37. Bundeswettbewerb Informatik

1. Runde

01.09.2018 - 26.11.2018

Aufgabe 2 - Twist

Nico Jeske (48312)

Team: ITler (00140)

Ich versichere hiermit, die vorliegende Arbeit selbstständig und unter den Wettbewerbsregeln des Bundeswettbewerbs Informatik, ohne fremde Hilfe und ohne die Verwendung anderer, als der in den Quellen angegebenen Hilfsmitteln, angefertigt zu haben.

Nico Jeske, den 12.10.2018

1 Lösungsidee

Die Aufgabe lässt sich in zwei Teilaufgaben unterteilen: Das twisten von Texten und das enttwisten von Texten. In den folgenden Abschnitten gehe ich nun genauer auf diese Teilaufgaben ein.

1.1 Das twisten von Texten

Das twisten von Texten basiert auf einem sehr simplen Prinzip, welches sich wie folgt kurz beschreiben lässt:

1. Das Filtern von Worten aus einem Text
2. Das Ersetzen dieser Wörter durch eine getwistete Variante dieser.

1.1.1 Finden von Wörtern

Es gibt viele Möglichkeiten, Wörter aus einem Text herauszufiltern. Man könnte den Text zum Beispiel an den Leerzeichen Splitten und auf diese Weise einzelne Wörter herausfiltern.

Diese Methode ist allerdings nicht praktisch anwendbar. So stellen besonders Satzzeichen, oder andere Semantische Elemente, wie Apostrophe ein Problem dar, da die Syntaktische Korrektheit dieser bewahrt werden soll.
Stattdessen habe ich mich für einen Ansatz unter Verwendung des folgenden simplen RegEx-Strings entschieden:

$$([A - z\ddot{o}\ddot{a}\ddot{u}\ddot{O}\ddot{A}\ddot{U}])+ \tag{1}$$

Dieser RegEx-String arbeitet auf einer sehr simplen Basis. So werden mit diesem RegEx-String an einander gekettete Buchstaben-folgen gematch. Dies hat den Vorteil, dass somit Semantische Elemente wie Kommata oder ähnliches nicht in dem Match enthalten sind und außerdem bei Wörtern wie "Rock'n'Roll" die Apostrophe ihre Position behalten und somit Syntakisch richtig positioniert sind. Bei dem Beispiel von "Rock'n'Roll" würde die Teilwörter "Rock", "n", und "Roll" von dem RegEx-String gematcht und schließlich im finalen Schritt getwistet werden, auf den ich nun im folgenden Abschnitt genauer eingehe.

1.1.2 Das Twisten von Wörtern

Nachdem nun also die einzelnen Wörter aus dem Text heraus gefiltert wurden, werden diese nun Schritt für Schritt durch ihre getwisteten Varianten ersetzt.

Sollte das Wort allerdings höchstens drei Buchstaben lang sein, muss es, beziehungsweise kann es nicht getwistet werden und das Originale Wort bleibt somit erhalten.

Der eigentliche Prozess des Twistens ist nun leicht zu erklären. Von dem gegebenen Wort werden der erste und der letzte Buchstabe abgespeichert. Die restlichen Buchstaben werden nun einfach Zufällig neu sortiert und die einzelnen Buchstaben wieder zusammengesetzt. Bei dieser Methode ist es Statistisch natürlich möglich, dass am Ende aufgrund des Zufalls wieder das Originale Wort heraus kommt. Man könnte dies umgehen, in dem Mann kontrolliert, ob das getwistete Wort das gleiche ist, wie das Original und in diesem Fall das Wort erneut twisten. Sollten Allerdings nun Wörter wie zum Beispiel "Anna" vorkommen, so ist die getwistete Variante **immer** das Original und die Kontrolle würde somit in eine Endlosschleife führen. Dies kann man wiederum lösen, in dem kontrolliert wird, ob sich das gegebene Wort überhaupt twisten lässt, also in anderen Worten muss einfach nur kontrolliert werden, dass die Buchstaben die getwistet werden nicht alle gleich sind.

