Reimwörterbücher

Sven Eric Panitz

15. Mai 2021

In diesem Modul soll eine kleine Bibliothek zur Erstellung von Reimwörterbüchern umgesetzt werden.[1]

Haskell: Rhyme

```
> module Rhyme where
```

Wir werden in diesem Modul mit Strings arbeiten. Der Typ String ist in Haskell ein Synonym für `[Char]` als eine Liste von Zeichen. Daher werden wir viele Standardfunktionen für die Typen Liste und Char benutzen und importieren die entsprechenden Standardmodule:

Haskell: Rhyme

```
> import Data.Char
> import Data.List
```

Auch die Standardtypen `Maybe`, der sich dazu eignen mit Nullwerten zu arbeiten, wird benötigt.

Haskell: Rhyme

```
> import Data.Maybe
```

Aufgabe 1 In dieser Aufgabe geht es darum, Reimwörter zu identifizieren.

a) Die erste Funktion, die entwickelt werden soll, soll einen String in die im String enthaltenen Wörter splitten. Das Ergebnis soll eine Wortliste sein.

[1] Es sind erzeit allerdings nur für ein Teil der Aufgaben Tests formuliert.

Haskell: Rhyme

```
> woerter :: String -> [String]
> woerter =  []
```

Es gibt zwar die Standardfunktion `words`, aber diese hat mehrere Nachteile:

- Die Ergebnisliste hat doppelte Einträge. Für ein Wörterbuch ist wünschenswert, dass jedes Wort nur einmal auftaucht.
- Die Methode `words` splittet einen String an Weißzeichen. Wir wollen aber Wörter an allen Zeichen, die keine Buchstaben darstellen, splitten. Es gibt zum Glück die Funktion `isLetter`, die testet, ob ein Zeichen ein Buchstabe ist.
- Groß- und Kleinschreibung soll für unser Wörterbuch keine Relevanz haben. Wir betrachten im besten Fall nur alle Buchstaben in Kleinschreibung. Es gibt zum Glück die Methode `toLower`
- In der deutschen Sprache gibt es keine Wörter aus nur einem Buchstaben. Daher sollen alle Buchstaben mit nur einem Zeichen nicht in der Ergebnisliste stehen.

b) Schreiben Sie eine Funktion, die testet, ob ein Zeichen ein deutscher Vokal ist.

Haskell: Rhyme

```
> isVowel :: Char -> Bool
> isVowel = False
```

Gegeben Sei hierzu die Liste der Vokale.

Haskell: Rhyme

```
> vowels = "aeiouüöäAEIOUÜÖÄ"
```

c) Schreiben Sie jetzt eine Funktion, die eine Liste von Paaren erhält. Es soll für alle Paare in dieser Liste Ersetzungen im zweiten Argument vorgenommen werden.

Beispielaufruf:

Shell:

```
*Rhyme> replace [("ss","SZ"),("pp","PP")] "mississippi"
"miSZiSZiPPi"
```

Haskell: Rhyme

```
> replace :: [([a],[a])] -> [a] -> [a]
> replace s1s2s str = str
```

Jetzt können wir phonetisch gleich klingende Buchstabenkombinationen der deutschen Sprache durch eine einheitliche Schreibung ersetzen.

Haskell: Rhyme

```
> phonetics
>  = [("ai","ei"),( "äu","eu"),( "aa","ah")
>    ,( "oo","oh"),("ee","eh")]

> normalize = replace phonetics . map toLower
```

d) Schreiben Sie jetzt eine Funktion, die testet, ob sich zwei Wörter reimen. Hierzu sind die Wörter zunächst zu normalisieren. Da Wörter, die sich reimen gleich enden, kann man die Buchstabenreihenfolge der Wörter umdrehen. Jetzt müssen die Wörter einen gemeinsamen Präfix haben. Dieser Präfix muss einen Vokal enthalten. Der Präfix soll auch länger als ein Zeichen sein, sonst würden sich alle Wörter, die auf einem gemeinsamen Vokal enden bereits reimen. Schließlich seien Wörter, die sich nur reimen, weil sie auf ›er‹ oder ›en‹ enden, als Reimwörter ausgenommen.

Haskell: Rhyme

```
> doRhyme :: String -> String -> Bool
> doRhyme cs1 cs2 = False
```

Aufgabe 2 In dieser Aufgabe soll ein Reimwörterbuch erstellt werden.

a) Zunächst soll eine Liste von Wörtern so sortiert werden, dass sie in normalisierter Form nach ihrer Endung sortiert sind.

Tipp: Verwenden Sie die Funktion `sortOn`.

Haskell: Rhyme

```
> endsort :: [String] -> [String]
> endsort xs = xs
```

b) Die folgende Funktion soll aus einer Wortliste eine Liste von Listen erzeugen. In einer inneren Liste sollen sich reimende Wörter gebündelt sein. Eine innere Liste soll dabei immer mehr als ein Element enthalten.

Haskell: Rhyme

```
> rhymeGroups :: [String] -> [[String]]
> rhymeGroups _ = []
```

Wir stellen eine Funktion bereit, die für diese Reimgruppen ein Paar mit einem Repräsentaten für die reimenden Wörter bereit stellt.

