Aufgabe 1

Vorwort

Sollte die Umsetzung nicht genau genug beschrieben worden sein, ist all mein Code umfassend kommentiert, um die Hintergründe des Programms zu erläutern.

Lösungsidee

Meine Idee war, die Lücken und Lösungsworte aufzuteilen, sodass jede Lücke und jedes Wort ein eigenes Element bzw. ein eigener String ist. Dann kann man die bereits gegebenen Stellen jeder Lücke gegen jedes Wort überprüfen. Wenn es dann nur noch genau eine mögliche Lösung für diese Stelle gibt, muss diese auch die richtige sein. Dieses gelöste Paar kann dann aus den noch zu lösenden Elementen gelöscht werden, da es ja nicht mehr gelöst werden muss. Dadurch ergeben sich möglicherweise neue eindeutige Lösungen, die dann wiederum zu neuen eindeutigen Lösungen führen, bis der gesamte Lückentext gelöst worden ist. Man könnte dieses Prinzip auch als „Ausschlussverfahren“ bezeichnen.

Umsetzung

Zur Lösung dieses Problems habe ich Python gewählt. Das Programm liest die Kommandozeilenargumente ein. Daraufhin wird die Funktion ´kombinieren´ aufgerufen, welche die Daten (Lückentext und Lösungsworte) aufbereitet, sodass ihr Format für die Lösung sinnvoll ist. Dann wird eine while-Schleife initiiert, die entweder so lange läuft, bis alle Lücken gelöst wurden, oder bis nach einem vollständigen Abgleichen von noch zu lösenden Worten und Lücken keine eindeutige neue Lösung mehr gefunden wird, woraufhin das Programm den bisherigen Fortschritt ausgibt. Die Lösung wird als Kopie des Lückentextes initiiert. Innerhalb der Schleife wird jede Lücke gegen jedes Wort geprüft, wobei Lücken als mögliche Lösung Worte haben können, die entweder gleich lang wie oder ein Zeichen kürzer sind als sie selbst, da man Satzzeichen miteinbeziehen muss. Hat nun ein Wort mehr Übereinstimmungen mit der Lücke als alle anderen, ist dieses Wort die eindeutige Lösung für diese Lücke. Eine weitere Möglichkeit ist, dass das gleiche Wort mehrmals Mal in den Lösungsworten vorkommt, weshalb man die Liste der potenziellen Lösungen, die potenzielle Lösungen beinhaltet, welche alle gleich viel Übereinstimmung mit der Lücke haben, gegen sich selbst vergleichen muss. Ist dies tatsächlich der Fall, wird dieses Wort auch als eindeutige Lösung für diese Lücke angesehen. Die eindeutige Lösung wird in der Liste ´loesung´ für die gelöste Lücke ausgetauscht, wobei die Satzzeichen (die immer am Ende eines Wortes vor einem Leerzeichen stehen) erhalten bleiben. Gibt es keine eindeutige Lösung, wird einfach mit der nächsten Lücke fortgefahren. Nach einer Überprüfung aller Lücken werden die gelösten Paare aus ihren jeweiligen Listen gelöscht.

Beispiele

Als Beispiele habe ich die gegebenen Beispiele der BWInf-Materialien gewählt:

$ python3 aufgabe1.py raetsel0.txt liefert:

oh je, was für eine arbeit!

$ python3 aufgabe1.py raetsel1.txt liefert:

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

$ python3 aufgabe1.py raetsel2.txt liefert:

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

$ python3 aufgabe1.py raetsel3.txt liefert:

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

$ python3 aufgabe1.py raetsel4.txt liefert:

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.