Da die Wahrscheinlichkeit, dass dieser Fall bei einem Wort mit n (unterschiedlichen) Buchstaben das Originale Wort beim Twisten raus liegt bei $\frac{1}{(n-2)!}$ und wäre somit nicht sehr unwahrscheinlich, weswegen ich die eben genannten Methoden implementiert habe.

1.2 Das Enttwisten von Wörtern

Während das Twisten von Wörtern der einfache Prozess ist, liegt die Herausforderung der Aufgabe darin, zu versuchen getwistete Wörter zu erkennen und den Original Text bestmöglich wieder herzustellen. Zum enttwisten ist natürlich eine Art Wörterbuch vorhanden, in dem man versucht ein Wort zu finden, was zu dem getwisteten Wort passt.

1.2.1 Repräsentation von Wörtern

Leider ist es natürlich nicht möglich einfach das getwistete Wort im Wörterbuch nachzuschlagen. So findet man zu ”Hrez” im Wörterbuch auch nicht sofort Herz. Es wird also eine Methode benötigt, um die Wörter aus dem Wörterbuch herauszufiltern, die zu dem getwisteten Wort passen können. So kam ich schließlich zu der Idee, die Wörter – ob getwistet oder nicht – zu generalisieren.

Die Methode für die ich mich schließlich entschieden hab, ist es das Wort zu sortieren. Also Die einzelnen Buchstaben des Wortes zu nehmen und in alphabetischer Reihenfolge neu anzuordnen. Wenn wir also zu meinem Beispiel mit ”Herz” zurückkommen, so wird sowohl aus ”Herz” als auch aus ”Hrez” das Wort ”ehrz”. Diese Form des sortierten Wortes wird dann schließlich so abgespeichert, dass man in einer Art neuem Wörterbuch unter dem Wort ”ehrz” das Wort ”Herz” findet. Sollten mehrere Wörter passend sein, wie zum Beispiel bei ”eeklmnntu” (”umlenkten” und ”munkelten”), werden beide Wörter gespeichert.

Figure 1: Repräsentation von ”Herz”

1.2.2 Enttwisten

Mit dieser Repräsentation der Wörter ist es nun also Problemlos möglich einen getwisteten Text zu enttwisten, indem nach und nach jedes getwistete Wort im oben beschriebenen neuen Wörterbuch nachgeschlagen wird. Dabei gibt es nun folgende Möglichkeiten.

1. Es wird kein passendes Wort im Wörterbuch gefunden. Das getwistete Wort ist also mit dem gegebenen Wörterbuch nicht übersetzbar und stattdessen wird das getwistete Wort – umschlossen mit Fragezeichen – zurückgegeben

2. Es wurde genau ein passendes Wort im Wörterbuch gefunden. Das getwistete Wort kann also durch das gefundene Wort ersetzt werden.

3. Es wurden mehrere passende Wörter im Wörterbuch gefunden. Es werden alle Möglichkeiten betrachtet. Es wird kontrolliert, ob der Anfangsbuchstabe und der Endbuchstabe der Möglichkeit mit dem des getwisteten Wort übereinstimmen. Alle Möglichkeiten die diese Bedingung erfüllen sind akzeptierte Übersetzungen des getwisteten Wortes.

2 Implementation

Die Aufgabe ist in Java implementiert. Das Twisten und Enttwisten wird hierbei von der Klasse Twister geregelt. Es kann vom Benutzer ein Text eingegeben, oder von einer Datei eingelesen werden. Anschließend kann der User entscheiden, ob er den Text twisten oder enttwisten will.

2.1 Twisten

Das Twisten eines Textes wird hierbei von der Funktion *twist(String text)* übernommen. In dieser Funktion wird jedes einzelne Wort, welches durch den in der Lösungsidee genannten RegEx-Strings gematcht wird mithilfe der Funktion *twistWord(String word)* getwistet. Das Twisten an sich ist nun wie folgt implementiert. Die Buchstaben zwischen dem Anfangs und dem Endbuchstaben werden in einer Liste gespeichert. Diese wird dann, mithilfe von *Collections.shuffle()* gemischt. Anschließend wird das getwistete Wort mithilfe aus dem Anfangs- und Endbuchstaben, sowie dem gemischten Mittelteil mit hilfe eines Stringbuilders wieder zusammengesetzt.

