Dokumentation:
Häufigkeit der Folgerelationen verschiedener Wörter

Fabian Hundertmark Matrikel-Nummer: 1769284

20. August 2007

Inhaltsverzeichnis

1 Fragestellung

Die Fragestellung, zu dem mein Programm die Antwort darstellt ist folgende:
„Wie kann man feststellen, welche Wörter in welcher Häufigkeit nach oder vor anderen Wörtern

auftauchen?“ In dieser Dokumentation werde ich zuerst aufzeigen wie ich diese Aufgabenstellung verstehe, dann werde ich darstellen, wie ich das Problem in Teilprobleme unterteilt habe. Es folgen die Darstellung einer Subroutine, zwei Testdurchläufe und zum Schluss werde ich kurz Selbstkritik üben und zeigen, was an meinem Programm besser gemacht werden kann.

1.1 Die dem Programm zugrunde liegende Definition von Wörtern

Die Definiton von Wörtern, die dem vorliegenden Programm zugrunde liegt ist folgende: Ein Wort ist eine Buchstabengruppe zwischen zwei Trennzeichen (bspw. Leerzeichen).[1]

Einzelne Buchstaben Ein Wort kann auch aus nur einem Zeichen bestehen, da sonst Wörter wie das englische „i“ (engl. für „ich“) nicht erfasst werden.

Komposita Es wird also kein Unterschied zwischen Komposita und anderen Wörtern gemacht. „Hausboot“ ist also genauso ein Wort, wie „Haus“ oder „Boot“.

Aphabetische und numerische Zeichen Zudem werde ich Zahlenfolgen (Beispielsweise „007“) oder Kombinationen aus Buchstaben und numerischen Zeichen (Beispielsweise „JamesBond007“) auch wie Wörter behandeln.

Worttrennung am Zeilenende Wörter die am Zeilenende getrennt werden, sollen zusammengesetzt werden. Beispielsweise wird aus
„Fahr-
rad“
das Wort „Fahrrad“.

1.2 Wie ich Satzzeichen behandle

Satzendzeichen Da das fertige Programm nicht nur einzelne Sätze, sondern ganze Texte analysieren soll, war zu entscheiden wie, ich mit diesen Satzenden umgehe. Das vorliegende Programm teilt den Text in Sätze ein, bevor er diese analysiert.[2] Beispiel:
„Dies ist ein Satz. Hier ist der nächste.“
Die Analyse sollte in diesem Fall das Wort „Satz“ und das Wort „Hier“ nicht als aufeinander folgend zählen. Daher werde ich bei Satzendzeichen wie Wörter behandeln. Es sollte also auf das Wort „Satz“ ein Punkt folgen. Vor dem „Hier“ steht hingegen weder ein Wort noch ein Zeichen. Als Satzendzeichen zählen: Punkte, Ausrufezeichen, Fragzeichen und Doppelpunkte.

[1] Damit entspricht die Definiton ziemlich genau der, die unter http://de.wikipedia.org/wiki/Wort (Stand: 16. November 2006, 01:48) als das ortographische Kriterium für Worte bezeichnet wird.

[2] Als „Analyse“ bezeichne ich in dieser Dokumentation die Untersuchung eines Textes. Genauer gesagt: Eine Untersuchung, die für jeden Worttype zählt, wie häufig dieser im Text vorkommt und wie häufig welcher andere Worttype im Text vor bzw. nach diesem steht.

Runde Klammern, Semikolons, Kommatas und Gedankenstriche werden auch wie Wörter behandelt. Die Häufigkeit mit der diese vor oder nach anderen Wörtern vorkommen wird also auch erfasst.

Sonstige Zeichen wie Beispielsweise mathematische Zeichen („+“, „=“ etc.) und eckige Klammern werden komplett ignoriert. So folgt in dem Beispiel „1 + 2 = 3“ die „2“ direkt auf die „1“ und die „3“ auf die „2“.

2 Aufteilung in Teilprobleme

2.1 Grobe Einteilung

Um das Problem zu lösen, braucht man drei Schritte:

1. Eingabe des zu analysierenden Textes
2. Analyse und Speicherung
3. Anzeige der Analyse

Diese Schritte sind in drei Perl-Dateien umgesetzt:

1. index.pl
2. worte.pl
3. show.pl

Hinzu kommt die Datei delete.pl, mit der sich nicht mehr gebrauchte Analysedateien löschen lassen und Gesamt.pm, in der sich einige Funktionen[3] befinden, welche von den anderen Dateien verwendet werden.

