Aufgabe 2: Twist

Team-ID: 00772

Team-Name: T.S

26. November 2018

Inhaltsverzeichnis

1 Lösungsidee

1.1 Twisten

Zunächst muss beachtet werden, dass in einem Text nur Wörter, die länger sind als 3 Buchstaben getwistet werden müssen. Um ein Wort der Länge n zu twisten, nimmt man vom 2. bis zum (n-1)-ten Buchstaben (damit der erste und letzte Buchstabe beibehalten werden) jeden einzelnen Buchstaben des nicht getwisteten Worts und ordnet im eine zufällig gewählte Position (zwischen 2 und n-1) im getwisteten Wort zu. Dabei können immer nur die Positionen zugeordnet werden, die davor nicht schon einem anderen Buchstaben zugeordnet wurden (damit Buchstaben im neuen Wort nicht überschrieben werden).

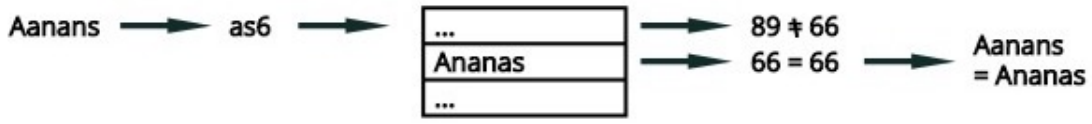
1.2 Enttwisten

Um ein Wort (mit mehr als 3 Buchstaben) in einem Text zu enttwisten, muss man sich zunächst überlegen wie man die Wörterliste darstellt, da ein Durchsuchen der gegebenen Liste bei jedem einzelnen Wort deutlich zu unübersichtlich und ineffizient wäre.

Ein getwistetes Wort und das ursprüngliche Wort haben zunächst 3 Eigenschaften gemeinsam:

den ersten Buchstaben, den letzten Buchstaben, und die Länge. Es bietet sich also an, die Wörterliste in kürzere Listen zu unterteilen, die alle jeweils in diesen 3 Eigenschaften übereinstimmen. Man hat also eine Abbildung die einem Schlüssel der Form [erster Buchstabe][letzter Buchstabe][Länge] (also z.b ab5, ez12, etc.) die zugehörige Liste zuordnet.

Hat man nun zu einem getwisteten Wort die zugehörige Liste mit möglichen ursprünglichen Wörtern gefunden (und enthält diese mehr als 1 Wort) vergleicht man sie weiter indem man zunächst die Summe der Unicodewerte der Buchstaben 2 bis (n-1) berechnet. Stimmt diese beim getwisteten Wort und einem Wort aus der Liste überein, wird als nächstes kontrolliert ob jeder Buchstabe der im Wort aus der Liste vorkommt auch im getwisteten Wort vorkommt (Summe der Unicodewerte allein reicht nicht, da auch verschiedene Wörter insgesamt die gleiche Summe haben können, der Schritt ist also nur zum Eingrenzen der möglichen Wörter). Ist das der Fall, wird das Wort aus der Liste als enttwistetes Wort ausgegeben. Zu allerletzt wird noch die Groß- oder Kleinschreibung des Wortes im Text (die nicht unbedingt mit der des gefundenen Worts aus der Liste übereinstimmen muss) angepasst. Das folgende Bild veranschaulicht den Ablauf für ein Beispiel:



Es gibt aber natürlich auch einige Wörter die sich mit dieser Methode nicht entschlüsseln lassen oder eventuell falsch entschlüsselt werden. Sie lassen sich in 2 Hauptgruppen einteilen:

- Wörter, die nicht in der Wörterliste enthalten sind: darunter auch Komposita aus Wörtern die schon in der Liste sind, Eigennamen, und Fachbegriffe. Eine mögliche Lösung wäre die Wörterliste zu erweitern, zum Beispiel falls man den Kontext und möglicherweise benutzte Wörter im Text kennt.
- Wörter, die aus den exakt gleichen Buchstaben bestehen wie ein anderes Wort, nur in einer etwas anderen Reihenfolge: zum Beispiel „frühen“ und „führen“. Es wäre sehr schwer dieses Problem zu lösen, da man um das Wort zu entschlüsseln Parameter wie Grammatik und Kontext maschinell interpretieren müsste.

Insgesamt ist diese Methode also bei weitem nicht ideal für Entschlüsslungen, aber vielleicht ein Ansatz um Teile des Textes zu verstehen.

2 Umsetzung

Die Lösungsidee wurde in Java implementiert.

Beim Twisten wird die Textdatei mit der Klasse Twist eingelesen, jedes Wort, das getwistet werden muss, getwistet und schließlich wird der getwistete Text als Textdatei mit dem Namen „twisted_[ursprünglicher Dateiname].txt“ ausgegeben.