2.2 Enttwisten

Um Wörter enttwisten zu können, muss erstmal das in der Lösungsidee angesprochene Wörterbuch erstellt werden. Dies geschieht in der Funktion *initializeDirectory*. Das Ergebnis ist dann eine HashMap¡String, List¡String¿¿ in der das sortierte Buchstabenreihenfolge auf eine Liste möglicher Wörter zeigt.

Mit diesem Wörterbuch ist es der Funktion *untwist* nun möglich, jedes einzelne Wort des getwisteten Textes nach der in der Lösungsidee genannten Methode zu entschlüsseln.

3 Beispiele

3.1 Twisten

Datei	Ergebnis
twist1.txt	Der Tsiwt (Eigsnlch twsit = Dneurhg, Vnerhrudeg) war ein Metnadoz im 4/4-Tkat, der in den feürhn 1960er Jeahrn ploäpur wudre und zu Rock'n'Rlol, Rhythm and Buels oedr szelipeelr Tswit-Musik gznatet wird.
twist2.txt	Hat der atle Hxmientseeer scih doch eainml weebegegbn! Und nun sleoln sinee Geister acuh ncah meienm Wlieln lbeen. Siene Wort und Wreke mkert ich und den Baurch, und mit Gikstetseräse tu ich Wenudr auch.
twist3.txt	Ein Raetrasunt, wleechs a la ctare aretiebt, betiet sien Abgonet ohne enie veorhr fgtltegeese Mhflereienoügne an. Ddcruah haebn die Gtsäe zwar mher Slrapiuem bei der Whal irher Siepsen, für das Rtasuranet eshetentn jcdeoh zhceätlusizr Awunfad, da wigeenr Pilrhaieegscnsnuht vaerhdonn ist.
twist4.txt	Agtuusa Ada Byorn Knig, Cunstoes of Lcvolaee, war eine btichirse Ailegde und Mrikehitaetman, die als die ertse Pemmrirrroagein üurbhpaet glit. Beeitrs 100 Jhrae vor dem Amoeukmfn der eretsn Porepracrsagmeihrmn ernasn sie enie Rcehen-Mhnaicek, der eniige Kezpnote merdneor Prcrimgesaaehrpmorn venrhgowam.
twist5.txt	siehe getwistet5.txt

3.2 Enttwisten

Anzumerken ist, dass in der gegebenen Wörterliste einige Wörter fehlen. Diese können logischerweise nicht übersetzt werden. Im Ergbnis sieht dies wie folgt aus: [?Originalwort?]. Dabei ist auch die Groß- und Kleinschreibung in der Wörterliste von Relevanz.

Datei	Ergebnis
enttwist.txt	Der Twist ([?Eigsnclh?] twist = Drehung, Verdrehung) war ein Modetanz im 4/4-Takt, der in den [frühen, führen] 1960er Jahren populär wurde und zu Rock'n'Roll, Rhythm and [?Bleus?] oder spezieller Twist-Musik getanzt wird.

Für die anderen Beispieltexte wird jeweils das Ergebnis des Twistens aus der obigen Tabelle als Grundlage zum enttwisten genommen.

Datei	Ergebnis
getwistet2.txt	Hat der alte Hexenmeister sich doch einmal wegbegeben! Und nun sollen seine [?Geister?] auch nach meinem Willen leben. [?Siene?] Wort und Werke merkt ich und den Brauch, und mit Geistesstärke tu ich [?Wenudr?] auch.
getwistet3.txt	Ein Restaurant, welches a la [?ctare?] [abrietet, arbeitet], bietet sein Angebot ohne eine vorher festgelegte Menüreihenfolge an. Dadurch haben die [?Gtsäe?] zwar mehr Spielraum bei der Wahl ihrer [?Siepsen?], für das Restaurant entstehen jedoch zusätzlicher Aufwand, da weniger Planungssicherheit vorhanden ist.
getwistet4.txt	[?Agtuusa?] Ada [?Byorn?] [?Knig?], [?Cunstoes?] of [?Lcvolaee?], war eine britische [?Ailegde?] und Mathematikerin, die als die erste Programmiererin überhaupt gilt. [?Beeitrs?] 100 Jahre vor dem Aufkommen der ersten Programmiersprachen ersann sie eine [?Rcehen?]-Mechanik, der einige Konzepte moderner Programmiersprachen vorwegnahm.
getwistet5.txt	siehe enttwistet5.txt

