Bundeswettbewerb Informatik 2. Runde: Aufgabe 3

1 Lösungsidee

Zunächst werden alle Anagramme erzeugt, die aus dem eingegebenen Wort gebildet werden können. Danach werden Texte untersucht und (anhand der Ergebnisse) nicht sinnvolle Anagramme entfernt. Die übrigen Anagramme sind dann mit großer Wahrscheinlichkeit sinnvoll und werden ausgegeben.

1.1 Multimengen

Wörter werden als Multimengen aufgefasst, in denen Elemente (Buchstaben) auch mehrfach vorkommen können.

Die Grundmenge des Wortes „hello“ ist gegeben durch:

$$hello = \{h, e, l, o\}$$

Die Darstellung des Wortes als Multimenge ist jedoch:

$$hello = \{h, e, l, l, o\}$$
$$mit\ |h| = 1, |e| = 1, |l| = 2, |o| = 1$$

1.2 Mengenoperatoren

Da ich Wörter als Multimengen definiert habe, muss ich auch eigene Mengenoperatoren definieren.

Zwei Mengen A und B sind gleich, wenn gilt:

$$A = B \rightarrow a \in A \land b \in B \land a = b \land |a| = |b|$$

Ein Beispiel dazu ist:

$$their = it\ her$$
$$\{t, h, e, i, r\} = \{i, t, h, e, r\}$$

Dementsprechend müssen bei der Teilmengenoperation alle Elemente der einen Menge auch in der zweiten Menge sein, jedoch kann die zweite Menge noch mehr Elemente enthalten:

$$tea \subseteq beauty$$
$$\{t, e, a\} \subseteq \{b, e, a, u, t, y\}$$

Bei der Mengendifferenz werden alle Elemente der einen Menge aus der anderen Menge „gestrichen“. Die übrigen Elemente sind dann Elemente der Differenzmenge:

$$\{b, e, l, l, m, a, n\} \setminus \{a, l, l\} = \{b, e, \cancel{l}, \cancel{l}, m, \cancel{a}, n\}$$
$$bellman \setminus all = bemn$$

1.3 Anagramm-Findung

Die Anagramme werden mithilfe eines Wörterbuches gefunden.

Zunächst wird eine Liste von Wörtern zusammengestellt, die in dem Anagramm vorkommen *können*. Dafür werden Wörter einzeln aus dem Wörterbuch gelesen und daraufhin überprüft, ob sie Teilmenge des eingegebenen Wortes sind. Ist das Wort aus dem Wörterbuch Teilmenge von dem eingegebenen Wort, so wird es in die Liste der möglichen Wörter aufgenommen. Dazu einige Beispiele:

$$\begin{aligned} the &\subseteq their \\ eval &\subseteq leave \\ tomb &\subseteq buttom \end{aligned}$$

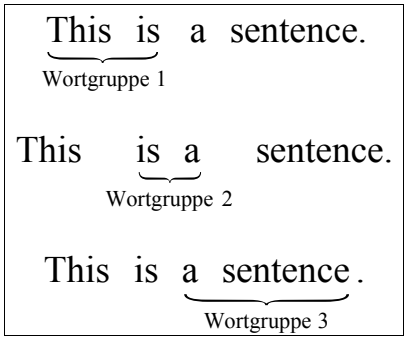
Mit der so gewonnenen Liste werden dann Anagramme gesucht:

Das Finden von Anagrammen geschieht rekursiv. Es werden alle Möglichkeiten durchlaufen, wie man die Wörter aus der oben beschriebenen Liste zusammenfügen kann. Die zusammengefügten Wörter werden auf Mengengleichheit mit dem eingegebenen Wort untersucht. Wenn die Differenzmenge der beiden Wörter die leere Menge ist, so ist ein Anagramm gefunden worden:

$$\begin{aligned} words \setminus sword &= \emptyset \\ snake \setminus sneak &= \emptyset \\ an\ example \setminus expel\ a\ man &= \emptyset \end{aligned}$$

1.4 Bildung sinnvoller Anagramme

Die Texte werden nach Wortgruppen durchsucht, die jeweils aus zwei hintereinanderfolgenden Wörtern bestehen. Kommen die Wörter in der Liste der möglichen Wörter vor, so wird sich gemerkt, welches Wort auf Welchem folgte. Das erste Wort der nächsten Wortgruppe ist dann das zweite Wort der vorherigen Wortgruppe:



Sind die Texte durchsucht worden, kann man nun die nicht sinnvollen Anagramme aus der Liste herausnehmen. Dabei werden die Anagramme auf Wortgruppen untersucht, die auch in den Texten vorkamen. Ist eine Wortgruppe des Anagramms auch in einem Text vorgekommen gilt das Anagramm als sinnvoll. Das eingegebene Wort „newspaper“ besitzt zum Beispiel das sinnvolle Anagramm „a pen sprew“. Weil in einem Text „a pen“ vorkommt wird das Anagramm als sinnvoll gekennzeichnet. Dabei ist es dann egal, ob „pen sprew“ in einem Text vorkam oder nicht.

Bessere Anagramme erhält man mit folgender Festlegung: alle Wortgruppen des Anagramms müssen in einem Text vorkommen, um als sinnvoll zu gelten. Das eingegebene Wort „cafeteria“ beispielsweise hat nach dieser Methode das sinnvolle Anagramm „I free a cat“. Es gilt als sinnvoll, weil die Wortgruppen „I free“, „free a“ und „a cat“ jeweils in einem Text vorkamen.

2 Programm-Dokumenation

Die Lösungsidee wurde von mir in der Programmiersprache C umgesetzt.

2.1 Strukturen

Mein Programm enthält folgende Strukturen: **`Element`** und **`List`**. Die Struktur **`Element`** stellt ein Element einer Liste dar und enthält einen **`String`** und einen Zeiger auf das nächste Listenelement.

Die Struktur **`List`** stellt eine Liste dar. Die Struktur hat einen Zeiger auf das Erste und das Letzte Element der Liste und eine Variable, die die Anzahl der Elemente in der Liste speichert.

2.2 `main`-Funktion

In der **`main`**-Funktion werden zunächst die übergebenen Parameter ausgewertet und dementsprechend Variablen gesetzt. Dann wird die Liste der möglichen Wörter für das Anagramm erstellt, indem die Funktion **`finde_moegliche_woerter`** aufgerufen wird. Diese Funktion findet die möglichen Wörter für das Anagramm anhand des Wörterbuches, fügt die gefundenen Wörter in eine Liste und gibt dann einen Zeiger auf die Liste zurück.

Als nächstes wird diese Liste der Funktion **`finde_anagramme`** übergeben, die alle Anagramme erstellt und in eine Liste einträgt.

Die Anagramme und die Dateinamen der Texte werden dann der Funktion **`filter`** übergeben. Diese Funktion „filtert“ mithilfe der Texte die sinnvollen Anagramme aus der Liste heraus und fügt sie in eine neue Liste, deren Zeiger von der Funktion zurückgegeben wird.

Zum Schluss wird die Liste der sinnvollen Anagramme ausgegeben.

2.3 Die Funktion `differenzmenge`

Die Funktion **`differenzmenge`** wird von den Funktionen **`finde_moegliche_woerter`** und **`finde_anagramme`** benutzt. Die Funktion bekommt zwei Wörter als Parameter und bestimmt deren Differenzmenge. Anhand des Rückgabewertes der Funktion kann geprüft werden, ob ein Wort Teilmenge eines anderen Wortes ist oder ob die Wörter gleich sind. Wird ein leerer **`String`** zurückgegeben, dann sind die beiden Wörter gleich. Ist der Rückgabewert **`NULL`**, dann liegt keine Teilmenge vor. Wird ein **`String`** zurückgegeben, der nicht leer ist, dann ist das erste Wort Teilmenge des zweiten Wortes.

3 Programm-Ablaufprotokoll

3.1 Texte

Für die Suche nach sinnvollen Anagrammen habe ich englische Texte von dem „Project Gutenberg“[1] genommen. Darunter sind z. B. die Bücher: „Romeo & Juliet“, „Moby Dick“, „Macbeth“, „Dracula“ und „Holy Bible“.

Die Textdatei **alle_texte.txt** enthält alle Texte, die ich aus dem „Project Gutenberg“ genommen habe.

3.2 Wörterbucher

Das Wörterbuch **woerterbuch.dict** habe ich mithilfe der **GNU**-Programme **sed** und **sort** aus der Textdatei **alle_texte.txt** erstellt. Ausserdem habe ich Wörterbücher aus der freien Software Ispell[2], sowie dem Freeware-Editor PSPad[3] genommen.