2.2 Die einzelnen Perl-Scripte und ihre Funktionen

2.2.1 index.pl

Die index.pl dient als Programmstart. Hier bietet sich die Möglichkeit einen neuen Text unter Angabe eines Titels zu analysieren, eine fertige Analyse anzuschauen oder eine fertige Analyse zu löschen.

2.2.2 worte.pl

Die Datei worte.pl dient der Analyse von Texten und dem Abspeichern dieser Analyse. Gibt man in index.pl also einen Text und einen Titel ein und klickt auf „Analyse Starten“, so wird worte.pl gestartet und sowohl Text als auch Titel werden übergeben.

[3] „Funktion“ verwende ich gleichbedeutend mit „Subroutine“

Die Funktion „Analysieren“ ist die Hauptfunktion der worte.pl. Hier wird der größte Teil der Analyse durchgeführt. Zudem werden die Funktionen „print_text“ und „Ausgabe“ gestartet.

Die Funktion „print_text“ speichert die Analyse unter dem vorgegebenen Dateinamen.

Die Funktion „Ausgabe“ dient dem Anzeigen eines Benachrichtigungsbildschirms, wenn die Analyse abgeschlossen ist. Von hier aus lassen sich alle Wörter des Textes nach Alphabet oder nach Häufigkeit sortiert anzeigen.

Die Funktion „german2ascii“ ersetzt in einem Wort die deutschen Sonderzeichen durch ein ASCII-kompatibles Äquivalent. So wird Beispielsweise aus einem „ä“ ein „ae“.

2.2.3 show.pl

Die Datei show.pl dient der Anzeige von fertigen Textanalysen.

Die Funktion „go“ wird automatisch beim Aufruf der show.pl geladen. Hier wird vor allem die Analysedatei geladen und geprüft ob alle Wörter mit ihrer Anzahl gezeigt oder ob nur ein einziges Wort mit den jeweils folgenden und vorhergehenden Wörtern betrachtet werden soll.

Die Funktion „All“ generiert in dem Fall, dass alle Wörter angezeigt werden sollen die ausgegebene Html-Datei. Allerdings wird die Erstellung der Tabelle entweder der Funktion „Alpha“ oder der Funktion „Quanti“ übergeben, je nachdem, ob die Wörter nach Alphabet oder nach Quantität angezeigt werden sollen.

Die Funktion „SpecWort“ generiert in dem Fall, dass nur ein Wort angezeigt werden soll die ausgegebene Html-Datei. Allerdings werden bei der Erstellung der Tabellen für die folgenden bzw. vorhergehenden Wörter die Funktionen „Alpha“ oder „Quanti“ benutzt, je nachdem, ob die Wörter nach Alphabet oder nach Quantität angezeigt werden sollen.

Die Funktion „Alpha“ dient alleine dem Zweck aus einem eingegebenen Datensatz eine alphabetisch sortierte Tabelle im Html-Format zu generieren.

Die Funktion „Quanti“ dient dem Zweck aus einem eingegebenen Datensatz eine quantitativ sortierte Tabelle im Html-Format zu generieren.

2.2.4 delete.pl

Die Datei delete.pl dient dem Löschen einer bereits angelegten Analysedatei. Zudem gibt sie eine Html-Seite aus, die den Erfolg des Löschens anzeigt.

2.2.5 Gesamt.pm

Die Datei Gesamt.pm ist eine Sammlung von Funktionen, die in den anderen Perl-Scripts gebraucht werden.

Die Funktion „Stil“ wird von jeder im Programm benutzten Html-Code generierenden Funktion eingesetzt um das Design des Programms festzulegen. Um dies zu erreichen benutzt sie CSS.

Die Funktion „DateiFehlt“ fängt den Fehler ab, dass eine angeforderte Analysedatei nicht vorhanden ist. Sie bietet die Möglichkeit zurück auf den Startbildschirm zu gehen oder eine andere Analysedatei anzuzeigen.

Die Funktion „NeuAnalyse“ generiert einen Teil einer Html-Seite, welcher ein Textfenster zum eingeben eines neuen Analysetextes, ein Textfeld zum Eingeben eines Titels und einen Button zum starten der Analyse bietet.