4 Quellcode

Listing 1: Die Klasse Twister

```
1  package jeske;
2
3  import com.esotericsoftware.minlog.Log;
4
5  import java.io.BufferedReader;
6  import java.io.IOException;
7  import java.io.InputStreamReader;
8  import java.util.*;
9  import java.util.regex.Pattern;
10
11 class Twister {
12
13   /**
14    * Dictionary.
15    * Works according to the following system:
16    * The keys are the letters of a word in ascending order. (e.g. for Auto -> acr)
17    * The value then represents all the words in a list that can be formed with these letters.
18    */
19   private static Map<String, List<String>> dict = new HashMap<>();
20
21   Twister() throws IOException {
22     Log.set(Log.LEVEL_INFO);
23
24     BufferedReader directoryReader = new BufferedReader(
25             new InputStreamReader(
26                     this.getClass().getResourceAsStream("/beispieldaten/woerterliste.txt")
27             )
28     );
29
30     initializeDictionary(directoryReader);
31   }
32
33   /**
34    * Twists a String
35    * @param string String to twist
36    * @return twisted string
37    */
38   String twist(String string) {
39     //Takes each word of the string and twists it with the function twistWord
40     return Util.replace(string, Pattern.compile("([A-zöäüÖÄÜ])+"),
41     match -> twistWord(match.group()));
42   }
43
44   /**
45    * Initializes the directory
46    * @param directoryReader BufferedReader from the worterliste.txt
47    * @throws IOException Error while reading the file
48    */
49   private static void initializeDictionary(BufferedReader directoryReader) throws IOException {
50     String currLine;
51     //For every word in the dictionary
52     while((currLine = directoryReader.readLine()) != null){
53       //Remove leading and trailing whitespaces and convert string to lowercase letters
54       String charsString = convertWordToDictionaryForm(currLine);
55       if(dict.containsKey(charsString)) {
```

```
56          dict.get(charsString).add(currLine);
57        } else {
58          dict.put(charsString, new ArrayList<>());
59          dict.get(charsString).add(currLine);
60        }
61      }
62      System.out.println();
63    }
64
65    /**
66     * Try's to untwist a String.
67     * @param encoded string
68     * @return The best possible decoded string
69     */
70    String untwist(String encoded) {
71      //Takes each word of the string and try's to untwist it with the function untwistWord
72      return Util.replace(encoded, Pattern.compile("([A-zöäüÖÄÜ])+"),
73      match -> untwistWord(match.group()));
74    }
75
76    /**
77     * Try's to untwist a given word
78     * @param word word
79     * @return With one possibility: The possibility in plain text.
80     *         For several possibilities: Option 1 | Option 2 | ...]
81     *         If no solution has been found: [?Original Word?]
82     */
83    private static String untwistWord(String word) {
84
85      //If the word has between 1 and 3 letters it can't be twisted.
86      if(word.length()<=3)
87        return word;
88
89      //Saves the first letter to keep upper and lower case later on
90      char firstChar = word.charAt(0);
91      //Convert word in dictionary form
92      String charsString = convertWordToDictionaryForm(word);
93      Log.debug("Finding Dict entry for -> " + word);
94      //If the word is in the dictionary, find fitting words.
95      if(dict.containsKey(charsString)){
96        Log.debug("Possibilities -> " + dict.get(charsString));
97        String ret;
98        //Try's to get a fitting Word from the possible words.
99        if(dict.get(charsString).size() > 1) {
100         ret = getFittingWord(dict.get(charsString), word.toCharArray());
101         if(ret.charAt(0) != '[') {
102           ret = firstChar + ret.substring(1);
103         }
104       } else {
105         ret = dict.get(charsString).get(0);
106         ret = firstChar + ret.substring(1);
107       }
108
109       Log.debug("Recognized " + word + " -> " + ret);
110       return ret;
111
112     //If the word is not in the dictionary
113     } else {
114       Log.info(word + " not found...");
115       return "[?" + firstChar + word.substring(1) + "?]";
116     }
117   }
118
119   /**
120    * Converts a word into the dictionary format: lowercase, sorted in ascending order.
