Aufgabe 2: Twist

Team-ID: 00381

Team-Name: EinSteine

Bearbeiter/-innen dieser Aufgabe:
Oliver Schirmer, Konrad Walter, Paskal Paesler, Corey Attireh

20. November 2018

Inhaltsverzeichnis

Lösungsidee - Twist

Beim Twisten eines Wortes sind zwei Dinge zu beachten:

1. Der erste und letzte Buchstabe bleibt unverändert.
2. Die Buchstaben zwischen dem Anfangs- & Endbuchstaben müssen nach dem Twisten in gleicher Menge wie vorher vorhanden sein, jedoch in einer anderen Reihenfolge, sofern dies möglich ist.

Zusätzlich muss auf Interpunktion geachtet werden, da diese möglichst nicht in den „Mischprozess“ integriert werden sollten.

Umsetzung - Twist

Aus den beiden vorherigen Bedingungen ergibt sich die Lösung. Zuerst werden alle Zeichen die nicht zum eigentlichen Wort gehören gefiltert und mit einem Index gespeichert. Dabei wird eine TreeMap verwendet, welche dafür sorgt, dass die Zeichen in der richtigen Reihenfolge gespeichert werden und somit nach dem Twisten wieder an der originalen Stelle eingefügt werden können.

Der eigentliche Shuffle-Prozess wird mithilfe eines Regex‘s umgesetzt. Dieser ist grundlegend in 3 Gruppen aufgeteilt. Die erste und letzte Gruppe, matcht Klein- & Großbuchstaben, sowie Umlaute, wie ä, ö, ü, ß. Die mittlere Gruppe matcht die gleichen Zeichen, jedoch können diese beliebig oft vorhanden sein.
Beim Erstellen des neuen Wortes lassen sich nun der Anfangs- & Endbuchstabe an Anfang / Ende setzen und die Buchstaben dazwischen durchlaufen einen Mischalgorithmus. Dabei wird eine Liste der Buchstaben erstellt und diese mithilfe der shuffle-Funktion der Collections-Klasse gemischt. Die nun gemischte Liste wird wieder zu einem Wort zusammengesetzt und zwischen den Anfangs- & Endbuchstaben eingefügt.

Lösungsidee - Untwist

Für die Lösung dieses Problems lassen sich auch hier wieder einige Voraussetzungen erschließen:

1. Der erste und letzte Buchstabe des gesuchten Wortes entsprechen dem getwisteten Wort.
2. Das gesuchte Wort hat die selbe Länge, wie das getwistete Wort.
3. Das gesuchte Wort muss exakt die selbe Anzahl an Buchstaben, wie das getwistete Wort besitzen.

Zusätzlich muss auch hier versucht werden die Interpunktion zu exludieren.

Umsetzung – Untwist

Zu Beginn wird über jede Zeile des vorgegebenen Textes iteriert und in dieser dann über jedes einzelne Wort (Leerzeichen gelten als Trenner für Worte).

Bei diesem Wort wird die Interpunktion nach dem selben Prinzip, wie beim Twisten, zu Beginn gefiltert und mit einem Index in einer TreeMap gespeichert. Daraufhin wird über jedes Wort der vorgegebenen Wortliste iteriert und auf die Merkmale geprüft. D.h. es werden erstmals alle Möglichkeiten gespeichert, die den Bedingungen 1 & 2 entsprechen. Daraufhin wird überprüft, ob eine der Möglichkeiten die exakt gleiche Anzahl der Buchstaben besitzt. Dies wird umgesetzt, indem die Möglichkeit und das getwistete Wort alphabetisch sortiert werden und dann ohne Beachtung von case-sensitivity auf Gleichheit überprüft werden. Ist dies der Fall, wird das Wort aus dem Wörterbuch als enttwistetes Wort in den Text eingesetzt.

Beispiele

In diesem Abschnitt gehe ich auf die Originalbeispiele der Webseite (https://bwinf.de/bundeswettbewerb/37-bwinf/1-runde/material-371/) ein:

Beispiel 1:

Beim getwisteten Text fällt auf, dass Worte, wie „der“, „im“, „in“ und „den“ vermeintlich nicht getwistet wurden. Das liegt daran, dass zum Mischen eines Wortes mindestens 4 Buchstaben notwendig sind: Anfangs- & Endbuchstabe und 2 Buchstaben in der Mitte des Wortes um die Reihenfolge überhaupt ändern zu können. Zudem kann es vorkommen, dass Wörter in der Originalreihenfolge verbleiben, was am Beispiel „frühen“ erkenntlich ist. Das liegt daran, dass auch eine geringe Chance besteht, dass die Originalreihenfolge auftritt.

