Aufgabe 2: Twist

Team: Richard Wohlbold
Team-ID: 00012

14. September 2018

Inhaltsverzeichnis

1 Lösungsidee

Um einen Text zu twisten, müssen zuerst die Wörter von anderen Zeichen, wie Kommas, Klammern und Ähnlichem getrennt werden, denn wir möchten nicht, dass solche Symbole in der Mitte eines Wortes stehen. Ebenfalls müssen beim Enttwisten eines Texts zuerst die getwisteten Wörter von Kommas und anderen Satzzeichen getrennt, um diese in einer Wörterliste zu finden.

1.1 Wördererkennung

Hinweis: Im Folgenden verwende ich die Schreibweise regulärer Ausdrücke, wie sie in Python mit dem `re`-Modul verwendet wird. Die Konzepte sind jedoch auch auf andere Schreibweisen übertragbar.

Um alle deutschen Wörter in einem Text zu erkennen, benutze ich reguläre Ausdrücke. Ein deutsches Wort ist demnach eine Folge von einem oder mehr deutschen Buchstaben (*a-z*, *A-Z*, *Ä*, *ä*, *Ö*, *ö*, *Ü*, *ü* und *ß*). Um damit einen deutschen Buchstaben zu erkennen, erstelle ich eine neue Buchstabenklasse, indem ich den regulären Ausdruck `[a-zA-ZÄäÖöÜü]` verwende. Um nun einen oder mehr als einen solchen Buchstaben hintereinander zu erfassen, füge ich dem regulären Ausdruck noch ein `+` hinzu, das *eins oder mehr* bedeutet: `[a-zA-ZÄäÖöÜü]+`.

Mit diesem regulären Ausdruck kann ich nun alle deutschen Wörter aus dem Text extrahieren. Die Zeichen zwischen zwei Wörtern werden weder getwistet noch enttwistet, sondern werden immer so gelassen wie sie sind. Die Benutzung eines regulären Ausdrucks hat den Vorteil, dass nicht spezifiziert werden muss, welche Zeichen Wörter trennen, sondern welche Zeichen Wörter ausmachen. Da es sehr viele Satzzeichen geben könnte, ist dieses Verfahren einfacher, solange die Aufgabe spezifiziert, welches Alphabet verwendet wird (hier ist es das Deutsche).

1.2 Twisten

Da wir nun alle Wörter in einem Text erkennen können, können wir alle Wörter getrennt betrachten. Das Twisten eines Worts geschieht in drei Schritten:

1. Falls das Wort nur einen oder zwei Buchstaben lang ist, ist Schluss und das Wort bleibt so wie es ist. Sonst wird der erste und letzte Buchstabe entfernt.
2. Die verbleibenden Buchstaben werden zufällig angeordnet.
3. Der erste und letzte Buchstabe werden wieder jeweils an den Anfang oder an das Ende hinzugefügt.

Mit diesem Verfahren lässt sich jedes Wort zufällig twisten.

1.3 Enttwisten

Die Wörterliste im Speicher Das Enttwisten ist herausfordernder als das Twisten, da das Programm ohne eine Wörterliste (oder Machine Learning) zurechtkommen kann, da es nicht weiß, welche Wörter überhaupt möglich sind. Als Vorbereitung zum Enttwisten muss eine Wörterliste eingelesen und vorbereitet werden. Damit das eigentliche Verfahren zum Enttwisten mit einer Geschwindigkeit von $O(n)$ (n ist die Anzahl der Wörter im Text) und unabhängig von der Länge der Wörterliste laufen kann, muss eine Datenstruktur verwendet werden, die gegeben einer Eingabe eines Worts spezieller Form eine Liste von möglichen Wörtern in $O(1)$ (konstanter Zeit) zuordnet. Eine geeignete Datenstruktur hierfür ist die Hashtabelle, da diese eine Zugriffszeit von $O(1)$ bietet, die ungenutzten Felder aufgrund geringer Datenmenge kein großes Problem darstellen und da diese nach initialer Befüllung nur noch zum Lesen benutzt wird.

