# Aufgabe 2
# „Twist“- Dokumentation

**37. Bundeswettbewerb Informatik 2018/19 - 1. Runde**

Sebastian Baron, Simon Fiebich, Lukas Rost

Team-ID: 00036

26. November 2018

## Inhaltsverzeichnis

# 1 Lösungsidee

## 1.1 Twisten

Das Twisten ist eine sehr einfache Angelegenheit. Da der erste und der letzte Buchstabe eines jeden Wortes beibehalten werden sollen, trennt man diese zunächst vom restlichen Wort ab. Die restlichen Buchstaben dürfen beliebig umgeordnet werden, es darf also eine beliebige Permutation dieser Buchstaben gebildet werden.

Für dieses sogenannte „Shufflen“ bieten viele Programmiersprachen bereits Bibliotheksfunktionen an. Ansonsten kann dies programmiertechnisch auch dadurch erreicht werden, dass man $n$ Mal (mit $n$ als Länge des Strings) die Buchstaben an jeweils zwei beliebigen, zufällig ausgewählten Positionen $i$ und $j$ miteinander vertauscht. Die entstehende Permutation ist garantiert zufällig.

Nun kann man das Wort wieder zusammenfügen. An den ersten Buchstaben wird das umgeordnete Wort angefügt und daran wiederum der letzte Buchstabe. Dieses Wort ist nun getwistet.

Ein Beispiel:

- Wir beginnen mit dem Wort $w =$ „Informatik“.
- Der erste Buchstabe ist $f =$ „I“, der letzte Buchstabe $l =$ „k“.
- Das restliche Wort lautet $r =$ „nformati“.
- Dieses wird umgeordnet. Dabei erhält man z.B. $r_{perm} =$ „ratominf“.
- Das getwistete Wort ist nun $t = f + r_{perm} + l$, in diesem Fall also „Iratominfk“.

## 1.2 Enttwisten

Beim Enttwisten muss man möglichst jedem getwisteten Wort ein ungetwistetes Wort aus der bereitgestellten Wörterliste zugeordnet werden. Um dies zu erreichen, muss man eine Funktion $f(w)$ finden, deren Ergebnis bei getwistetem und ungetwistetem Wort übereinstimmt. Diese wird im folgenden Hashfunktion genannt[1]. Hierbei erfahren wir aus einem getwisteten Wort folgendes über den Klartext:

- Anfangs- und Endbuchstabe
- Länge
- Häufigkeit der restlichen Buchstaben

Es wäre nun zwar möglich, das gesamte Wörterbuch linear auf Übereinstimmung dieser Eigenschaften zu durchsuchen, dies wäre aber laufzeittechnisch insbesondere bei längeren Texten inakzeptabel. Stattdessen lässt sich mithilfe der Eigenschaften die Hashfunktion bestimmen. Zum Beispiel erfüllt die Funktion $f(w) = f + sort(r) + l$ diese Bedingung. Dabei stellen

[1] Dies ist zwar nicht ganz korrekt, da eine Hashfunktion einem beliebigen Objekt einen Zahlenwert zuweist, während hier ein String bestimmt wird. Trotzdem erfüllt die Funktion einen ähnlichen Zweck wie eine Hashfunktion.

- $f$ den ersten Buchstaben,
- $r$ die mittleren Buchstaben,
- $l$ den letzten Buchstaben und
- $sort(r)$ eine Funktion, die die Buchstaben des Wortes $r$ alphabetisch sortiert,

dar.

Mithilfe dieser Funktion ist das Enttwisten nun möglich. Durch eine kurze Vorverarbeitung lässt sich damit die Suche beschleunigen. Dazu legt man eine sogenannte Map an, in der man Objekten (in diesem Fall Wörtern/Strings) andere Objekte (ebenfalls Strings) zuordnen kann, wobei diese Zuordnung natürlich eindeutig ist. Nun kann man für jedes Wort des Wörterbuchs in der Map eine Zuordnung $f(w) -> w$ anlegen. Dem „Hashwert“ wird also das ungetwistete Wort zugeordnet.

