man, dass die Rekursion funktioniert, da das dritte Wort im Originaltext, „____e“, durch „Leute“ ersetzbar ist, allerdings muss „Leute“ für das zehnte Wort, „__u__“ verwendet werden.

raetsel2.txt „Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt.“

raetsel3.txt „Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, besonders der automatischen Verarbeitung mit Digitalrechnern.“

raetsel4.txt „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.“

Das Programm weist keine Probleme mit längeren Sätzen auf.

Eigenes Beispiel 1 Die Beispieldatei sieht wie folgt aus:

> !!!Das ...,,,,,ist,,,!!!!...! ein .,.,..,.tolles __i__iel!!!
>
> Beispiel

Mit dem Ergebnis „!!!Das ...,,,,,ist,,,!!!!...! ein .,.,..,.tolles Beispiel!!!“

Hier wird deutlich, dass das Programm kein Problem mit Wörtern ohne Lücken aufweist und ebenfalls Nicht-Wörter ganz am Anfang akzeptiert. Zudem wird deutlich, dass die Länge der Nicht-Wörter und deren Bestandteile (solange sie keine Buchstaben oder „_“ enthalten) irrelevant sind.

Eigenes Beispiel 2 Dieses Beispiel ist **raetsel4.txt** mit einer kleinen Abänderung, ein Wort aus der Wörterbank fehlt, es kann also keine Lösung geben. Dies gibt das Programm korrekt aus, „No solution could be found!“.

Eigenes Beispiel 3 Genau wie das zweite eigene Beispiel ist dieses eine Kopie von **raetsel4.txt** mit einer kleinen Abänderung, ein Wort aus der Wörterbank wurde abgeändert, „gegeben“ wurde durch „aaaaben“ ausgetauscht, es kann also keine Lösung geben. Dies gibt das Programm ebenfalls korrekt aus, „No solution could be found!“.

4 Quellcode

Es folgt der wichtigste Teil des Programmes, die rekursive Funktion, mit gekürzten Kommentaren. Das komplette Programm mit ausführlichen Kommentaren findet sich in **main.py**.

```python
def replace_incomplete_words(words, word_bank):
    # there are no words left to be replaced -> break recursion
    if len(words) == 0:
        return []

    result = []

    # no blanks in the current word -> word is already complete
    if "_" not in words[0]:
        following_replacements = replace_incomplete_words(words[1:], word_bank)
        if following_replacements is not None:
            result = [words[0]] + following_replacements
            hit = True
        else:
            hit = False
    else:
        replacement_indices = find_replacements(words[0], word_bank)

        # test every replacement
        hit = False
        for replacement_idx in replacement_indices:
            current_word_bank = word_bank.copy()
            replacement = current_word_bank.pop(replacement_idx)
            following_replacements = replace_incomplete_words(words[1:], current_word_bank)
            if following_replacements is not None:
                result = [replacement] + following_replacements
                hit = True
                break
    if hit:
        return result
    else:
        return None
```