Junior 2: Baywatch

Symbroson
Team-ID: 00165

10. September 2018

Inhaltsverzeichnis

1 Lösungsidee

Bei jedem Wort sind Anfangsbuchstabe, Endbuchstabe und die Buchstabenanzahl gegeben. Man kann also durch diese Eigenschaften 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. Da die längsten Wörter 34 Buchstaben lang sind gehe ich großzügig von einer maximalen Wortlänge von 40 aus.

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.

2.2 Wortentdrehung

2.2.1 Findung

Die Eingabedatei wird zeilenweise eingelesen und behandelt. Für jede Zeile werden dann zunächst alle Sonderzeichen und Wörter bestehend aus registrierten Zeichen (Alphabet und Umlaute äöüß) extrahiert. Sonderzeichen werden sofort ausgegeben und Wörter werden zunächst auf die Länge geprüft. Ist es kürzer als 4 Zeichen wird es ebenfalls sofort ausgegeben da es keine vertauschten Buchstaben geben kann, ist es zu lang, kann es nicht in der Wortliste verzeichnet worden sein.

2.2.2 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.

2.2.3 Zeichenauszählung

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

2.2.4 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 das Wort erst im nächsten durchlauf, wenn ein weiteres mögliches Wort gefunden wurde, oder nach der Suche ausgegeben wird. Des weiteren kann man dies dafür nutzen um versehentliche Dopplungen in der Wortliste (die leider vorkamen) auf einfachem Wege zumindest teilweise zu überspringen.

2.2.5 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 wird das Wort getrennt von einem '|' gelb ausgegeben, und falls es gar keine Übereinstimmung gab, wird das Eingabewort unverändert in rot ausgegeben.

3 Beispiele

3.1 twist1.txt | enttwist.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.

3.2 twist2.txt

Hier ist 'wegbegeben' das einzige nicht gefundene Wort. Man müsste dem Programm beibringen Wörter zu konjugieren, zu steigern, die Mehrzahl zu bilden etc. - 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.

3.3 twist3.txt

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

3.4 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.

3.5 twist5.txt

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

4 Quellcode

```
// Includes
/* ... */

using namespace std;

#define error(a, ...) \
    fprintf(stderr, "\033[0;33m" a "\033[0;37m\n", ##__VA_ARGS__)

#define MAXLEN 40       // maximale Wortlänge
#define CHRCNT (26 + 5) // = lowercase + äöüß + 1 for others

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

bool tryOpen(const char* name, FILE*& fp) {
    /* ... */
}

// gibt Zeichennummer zurück (nicht Zeichencode)
uint8_t charNum(char* cp) {
    static char lc;

    if (*cp >= 'A' && *cp <= 'Z') return *cp - 'A';
    if (*cp >= 'a' && *cp <= 'z') return *cp - 'a';

    uint8_t sub = 0;            //  Ergebnis-Subtrahent
    bool isSpec = *cp == *"ü"; // Status: ist Umlaut

    // Umlaute sind nicht in ASCII enthalten, stattdessen bestehen sie aus zwei
    // ASCII-Zeichen, wobei der erste (hier) bei alles gleich ist
    if (isSpec || lc == *"ü") {
        /* ... */
    }
    return CHRCNT - sub;
}

// zählt Umlaute
uint16_t cntUml(char* word, uint16_t len = -1) {
    /* ... */
}

void freeWordMap() {
    /* ... */
}

bool initWordMap() {
    FILE* fp;
    char* line = NULL;
    ssize_t read;
```

```
    size_t len = 0;

    if (tryOpen("res/woerterliste.txt", fp)) return true;
    while ((read = getline(&line, &len, fp)) != -1) {
        if (read > 1) {
            if (line[read - 1] == '\n') read--;
            if (read > MAXLEN) error("word %.*s too long", (int)read, line);
            wordMap[read - cntUml(line, read)][charNum(line)].push_front(
                strndup(line, read));
        }
    }
    fclose(fp);
    return false;
}

int main(int argc, const char* argv[]) {
    FILE* fp   = NULL;
    char* line = NULL;
    size_t len = 0;
    ssize_t read;
    bool chkAlt = false;

    // Argumente einlesen
    /* ... */

    if (initWordMap()) return 1;

    if (fp == NULL && tryOpen("res/enttwist.txt", fp)) {
        error("initialization failed");
        fclose(fp);
        freeWordMap();
        return 1;
    }

    // lese Eingabedatei zeilenweise
    while ((read = getline(&line, &len, fp)) != -1) {
        if (read > 1) {
            char *c = line, *b;
            uint16_t l, corr;

            do {
                // lese nicht-Wörter
                l = 0;
                b = c;
                while (*c && charNum(c) == CHRCNT) c++, l++;
                printf("%.*s", l, b);

                // lese Wörter
                l = 0;
                b = c;
                while (*c && charNum(c) != CHRCNT) c++, l++;

                // Keine vertauschten Bst. möglich
                if (l < 3 || l - cntUml(b, l) <= 3) {
                    printf("\033[0;32m%.*s\033[0;37m", l, b);
                    continue;

                    // Wort zu lang
                } else if (l > MAXLEN) {
                    printf("\033[0;31m%.*s\033[0;37m", l, b);
                    continue;
                }

                char p[l], t[l], *match = NULL;
                uint8_t i, j, found     = 0;
                corr = 0;

            search:
                // suche Basiswörter
                for (char* word: wordMap[l - cntUml(b, l)][charNum(b)]) {
                    if ((b[l - 1] != word[l - 1]) || // Endbst. verschieden
                        ((b[0] != word[0]) &&        // Anfangsbst. verschieden
                         (b[0] != word[0] - 'a' + 'A')) // klein->groß
```

```
                    )
                        continue;

                    // kopiere Eingabewort zu p für Buchstabenzählung
                    strncpy(p, b, 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 (!strncmp(match + 1, word + 1, l - 1)) continue;

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

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

                // teste Schreibalternativen
                if (chkAlt && !found) {
                    /* ... */
                }

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

            } while (*c);
        } else
            printf("\n");
    }

    printf("\n");
    fclose(fp);
    free(line);
    freeWordMap();
    return 0;
}
```