Dadurch lässt sich nun jedes getwistete Wort $t$, dessen Klartext im Wörterbuch hinterlegt ist, enttwisten, indem man $f(t)$ bestimmt und $t$ durch den $f(t)$ zugeordneten Eintrag in der Map ersetzt. Dieses Verfahren hat natürlich auch seine Grenzen. Wenn der Klartext eines Wortes nicht im Wörterbuch enthalten ist, kann das Wort auch nicht enttwistet werden. Um möglichst gute Ergebnisse zu erzielen, sollte das Wörterbuch deshalb möglichst viele Begriffe enthalten.

# 2 Umsetzung

Das Programm wurde in der Programmiersprache C++ umgesetzt. Im Implementierungsordner ist ein vorkompiliertes ausführbares Programm für Linux enthalten. Auf anderen Systemen müsste entsprechend neu kompiliert werden.

Das Programm verwendet die auf jedem System vorhandene Standard-Library (STL) sowie die Boost-Bibliothek für C++, die für den Umgang mit regulären Ausdrücken benötigt wird. Diese lässt sich über den Paketmanager der verwendeten Linux-Distribution nachinstallieren, unter Manjaro Linux beispielsweise mit `sudo pacman -S boost boost-libs`.

Das Programm muss über die Konsole gestartet werden. Zum Twisten wird das Programm folgendermaßen aufgerufen: `./aufgabe2 -t <zu twistende Datei> <Ausgabedatei>`, wärend man zum Enttwisten folgenden Befehl verwendet: `./aufgabe2 -d <zu twistende Datei> <Ausgabedatei>`. Die Ausdrücke in spitzen Klammern sind dabei durch die Pfade zu den entsprechenden Dateien zu ersetzen.

Es soll noch angemerkt werden, dass das Programm nur mit Windows-1252-Dateien (ANSI) funktioniert. Unicode-Dateien verwenden leider zwei Bytes, um Umlaute darzustellen, wodurch diese beim Twisten auseinandergerissen werden können. Demzufolge werden sie dann nicht mehr korrekt dargestellt.

Das Programm besteht aus mehreren Funktionen, die im folgenden erläutert werden.

- `read()` liest eine Datei in einen einzigen String ein, während *write*() einen String in eine Datei schreibt.

- `gen_wordmap()` erstellt die Map zum Enttwisten. Dabei wird für jedes Wort in der Wörterliste der Map wie beschrieben ein Eintrag $f(w) -> w$ hinzugefügt. Die Datei, welche die Wörterliste beinhaltet, wird durch die Preprocessor-Anweisung in Zeile 3 festgelegt.

- `main()` ist die Hauptfunktion, welche durch das Betriebssystem aufgerufen wird. Zunächst überprüft sie, ob die Anzahl der Konsolenargumente stimmt und liest diese Argumente dann entsprechend ein.
  Um die Bearbeitung des Textes einfach zu gestalten, werden reguläre Ausdrücke und die `regex_replace()`-Funktion der Boost-Library verwendet. Diese ersetzt alle Bestandteile eines Strings, auf die der reguläre Ausdruck passt, durch den Rückgabewert einer festgelegten Funktion. Der verwendete Ausdruck beschreibt ein Wort und hat folgende Form:
  `sregex reg = _b >> +(_w | "Ä" | "ä" | "Ü" | "ü" | "Ö" | "ö") >> _b;`
  Dieser Ausdruck erkennt alle Textbestandteile, die von Wortgrenzen (`_b`) umgeben sind und aus mindestens einem Buchstaben (`_w`) oder Umlaut bestehen, also alle Wörter. Dementsprechend übergibt man der `regex_replace()`-Funktion je nach Modus entweder `twist()` oder `detwist()`. Außerdem werden in `main()` die Ein- bzw. Ausgaberoutinen für die Textdateien aufgerufen.

- `twist()` twistet je ein Wort, das von `regex_replace()` in Form eines speziellen Objekts übergeben wird. Wenn ein Wort nur 2 Zeichen oder weniger lang ist, kann es direkt zurückgegeben werden, denn es verändert sich durch das Twisten nicht. Sonst erstellt man mittels der STL-Funktion `random_shuffle()` eine zufällige Permutation der mittleren Buchstaben und gibt entsprechend das getwistete Wort zurück.