Die Funktion „WortFehlt“ fängt den Fehler ab, dass ein angefordertes einzelnes Wort nicht vorhanden ist. Sie generiert Html-Code, durch welchen sich die Möglichkeit bietet, zurück auf den Startbildschirm zu gehen, eine andere Analysedatei bzw. alle Wörter alphabetisch oder quantitativ sortiert anzuzeigen.

Die Funktion „AnalyseDateienAnzeigen“ generiert einen Teil einer Html-Seite. Dieser Teil enthält ein Dropdown-Menü, welches die fertigen Analysedateien enthält und einen Button, mit dem man sich die ausgewählte Datei anzeigen lassen kann.

Die Funktion „LoeschDateienAnzeigen“ generiert einen Teil einer Html-Seite. Dieser Teil enthält ein Dropdown-Menü, welches die fertigen Analysedateien enthält und einen Button, mit dem man die ausgewählte Datei löschen lassen kann.

Die Funktion „ZurueckZurHauptseite“ generiert einen Link als Teil einer Html-Seite, welcher einen auf die Startseite (index.pl) zurückschickt.

3 Die Subroutine „Analysieren“ zur Veranschaulichung

Zur Veranschaulichung werde ich die wichtigste Subroutine „Analysieren“ im folgenden darstellen. „Analysieren“ ist die Hauptfunktion des „worte.pl“-Scripts.

3.1 Der Quelltext

Listing 1: Die Funktion „Analysieren“. Kommentare aus dem Originalquelltext wurden entfernt.

```
1  sub Analysieren{
2   my ($Titel, $Text) = @_;
3   my @Woerter;
4   my %Datensatze = ();
5   my $IDWort;
6
7   if($Titel eq ""){
8    DateiFehlt();
9    return;
10  }
11
12  #----------------------
13  $Text =~ s/-\s*\n\s*//g;
14  $Text =~ s/\n/ /g;
15  $Text =~ s/[<>»«="+~#']/␣/g;
16 ␣$Text␣=~␣s/\s\s/␣/g;
17 ␣$Text␣=~␣s/\./\.\n/g;
18 ␣$Text␣=~␣s/\!/\!\n/g;
19 ␣$Text␣=~␣s/\?/\?\n/g;
20 ␣$Text␣=~␣s/:/:\n/g;
21
22 ␣do{
23 ␣$Text␣=~␣s/\n␣/\n/g;
24 ␣}␣while␣($Text␣=~␣/\n␣/);
25
26 ␣$Text␣=~␣s/,/␣,/g;
27 ␣$Text␣=~␣s/;/␣<,,>/g;
28 ␣$Text␣=~␣s/\./␣\./g;
29 ␣$Text␣=~␣s/!/␣!/g;
30 ␣$Text␣=~␣s/\?/␣\?/g;
31 ␣$Text␣=~␣s/\(/\(␣/g;
32 ␣$Text␣=~␣s/\)/␣\)/g;
```

```
33 ␣$Text␣=~␣s/␣␣/␣/g;
34
35 ␣@Woerter␣=␣split␣("[\n ]",␣$Text);
36
37 ␣foreach␣my␣$Wort␣(@Woerter)
38 ␣{
39 ␣␣$IDWort␣=␣german2ascii(lc($Wort));
40 ␣␣$Datensatze{$IDWort}->{HowMany}␣+=␣1
41 ␣␣if␣(!($Wort␣=~␣/^[ÄÖÜA–Z]/)␣or
42 ␣␣␣(␣$Datensatze{$IDWort}->{HowMany}␣==␣1))
43 ␣␣␣{
44 ␣␣␣$Datensatze{$IDWort}->{OWort}␣=␣$Wort;
45 ␣␣␣}
46 ␣}
47 ␣#----------------------
48 ␣my␣@Saetze;
49 ␣my␣@TextStruk;
50
51 ␣$Text␣=␣lc(␣german2ascii␣($Text));
52
53 ␣@Saetze␣=␣split("\n",␣$Text);
54
55 ␣my␣$SN␣=␣0;
56 ␣while␣($SN␣<␣@Saetze)
57 ␣{
58 ␣␣my␣@test␣=split("␣",␣$Saetze[$SN]);
59 ␣␣$TextStruk[$SN]␣=␣\@test;
60 ␣␣$SN␣+=␣1;
61 ␣}
62
63 ␣foreach␣my␣$AktSatz␣(␣@TextStruk␣)
64 ␣{
65 ␣␣my␣$AktWort␣=␣0;
66 ␣␣while␣($AktWort␣<␣@$AktSatz)
67 ␣␣{
68 ␣␣␣if␣($AktWort␣!=␣0)
69 ␣␣␣{
70 ␣␣␣␣$Datensatze{@$AktSatz[$AktWort]}->{'davor'}->
71 ␣␣␣␣␣{@$AktSatz[$AktWort␣-␣1]}␣+=␣1;
72 ␣␣␣}
```

```
73
74  ␣␣␣if␣($AktWort␣<␣@$AktSatz␣-␣1)
75  ␣␣␣{
76  ␣␣␣␣$Datensatze{@$AktSatz[$AktWort]}->{'danach'}->
77  ␣␣␣␣{@$AktSatz[$AktWort␣+␣1]}␣+=␣1;
78  ␣␣␣}
79  ␣␣␣$AktWort++;
80  ␣␣}
81  ␣}
82
83  ␣#----------------------
84  ␣my␣$Datei;
85  ␣use␣Data::Dumper;
86  ␣$Data::Dumper::Purity␣=␣1;
87  ␣$Data::Dumper::Varname␣=␣"";
88  ␣$Datei␣=␣Dumper(%Datensatze);
89
90  ␣$Datei␣=~␣s/\$[0-9]*␣=//g;
91  ␣$Datei␣=~␣s/\'\;\n/'␣=>/g;
92  ␣$Datei␣=~␣s/\;/,/g;
93  ␣$Datei␣=~␣s/://g;
94  ␣$Datei␣=~␣s/\s//g;
95  ␣#----------------------
96  ␣print_text(␣"$Titel.aly",␣$Datei);
97  ␣Ausgabe($Titel);
98  }
```

