# Aufgabe 2: Twist

Symbroson
Team-ID: 00165

24. November 2018

## Inhaltsverzeichnis

# 1 Lösungsidee

Bei jedem Wort sind Anfangsbuchstabe, Endbuchstabe und die Buchstabenanzahl gegeben. Man kann also durch diese Eigenschaften bei jedem getwisteten Wort die Wortliste bereits auf wenige mögliche Wörter einschränken.

Die zweite Auswahl erfolgt durch den Vergleich der Buchstaben dazwischen, sofern das Wort länger als 3 Zeichen ist, da sonst keine Buchstaben vertauscht worden sein können. Somit kann in den meisten Fällen das gesuchte Wort gefunden und (grün) ausgegeben werden. Falls nicht, wird das Wort unverändert (rot), bzw. wenn mehrere Wörter möglich sind, werden diese alle (gelb, durch ’|’ getrennt) ausgegeben. Alle nicht-alphabetischen Zeichen werden weiß notiert.

# 2 Umsetzung

## 2.1 Wortlisten-Vorsortierung

### 2.1.1 Per Wortlänge

Aufgrund der Großen Anzahl an Wörtern wäre es vermessen bei jedem Wort die gesamte Liste zu durchsuchen. Deshalb kann man bereits beim Einlesen der Wortliste nach Wortlänge sortieren. Somit würden aus ca. 312.000 zu durchsuchenden Wörtern maximal etwa 39.000 bei einer Wortlänge von 12. Die maximal zu betrachtende Wortlänge in der Wortliste beträgt 34, wird im Programm aber vorsichtshalber auf 40 erhöht. Das Maximum kann auch dynamisch ermittelt werden, wurde aber der Einfachheit halber nicht umgesetzt.

### 2.1.2 Per Wortlänge und Anfangsbuchstabe

Man kann dies allerdings noch Erweitern, indem man den Anfangsbuchstaben ebenfalls in die Erstauswahl mit einbezieht. Dabei entstehen maximal ca. 6000 Einträge für 12 Buchstaben bei dem Anfangsbuchstaben a bzw. A. Dies macht den Wortfindeprozess bei twist5.txt fast 15x schneller (ohne Ausgabe) als ohne. Deshalb ist dies auch die Variante meiner Einsendung.

### 2.1.3 Per Wortlänge, Anfangs- und Endbuchstabe

In einer dritten Stufe kann man dann auch noch den Endbuchstaben betrachten. Somit könnte man die maximal zu durchsuchenden Wörter auf etwa 1500 einschränken. Auch hier werden die Wörter im Verhältnis zur Variante mit nur Länge und Anfangsbuchstabe etwa 3x schneller gefunden, allerdings entstehen hier schon eine Menge unbenutzter Listen, die dementsprechend unnötig Speicher blockieren, Weshalb im Programm Variante 2 Umgesetzt wurde.

## 2.2 Wortverdrehung

Standardmäßig wird die Eingabe per Eingabedatei oder durch lesen des Input-Streames (stdin) getwistet. Hierbei werden abwechselnd Sonderzeichen und Alphabetische Zeichen bis zum nächsten Leerzeichen eingelesen und in letzterem Fall alle Zeichen zwischen dem ersten und letzten Buchstaben des Wortes mit einem zufälligen Buchstaben aus demselben Bereich vertauscht, bevor sie ausgegeben werden.

## 2.3 Wortentdrehung

Die Wortentdrehung erfolgt mit der –untwist Option und folgt beim Einlesen demselben Prinzip wie bei der Verdrehung.

### 2.3.1 Findung

Jedes Wort wird zunächst auf seine Länge geprüft. Ist es kürzer als 4 Zeichen oder länger als die maximal definierte Länge, wird ebenfalls sofort ausgegeben, da es keine vertauschten Buchstaben geben bzw. nicht in der Wortliste verzeichnet worden sein kann.

### 2.3.2 Erstauswahl

Ist all dies nicht der Fall, wird dass Wort in der Wortliste gesucht. Dabei werden zuerst Anfangs- und Endbuchstaben verglichen. Bei den Anfangsbuchstaben muss beachtet werden, dass ein in der Wortliste klein geschriebenes Wort im Text großgeschrieben sein kann, wenn es am Satzanfang steht, allerdings kann ein inder Wortliste groß geschriebenes Wort niemals klein geschrieben sein.

