Übung zur Vorlesung “Einführung in die Computerlinguistik und Sprachtechnologie”

Wintersemester 2017/2018, Prof. Dr. Udo Hahn, Sven Büchel
Übungsblatt 5 vom 24.11.2017
Abgabe bis 28.11.2017, 23.59 Uhr; per Email (PDF-Format) an sven.buechel@uni-jena.de

---

Aufgabe 1 : Silbentrennung **5**

Definieren Sie eine Funktion, die geeignet ist eine vereinfachte Version der 1. Regel für Silbentrennung im Deutschen anzuwenden. Diese lautet (vereinfacht):

> Es wird vor jedem Konsonanten getrennt, auf den ein Vokal folgt.

Betrachten Sie dabei die Liste `vokale` als gegeben, die alle Vokale (auch die Umlaute) des Deutschen als Strings enthält.
Ihre Funktion soll einen String entgegennehmen und alle Stellen finden, an denen laut dieser Regel getrennt werden könnte. Schreiben Sie dazu einfach an den passenden Stellen `Trennstelle eintragen`.
Anmerkungen:

- Sie können für diese Übung mit `String[n-1]` den n-ten Buchstaben eines Strings abfragen, also etwa für `"Mama"[0]` das ”M”.
- Zudem können Sie mit `length(String)` die Länge eines Strings abfragen, im obigen Beispiel also 4.
- Sie können mit `in` bzw. `not in` testen, ob etwas (nicht) in einer Liste ist. So würde etwa ein eingerückter Block unter `if "k" not in vokale` ausgeführt werden.
- Wörter können nicht vor dem ersten Buchstaben getrennt werden!

Aufgabe 2 : Porter-Stemmer **5**

Unten stehend sehen Sie eine einfache Variante des Porter-Stemmers in Pseudocode.

Geben Sie für die nachstehenden Wörter an, in welchen Zeilen des Pseudocodes jeweils eine Veränderung des Eingabeworts durchgeführt wird.

- hospitalization
- unpredictability
- hopelessness
- replacements

Dabei reicht es, wenn Sie die jeweiligen Zeilen innerhalb der Funktionen `step1`...`step6` angeben.

