

Aufgabe 1: Wörter aufräumen

Team-ID: ?????

Team-Name: ?????

Bearbeiter/-innen dieser Aufgabe:
Katharina Libner , Christopher Besch

12. November 2020

Inhaltsverzeichnis

1	Lösungsidee	1
2	Umsetzung	1
3	Beispiele	2
4	Quellcode	3

1 Lösungsidee

Eingegeben werden Schüler und Pakete, die mit 1 anfangend durchnummeriert sind. Jeder Schüler hat drei Wünsche, die jeweils einem der gegebenen Pakete entsprechen. Nun ist es die Aufgabe des Programmes, die/eine optimale Anordnung der Pakete und Schüler zu finden. Die Ausführung dieser Aufgabe lässt sich in drei Subaufgaben einteilen:

1. Eine Anordnung ist besser als eine andere, wenn mehr Schüler ihren ersten Wunsch bekommen haben, die anderen beiden Wünsche sind hierbei irrelevant. Zuerst können alle Pakete, die nur von einem Schüler am meisten (erster Wunsch) gewünscht werden, zugeteilt werden. Das Gleiche kann für die anderen Wünsche durchgeführt werden. (Jeder einzelne Wunsch wird im Folgenden „aktueller Wunsch“ genannt.) Hierbei ist allerdings zu beachten, dass die Pakete, die von mindestens einem Schüler als ein wichtigerer Wunsch als der aktuelle gewählt wurden, ausgenommen werden müssen. Wenn nun ein Schüler einem Paket zugeteilt worden ist, überprüft das Programm, ob nun ein Paket, dass von den zugeteiltem Schüler als wichtigerer Wunsch gewählt wurde, nur noch von einem einzelnen Schüler gewünscht wird, ist dies der Fall, wird dieses Paket ebenfalls zugeordnet. Diese Überprüfung wird immer für alle Zuteilungen durchgeführt.

2.

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. 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.“

Hier erlangt man keine neuen Erkenntnisse über das Programm.

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.“

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**.

```
1 def replace_incomplete_words(words, word_bank):
2     # there are no words left to be replaced -> break recursion
3     if len(words) == 0:
4         return []
5
6     result = []
7
8     # no blanks in the current word -> word is already complete
9     if "_" not in words[0]:
10         following_replacements = replace_incomplete_words(words[1:], word_bank)
11         if following_replacements is not None:
12             result = [words[0]] + following_replacements
13             hit = True
14         else:
15             hit = False
16     else:
17         replacement_indices = find_replacements(words[0], word_bank)
18
19         # test every replacement
20         hit = False
21         for replacement_idx in replacement_indices:
22             current_word_bank = word_bank.copy()
23             replacement = current_word_bank.pop(replacement_idx)
24             following_replacements = replace_incomplete_words(words[1:], current_word_bank)
25             if following_replacements is not None:
26                 result = [replacement] + following_replacements
27                 hit = True
28                 break
29     if hit:
30         return result
31     else:
32         return None
```