### 2.3.3 Zeichenauszählung

Nun werden die inneren Zeichen miteinander verglichen. Dabei werden in einer Kopie der Zeichenkette alle Buchstaben des Wortlisten-Wortes aus dem Eingabewort überschrieben, um sicherzustellen, dass die Vorkommensanzahl jedes Buchstabens im Wort übereinstimmt. So wird der Vorgang entweder abgebrochen, wenn alle Buchstaben ersetzt wurden, oder wenn ein Zeichen nicht mehr im Eingabewort gefunden werden konnte.

### 2.3.4 Mehrere Treffer

Wurden alle Buchstaben gefunden, kann das gefundene Wort theoretisch ausgegeben werden. Allerdings kann ein verdrehtes Wort mehrere Bedeutungen haben, die bestenfalls alle dementsprechend hervorgehoben ausgegeben werden sollen. Dafür wird das Wort in einen Zwischenspeicher kopiert und erst im nächsten Durchlauf, wenn ein weiteres mögliches Wort gefunden wurde oder nach der Suche, ausgegeben. Des weiteren kann man dies dafür nutzen um versehentliche Dopplungen in der Wortliste (die leider vorkamen) auf einfachem Wege zumindest teilweise zu überspringen.

### 2.3.5 Ausgabe

Wenn nach der Suche nur ein Wort gefunden wurde, ist das gefundene Wort im Zwischenspeicher der einzige Treffer und wird grün ausgegeben. Sind mehrere gefunden worden, wird das Wort getrennt von einem ’|’ gelb und bei keiner Übereinstimmung unverändert in rot ausgegeben.

# 3 Beispiele

## 3.1

lstlisting @SH*makerunARGS* = *"res/twist*1*.txt"@SH@gDer@g@rTiswt@r(@gEnglisch@g@rtwist@r =
@gDrehung@g, @gVerdrehung@g)@gwareinModetanzim@g4/4−@gTakt@g, @gderinden@g@yfhre
@gMusikgetanztwird@g.

## 3.2 twist1.txt

In diesem Beispiel wurden anderssprachige Wörter (hier Englisch) verwendet. Diese können natürlich zumeist nicht gefunden werden oder führen sogar zu deutschen Wörtern mit einer völlig anderen Bedeutung. Das lässt sich leider nur schwer verhindern.

```
$ make; ./build/twist.out res/twist1.txt | ./build/twist.out --untwist
Der Tiswt
(Englisch twist = Drehung, Verdrehung)
war ein Modetanz im 4/4-Takt,
der in den führen|frühen 1960er Jahren populär
wurde und zu
Rock'n'Rlol, Rhyhtm and Blues oder spezieller
Tiwst-Musik getanzt wird.
```

## 3.3 twist2.txt

Hier ist 'wegbegeben' das einzige nicht gefundene Wort, weil es zumindest in dieser Form nicht in der Wortliste verzeichnet ist, sondern nur die Form 'begeben'. Man müsste entweder alle Nebenformen eines Wortes der Wortliste hinzufügen oder dem Programm beibringen diese selbstständig zu bilden, also die deutsche Sprache beibringen, um wirklich alle Wörter erkennen zu können. Dies kann höchstens durch extremes Hardcoding oder durch lernfähige Algorithmen erreicht werden, was beides hier nicht geeignet ist.

```
$ make; ./build/twist.out res/twist2.txt | ./build/twist.out --untwist
Hat der alte Hexenmeister sich doch einmal wgegebeben! 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.
```

## 3.4 twist3.txt

Auch in diesem Fall wurden andersprachige (französische) Ausdrücke verwendet. Das eigentliche Problem besteht aber aus den zusammengesetzten Substantiven. Auch hier lässt sich nicht ohne weiteres feststellen, ob dies der Fall ist und wo man diese trennen müsste, um auf die Basiswörter schließen zu können.

```
$ make; ./build/twist.out res/twist3.txt | ./build/twist.out --untwist
Ein Restaurant, welches a la carte abrietet|arbeitet, bietet sein Angebot ohne eine vorher
    festgelegte Mlefrnoiüeneghe an. Dadurch haben die Gäste zwar mehr Spielraum bei der
    Wahl ihrer Speisen, für das Restaurant entstehen jedoch zusätzlicher Aufwand, da
    weniger Paesrhgulneichnist vorhanden ist.
```

## 3.5 twist4.txt

Mit Eigennamen verhält es sich genauso wie bei den vorherigen Beispielen. Man bräuchte wohl noch einige Lexika mehr um Eigennamen mit in die Suche einzubeziehen.

```
$ make; ./build/twist.out res/twist4.txt | ./build/twist.out --untwist
Atgusua Ada Broyn Knig, Cuotenss of Llvaceoe, war eine britische Adelige und Mathematikerin
    , die als die erste Programmiererin überhaupt gilt. Breites|Bieters|Bereits 100 Jahre
    vor dem Aufkommen der ersten Programmiersprachen ersann sie eine Rechen-Mechanik, der
    einige Konzepte moderner Programmiersprachen vorwegnahm.
```