Der enttwistete Text weist nun einige Fehler auf bzw. Wörter die nicht erkannt wurden. Bspw. „twist“ und „Modetanz“ wurden nicht enttwistet. Dies lässt sich auf die Wörterliste zurückführen, welche rein technisch nicht den gesamten Wortschatz aller Kulturen umfassen kann.
Zudem wurde „Musik“ im Wort „Twist-Musik“ nicht erkannt, was daran liegt, dass die Interpunktion nur entfernt wird und nicht versucht wird die Wörter vor und nach diesem Zeichen, insofern sie vorhanden sind, zu enttwisten. Dies würde sich zwar im Algorithmus integrieren lassen, verkompliziert diesen jedoch erheblich. Trotz dessen hat der Untwist-Algorithmus eine relativ hohe Erfolgsquote, welche jedoch wie bereits erwähnt von der Wortwahl des Textes und der Wörterliste abhängig ist.

Bei den folgenden Beispielen wurde nicht weiter erläutert, da diese keine zusätzlichen Besonderheiten aufweisen.

Eingabe:	Twisted:	Untwisted:
Der Twist (Englisch twist = Drehung, Verdrehung) war ein Modetanz im 4/4-Takt, der in den frühen 1960er Jahren	Der Twsit (Esngiclh twsit = Durnheg, Vhnedrrueg) war ein Mtdenoaz im 4/4-Tkat, der in den frühen 1960er Jhraen	Der Twsit (englisch twsit = Drehung, Verdrehung) war ein Mtdenoaz im 4/4-Takt, der in den frühen 1960er Jahren

populär wurde und zu Rock'n'Roll, Rhythm and Blues oder spezieller Twist-Musik getanzt wird.	plpuoär wrdue und zu Rcok'n'Rlol, Rhtyhm and Belus oder spzleeiler Tswit-Muisk ganztet wird	populär wurde und zu Rcok'n'Rlol, Rhtyhm and Blues oder spezieller Tswit-Muisk getanzt wird

Beispiel 2:

Eingabe:	Twisted:	Untwisted:
Hat der alte Hexenmeister sich doch einmal wegbegeben! Und nun sollen seine Geister auch nach meinem Willen leben. Seine Wort und Werke merkt ich und den Brauch, und mit Geistesstärke tu ich Wunder auch.	Hat der alte Hesixneemetr sich dcoh eimanl weegbbegen! Und nun sloeln seine Geesitr auch ncah mieenm Weilln leebn. Sniee Wrot und Wrkee mkert ich und den Bcaruh, und mit Gktesrsäieste tu ich Wdneur acuh	Hat der alte Hexenmeister sich doch einmal weegbbegen! Und nun sollen seine Geister auch nach meinem willen leben. seine Wort und werke merkt ich und den Brauch, und mit Geistesstärke tu ich wunder auch

Beispiel 3:

Eingabe:	Twisted:	Untwisted:
Ein Restaurant, welches a la carte arbeitet, bietet sein Angebot ohne eine vorher festgelegte Menüreihenfolge an. Dadurch haben die Gäste zwar mehr Spielraum bei der Wahl ihrer Speisen, für das Restaurant entstehen jedoch zusätzlicher Aufwand, da weniger Planungssicherheit vorhanden ist.	Ein Resanuartt, weclehs a la ctare aeetribt, beetit sein Anebogt ohne eine vhorer flgettsgeee Mrogfeneünilehe an. Dcudarh hbaen die Gstäe zwar mher Saelpruim bei der Whal iehrr Sseipen, für das Reuaartsnt esttneehn jodech zhutsälcezir Afanuwd, da wgineer Pehrsuelcinghsiant veordhann ist	Ein Restaurant, welches a la ctare abrietet, bietet sein Angebot ohne eine vorher festgelegte Mrogfeneünilehe an. dadurch haben die Gäste zwar mehr Spielraum bei der Wahl ihrer speisen, für das Restaurant entstehen jedoch zusätzlicher Aufwand, da weniger Pehrsuelcinghsiant vorhanden ist

Quellcode - Twist

```
package de.ncrypted.twist;

/**
 * @author ncrypted
 */
public class Twist {

    private static final Pattern PATTERN = Pattern.compile("([a-
zA-ZäÄöÖüÜßß])([a-zA-ZäÄöÖüÜßß]*)([a-zA-ZäÄöÖüÜßß])");
    private static String text = "";

    public static void main(String[] args) {
        readText(textFileName);
        System.out.println("---UNTWISTED---");
```

```
            System.out.println(text);
            System.out.println("---TWISTED---");
            System.out.println(twist(text));
        }

        private static String twist(String input) {
            final StringBuilder builder = new StringBuilder();
            final Matcher matcher = PATTERN.matcher(input);

            // Filtere alle anderen Zeichen, die keine Buchstaben sind
    und speichere sie mit entsprechendem Index
            // Notiz: Die TreeMap sorgt dafür, dass die gefilterten
    Zeichen in der aufsteigenden Reihenfolge gespeichert werden
            Map<Integer, String> filtered = new TreeMap<>();
            Matcher matcherPunctuation = Pattern.compile("[^a-zA-
    ZäÄöÖüÜßß]").matcher(input);
            while (matcherPunctuation.find()) {
                filtered.put(matcherPunctuation.start(),
    matcherPunctuation.group());
            }

            // Filtere alle einzelnen Buchstaben und speichere sie mit
    entsprechendem Index
            Matcher matcherSingleLetters = Pattern.compile("(?<![a-zA-
    ZäÄöÖüÜßß])([a-zA-ZäÄöÖüÜßß])(?![a-zA-
    ZäÄöÖüÜßß])").matcher(input);
            while (matcherSingleLetters.find()) {
                filtered.put(matcherSingleLetters.start(),
    matcherSingleLetters.group());
            }

            while (matcher.find()) {
                builder
                        // Behalte 1. Buchstaben des Wortes bei
                        .append(matcher.group(1))
                        // Mische Buchstaben in der Mitte des Wortes
                        .append(shuffleString(matcher.group(2)))
                        // Behalte letzten Buchstaben des Wortes bei
                        .append(matcher.group(3));
            }

            // Fuege die gefilterten Zeichen & einzelnen Buchstaben
    wieder zum Ergebnis hinzu
            for (Map.Entry<Integer, String> entry :
    filtered.entrySet()) {
                builder.insert(entry.getKey(), entry.getValue());
            }

            return builder.toString().trim();
        }

        private static String shuffleString(String input) {
            final List<String> letters =
    Arrays.asList(input.split(""));
            Collections.shuffle(letters);
            return String.join("", letters);
        }
    }
```