3.2 Die Erläuterung des Quelltextes

In den Zeilen 2-5 werden einige Variablen deklariert und zum Teil werden diesen schon Werte zugewiesen. So erhalten die Variablen $Titel und $Text die Werte, welche beim Aufruf der Subroutine übergeben werden. Es sollte sich bei $Titel um einen kurzen String handeln, aus dem sich der Name ergibt unter dem die Analysedatei gespeichert wird. (Beispielsweise „Tröte“) Der Variable $Text hingegen sollte ein längerer String zugewiesen werden. (Beispielsweise „Die Tröte trötet.“ Dieser wird im folgenden darauf untersucht, welche Wörter mit welcher Häufigkeit vor oder nach welchen anderen Wörtern auftauchen. Das Array @Woerter soll im Folgenden die einzelnen Worttokens des Text enhalten. Der Hash %Datensätze soll später eine Datenstruktur enthalten, welche am Ende den kompletten Inhalt der Analysedatei wiedergibt. Die Variable $IDWort wird in Schleifen benutzt, um dort das aktuell behandelte Wort in vereinfachter Form zu speichern.

Die Zeilen 7-10 testen, ob ein Titel für die aktuelle Analyse eingegeben wurde. Ist dies nicht der Fall wird eine Fehlermeldung zurückgegeben und die die Subroutine beendet.

In den Zeilen 13-33 werden verschiedene Vorbereitungen im zu analysierenden Text getroffen. So werden getrennte Wörter am Zeilenende zusammengefasst, Zeilenumbrüche, Sonderzeichen und überflüssige Leerzeichen entfernt. Zudem werden Zeilenumbrüche am Satzende und Leerzeichen vor oder nach bestimmten Zeichen eingefügt. Da keine Semikolons in der Analysedatei stehen dürfen, da es sonst zu Fehlern kommt, wird aus jedes Semikolon durch „{,,}“ ersetzt. Diese Änderung wird vor dem Anzeigen in show.pl wieder rückgängig gemacht, sodass das fertige Programm Semikolons anzeigt.

In Zeile 35 wird der so bearbeitete Text „zerschnitten“ und einzelnen Worttokens werden in @Woerter gespeichert.