121    * @param word word
122    * @return dictionary format
123    */
124   private static String convertWordToDictionaryForm(String word) {
125     word = word.trim();
126     word = word.toLowerCase();
127     char[] chars = word.toCharArray();
128     Arrays.sort(chars);
129     return new String(chars);
130   }
131
132   /**
133    * Gets the possible words from the dictionary and returns them if possible.
134    * Given is only that the word from the dictionary contains all the same chars.
135    * Now must be checked, if the start and the end char are the same.
136    * @param words Word list from the dictionary fitting to the word you are trying to decode
137    * @param chars char[] from the word you are trying to decode
138    * @return With one possibility: The possibility in plain text.
139    *         For several possibilities: [Option 1 | Option 2 | ...]
140    *         If no solution has been found: [?Original Word?]
141    */
142   private static String getFittingWord(List<String> words, char[] chars) {
```

```
143     char firstLetter = chars[0];
144     char lastLetter = chars[chars.length-1];
145     ArrayList<String> ret = new ArrayList<>();
146     for(String word : words){
147       //Beginning and ending letter match -> the word is a solution.
148       if(word.charAt(0) == firstLetter && word.charAt(word.length()-1) == lastLetter) {
149         ret.add(word);
150       }
151     }
152
153     //Exactly one solution -> return solution
154     if(ret.size() == 1) {
155       return ret.get(0);
156     //More then one solution -> return List representation from the solutions.
157     } else if(ret.size() > 1) {
158       return ret.toString();
159     //No solution -> [?word?]
160     } else {
161       Log.warn("No fitting word found for " + new String(chars));
162       return  "[?"+ new String(chars)+"?]";
163     }
164   }
165
166   /**
167    * Twists a given word
168    * @param word word
169    * @return twisted word
170    */
171   private static String twistWord(String word) {
172     //If word is between 1 and 3 chars long, it can't be twisted.
173     if(word.length() <= 3)
174       return word;
175
176     //Twisting the word.
177
178     List<Character> charsToTwist = new ArrayList<>();
179     for (char c : word.substring(1, word.length() - 1).toCharArray()) {
180       charsToTwist.add(c);
181     }
182
183     Collections.shuffle(charsToTwist);
184
185     StringBuilder twistedPartBuilder = new StringBuilder();
186     charsToTwist.forEach(twistedPartBuilder::append);
187
188
189     //If the twisted Word is the same as the original, twist it again.
190     if (twistedPartBuilder.toString().equals(word.substring(1, word.length() - 1))) {
191       boolean possible = false;
192       char firstChar = charsToTwist.get(0);
193       for (Character character : charsToTwist) {
194         if (character != firstChar) {
195           possible = true;
196           break;
197         }
198       }
199
200       if (possible) {
201         twistWord(word);
202       }
203     }
204
205
206     return ""+word.charAt(0) + twistedPartBuilder.toString() + word.charAt(word.length()-1);
207   }
208 }
```

Listing 2: Util.replace

```
1  /**
2     * Given a string and a RegEx pattern, apply a given method to each hit and replace
3     * it with the result of the method.
4     * @param input input string
5     * @param regex regex pattern
6     * @param callback replace method
7     * @return replaced string
8     */
9    static String replace(String input, Pattern regex, StringReplacerCallback callback) {
10     StringBuffer resultString = new StringBuffer();
11     Matcher regexMatcher = regex.matcher(input);
12     while (regexMatcher.find()) {
13       regexMatcher.appendReplacement(resultString, callback.replace(regexMatcher));
14     }
15     regexMatcher.appendTail(resultString);
16
17     return resultString.toString();
```

```
18   }
```

Listing 3: StringReplacerCallback

```
1  package jeske;
2
3  import java.util.regex.Matcher;
4
5  /**
6   * Interface for a method that replaces a RegEx match.
7   */
8  public interface StringReplacerCallback {
9    String replace(Matcher match);
10 }
```

5 Quellen

- minlog: https://github.com/EsotericSoftware/minlog