Da jedes getwistete Wort eine beliebige Reihenfolge der mittleren Buchstaben innehaben kann, muss ein Verfahren bestehen, jede getwistete Version eines Originalworts in die gleiche Zeichenfolge umzuwandeln, ohne das Originalwort zu kennen. Währenddessen muss jedoch auch so viel Information wie möglich behalten werden (vor Allem der erste und letzte Buchstabe). Dies ist möglich, indem die mittleren Buchstaben eines getwisteten Worts alphabetisch sortiert werden, der erste und letzte jedoch gleich bleiben. Somit ist immernoch klar, was der erste und letzte Buchstabe waren und welche anderen Buchstaben in dem Wort enthalten sind. Dieses Wortform nenne ich **reduzierte Form**.

Dieses Verfahren eignet sich als Schlüssel für die Hashtabelle. Aus jedem getwistenen Wort kann diese Form abgeleitetet werden und eine Liste von enttwisteten Wörtern kann in $O(1)$ geholt werden.

Somit erstellen wir aus der Wörterliste eine Hashtabelle: Für jedes Wort wird die reduzierte Form als Schlüssel verwendet und der Liste von Wörtern mit diesem Schlüssel wird das richtige Wort angehängt.

Bestimmung der enttwisteten Wörter Wenn ein getwistetes Wort gegeben ist, muss dieses Wort nun die reduzierte Form des Wortes gebildet werden, die der Hashtabelle als Schlüssel gegeben wird. Nun gibt es drei Möglichkeiten:

1. Es gibt kein Wort für die reduzierte Form in der Hashtabelle. Hier wird das getwistete Wort so belassen, wie es ist. Je nach Länge der Wörterliste tritt dieses Problem unterschiedlich häufig auf.
2. Es gibt genau ein Wort für die reduzierte Form in der Hashtabelle. Dies ist der Idealfall, das Wort wird einfach in den Ergebnistext übernommen.
3. Es gibt mehrere Wörter für die reduzierte Form. Wir können irgendeins der Wörter nehmen, da wird nicht wissen, welches hier richtig ist. Die anderen Möglichkeiten sollten als Warnung ausgegeben werden.

Optimierung Wenn ein Wort am Satzanfang steht, wird es immer großgeschrieben, nach manchen Satzzeichen, wie einem Doppelpunkt manchmal auch. Verben werden nach einem Artikel auch großgeschrieben (*das Schwimmen*). Damit das Verfahren nicht alle diese Sonderfälle einzeln betrachtet, werden alle Schlüssel der Hashtabelle kleingeschrieben. Damit der Ausgabetext trotzdem zu einem Großteil richtig geschrieben ist, wird die Groß- und Kleinschreibung der Wörter in der Wörterliste in den Ergebnislisten der Hashtabelle übernommen. Außerdem wird ein Wort im Ausgabetext großgeschrieben, wenn das getwistete Wort im Eingabetext großgeschrieben ist, wie z.B. am Satzanfang.

2 Umsetzung

Ich habe das Verfahren in Python umgesetzt. Zur Erkennung von Wörtern mithilfe regulärer Ausdrücke benutze ich das integrierte Modul `re`. Um die Wörter von Satzzeichen zu trennen, benutze ich die Funktion `split`, die alle Treffer des Musters von allen anderen Zeichen trennt und in einer Liste zurückgibt. Zur zufälligen Anordnung mittlerer Buchstaben beim twisten benutze ich die Funktion `shuffle` aus dem Modul `random`. Als Hashtabelle verwende ich die standardmäßigen Python-`dict`s.

Um mit einem Skript zu twisten, enttwisten und beides nacheinander zu tun, parse ich mit `argparse` die Argumente. Das Skript wird mit `twist.py <aktion> <eingabedatei> [-w woerterliste]` aufgerufen, wobei Aktion entweder `twist`, `enttwist` oder `beides` sein kann. Eine Wörterliste wird nur bei `enttwist` oder `beides` benötigt.

3 Beispiele

Die Beispiele zur Twistfunktion des Programms von der bwinf-Website.

