Junioraufgabe 1: Reimerei

Team-ID: 00772

Team: Aino Spring

Bearbeiter/-innen dieser Aufgabe:
Aino Spring

12. November 2022

Inhaltsverzeichnis

Lösungsidee

Die 3 Regeln sind einfache Funktionen. Um alle möglichen Reimpaare zu finden, werden alle möglichen Kombinationen überprüft. Die Wörter werden als Wortliste importiert.

Umsetzung

Reimerkennung

Als erstes habe ich eine Funktion geschrieben, welche alle Vokalgruppen in einem Wort findet (GetVowelGroups(word string)). Diese gibt als Rückgabe eine „VowelGroup“. Dies ist ein

struct, welches die Vokalgruppe und den Index, an dem die Vokalgruppe im Wort erscheint, angibt.

Die nächste Funktion findet aus den Vokalgruppen die maßgebliche Vokalgruppe (GetEssentialVowelGroup(word string)). Dafür wird überprüft, wie viele Vokalgruppen es gibt. Gibt es eine, wird diese zurückgegeben. Gibt es mehrere, wird die Vorletzte zurückgegeben. Wenn es keine gibt, wird eine leere „VowelGroup“ zurückgegeben.

Die Funktion „CheckIfEndSame(word1 string, word2 string)“ prüft, ob die maßgebliche Vokalgruppe und alle ihr folgenden Buchstaben der beiden Wörter gleich sind. Dafür wird der Index in der „VowelGroup“ genutzt, um in den in ein character array (in diesem Fall ein string slice) umgewandelten Wort-String alle Buchstaben der maßgeblichen Vokalgruppe und danach mit dem anderen Wort zu überprüfen.

Die Funktion „CheckIfEssentialVowelGroupValid(word string)“ überprüft die in der Aufgabenstellung erwähnte Regel 2 und ob die „VowelGroup“ leer ist. Dazu wird erst überprüft, ob der Wert der maßgeblichen Vokalgruppe gleich null ist, da die Funktion „GetEssentialVowelGroup“, welche im 2. Punkt genannt wurde, eine leere Vokalgruppe zurückgibt, wenn keine im Wort vorhanden ist. Als nächstes wird überprüft, ob die Länge der maßgeblichen Vokalgruppe und allen ihr folgenden Buchstaben länger oder gleich ist, wie die Hälfte der Länge des Wortes. Dafür werden Fließkommazahlen verwendet, da sonst in manchen Fällen Fehler auftreten können.

Die Funktion „CheckIfWordsIncludeThem(word1 string, word2 string)“ überprüft die in der Aufgabenstellung erwähnte Regel 3. Dafür wird überprüft, ob das erste Wort mit dem zweiten Wort endet oder ob das zweite Wort mit dem Ersten endet.

Die Funktion „CheckIfRhyme(word1 string, word2 string)“ fügt alle Regeln zusammen und entfernt ein mögliches Leerzeichen am Ende der Wörter. Wichtig wäre noch hinzuzufügen, dass die Funktion „CheckIfEndSame“ nur ausgeführt wird, wenn die Funktion „CheckIfEssentialVowelGroupValid“ bei beiden Wörtern „true“ zurückgegeben hat. Wenn dies nicht der Fall wäre, könnte es zu Fehlern kommen, da der Index der „VowelGroup“ abgefragt wird und dieser nicht vorhanden ist, wenn es keine Vokalgruppe/keinen Vokal in dem Wort gibt. Dies wurde im Punkt zur Funktion „CheckIfEssentialVowelGroupValid“ ausführlicher verdeutlicht.

Die Funktion „CalculateRhymes(words []string)” berechnet alle Reimpaare einer Liste aus Wörtern (strings). Dazu wird über jene Liste iteriert und jedes Mal wird noch einmal über alle Wörter nach dem aktuellen Wort iteriert und mit der Funktion „CheckIfRhyme“ überprüft, ob sich beide aktuellen Wörter reimen. Die sich reimenden Wörter werden als Liste aus Wörtern zurückgegeben.

Wortimportierung

Die Funktion „importFiles()“ importiert die Wortlisten aus den in der config-Datei angegebenen Dateien. Die Dateien sind in der config-Datei als relative Pfade angegeben, jedoch können es auch absolute Pfade sein. Diese Dateien werden eingelesen und jede Zeile des Inhalts ist ein Wort. Die Wortlisten werden als 2-dimensionales array aus strings (in diesem Fall wieder ein 2-dimensionaler slice aus strings) zurückgegeben.

Alles zusammenfügen

In der main-Funktion wird alles zusammengefügt. Die Wortlisten werden mithilfe der im obigen Punkt genannten Funktion „importFiles“ importiert und für jede Wortliste werden alle Reimpaare mit der „CalculateRhymes“ Funktion berechnet und diese werden mit dem Index der Wortliste über die Standartausgabe (stdout) ausgegeben.