- `detwist()` enttwistet je ein Wort, dass ähnlich wie bei `twist()` übergeben wird. Wörter mit 2 oder weniger Zeichen ändern sich nicht. Sonst wird die Hashfunktion auf das Wort angewendet und, falls vorhanden, das Wort durch den entsprechenden Eintrag in der Map ersetzt.

# 3 Beispiele

## 3.1 Twisten

`Getwisteter Beispieltext 1`

```
1 Der Twsit
2 Esiglcnh tswit = Dnehurg, Vnrudeerhg
3 war ein Modntaez im 4/4-Tkat,
4 der in den frühen 1609er Jahern puoplär
5 wdrue und zu
6 Rock'n'Rlol, Rhhytm and Beuls oder selipeezlr
7 Tiswt-Msuik geznatt wird.
```

`Getwisteter Beispieltext 2`

```
1 Hat der alte Heexisetnmer scih doch enmial weegbeegbn! Und nun slleon siene Gesiter
  ↪ acuh ncah mienem Willen leebn. Sniee Wort und Wrkee mrekt ich und den Bucrah, und
  ↪ mit Gteessistärke tu ich Wneudr auch.
```

Getwisteter Beispieltext 3

```
Ein Ruanaertst, wcheels a la carte aebiertt, betiet sien Aogbnet onhe eine voerhr fgegltsetee Menürhlieegonfe an. Ddarcuh hbean die Gäste zawr mher Spaueirlm bei der Wahl iehrr Sepsien, für das Rrstaaunet eethenstn joecdh zusätiheczlr Aanuwfd, da wgeenir Phcesinuerlgsinaht vonadrhen ist.
```

Getwisteter Beispieltext 4

```
Atsguua Ada Byorn Knig, Cnseouts of Lcaoleve, war enie bithcisre Aedigle und Maemetiairkhtn, die als die etrse Pimeriomaegrrrn übrapehut glit. Beierts 100 Jahre vor dem Auofmkemn der etersn Pmrhraorapsecmiergn enrsan sie enie Rehecn-Mnecaihk, der eniige Ktnpzeoe mnreoder Peireaamrgcsrpmrhon vrawhengom.
```

Auszug aus dem getwisteten Beispieltext 5

```
Alcie fing an sich zu lagelwnien; sie saß schon lange bei iehrr Sesthwecr am Ufer und htate nchits zu thun. Das Buch, das irhe Setschewr las, geefil ihr nhcit; denn es wrean wdeer Bdeilr ncoh Gespräche diarn. „Und was nütezn Bücehr,\ dctahe Alice, „ohne Bliedr und Gepsräche?\

Sie üblrgteee scih eben, so gut es gnig, dnen sie war schläfirg und dumm von der Hzite, ob es der Mühe wreth sei aztfuheusen und Gänebslümhecn zu pflücekn, um enie Kttee dmait zu mcehan, als plötzcilh ein weißes Knchniaen mit rheotn Aguen dhict an ihr voirrnnbteae.

Deis war gdare nciht sher merkwüridg; Aclie fnad es auch nicht sher außeerrltcondih, daß sie das Kcenaihnn sgaen hörte: „O weh, o weh! Ich werde zu spät komemn!\ Als sie es später wdieer übeglerte, fiel ihr ein, daß sie sich darüber hätte wndreun slolen; dcoh zur Zeit kam es ihr Alels ganz natürcilh vor. Aber als das Kheinancn sinee Uhr aus der Wseancstehte zog, ncah der Zeit sah und eilig fiotlref, snparg Alcie auf; dnen es war ihr doch noch nie voekeommrgn, ein Knceainhn mit eenir Wehnsastctee und eneir Uhr diran zu sheen. Vor Ngidereue bennrned, rtanne sie ihm nach über den Gpltasraz, und kam noch zur rehtecn Zeit, um es in ein großes Loch unter der Hkece schlüpfen zu sheen.

Den nächsetn Acuigbenlk war sie ihm nach in das Loch huegpsneiirngnen, onhe zu bnedeekn, wie in alelr Welt sie wedier heuasmkroemn köntne.

Der Eangnig zum Kbnaeicnanhu lief esrt grdaueaes, wie ein Tnenul, und ging dnan plötliczh abwärts; ehe Ailce noch den Gkdnaeen feassn kotnne sich shecnll fleethsutzan, fühtle sie shocn, daß sie feil, wie es sicehn, in eeinn tiefen, tefein Bneunrn.
```