3.3 Parameter

Durch Aufrufen des Programms mit dem Paramter **-h** bekommt der Benutzer eine Auflistung der möglichen Parameter für das Programm:

```
$ ./anagramme -h
Benutzung: ./anagramme -w "<Wort(gruppe)>" [Optionen]
Optionen:
-a <Anzahl>             maximale Anzahl der Woerter pro
                        Anagramm
                        Default: 0 (kein Limit)

-d <Dateiname>          Dateiname des Woerterbuches
                        Default: english_pspad.dict

-b "<Dateien>"          Textdateien, die beim Herausfinden
                        sinnvoller Anagramme benutzt werden
                        sollen
                        Default: holy_bible.txt

-p                      Ausgeben von Zwischenergebnissen

-s                      Strenges Aussortieren von nicht
                        sinnvollen Anagrammen

-h                      Ausgeben dieser Hilfe
```

Dem Programm muss mit dem Parameter **-w** ein Wort angegeben werden, ansonsten ist es nicht notwendig das Programm auszuführen.

1 http://www.gutenberg.org/wiki/Main_Page
2 http://ficus-www.cs.ucla.edu/geoff/ispell.html
3 http://www.pspad.com/

Der Parameter **-a** ist bei langen Wörtern sinnvoll um die Laufzeit zu verringern. Bei dem Paramter **-a 3** werden beispielsweise nur Anagramme untersucht, die aus maximal 3 Wörtern bestehen:

```
$ ./anagramme -w "loud noise" -b alle_texte.txt -a 3
Sinnvolle Anagramme fuer das Wort loud noise:
de louis on
dei no soul
dei soul on
deloo in us
deloo us in
deul i soon
deul soon i
deus in loo
deus oil on
...
use on olid
used in loo
used no oil
used oil on
used on oil
Anzahl: 457
```

In dem Fall wurde auch noch die Option **-b** angegeben, die angibt welche Texte untersucht werden sollen. An diesem Beispiel erkennt man, dass es lange dauern würde sich die Anagramme durchzulesen, des Weiteren sind Anagramme vorhanden, die kaum einen Sinn ergeben. Besser wäre es das Programm mit der Option **-s** aufzurufen, was bedeutet, dass die Anagramme „strenger“ aussortiert werden (siehe 1.4):

```
$ ./anagramme -w "loud noise" -b alle_texte.txt -s -a 3
Sinnvolle Anagramme fuer das Wort loud noise:
doe in soul
due no soil
loud noise
louis on de
no soul die
soul die on
soul i done
sound i loe
use no idol
Anzahl: 9
```

Bei dieser Option erhält man dann nur noch 9 Anagramme, jedoch sind diese besser.

Mit der Option **-d** kann man bestimmen, welches Wörterbuch verwendet werden soll:

```
$ ./anagramme -w "loud noise" -d english_ispell.dict -b
alle_texte.txt -s -a 3
Sinnvolle Anagramme fuer das Wort loud noise:
Louis on de
doe in soul
```

```
due no soil
loud noise
no soul die
soul die on
use no idol
Anzahl: 7
```

In diesem Wörterbuch ist das Wort „I“ beispielsweise nicht enthalten, deshalb werden auch keine Anagramme erstellt, die „I“ enthalten.

Wie sinnvoll die gefundenen Anagramm sind, hängt stark von dem Wörterbuch und den Texten ab. Das Anagramme „tomb in corsica“ aus der Aufgabenstellung kann beispielsweise nur reproduziert werden, wenn als Wörterbuch die Datei **`englisch_ispell.dict`** und als Texte die Dateien **`dracula.txt`** und **`the_count_of_monte_cristo.txt`** angegeben werden:

```
$ ./anagramme -w combinatorics -b
"the_count_of_monte_cristo.txt dracula.txt" -a 3 -d
english_ispell.dict

Sinnvolle Anagramme fuer das Wort combinatorics:
as not microbic
at robin cosmic
Bim Corsican to
boracic in most
boracic most in
brain to cosmic
cambric it soon
Corsica tomb in
Corsican to Bim
cosmic at robin
cosmic brain to
in most boracic
it soon cambric
microbic as not
microbic not as
microbic sat on
microbic to San
most in boracic
not as microbic
sat on microbic
to San microbic
tomb in Corsica
Anzahl: 22
```