Beispiele

reimerei0.txt

"bemühen", "glühen"

"Biene", "Hygiene" Dieses Beispiel ist interessant, da es sich nicht reimt, wenn man es spricht, jedoch wenn man „Hygiene“ so ausspricht, wie man es schreibt und das „ie“ wie ein langes „i“ spricht, reimt es sich. Es wird jedoch nicht so ausgesprochen, da es aus dem griechischen „Hygieia“ kommt. Dies ist die griechische Göttin der Gesundheit.

"Biene", "Schiene"

"Hygiene", "Schiene" Hier das gleiche wie bei „Biene“ und „Hygiene“.

"Knecht", "Recht"

reimerei1.txt

"Bildnis", "Wildnis"

"Brote", "Note"

reimerei2.txt

"Epsilon", "Ypsilon"

reimerei3.txt

"Absender", "Kalender"

"Ansage", "Frage"

"Ansage", "Garage" Das „g“ wird unterschiedlich ausgesprochen. Bei „Garage“ ist es ein [ʒ] (stimmhafter postalveolarer Frikativ), bei „ansage“ ist es in normales [g].

"Bahn", "Zahn"

"Bank", "Dank"

"Baum", "Raum"

"Bein", "Wein"

"Bier", "Tier"

"Bild", "Schild"

"Bitte", "Mitte"

"Butter", "Großmutter"

"Butter", "Mutter"

"Dame", "Name"

"Dezember", "November"

"Dezember", "September"

"Drucker", "Zucker"

"Durst", "Wurst"

"Ermäßigung", "Kündigung"

"Ermäßigung", "Reinigung"

"Fest", "Test"

"Feuer", "Steuer"

"Fisch", "Tisch"

"Flasche", "Tasche"

"Frage", "Garage" Das gleiche wie bei „Garage“ und „ansage“

"Fuß", "Gruß"

"Gas", "Glas"

"Glück", "Stück"

"Gleis", "Kreis"

"Gleis", "Preis"

"Gleis", "Reis"

"Großmutter", "Mutter"

"Gruppe", "Suppe"

"Hand", "Land"

"Hand", "Strand"

"Hose", "Rose"

"Hund", "Mund"

"Kündigung", "Reinigung"

"Kanne", "Panne"

"Kasse", "Klasse"

"Kasse", "Tasse"

"Kassette", "Kette"

"Kassette", "Toilette"

"Keller", "Teller"

"Kette", "Toilette"

"Kind", "Rind"

"Kind", "Wind"

"Klasse", "Tasse"

"Kopf", "Topf"

"Kreis", "Preis"

"Kunde", "Stunde"

"Land", "Strand"

"Lohn", "Sohn"

"Magen", "Wagen"

"Nachmittag", "Vormittag"

"November", "September"

"Platz", "Satz"

"Rind", "Wind"

"Rock", "Stock"

"S-Bahn", "Zahn"

"Sache", "Sprache" Das „a“ wird bei „Sache“ kurz gesprochen und bei „Sprache“ lang.

"See", "Tee"

"Sekunde", "Stunde"

Eigene Wortliste

"Löwe", "Möwe" Hier habe ich getestet, ob Umlaute auch als Vokale erkannt werden.

"Tür", "Für " Hier habe ich getestet, ob Leerzeichen am Ende eines Wortes entfernt werden.

In dieser Wortliste sind auch andere Wörter enthalten, welche bestimmte Regeln verletzen:

- Wörter mit maßgeblicher Vokalgruppe nach der Mitte Bsp.: Überaschungsei
- Wörter mit keinem Vokal Bsp.: Fynn (Name), Nyx (griech. Göttin der Nacht)
- Wörter, welche mit anderen Wörtern enden Bsp.: laufen und verlaufen

Quellcode

Reimerkennung

Alle Vokale:

```
var (
   VOWELS = []string{"a", "e", "i", "o", "u", "ä", "ö", "ü"}
)
```

Das VowelGroup struct:

```
type VowelGroup struct {
   value string
   idx   uint
}
```

Das Rhyme struct:

```
type Rhyme struct {
   word1 string
   word2 string
}
```

Die Funktion, um die Vokalgruppen eines Wortes erkennen:

```
func GetVowelGroups(word string) (vowelGroups []VowelGroup) {
   word += " "
   vowelGroups = make([]VowelGroup, 0)
   var currentVowelGroup VowelGroup = VowelGroup{value: ""}
   for idx, character := range strings.Split(word, "") {
      if slices.Contains(VOWELS, strings.ToLower(character)) {
         if "" == currentVowelGroup.value {
            currentVowelGroup = VowelGroup{
               value: character,
               idx:   uint(idx),
            }
         } else {
            currentVowelGroup.value += character
         }
      } else {
         if "" == currentVowelGroup.value {
            continue
         }
         vowelGroups = append(vowelGroups, currentVowelGroup)
         currentVowelGroup = VowelGroup{value: ""}
      }
   }
   return
}
```