Hier schon einmal ein kleiner Beispielaufruf für die Funktion:

Wir nehmen folgende Wortliste:

Haskell: Rhyme

```
> ws1 = ["zart","haus","aal","butter","art","vater","bär"
>       ,"maus","mär","wiese","klaus","zahl","mutter"
>       ,"klein","mein","riese"]
```

Sie führt zu 7 Reimgruppen:

Shell:

```
*Rhyme> rhymeGroups ws1
[["riese","wiese"],["aal","zahl"],["klein","mein"],["butter","mutter"]
,["bär","mär"],["haus","klaus","maus"],["art","zart"]]
*Rhyme>
```

c) Schreiben Sie eine Funktion

Haskell: Rhyme

```
> lookupBy              :: (a->c->Bool) -> a -> [(c,b)] -> Maybe b
> lookupBy _ _key []    =  Nothing
```

Mit dieser Funktion können wir bereits alle Reimwörter aus dem erzeugten Wörterbuch erfragen:

Haskell: Rhyme

```
> findRhymes :: String -> [(String,[String])] -> [String]
> findRhymes w =  (fromMaybe [])
>                .(lookupBy (\x y->x==y||doRhyme x y) w)
```

Aufgabe 3 Diese Aufgabe erstellt ein Lexikon, in dem Schüttelreimpaare vermerkt sind. Ein Schüttelreimpaar sind zwei Paare von Reimwörter, die mit denselben Konsonanten beginnen. Zum Beispiel für das erste Reimpaar ›Nacht‹ und ›lacht‹, passt für ein Schüttelreimpaar das zweite Paar von Reimwörtern ›nieder‹ und ›Lieder‹. Es sind Reimpaare

die mit ›n‹ und ›l‹ beginnen. Die Reimwörter überkreuz ergeben das Schüttelreimpaar ›Liedernacht‹ und ›nieder lacht‹, der vielleicht in einem Zweizeiler seine Verwendung findet:

Der ESC die Liedernacht,
in der man vieles nieder lacht.

a) In dieser Aufgabe geht es darum, für zwei Wörter, wenn sie sich reimen, die Konsonanten vor der Reimendung zu finden. Das Ergebnispaar hat zwei leere String, wenn die Wörter sich nicht reimen. Die beiden Konsonantenanfänge im Ergebnispaar sollen alphabetisch sotriert sein.

Haskell: Rhyme

```
> startConsonants :: String -> String -> (String,String)
> startConsonants ws1 ws2 = ("","")
```

Hier ein kleiner Beispielaufruf:

Shell:

```
*Rhyme> startConsonants "streifen" "begreifen"
("gr","str")
*Rhyme> startConsonants "Aal" "Zahl"
("","z")
*Rhyme>
```

b) Ähnlich wie das Reimlexikon aus der letzten Aufgabe, soll jetzt ein Schüttelreimlexikon erstellt werden.

Ein Schüttelreimlexikon ist eine Liste von Schlüssel-/Wertpaaren, deren Schlüssel die Konsonantenanfänge sind. Die Werte sind eine Liste der Reimwörter, die mit diesen KOnsonantenkombinationen beginnnen.

Haskell: Rhyme

```
> type SchuettelLexikon = [((String, String), [(String, String)])]
```

Das Argument der Funktion ist eine Liste, deren Elemente Listen von reimenden Wörtern ist, wie wir sie mit `rhymeGroups` erstellt haben.

Haskell: Rhyme

```
> schuettelGroups :: [[String]] -> SchuettelLexikon
> schuettelGroups xss = []
```

Ein Beispielaufruf hilft, die Erstellung eines Schüttelreimlexikons zu erstellen.

Shell:

```
*Rhyme> schuettelGroups$rhymeGroups ws1
[(("","z"),[("aal","zahl"),("art","zart")])
,(("b","m"),[("butter","mutter"),("bär","mär")])
,(("h","kl"),[("haus","klaus")])
,(("h","m"),[("haus","maus")])
,(("kl","m"),[("klein","mein"),("klaus","maus")])
,(("r","w"),[("riese","wiese")])]
*Rhyme>
```

Wie man sieht, sind einelementige Wertlisten dieses Mal erlaubt.

Wir können jetzt alles einmal zusammenfassen und eine Funktion schreiben, die eine Datei mit einer sehr großen Wortliste einliest und das Reimlexikon und das Schüttelreimlexikon wieder in Dateien schreibt.

Haskell: Rhyme

```
> generateRhymeLexicon inFile outReimFile outSchuettelFile = do
>   xs <- readFile inFile
>   let rg = rhymeGroups$woerter xs
>   let grs = taggedRhymeGroups rg
>   writeFile  outReimFile (show grs)
>   let schuettel = schuettelGroups rg
>   writeFile  outSchuettelFile (show schuettel)
```

Im Internet gibt es große Teytdateien mit allen deutschen Wörtern, die sich nun gut eignen, um die Reimlexika zu generieren:

Haskell: Rhyme

```
> moin = generateRhymeLexicon
>  "openthesaurus.txt" "reim.txt" "schuettel.txt"
```