In den Zeilen 37-46 wird für jedes Worttoken mit Hilfe der Subroutine german2ascii eine ASCII-kompatible Version des Wortes erstellt. Diese enthält keine deutschen Sonderzeichen und nur Kleinbuchstaben.[4] Nun wird die Anzahl dieses Worts in dem Hash %Datensaetze um 1 erhöht. Zuletzt wird getestet, ob dieses Wort neu ist oder ob es in der Originalversion Großbuchstaben enthält. Ist dies der Fall, wird dieses Originalwort unter „OWort“ in den Hash %Datensaetze eingetragen.

In den Zeilen 48 und 49 werden zwei Arrays deklariert. So soll das Array @Saetze die einzelnen Saetze des Textes enthalten. Das Array @TextStruk soll später Zeiger auf einzelne Arrays speichern, die in den einzelnen Elementen die Wörter eines Satzes enthalten.

In Zeile 51 wird der gesamte Text so umgewandelt, dass er nur ASCII-kompatible Kleinbuchstaben enthält.

In Zeile 53 werden die einzelnen Sätze jeweils als ein Element in @Saetze gespeichert.

In den Zeilen 55 bis 61 befindet sich eine Schleife. Diese geht jeden einzelnen Satz in @Saetze durch, zerschneidet diesen in einzelne Wörter und speichert einen Zeiger auf das so gewonnene Array in @TextStruk.

Die Zeilen 63-81 enthalten eine foreach-Schleife, welche eine Unterschleife enthält, welche wiederum zwei bedingte Anweisungen enthält. Erstere Schleife geht die einzelnen, in @TextStruk enthaltenen Sätze durch. Die Unterschleife geht nun die einzelnen Worttokens dieser Sätze durch. Ist das aktuelle Wort nicht das erste Wort des Satzes, wird im Hash %Datensaetze das der Zähler des vorhergehendes Wortes im Unterpunkt „davor“ um eins erhöht. Ist das aktuelle Wort nicht das

[4] So wird beispielsweise aus dem Wort „Tröte“ das Wort „troete“.

letzte Wort des Satzes, wird im der Zähler des folgenden Wortes im Unterpunkt „danach“ um eins erhöht. Damit ist die Analyse des Textes beendet. Die gewonnene Datenstruktur sieht nun so aus:

$$\begin{bmatrix} & \begin{bmatrix} OWort & Wort_X \\ HowMany & Anzahl \\ & \begin{bmatrix} wort_1 & Anzahl \\ ... & ... \\ wort_n & Anzahl \end{bmatrix} \\ davor & \\ & \begin{bmatrix} wort_1 & Anzahl \\ ... & ... \\ wort_n & Anzahl \end{bmatrix} \\ danach & \end{bmatrix} \\ wort_x & \\ & \begin{bmatrix} OWort & Wort_Y \\ HowMany & Anzahl \\ & \begin{bmatrix} wort_1 & Anzahl \\ ... & ... \\ wort_n & Anzahl \end{bmatrix} \\ davor & \\ & \begin{bmatrix} wort_1 & Anzahl \\ ... & ... \\ wort_n & Anzahl \end{bmatrix} \\ danach & \end{bmatrix} \\ wort_y & \\ ... & \end{bmatrix} \tag{1}$$

Um dieses zu veranschaulichen, hier ein Beispiel:

$$\begin{bmatrix} & \begin{bmatrix} OWort & \text{Tröte} \\ HowMany & 1 \\ davor & \begin{bmatrix} die & 1 \end{bmatrix} \\ danach & \begin{bmatrix} troetet & 1 \end{bmatrix} \end{bmatrix} \\ troete & \end{bmatrix} \tag{2}$$

In den Zeilen 84-88 wird die externe Dumper-Funktion verwendet um aus den Hash %Datensaetze einen String zu machen, der nun in der neu erstellten Variable $Datei zwischengelagert wird. Unser Beispiel sieht nun so aus:

Listing 2: Beispiel

```
1 ...
2 $5 = 'troete';
3 $6 = {
4         'danach' => {
5                       'troetet' => 1
6                     },
7         'HowMany' => 1,
8         'davor' => {
```

```
9                         'die' => 1
10                      },
11          'OWort' => 'Tröte'
12        };
13 ...
```