## 3.2 Enttwisten

Enttwisteter Text aus der Aufgabenstellung

```
Der Twsit
Eigsnclh tiwst = Drehung, Verdrehung
```

```
war ein Mdaotenz im 4/4-Takt,
der in den frhüen 1960er Jahren populär
wurde und zu
Rock'n'Roll, Ryhthm and Blues oder spezieller
Twsit-Musik getanzt wird.
```

---

Enttwisteter Beispieltext 1

```
Der Twsit
Esiglcnh tswit = Drehung, Verdrehung
war ein Modntaez im 4/4-Takt,
der in den frühen 1609er Jahren puoplär
wurde und zu
Rock'n'Rlol, Rhhytm and Blues oder spezieller
Tiswt-Musik getanzt wird.
```

---

Enttwisteter Beispieltext 2

```
Hat der alte Hexenmeister sich doch einmal weegbeegbn! Und nun sollen seine Geister
↪ auch nach meinem Willen leben. Sniee Wort und Wrkee merkt ich und den Brauch, und
↪ mit Gteessistärke tu ich Wneudr auch.
```

---

Enttwisteter Beispieltext 3

```
Ein Restaurant, welches a la carte abrietet, bietet sein Angebot ohne eine vorher
↪ festgelegte Menürhlieegonfe an. Ddarcuh haben die Gäste zwar mehr Spielraum bei
↪ der Wahl ihrer Sepsien, für das Restaurant entstehen jedoch zusätiheczlr Aufwand,
↪ da weniger Phcesinuerlgsinaht vorhanden ist.
```

---

Enttwisteter Beispieltext 4

```
Atsguua Ada Byorn Knig, Cnseouts of Lcaoleve, war eine britische Aedigle und
↪ Mathematikerin, die als die erste Programmiererin übrapehut gilt. Bieters 100
↪ Jahre vor dem Auofmkemn der ersten Programmiersprachen ersann sie eine
↪ Rehecn-Mechanik, der einige Konzepte moderner Programmiersprachen vorwegnahm.
```

---

Auszug aus dem enttwisteten Beispieltext 5

```
Alice fing an sich zu langweilen; sie saß schon lange bei ihrer Schwester am Ufer und
↪ hatte nichts zu thun. Das Buch, das ihre Schwester las, gefiel ihr nicht; denn es
↪ waren weder Bilder noch Gespräche darin. „Und was nütezn Bücehr,\ dachte Alice,
↪ „ohne Bilder und Gepsräche?\

Sie üblrgteee sich eben, so gut es ging, denn sie war schläfirg und dumm von der
↪ Hitze, ob es der Mühe wreth sei aufzustehen und Gänebslümhecn zu pflücekn, um eine
↪ Kttee damit zu machen, als plötzcilh ein weißes Kaninchen mit rheotn Augen dicht
↪ an ihr voirrnnbteae.
```

```
4
5  Deis war gdare nicht sehr merkwüridg; Alice fand es auch nicht sehr außeerrltcondih,
   ↪  daß sie das Kaninchen sagen hörte: „O weh, o weh! Ich werde zu spät kommen!\ Als
   ↪  sie es später wieder übeglerte, fiel ihr ein, daß sie sich darüber hätte wundern
   ↪  sollen; doch zur Zeit kam es ihr Alels ganz natürcilh vor. Aber als das Kaninchen
   ↪  seine Uhr aus der Westentasche zog, nach der Zeit sah und eilig fiotlref, sprang
   ↪  Alice auf; denn es war ihr doch noch nie vorgekommen, ein Kaninchen mit einer
   ↪  Westentasche und einer Uhr darin zu sehen. Vor Neugierde brennend, rannte sie ihm
   ↪  nach über den Gpltasraz, und kam noch zur rechten Zeit, um es in ein großes Loch
   ↪  unter der Hkece schlüpfen zu sehen.
6
7  Den nächsetn Augenblick war sie ihm nach in das Loch huegpsneiirngnen, ohne zu
   ↪  bedenken, wie in aller Welt sie wieder herauskommen köntne.
8
9  Der Eingang zum Kbnaeicnanhu lief erst geradeaus, wie ein Tunnel, und ging dann
   ↪  plötliczh abwärts; ehe Alice noch den Gedanken fassen konnte sich schnell
   ↪  festzuhalten, fühtle sie schon, daß sie fiel, wie es schien, in einen tiefen,
   ↪  tiefen Brunnen.
10
```