Beim Enttwisten wird zunächst in der Klasse UnTwist die gegebene Wörterliste eingelesen und die HashMap mit den Daten aus der Liste erstellt. Anschließend folgt der Entschlüsselungsprozess und auch hier wird das Ergebnis als neue Textdatei mit dem Namen „entwist_[ursprünglicher Dateiname].txt“ ausgegeben. Dabei werden Wörter, die nicht enttwistet werden konnten mit einem (n) markiert.

3 Beispiele

3.1 Twisten

Beispiel 1 (twist1.txt -> twisted_twist1):
Der Twsit (Ecgilsnh twist = Dernhug, Vrnhueedrg) war ein Meontadz im 4/4-Takt, der in den fhüern 1960er Jearhn päolpur wdure und zu Rcok'n'Roll, Rythhm and Buels oder seeziellpr Tsiwt-Miusk gtnzaet wrid.

Beispiel 2 (twist2.txt -> twisted_twist2.txt):
Hat der alte Hxnmeesteeir sich doch eminal weeggbeebn! Und nun sellon siene Gsieetr acuh nach mienem Wlelin lbeen. Sinee Wrot und Wrkee mkert ich und den Bcruah, und mit Gssstkeriätee tu ich Wnduer auch.

Beispiel 3 (twist3.txt -> twisted_twist3.txt):
Ein Rearstunat, wlceehs a la carte aebtiret, biteet sien Agbneot onhe enie vheror fgetlegeste Moheneneülifgre an. Drcudah heabn die Gstäe zawr mher Suialrepm bei der Wahl irher Spieesn, für das Rtuarneast eethenstn jocdeh zscätuizlehr Anfuwad, da weeingr Phcgreiueshnnalsit vdehoanrn ist.

Beispiel 4 (twist4.txt -> twisted_twist4.txt):
Astguua Ada Boyrn King, Cotsnues of Lelocvae, war enie bsirtiche Aigldee und Mktreiehtamain, die als die esrte Primerimogarren üabepuhrt gilt. Betries 100 Jrhae vor dem Aomkefmun der etrsen Pasmrahreeopircgrmn esrann sie enie Rceehn-Meinchak, der einige Kzetnpoe meeodnrr Pararmrsecimpreohgn vgarnhoewm.

Beispiel 5 (twist5.txt -> twisted_twist5.txt):
Zu lang um hier abgedrucukt zu werden, lässt sich bei den Dateien finden.

3.2 Enttwisten

Beispiel 1 (enttwist.txt -> enttwist_enttwist.txt):
Der Twsit(n) (Englisch tiwst(n) = Drehung, Verdrehung) war ein Mdaotenz(n) im 4/4-Takt, der in den führen 1960er Jahren populär wurde und zu Rock'n'Roll(n), Ryhthm(n) and Blues oder spezieller Twsit(n)-Musik getanzt wird.

Beispiel 2 (twisted_twist2.txt -> enttwist_twisted_twist2.txt.txt):
Hat der alte Hexenmeister sich doch einmal weeggbeebn(n)! 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 (twisted_twist4.txt -> enttwist_twisted_twist4.txt.txt):
Astguua(n) Ada Boyrn(n) King(n), Cotsnues(n) of Lelocvae(n), war eine britische Adelige und Mathematikerin, die als die erste Programmiererin überhaupt gilt. Breites 100 Jahre vor dem Aufkommen der ersten Programmiersprachen ersann sie eine Rechen-Mechanik, der einige Konzepte moderner Programmiersprachen vorwegnahm.

Auch die anderen im ersten Teil getwisteten Beispiele lassen sich bei den Dateien enttwistet vorfinden.