Die Funktion, um maßgebliche Vokalgruppe zu erkennen:

```
func GetEssentialVowelGroup(word string) VowelGroup {
   var vowelGroups = GetVowelGroups(word)
   if len(vowelGroups) > 1 {
      return vowelGroups[len(vowelGroups)-2]
   }
```

```
    if len(vowelGroups) < 1 {
        return VowelGroup{value: ""}
    }
    return vowelGroups[0]
}
```

Die Funktion, um zu prüfen, ob die maßgebliche Vokalgruppe und alle folgenden Buchstaben gleich sind:

```
func CheckIfEndSame(word1 string, word2 string) bool {
    var word1EssentialVowelGroup = GetEssentialVowelGroup(word1)
    var word2EssentialVowelGroup = GetEssentialVowelGroup(word2)
    return strings.Join(strings.Split(word1, "")[word1EssentialVow-
elGroup.idx:], "") == strings.Join(strings.Split(word2, "")[word2Essential-
VowelGroup.idx:], "")
}
```

Die Funktion, um zu prüfen, ob die maßgebliche Vokalgruppe vorhanden ist und die maßgebliche Vokalgruppe und alle folgenden Buchstaben mindestens die Mitte des Wortes sind:

```
func CheckIfEssentialVowelGroupValid(word string) bool {
    var essentialVowelGroup = GetEssentialVowelGroup(word)
    if "" == essentialVowelGroup.value {
        return false
    }
    return float32(len(strings.Split(word, "")[essentialVowelGroup.idx:]))
>= ((float32(len(strings.Split(word, "")))) / 2)
}
```

Die Funktion, um zu prüfen, ob ein Wort mit dem anderen endet:

```
func CheckIfWordsIncludeThem(word1 string, word2 string) bool {
    word1 = strings.ToLower(word1)
    word2 = strings.ToLower(word2)
    var word1Length = len(strings.Split(word1, ""))
    var word2Length = len(strings.Split(word2, ""))
    if word1Length >= word2Length {
        return word1[word1Length-word2Length:] == word2
    } else if word1Length < word2Length {
        return word1 == word2[word2Length-word1Length:]
    } else {
        return false
    }
}
```

Die Funktion, die alle anderen Funktionen zusammenfügt und damit prüft, ob sich zwei Wörter reimen:

```
func CheckIfRhyme(word1 string, word2 string) (bool, []uint8) {
    word1 = strings.TrimSuffix(word1, " ")
    word2 = strings.TrimSuffix(word2, " ")
    var rhyme = true
    var rules = make([]uint8, 0)
    for _, word := range []string{word1, word2} {
        if !CheckIfEssentialVowelGroupValid(word) {
            rhyme = false
```

```
            rules = append(rules, 2)
        }
    }
    if rhyme {
        if !CheckIfEndSame(word1, word2) {
            rhyme = false
            rules = append(rules, 1)
        }
    }
    if CheckIfWordsIncludeThem(word1, word2) {
        rhyme = false
        rules = append(rules, 3)
    }
    return rhyme, rules
}
```

Die Funktion, welche alle Reimpaare einer Wortliste berechnet:

```
func CalculateRhymes(words []string) []Rhyme {
    var rhymes = make([]Rhyme, 0)
    for idx, word := range words {
        for i := idx + 1; i < len(words); i++ {
            if ok, _ := CheckIfRhyme(word, words[i]); ok {
                rhymes = append(rhymes, Rhyme{word1: word, word2: words[i]})
            }
        }
    }
    return rhymes
}
```

Wortimportierung

Der Pfad zur config-Datei:

```
const (
    CONFIG = "config"
)
```

Die Funktion, welche die Wortlisten, welche in der config-Datei angegeben sind, importiert:

```
func importFiles() [][]string {
    var wordLists = make([][]string, 0)
    data, err := os.ReadFile(CONFIG)
    if err != nil {
        log.Panic(err)
    }
    for _, line := range strings.Split(string(data), "\n") {
        line = strings.TrimSuffix(line, "\r")
        wordListData, wordListErr := os.ReadFile(line)
        if wordListErr != nil {
            log.Panicf("%#v: %v", line, wordListErr)
        }
        wordLists = append(wordLists, strings.Split(strings.Re-
place(string(wordListData), "\r", "", -1), "\n"))
```

```
    }
    return wordLists
}
```

Alles zusammenfügen

Die main-Funktion, welche alle Reimpaare aus den importierten Wortlisten berechnet:

```
func main() {
    var wordLists = importFiles()
    for idx, wordList := range wordLists {
        var rhymes = CalculateRhymes(wordList)
        fmt.Printf("Rhymes from wordlist %v: \n", idx)
        for _, rhyme := range rhymes {
            fmt.Printf("%#v, %#v\n", rhyme.word1, rhyme.word2)
        }
        fmt.Print("\n")
    }
}
```