Quellcode

```
# Zum Einlesen von Kommandozeilenargumenten
import sys
# Zum Kopieren einer Liste, deren Elemente später noch verändert werden
from copy import deepcopy


# Satzzeichen, die von Worten unterschieden werden sollen
satzzeichen = ",.-!:"



def finden(zufinden, finden_in):
    """Gibt die Stellen aus, an denen die beiden Strings ´zufinden´ und ´finden_in´ die gleichen Buchstaben innehaben."""
    # Initialisierung der Ausgabe
    uebereinstimmungen = []
    for i in range(0, len(zufinden)):
```

```
        if i <= len(finden_in)-1 and zufinden[i] == finden_in[i]:
            uebereinstimmungen.append(i)
    return uebereinstimmungen


def dopplung(liste):
    """Gleicht alle Elemente einer Liste ´liste´ mit sich selbst ab und gibt ´True´ aus, wenn sie sich entsprechen."""
    for x in liste:
        for y in liste:
            if x != y:
                return False
    return True


def ersetzen(ausgabe, luecke, ersetzenMit):
    """Tauscht das erste Elemente ´luecke´ in der Lösung ´loesung´ mit ´ersetzenMit´ aus, behält jedoch die Satzzeichen, da diese
    immer am Ende der Lücke ´luecke´ stehen."""
    # Listenweise Speicherung der Zeichen in ´luecke´ und ´ersetzenMit´
    zuErsetzen = list(luecke)
    ersetzenMit = list(ersetzenMit)
    # Ersetze jeden Buchstaben von ´zuErsetzen´ mit dem Buchstaben, den ´ersetzenMit´ an dieser Stelle hat, bis ´ersetzenMit´
    # keine Zeichen mehr hat
    for element in ausgabe:
        if element == luecke:
            for i in range(0, len(ersetzenMit)):
                zuErsetzen[i] = ersetzenMit[i]
    # Setze die Liste wieder zu einem String zusammen und füll mit diesem die gewollte Lücke
    ausgabe[ausgabe.index(luecke)] = "".join(zuErsetzen)
    return ausgabe


def zuordnen(loesung, luecken, woerter):
    """Ordnet Lücken ein Wort zu, wenn dieses die meisten übereinstimmenden Zeichen hat."""
    # Initialisierung der Liste der gelösten Stellen
    geloeste = []
```

```
    for luecke in luecken:
        # Initalisierung der Liste der potenziell passenden Worte
        passend = []
        for wort in woerter:
            # Die nachfolgenden Befehle sind nur relevant, wenn das Wort ´wort´ entweder so lang ist, wie die Lücke ´luecke´,
            # die nicht mit einem Satzzeichen endet, oder wenn das Wort ´wort´ ein Zeichen kürzer ist, nämlich das Satzzeichen
            # am Ende der Lücke ´luecke´
            if len(wort) == len(luecke) and luecke[-1] not in satzzeichen or len(wort)+1 == len(luecke) and luecke[-1] in satzzeichen:
                # Finde die Stellen heraus, an denen die aktuelle Lücke ´luecke´ und das aktuelle Wort ´wort´ gleich sind
                matches = finden(luecke, wort)
                # Ist die Liste der potenziell passenden Worte ´passend´ noch leer, oder hat das aktuelle Wort genau so viele
                # Übereinstimmungen wie die bisherige potenzielle Lösung, füg das aktuelle Wort ´wort´ hinzu
                if len(passend) == 0 or len(matches) == len(passend[-1][1]):
                    passend.append([wort, matches])
                # Hat es mehr Übereinstimmungen, überschreibe die bisherige potenzielle Lösung ´passend´ mit dem aktuellen Wort ´wort´
                elif len(matches) > len(passend[-1][1]):
                    passend = [[wort, matches]]
        # Gibt es genau einen Eintrag in der Liste der potenziellen Lösungen ´passend´, ist dieser die Lösung
        if len(passend) == 1 or dopplung(passend):
            # Ersetze die Lücke in ´loesung´ mit dem gerade bestimmten Lösungspaar ´passend´
            loesung = ersetzen(loesung, luecke, passend[0][0])
            # Hänge an ´geloeste ´ das gerade gelöste Paar von ´luecke´ und ´wort´ an
            geloeste.append([luecke, passend[0][0]])
    # Um die for-Schleifen intakt zu halten, werden die gelösten Stellen ´geloeste´ erst jetzt aus ihren Listen gelöscht
    for geloest in geloeste:
        luecken.remove(geloest[0])
        woerter.remove(geloest[1])
    return [loesung, luecken, woerter]


def kombinieren(lueckentext, worte):
    """Ruft ´zuordnen´ auf, bis eine Lösung gefunden wurde oder es keine Veränderung mehr gibt."""
    # Teile den Lückentext ´lueckentext´  und die Worte ´worte´ an jeder Leerstelle zu einer Liste
```

```
    lueckentext = lueckentext.split(" ")
    worte = worte.split(" ")
    # Initialisierung der Lösung, des Lückentextes und der Worte
    daten = [lueckentext[:], lueckentext, worte]
    # Solange es noch ungelöste Lücken gibt, rufe ´zuordnen´ auf
    while daten[1] != []:
        # Erstelle eine Kopie der Lösung für spätere Überprüfung
        daten_alt = deepcopy(daten)
        # Ruf ´zuordnen´ mit den gegeben Daten ´daten´ auf
        daten = zuordnen(daten[0], daten[1], daten[2])
        # Wird nach einer vollständigen Iteration keine neue Lösung gefunden, dann gib die aktuelle aus, um eine
        # Endlosschleife zu verhindern
        if daten == daten_alt:
            return " ".join(daten[0])
    return " ".join(daten[0])


# Wenn genau zwei Argumente gegeben wurden, lies die ersten beiden Zeilen der gegebenen Datei ein und ruf
# ´kombinieren´ damit auf, sonst weise auf die korrekte Syntax hin
if len(sys.argv) == 2:
    f = open(sys.argv[1], "r")
    print(kombinieren(f.readline().strip("\n"), f.readline().strip("\n")))
    f.close()
else:
    print("Korrekte Nutzung: aufgabe1.py <Textdatei>")
```