In den Zeilen 90-94 werden aus diesem String Variablenbezeichnungen (im Beispiel „$5 und $6“, Doppelpunkte und Leerzeichen entfernt. Semikolons am Zeilenende werden in Pfeile, andere Semikolons werden in Kommatas umgewandelt. Sodass unser Beispiel nun wie folgt aussieht:

Listing 3: Ausschnitt aus der fertigen Analysedatei. Aus Darstellungsgründen in zwei Zeilen.

```
1           ... 'troete'=>{'danach'=>{'troetet'=>1},'HowMany'=>1,
2           'davor'=>{'die'=>1},'OWort'=>'Tröte'} ,...
```

Zuletzt wird in den Zeilen 96 und 97 der so erzeugte String und der Titel der Subroutine „print_text“ übergeben, welcher den String nun unter dem Titel und der Dateinamenerweiterung „.aly“ abspeichert. Zudem wird die Subroutine „Ausgabe“ aufgerufen, welche dem Benutzer mitteilt, dass die Analyse erfolgreich war.

4 Zwei Testdurchläufe

In diesem Abschnitt werde ich zwei Testdurchläufe durchführen um zu zeigen, wie das Programm funktioniert. Dabei werde ich zunächst einen kurzen Text analysieren lassen um die Bedienung klar zu machen. Dann folgt – kürzer gefasst – die Analyse eines längeren Textes.

4.1 Kurzer Text

Der zu analysierende Text sei: „Die Tröte trötet. Dieses Tröten weckt Johanna, die gerade in ihrem Bett liegt.“ Ich setze voraus, dass die fünf notwendigen Dateien (siehe oben) in einem Ordner sind, in dem der bereits gestartete Webserver auf sie zugreifen kann. Nun öffne man mit dem Internetbrowser die index.pl.[5] Es sollte nun eine Seite mit einem großen Textfenster, einem kleinen Eingabefeld, drei Buttons und zwei Dropdownmenüs angezeigt werden. Dies ist die Hauptseite das Programms. Will man nun einen Text analysieren befolgt man folgende Schritte:

1. Man gebe den zu analysierenden Text in das große Textfenster ein.
2. Man gebe dem Text einen Titel und gebe ihn in das entsprechende Feld ein.
3. Man klicke auf „Analyse Starten“.

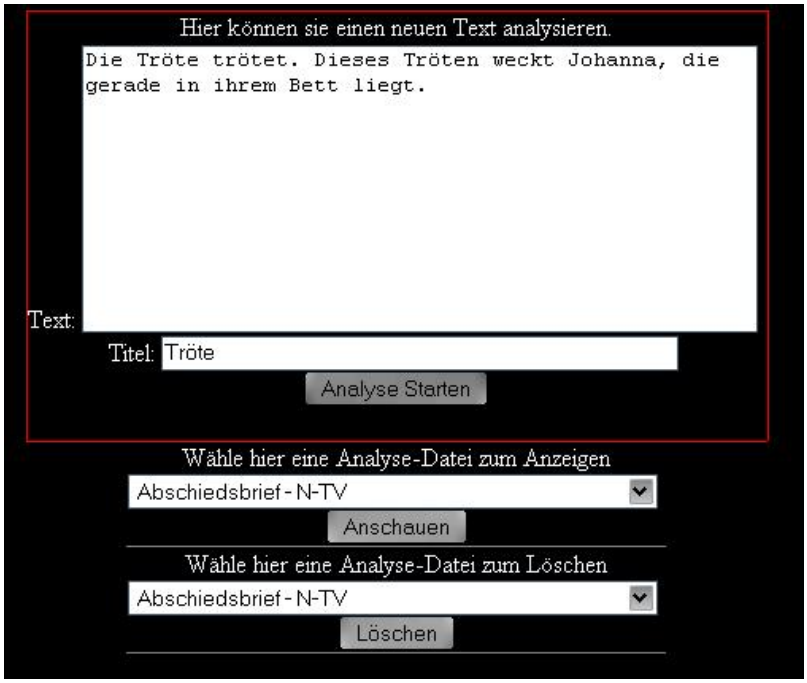

Abbildung 1: Bevor man auf „Analyse Starten“ klickt, müsste es so aussehen.