## 3.6 twist5.txt

Altteutsch ist ebenso schwierig zu erkennen und eine weitere Wortliste wäre vonnöten. Allerdings treten einige Ersetzungen vom Altdeutschen in die heutige Screibweise öfters auf. Die häufigsten sind beispielsweise `"ß"` zu `"ss"` oder `"th"` zu `"t"`. Diese Abfrage kann optional in dem Programm durch die `--ckeck-alt` Flag eingeschalten werden, da man standardmäßig von Hochdeutsch ausgeht und andernfalls woanders ungewollt Fehler entstehen können, die sonst nicht aufträten.

Ohne –chek-alt Option:

```
$ make; ./build/twist.out res/twist5.txt | ./build/twist.out --untwist
...
Entweder mtßue der Brunnen sehr tief sein, oder sie fiel|feil sehr langsam; denn sie hatte
    Zeit genug, sich beim Fallen umzusehen und sich zu wundern, was nun wohl geschehen wü
    rde. Zuerst versuchte|vertusche sie hinunter zu sehen, um zu wissen wohin sie käme,
    aber es war zu dunkel etwas zu erkennen. Da besah sie die Wände des Brunnens und
    bemerkte, daß sie mit Khnckhcärüeesnn und Bücherbrettern bedeckt waren; hier und da
    erblickte sie Landkarten und Bilder, an Haken aufgehängt. Sie nahm im Vofrleaeibln von
    einem der Bretter ein Tcefpöhn mit der Aufschrift|Auffrischt: „Eingemachte Apfelsinen“,
     aber zu ihrem großen Vrerduß war es leer. Sie wollte es nicht fallen lassen, aus
    Furcht|Frucht Jemand unter sich zu tedtön; und es gelang ihr, es in einen andren|andern
     Schrank, an dem sie veaikobrm, zu schieben.
...
```

Mit –chek-alt Option:

```
$ make; ./build/twist.out res/twist5.txt | ./build/twist.out --untwist --check-alt
...
Entweder musste der Brunnen sehr tief sein, oder sie fiel|feil sehr langsam; denn sie hatte
     Zeit genug, sich beim Fallen umzusehen und sich zu wundern, was nun wohl geschehen wü
    rde. Zuerst versuchte|vertusche sie hinunter zu sehen, um zu wissen wohin sie käme,
    aber es war zu dunkel etwas zu erkennen. Da besah sie die Wände des Brunnens und
    bemerkte, daß sie mit Kkercnhehäüncsn und Bücherbrettern bedeckt waren; hier und da
    erblickte sie Landkarten und Bilder, an Haken aufgehängt. Sie nahm im Vbrfilelaoen von
    einem der Bretter ein Töpchfen mit der Aufschrift|Auffrischt: „Eingemachte Apfelsinen“,
     aber zu ihrem großen Verdruss war es leer. Sie wollte es nicht fallen lassen, aus
    Furcht|Frucht Jemand unter sich zu töten; und es gelang ihr, es in einen andren|andern
    Schrank, an dem sie vrbeiokam, zu schieben.
...
```