4 Quellcode

Twisten

```
public class Twist {

    public static void main(String[] args) throws FileNotFoundException, IOException {
        //Einlesen des Textes durch BufferedReader, neue Datei durch BufferedWriter
        (...)

        int s;
        String word = "";
        String twistedWord;
        while ((s = in.read()) != -1) {
            char l = (char) (s);
            if (Character.isLetter(l)) {
                word = word + l;
            } else {
                //bei einem Wort mit weniger als 4 Zeichen muss nicht getwistet werden
                if (word.length() > 3) {
                    char[] wordArray = word.toCharArray();
                    twistedWord = twistWord(wordArray);
                } else {
                    twistedWord = word;
                }
                writer.write(twistedWord+l);
                word = "";
            }
        }
        writer.close();
    }

    //twistet ein einzelnes Wort
    static String twistWord(char[] wordArray) {
        char[] wordArrayTwist = new char[wordArray.length]; //getwistetes Wort
        wordArrayTwist[0] = wordArray[0]; //erster Buchstabe bleibt gleich
        wordArrayTwist[wordArray.length - 1] = wordArray[wordArray.length - 1];
        //Liste der noch nicht besetzen Positionen
        LinkedList<Integer> positions = new LinkedList();
        for (int j = 1; j < wordArray.length - 1; j++) {
            positions.add(j);
        }
```

```
39          int pos;
            //allen Buchstaben von der 2. bis zur vorletzen Position
41          //wird eine neue zufaellige Position zugeordente
            for (int i = 1; i < wordArray.length - 1; i++) {
43              pos = randomPosition(positions);
                wordArrayTwist[pos] = wordArray[i];
45          }
            String twistedWord = new String(wordArrayTwist);
47          return twistedWord;
        }
49
        //generiert eine zufaellige neue Position und
51      //entfernt diese aus der Liste der noch freien neuen Positionen
        static int randomPosition(LinkedList<Integer> positions) {
53          Random rand = new Random();
            int index = rand.nextInt(positions.size());
55          int newPos = positions.get(index);
            positions.remove(index);
57          return newPos;
        }
59  }
```

Enttwisten

```
1   public class UnTwist {

3       static HashMap<String, ArrayList<String>> wortliste;

5       public static void main(String[] args) throws FileNotFoundException, IOException {
            buildWordMap();
7           //Einlesen des Textes durch BufferedReader, neue Datei mit BufferedWriter
            (...)
9
            int s;
11          String word = "";
            String enttwist;
13          while ((s = in.read()) != -1) {
                char l = (char) (s);
15              if (Character.isLetter(l)) {
                    word = word + l;
17              } else {
                    if (word.length() > 3) {
19                      enttwist = untwistWord(word);
                    } else {
21                      enttwist = word;
                    }
23                  //Kontrolle Grossschreibung
                    if (!word.equals("")) {
25                      String uWord = word.substring(0, 1).toUpperCase()
                        + word.substring(1, word.length());
27                      if (word.equals(uWord)) {
                            enttwist = enttwist.substring(0, 1).toUpperCase() +
29                          enttwist.substring(1, enttwist.length());
                        }
```

```
31                  }
                    writer.write(enttwist + 1);
33                  word = "";
                }
35          }
            writer.close();
37      }

39      //enttwistet ein einzelnes Wort
        static String untwistWord(String word) {
41          String enttwist = word + "(n)";
            String k = computeKey(word);
43          if (wortliste.containsKey(k)) {
                ArrayList<String> possible = wortliste.get(k);
45              if (possible.size() == 1) {
                    enttwist = possible.get(0);
47              } else {
                    int hword = computeWordHash(word);
49                  for (int j = 0; j < possible.size(); j++) {
                        if (hword == (computeWordHash(possible.get(j)))) {
51                          if (compareLetters(possible.get(j), word))
                                enttwist = possible.get(j);
53                      }
                    }
55              }
            } else {
57              enttwist = word + "(n)";
            }
59          return enttwist;
        }
61
        //enthaelt Wort w1 alle Buchstaben aus w2?
63      static boolean compareLetters(String w1, String w2) {
            w1 = w1.substring(1, w1.length() - 1);
65          w2 = w2.substring(1, w2.length() - 1);
            for (int i = 0; i < w1.length(); i++) {
67              char c = w1.charAt(i);
                boolean in = false;
69              for (int j = 0; j < w2.length(); j++) {
                    if (w2.charAt(j) == c) {
71                      in = true;
                    }
73              }
                if (in == false) {
75                  return false;
                }
77          }
            return true;
79      }

81      //Schluessel der Form [erster Buchstabe][letzter Buchstabe][Laenge]
        static String computeKey(String word) {
83          word = word.toLowerCase();
            String fl = Character.toString(word.charAt(0));
85          String ll = Character.toString(word.charAt(word.length() - 1));
```

```
        int length = word.length();
87      return (fl + ll + length);
    }
89
    //Summe der Unicodewerte
91  static int computeWordHash(String w) {
        int h = 0;
93      for (int i = 1; i < w.length() - 1; i++) {
            int n = Character.getNumericValue(w.charAt(i));
95          h = h + n;
        }
97      return h;
    }
99
    //erstellt Hashmap zur Darstellung der Daten aus der W[U+FFFD]rterliste
101 static void buildWordMap() throws IOException {
        (...)
103     while ((line = in.readLine()) != null) {
            String k = computeKey(line);
105         if (wortliste.containsKey(k)) {
                wortliste.get(k).add(line);
107         } else {
                ArrayList<String> w = new ArrayList<>();
109             wortliste.put(k, w);
            }
111     }
    }
113 }
```