Beispiel 1:

```
$ python twist.py twist ../beispieldaten/twist1.txt
Der Twist
(Eiscnlgh tiswt = Dhureng, Vreuedhnrg)
war ein Moedtnaz im 4/4-Tkat,
der in den fheürn 1960er Jaerhn päolupr
wrdue und zu
Rcok'n'Rlol, Rhtyhm and Beuls oder sieezlpler
Tsiwt-Misuk gazentt wrid.
```

Beispiel 2:

```
$ python twist.py twist ../beispieldaten/twist2.txt
Hat der alte Hmeixeenster scih doch eniaml wegebeegbn! Und nun soelln sneie Getsier auch
    nach meniem Wellin leebn. Sinee Wort und Werke mkret ich und den Bruach, und mit
    Gietsästksere tu ich Wuednr acuh.
```

Beispiel 3:

```
$ python twist.py twist ../beispieldaten/twist3.txt
Ein Rrauseatnt, wehecls a la crate aebirett, beeitt sien Agnboet ohne eine vhreor
    fleeggestte Mrfnneoehiülgee an. Dducarh heabn die Gätse zawr mher Sielpaurm bei der
    Whal ierhr Seepisn, für das Rnrauaetst eeteshtnn jeodch zueäihczstlr Afnuwad, da
    wengeir Panrisgnhlhucsieet vohneadrn ist.
```

Beispiel 4:

```
$ python twist.py twist ../beispieldaten/twist4.txt
Autusga Ada Bryon Knig, Ctnouess of Levlaoce, war enie birsthice Adliege und
    Mitetiakmerahn, die als die etsre Pragrioemriermn üpubaerht glit. Brtiees 100 Jahre
    vor dem Akuomefmn der etsren Paerprmrgceahomsrin enarsn sie enie Rcehen-Mhnaceik, der
     egniie Kzponete mneeordr Paprhoariemcrmregsn vgowanhrem.
```

4 Quellcode (ausschnittsweise)

Die Funktion `wort_twisten`, die ein Wort twistet:

```python
def wort_twisten(wort):
    if len(wort) <= 2:
        return wort
    mittlere_buchstaben = list(wort[1:-1])
    shuffle(mittlere_buchstaben)
    return wort[0] + ''.join(mittlere_buchstaben) + wort[-1]
```

code/wort_twisten.py

Die Funktion `text_twisten`, die einen Text twistet:

```python
import re

def text_twisten(text):
    # Der reguläre Ausdruck zum Finden von Wörtern im Text
    muster = re.compile(r'([a-zA-ZäöüÄÖÜß]+)')
    result = ""
    elemente = re.split(muster, text)
    for element in elemente:
        if re.fullmatch(muster, element):
            result += wort_twisten(element)
        else:
            result += element
    return result
```

code/text_twisten.py

Die Funktion `wort_zu_schluessel`, die einen Schlüssel für die Hashtabelle aus dem Wort macht:

```python
def wort_zu_schluessel(wort):
    # Sortiere mittlere Buchstaben des Worts
    mittlere_buchstaben = ''.join(sorted(list(wort[1:-1])))
    return wort[0].lower() + mittlere_buchstaben + wort[-1].lower()
```

code/wort_zu_schluessel.py

Die Funktion `enttwist_text`, die einen Text versucht zu enttwisten:

```python
import re

def enttwist_text(text: str, woerterbuch_text: str):
    woerterbuch = woerterbuch_erstellen(woerterbuch_text.split("\n"))
    enttwisteter_text = []
    # Der reguläre Ausdruck zum Finden von Wörtern im Text
    muster = re.compile(r'([a-zA-ZäöüÄÖÜß]+)')
    woerter_und_zeichen = re.split(muster, text)
    for element in woerter_und_zeichen:
```

```
10          # Ist das Element ein Wort?
            if re.fullmatch(muster, element):
12              schluessel = wort_zu_schluessel(element)
                moegliche_woerter = woerterbuch.get(schluessel)
14              # Keine möglichen Wörter gefunden
                if moegliche_woerter is None:
16                  print("Kein enttwistetes Wort für", '"' + element + '"', "gefunden...")
                    enttwisteter_text.append(element)
18              # Ein mögliches Wort gefunden
                elif len(moegliche_woerter) == 1:
20                  if (element[0].isupper()):
                        moegliche_woerter[0] = moegliche_woerter[0][0].upper() +
    moegliche_woerter[0][1:]
22                  enttwisteter_text.append(moegliche_woerter[0])
                # Mehrere mögliche Woerter gefunden
24              else:
                    benutztes_wort = moegliche_woerter[0]
26                  print("Mehrere enttwistete Wörter für", '"' + element + '"', "gefunden: "
    , moegliche_woerter, ", benutze", '"' + benutztes_wort + '"', "...")
                    enttwisteter_text.append(benutztes_wort)
28          # Das Element ist kein Wort
            else:
30              enttwisteter_text.append(element)
    print()
32  return ''.join(enttwisteter_text)
```

code/enttwist_text.py