Quellcode - Untwist

```
package de.ncrypted.twist;

/**
 * @author ncrypted
 */
public class Untwist {

    private static Set<String> wordlist = new HashSet<>();
    private static List<String> text = new ArrayList<>();

    public static void main(String[] args) {
        // Lese Wortliste und getwisteten Text ein
        readWordList();
        readText(textFileName);
        System.out.println("---TWISTED---");
        for (String line : text) {
            System.out.println(line);
        }
        untwist();
        System.out.println("---UNTWISTED---");
        for (String line : text) {
            System.out.println(line);
        }
    }

    private static void untwist() {
        // Iteriere ueber jede Zeile des Textes
        for (int i = 0; i < text.size(); i++) {
            String line = text.get(i);
            String newLine = "";
            // Iteriere ueber jedes Wort der momentanen Zeile
            for (String lineWord : line.split(" ")) {
                // Filtere alle anderen Zeichen, die keine
Buchstaben sind und speichere sie mit entsprechendem Index
                LinkedHashMap<Integer, String> punctuation = new
LinkedHashMap<>();
                String[] characters = lineWord.split("");
                for (int j = 0; j < characters.length; j++) {
                    String character = characters[j];
                    if (character.matches("[^a-zA-ZäÄöÖüÜß]")) {
                        punctuation.put(j, character);
                    }
                }
                // Entferne alle anderen Zeichen, die keine
Buchstaben sind, des momentanen Wortes
                lineWord = lineWord.replaceAll("[^a-zA-ZäÄöÖüÜß]",
"");

                // Filtere Wortliste nach Möglichkeiten,
entsprechend zur Laenge, Anfangs- & Endbuchstaben des
Ausgangswortes
                Set<String> possibles = new HashSet<>();
                for (String word : wordlist) {
                    if (word.length() > 3
                            && word.length() == lineWord.length()
                            && word.substring(0,
1).equalsIgnoreCase(lineWord.substring(0, 1))
                            && word.substring(word.length() - 1)
```

```
                                .equalsIgnoreCase(lineWord.substring(l
ineWord.length() - 1))) {
                            possibles.add(word);
                        }
                    }

                    // Pruefe, ob ein Wort aus dem Woerterbuch die
exakt selben Buchstaben wie das Ausgangswort hat
                    String chosen = null;
                    for (String possible : possibles) {
                        // Notiz: Die Sortier-Methode behandelt jeden
Buchstaben in Kleinschreibung
                        if
(sortAlphabetically(possible).equalsIgnoreCase(sortAlphabetically(
lineWord))) {
                            chosen = possible;
                        }
                    }
                    // Falls kein Wort gefunden wurde, wird das
Original-Wort wieder eingesetzt
                    if (chosen == null) {
                        chosen = lineWord;
                    }

                    // Fuege die gefilterten Zeichen wieder zum
Ergebnis hinzu
                    StringBuilder builder = new StringBuilder(chosen);
                    for (Map.Entry<Integer, String> entry :
punctuation.entrySet()) {
                        builder.insert(entry.getKey(),
entry.getValue());
                    }
                    chosen = builder.toString();
                    // Fuege das Ergebnis zur neuen Zeile hinzu
                    newLine += chosen + " ";
                }
                // Entferne ueberstehendes Leerzeichen am Ende der
neuen Zeile
                newLine = newLine.substring(0, newLine.length() - 1);
                // Aktualisiere die Originalzeile
                text.set(i, newLine);
            }
        }

        private static String sortAlphabetically(String string) {
            return string.toLowerCase().chars().sorted()
                    .collect(StringBuilder::new,
StringBuilder::appendCodePoint, StringBuilder::append)
                    .toString();
        }
}
```