Nun sollte eine schlichte Seite folgen, welche einem drei Möglichkeiten bietet:

[5] Bei mir beispielsweise unter „http://localhost/cgi-bin/perl/index.pl“

- „Alle Wörter - nach Alphabet geordnet anzeigen“
- „Alle Wörter - nach Anzahl geordnet anzeigen“
- „Zurück zur Startseite“

Wählt man nun die Möglichkeit, sich alle Wörter nach Anzahl sortiert anzuschauen, bietet sich einem folgendes Bild: Oben lässt sich eine andere Analysedatei auswählen. Tut man dies, bleiben alle

Abbildung 2: Alle Wörter - nach Anzahl geordnet

anderen Optionen wie die Ordnung der Tabelle oder später auch das ausgewählte Wort gleich. Unter der Möglichkeit eine andere Analysedatei anzuzeigen findet sich die Tabelle, welche alle im Text enthaltenen Worttokens nebst Anzahl anzeigt. Klickt man eine der beiden Tabellenüberschriften an, lässt sich die Tabelle anders sortieren. Klickt man hingegen auf ein Wort, öffnet sich eine neue Seite, welche etwa wie folgt aussieht: Oben finden sich hier bereits bekannte Elemente. In der Mitte steht das Wort, welches angeklickt wurde und zu dem nun die Details angezeigt werden – in diesem Fall „Tröte“. Noch zwei Zeilen weiter dadrunter kann man ablesen, wie oft dieses Wort im Text gefunden wurde – hier ein mal. Links und rechts bedinden sich zwei Tabellen, welche die folgenden und vorhergehenden Wörter anzeigen. So lässt sich beispielsweise aus der linken Tabelle ablesen, dass das Wort „die“ ein Mal vor dem Wort „Tröte“ vorkommt. In der rechten Tabelle kann man sehen, dass „trötet“ einmal im Text auf das Wort „Tröte“ folgt. Auch diese Tabellen lassen sich mit einem Klick auf die Tabellenüberschriften umsortieren.

Zum Löschen einer bereits erstellten Analysedatei geht man auf die Startseite (index.pl). Dort wählt man unten [6] den Titel der zu löschenden Analyse aus und klickt auf „Löschen“. Nun wird man auf eine Seite geleitet, die den Erfolg des Löschvorgangs bestätigt.

[6] unter der Überschrift „Wähle hier eine Analyse-Datei zum Löschen“

Abbildung 3: Hier beispielsweise das Wort „Tröte“

4.2 Langer Text

Der lange Text sei hier nicht komplett dargestellt. Es handelt sich aber um die „Untersuchung in Betreff des menschlichen Verstandes“ von David Hume. Dieses Werk enthält, schenkt man „Wörter zählen“ in Microsoft Word Glauben. 50 533 Wörter, was mit Leerzeichen 341 686 Zeichen macht. Gibt man diesen Text nun in mein Programm ein, stellt man fest, dass die Analyse mit Speichervorgang etwa eine Minute dauert. Jede Anzeige eines Wortes oder auch aller Wörter dauert ca. 15 Sekunden. Dies ist zwar keine Zeit, die ein schnelles herumklicken im Programm ermöglicht, jedoch ist sie ganz akzeptabel. Die Analysezeit sollte etwa proportional zur Menge der Worttokens, die Anzeigezeit proportional zur Anzahl der Worttypes steigen.

5 Selbstkritik

Ich habe zwei Kritikpunkte an meinem eigenen Programm, die ich kurz darstellen werde:

Geschwindigkeit Die eben dargestellte Zeit, die mein Programm zur Analyse und Anzeige braucht, ist gerade für große Korpora – welche natürlich interessantere Ergebnisse liefern würden – unpraktisch. Ist eine solche Nutzung also gewollt, wäre es möglicherweise sinnvoll die Analyse in einer Datenbank zu speichern, da diese deutlich schnellere Zugriffszeiten bietet, als die hier verwendete Speicherung als Datei auf der Festplatte.

Eine klarere Strukturierung Zwischen Programmierung und Dokumentation meines Programms lag eine relativ lange Zeit. Dies hatte zur Folge, dass ich mich neu in den Quelltext einfinden musste. Hierbei habe ich gemerkt, dass meine Kommentare im Quelltext zwar nützlich waren, dass aber eine klarere Strukturierung der Subroutinen und eine endeutigere Bennennung dieser (Als Beispiel die Subroutine „Go“) und der Variablen zu mehr Übersicht führen würde.