Porter Stemmer

```
    //prueft ob aktuelles wort auf doppelkonsonant endet
V1  def star_d(↓word, ↑out)
2       out ← word[-1] = word[-2]

W1  def replace(↓word, ↓original, ↓replacement, ↑new_word)
```

```
2       if word ends with replacement
3           //ersetzt die Endung von word durch replacement
4           word ← word[:-length(ending)] + replacement
5       new_word ← word

    //zaelt Vokal-Konsonanten-Sequenzen im akutellen Wort (entspricht m() )
X1  def m(↓ word, ↑ count)
2       vowels ← ["a", "e", "i", "o", "u", "y"]
3       count ← 0
4       for i in 1...length(word)-1
5           if word[i] not in vowels and word[i-1] in vowels
6               count ← count+1

    // Prueft ob ein Vokal innerhalb des Stamms ist (entspricht *v*)
Y1  def star_vowel_star(↓word, ↓suffix, ↑out)
2       vowels ← ["a", "e", "i", "o", "u", "y"]
3       out ← FALSE
4       stem ← word[:-length(suffix)]
5       for v in vowels
6           if v in stem
7               out ← TRUE

    //prueft ob aktuelles Wort auf mit KONSONANT-VOKAL-KONSONANT endet
Z1  def star_o(↓word, ↑out)
2       a ← ["w", "x", "y"]
3       vowels ← ["a", "e", "i", "o", "u", "y"]
4       out ← word[-3] not in vowels and word[-2] in vowels and\
            word[-1] not in vowels and word[-1] not in a

A1  def step1 (↓raw, ↑processed)
2       raw ← replace(raw, "sses", "ss")
3       raw ← replace(raw, "ies", "i")
4       if not raw[-2] = "s"
5           raw ← replace(raw, "s", "")
6       processed ← raw

B1  def step2 (↓raw, ↑processed)
2       if m(raw) > 1
3           raw ← replace(raw, "eed", "ee")
4       if (star_vowel_star(raw, "ed") and raw ends with "ed") or\
            (star_vowel_star(raw, "ing") and raw ends with "ing")
5           raw ← replace(raw, "ed", "")
6           raw ← replace(raw, "ing", "")
7           //Sonderschritt zum Wiederdranhaengen von "e"
8           raw ← replace(raw, "at", "ate")
9           raw ← replace(raw, "bl", "ble")
10          raw ← replace(raw, "iz", "ize")
11          //Doppelkonsonant vereinfachen
12          if (not raw[-1] in ["l", "s", "z"]) and (star_d(raw))
13              raw ← raw[:-1]
14          if m(raw) = 1 and star_o(raw)
15              raw ← raw + "e"
16      processed ← raw

C1  def step3(↓ raw, ↑ processed)
2     if star_vowel_star(raw, "y")
3       raw ← replace(raw, "y", "i")
4     processed ← raw

D1  def step4 (↓ raw, ↑ processed)
```

```
2     if m(raw) > 0
3       raw ← replace(raw, "ational", "ate")
4       raw ← replace(raw, "tional", "tion")
5       raw ← replace(raw, "enci", "ence")
6       raw ← replace(raw, "anci", "ance")
7       raw ← replace(raw, "izer", "ize")
8       raw ← replace(raw, "abli", "able")
9       raw ← replace(raw, "ational", "ate")
10      raw ← replace(raw, "alli", "al")
11      raw ← replace(raw, "entli", "ent")
12      raw ← replace(raw, "eli", "e")
13      raw ← replace(raw, "ousli", "ous")
14      raw ← replace(raw, "ization", "ize")
15      raw ← replace(raw, "ation", "ate")
16      raw ← replace(raw, "ator", "ate")
17      raw ← replace(raw, "alism", "al")
18      raw ← replace(raw, "iveness", "ive")
19      raw ← replace(raw, "fulness", "ful")
20      raw ← replace(raw, "ousness", "ous")
21      raw ← replace(raw, "aliti", "al")
22      raw ← replace(raw, "iviti", "ive")
23      raw ← replace(raw, "biliti", "ble")
24      raw ← replace(raw, "icate", "ic")
25      raw ← replace(raw, "ative", "")
26      raw ← replace(raw, "alize", "al")
27      raw ← replace(raw, "iciti", "ic")
28      raw ← replace(raw, "ful", "")
29      raw ← replace(raw, "ness", "")
30    processed ← raw

E1  def step5 (↓ raw, ↑ processed)
2     if m(raw) > 1
3       raw ← replace(raw, "al", "")
4       raw ← replace(raw, "ance", "")
5       raw ← replace(raw, "ence", "")
6       raw ← replace(raw, "er", "")
7       raw ← replace(raw, "ic", "")
8       raw ← replace(raw, "able", "")
9       raw ← replace(raw, "ant", "")
10      raw ← replace(raw, "ement", "")
11      raw ← replace(raw, "ent", "")
12      raw ← replace(raw, "tion", "")
13      raw ← replace(raw, "sion", "")
14      raw ← replace(raw, "ou", "")
15      raw ← replace(raw, "ism", "")
16      raw ← replace(raw, "ate", "")
17      raw ← replace(raw, "iti", "")
18      raw ← replace(raw, "ous", "")
19      raw ← replace(raw, "ive", "")
20      raw ← replace(raw, "ize", "")
21    processed ← raw

F1  def step6 (↓ raw, ↑ processed)
2     if m(raw) > 1
3       raw ← replace(raw, "e", "")
4     if m(raw) = 1 and not star_o(raw)
5       raw ← replace(raw, "e", "")
6     if m(raw) > 1 and raw ends with "ll"
7       raw ← replace(raw, "ll", "l")
8     processed ← raw
```

```
P1  def stem_word (↓word, ↑stem)
2     stem ← lower_case(word) //alle Grossbuchstaben zu Kleinbuchstaben
3     stem ← step1(stem)
4     stem ← step2(stem)
5     stem ← step3(stem)
6     stem ← step4(stem)
7     stem ← step5(stem)
8     stem ← step6(stem)
```