# 4 Quellcode

```
/* ... */

#define MAXLEN 40 // maximale Wortlänge
#define CHRCNT (26 + 10) // = 26 Buchstaben + 9 verwendete Umlaute + 1 für andere
    Sonderzeichen/Umlaute


// speichert Wörter sortiert nach Länge
forward_list<wchar_t*> wordMap[MAXLEN][CHRCNT];

wchar_t uml[] = L"ÜÄÖßÉÀÊÂÑ";


// gibt Zeichennummer zurück (nicht Zeichencode)
uint charNum(wchar_t wc) {
    /* ... */
}

// zählt Umlaute
uint cntUml(wchar_t* word, int len = -1) {
    /* ... */
}


int main(int argc, const char* argv[]) {
    FILE* fp = NULL;
    uint i;
    bool chkAlt = false, untwist = false;
    setlocale(LC_CTYPE, "");

    // Argumente einlesen
    /* ... */

    // lese Eingabedatei zeilenweise
    wint_t wc;
    wchar_t buf[MAXLEN];
    wc = fgetwc(fp);

    // randomize
    srand(clock() + (long)fp);

    while (wc && wc != WEOF) {
        uint l;
        int corr;

        // lese & schreibe nicht-Wörter
        l    = 0;
        *buf = wc;
        while (wc != WEOF && charNum(wc) == CHRCNT) {
            if (wc == '\033') // ignore color escape sequences
                fwscanf(fp, L"[%i;%im", &i, &i);
            else
                buf[l++] = wc;
            wc = fgetwc(fp);
        }
        wprintf(L"%.*ls", l, buf);

        // lese Wörter
        l    = 0;
        *buf = wc;
        while (wc != WEOF && charNum(wc) != CHRCNT) {
            buf[l++] = wc;
            wc       = fgetwc(fp);
        }


        // Keine vertauschten Bst. möglich
        if (l <= 3) {
            wprintf(L"\033[0;%im%.*ls\033[0;37m", untwist ? 32 : 37, l, buf);
            continue;
```

```
        // Wort zu lang
    } else if (untwist && l > MAXLEN) {
        wprintf(L"\033[0;31m%.*ls\033[0;37m", l, buf);
        continue;
    }

    // twist word
    if (!untwist) {
        for (i = 1; i < l - 1; i++) swap(buf[i], buf[rand() % (l - 2) + 1]);
        wprintf(L"%.*ls", l, buf);

        // untwist
    } else {
        wchar_t p[MAXLEN], tmp[MAXLEN], *match = NULL;
        uint j, found = 0;
        corr = 255;

    search:
        // suche Basiswörter
        for (wchar_t* word: wordMap[l][charNum(*buf)]) {
            if ((buf[l - 1] != word[l - 1]) || // Endbst. verschieden
                ((buf[0] != word[0]) &&        // Anfangsbst. verschieden
                 (buf[0] != word[0] - 32)) ||  // klein->groß
                (cntUml(buf, l) != cntUml(word, l)) // gleiche Umlautzahl
            )
                continue;

            // kopiere Eingabewort zu p für Buchstabenzählung
            wcsncpy(p, buf, l);

            // von 1 - l-1 weil Anfangs- und Endbuchstaben bereits
            // überprüft worden sind
            for (i = 1; i < l - 1; i++) {
                for (j = 1; j < l - 1; j++) {
                    if (charNum(word[i]) == charNum(p[j])) {
                        p[j] = ' '; // überschreibe gefundene Zeichen
                        break;
                    }
                }

                // Zeichen nicht gefunden
                if (j == l - 1) break;
            }

            // alle Zeichen gefunden
            if (i == l - 1) {
                if (match) {
                    // gleiches Wort gefunden -> überspringen
                    if (!wcsncmp(match + 1, word + 1, l - 1)) continue;

                    // bereits anderes Wort gefunden -> gelb
                    wprintf(L"\033[0;33m%.*ls\033[0;37m|", l, match);
                }

                // kopiere richtiges Wort temporär in Eingabezeile
                // Anfangs- und Endbuchstabe wird beibehalten
                wcsncpy(buf + 1, word + 1, l - 2);
                match = buf;
                found++;
            }
        }

        // teste Schreibalternativen
        if (chkAlt && !found) {
            if (corr == 255) {
                corr = 0;
                // Temp-Speicher
                wcsncpy(tmp, buf, l);

                wchar_t* f;
                uint d;
```

```
                    // ß -> ss
                    if ((d = (f = wcschr(buf, L'ß')) - buf) < l) {
                        wcsncpy(f + 1, tmp + d, l - d);
                        *f   = L's';
                        f[1] = L's';
                        corr--;
                        l++;
                    }
                    // h ->
                    else if ((d = (f = wcschr(buf, L'h')) - buf) < l) {
                        wcsncpy(f, f + 1, l - d + 1);
                        corr++;
                        l--;
                    }
                    // dt -> t
                    else if (
                        (wcschr(buf, L't') - buf < l) &&
                        (d = (f = wcschr(buf, L'd')) - buf) < l) {
                        wcsncpy(f, f + 1, l - d);
                        corr++;
                        l--;
                    }

                    goto search;
                } else {
                    // Rückkopieren falls erfolglos
                    l += corr;
                    wcsncpy(buf, tmp, l);
                }
            }

            if (found == 1) // eine Übereinstimmung -> grün
                wprintf(L"\033[0;32m%.*ls\033[0;37m", l, match);
            else if (!found) // keine Übereinstimmung -> rot
                wprintf(L"\033[0;31m%.*ls\033[0;37m", l, buf);
            else // mehrere Übereinstimmungen -> gelb
                wprintf(L"\033[0;33m%.*ls\033[0;37m", l, match);
        }
    }

    wprintf(L"\n");
    fclose(fp);
    if (untwist) freeWordMap();
    return 0;
}
```