Aufgabe 1: Wörter aufräumen

Team-ID: ?????

Team-Name: ?????

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

19. November 2020

Inhaltsverzeichnis

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 werde. (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. Nun liegen nur noch Pakete vor, die von niemandem gewünscht werden, und solche, die von mehreren gewünscht werden. Es werden alle Pakete, die von mehren Schülern gewünscht worden, durchgegangen. Hierbei werden erst alle ersten und danach alle unwichtigeren Wünsche untersucht. Für alle diese Pakete wird willkürlich ein Wünscher, ein Schüler, der sich das Paket wünscht, ausgewählt.

3. Es bleiben ausschließlich Pakete übrig die von niemandem, der noch nicht zugeteilt wurde, gewünscht werden. Diese werden komplett willkürlich auf die verbleibenden Schüler verteilt.

2 Umsetzung

Die Lösungsidee wird in Python implementiert. Zuerst werden die Pakete aus der Datei gelesen und in Student (sie enthalten die Nummer des Schülers und dessen Wünsche) und Package (sie enthalten die Nummer des Paketes und die Nummern aufgeschlüsselt nach der Wichtigkeit des Wünsche aller Schüler, die sich das Paket wünschen.) Objekte konvertiert. Es werden zwei Dictionaries verwendet, eins enthält alle Schüler Objekte und eins alle Schüler Objekte. Um die rohen Daten aus der Datei in diese beiden Dictionaries zu speichern wird die Funktion load_students_and_packages verwendet.

Diese Dictionaries werden in einem Selection Objekt gespeichert. Dieses Objekt enthält die in Abschnitt 1 beschriebenen Methoden, assign_all_cleanly, assign_all_uncleanly und assign_all_dirtily in der richtigen Anordnung. Zudem enthält es ein Dictionary (assigned_students), dieses enthält die Zuordnungen der Schüler mit den Paketen. assigned_students wird somit von den genannten Methoden angepasst. Die in Abschnitt 1 genannte Überprüfung wird mithilfe zweier rekursiver Methoden umgesetzt, assign_package_if_possible und resolve_after_assignment. assign_package_if_possible versucht ein Paket zuzuteilen, ohne einem Schüler dessen Wunsch zu nehmen, wenn dieser genauso wichtig oder wichtiger ist als der aktuelle. resolve_after_assignment führt die Überprüfung rekursiv aus, die Methode führt sich selber für alle wichtigeren Wünschen aus und führt assign_package_if_possible aus, um ein Paket einem Schüler zuzuordnen. assign_package_if_possible ruft wiederum resolve_after_assignment auf, um weitere Pakete zu überprüfen.

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**.

```
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
```