---

Als Ausgangstexte wurden jeweils die durch das Programm getwisteten Beispieltexte benutzt. Bei den Beispieltexten fehlen hier teilweise Klammern oder wurden Anführungszeichen durch Slashes ersetzt. Dies ist in den originalen Ausgabedateien nicht so, sondern hängt mit dem verwendeten LaTeX-Paket zusammen.

# 4 Quellcode

```cpp
#include <bits/stdc++.h>
#include <boost/xpressive/xpressive.hpp>
#define WORDLIST "woerterliste_ansi.txt"

using namespace std;
using namespace boost::xpressive;

map<string,string> wordmap;
//nur Windows 1252 (= ANSI) Dateien funktionieren

// Eingabe
string read(string infile){
    ifstream in(infile);
    stringstream buffer;
    buffer << in.rdbuf();
    return buffer.str();
}

// Ausgabe
void write(string output, string outfile){
    ofstream out(outfile);
    out << output << flush;
}

// Map generieren
void gen_wordmap(void){
```

```
27      ifstream in(WORDLIST);
28      string word;
29      while(in >> word){
30          string key = string(word);
31          sort(key.begin()+1,key.end()-1);
32          wordmap[key] = word;
33      }
34  }
35
36  string twist(const boost::xpressive::smatch &m){
37      auto word = m[0].str();
38      char first = word[0], last = word[word.length()-1];
39      if(word.length() <= 2){
40          // Sonderfall: mittlerer Teil leer
41          return word;
42      } else {
43          // zufälliges Shufflen des mittleren Teils
44          word = word.substr(1,word.length()-2);
45          random_shuffle(word.begin(),word.end());
46          return first + word + last;
47      }
48  }
49
50  string detwist(const boost::xpressive::smatch &m){
51      string word = m[0].str();
52      if(word.length() <= 2) {
53          // Sonderfall: mittlerer Teil leer
54          return word;
55      } else {
56          // Anwendung der Hashfunktion
57          sort(word.begin()+1,word.end()-1);
58          // Nachschlagen in der Map
59          return (wordmap.count(word) == 1) ? wordmap[word] : m[0].str();
60      }
61  }
62
63  int main(int argc, char** argv){
64      // Benutzungshinweise
65      cout << "37. BwInf, 1. Runde, Aufgabe 2\n";
66      if(argc != 4){
67          cout << "Usage: -t <File to twist> <Output file> or -d <File to
            ↪  detwist> <Output file>" << "\n";
68          return 0;
69      }
70
71      // Regex + Konsolenargumente
72      sregex reg = _b >> +(_w | "Ä" | "ä" | "Ü" | "ü" | "Ö" | "ö") >> _b;
73      string mode = argv[1];
74      string inputfile = argv[2];
75      string outputfile = argv[3];
76
77      if(mode == "-t"){
78          // Twisten und I/O
```

```
79          auto str = read(inputfile);
80          string replaced = regex_replace(str, reg, twist);
81          write(replaced,outputfile);
82      } else if(mode == "-d"){
83          // Enttwisten und I/O
84          gen_wordmap();
85          auto str = read(inputfile);
86          string replaced = regex_replace(str, reg, detwist);
87          write(replaced,outputfile);
88      }
89      return 0;
90  }
```

Quellcode 1: Implementierung des Twist-Programms: `aufgabe2.cpp`