Matthias Rüster
Lena Kristin Zander

Bundeswettbewerb Informatik: Aufgabe 4

Lösungsidee

Bei der Entschlüsselung des Buches werden nur die Buchstaben des deutschen Alphabets und die Umlaute (ä, ö, ü, ß) betrachtet. Groß- und Kleinschreibung ist dabei irrelevant. Alle anderen Buchstaben, Sonderzeichen und Leerzeilen werden also übersprungen.

Führt man für jeden Zeilenanfang im Buch die Entschlüsselung aus, erhält man eine Menge von möglichen Lösungen. Jede mögliche Lösung wird dann mit Hilfe eines Wörterbuches bewertet.

Besteht der entschlüsselte Satz aus N Buchstaben, dann gibt es genau $\frac{N \cdot (N+1)}{2}$ mögliche Zeichenketten, die der Satz enthalten kann. Für das Wort „sauer“ beispielsweise können die Zeichenketten: s, sa, sau, saue, sauer, a, au, aue, auer usw. mögliche deutsche Wörter sein.

Jede mögliche Zeichenkette wird dann in einem deutschen Wörterbuch nachgeschlagen. Wird eine Zeichenkette als deutsches Wort erkannt, werden die Buchstaben der gerade betrachteten Zeichenkette im Satz als „gefunden“ markiert.

Zuletzt wird für jede Zeichenkette ein Wert berechnet, der aussagt mit welcher Wahrscheinlichkeit ein deutscher Satz gefunden wurde. Berechnet wird diese Wahrscheinlichkeit indem man die Anzahl der „gefundenen“ Buchstaben durch die Gesamtanzahl der Buchstaben der Zeichenkette teilt.

Wurde beispielsweise bei der entschlüsselten Zeichenkette „qundiy“ nur das Wort „und“ (3 Buchstaben) im Wörterbuch gefunden, so erhält diese Zeichenkette eine Wahrscheinlichkeit von 3 ÷ 6 = 0,5 (bzw. 50%). Eine Wahrscheinlichkeit von 1 (bzw. 100 %) würde also bedeuten, dass alle Buchstaben des Satzes als Teil eines Wortes im Wörterbuch gefunden wurden.

Der entschlüsselte Satz mit der höchsten Wahrscheinlichkeit ist in den meisten Fällen dann auch der deutsche Klartext. Es hängt natürlich stark davon ab, wie vollständig das Wörterbuch ist. Sind in dem Wörterbuch beispielsweise Abkürzungen oder Anglizismen enthalten, wirkt sich das negativ auf die Bewertung der Sätze aus. Das Programm könnte mit einem Wörterbuch, das beispielsweise Abkürzungen, wie „zzgl“ oder „dgl“, nicht aber häufige Wörter wie „du“ oder „ich“ enthält, die Zeichenkette „tzzgl“ („zzgl“ wurde gefunden: 80%) besser bewerten als „unddu“ (nur „und“ wurde gefunden: 60%).

Programm-Dokumentation

Das Einlesen des Buches übernimmt die Funktion **`read_book`**. Um sich das Entschlüsseln der Sätze möglichst einfach zu machen, wird das gesamte Buch zunächst als eine Zeichenkette einer **`book`**-Struktur gespeichert. Das Buch wird zeichenweise eingelesen und nur die zulässigen Buchstaben (Umlaute und Buchstaben von A-Z) werden in der Zeichenkette hintereinander abgespeichert. Anschließend werden alle Buchstaben mit Hilfe der Funktion **`towlower`** in Kleinbuchstaben umgewandelt. Zusätzlich werden sich noch die Zeilenanfänge, als „Anfangspunkte“ für den Entschlüsselungsalgorithmus gemerkt. Diese Information wird in dem **`int`**-Array der **`book`**-Struktur gespeichert.

Die Datenhaltung wird hier nochmal an einem Beispieltext erklärt:

```
Im Sommer 2011
war es sehr warm.
```

Das **`char`**-Array der **`book`**-Struktur sieht dann folgendermaßen aus:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
i	m	s	o	m	m	e	r	w	a	r	e	s	s	e	h	r	w	a	r	m

Das **`int`**-Array der **`book`**-Struktur besteht lediglich aus zwei Elementen, da es nur zwei Zeilenanfänge gibt:

0	8

Die 0 und 8 sagt also aus, dass die Elemente mit dem Index 0 und 8 in dem **`char`**-Array Zeilenanfänge und somit „Anfangspunkte“ beim Entschlüsseln der Zahlenfolge sind.

Soll beispielsweise die Zahlenfolge „4 2 4“ entschlüsselt werden, beginnt man die Entschlüsselung bei den Arrayvariablen mit dem Index 0 und 8. Man erhält zwei mögliche Lösungen: „oma“, „esw“.

Zusätzlich zu diesen beiden Arrays enthält die **`book`**-Struktur auch noch zwei **`int`**-Variablen, die die jeweiligen Größen der Arrays enthalten.

Nachdem das Buch eingelesen wurde, wird die Zahlenfolge eingelesen. Dies übernimmt die Funktion **`read_code`**. Die einzelnen Zahlen der Zahlenfolge werden aus der Datei gelesen und in dem **`int`**-Array der **`code`**-Struktur gespeichert. In der **`code`**-Struktur wird in einer zusätzlichen **`int`**-Variablen die Größe dieses Arrays gespeichert.

Die Funktion **`decrypt`** bekommt das eingelesene Buch und die Zahlenfolge in Form einer **`book`**- und **`code`**-Struktur übergeben. Beginnend bei jedem Zeilenanfang im Buch wird jeder mögliche Lösungssatz mit Hilfe der Zahlenfolge zusammengestellt und in einer **`sentence`**-Struktur abgespeichert. Ein Array von **`sentence`**-Strukturen wird dann zurückgegeben.

Danach wird jeder mögliche Lösungssatz mit der Funktion **`dict_search`** bewertet. Diese Funktion benötigt eine **`sentence`**-Struktur deren Satz bewertet werden soll und ein Wörterbuch, welches vorher mit der Funktion **`read_dictionary`** eingelesen wurde. Diese Funktion bewertet die Sätze so, wie es in der Lösungsidee bereits beschrieben wurde: Jedes mögliche Wort des Satzes wird in dem Wörterbuch gesucht. Wurde das Wort gefunden, werden sich die Buchstaben des gefundenen Wortes gemerkt. Zum Schluss wird die Anzahl der „gefunden Buchstaben“ durch die Gesamtanzahl der Buchstaben des Satzes geteilt. Dieser Wert wird auch in den jeweiligen **`sentence`**-Strukturen gespeichert.

Das Suchen eines Wortes in einem umfangreichen Wörterbuch ist ohne die richtigen Strukturen mit

viel Zeitaufwand verbunden. Um dies zu umgehen, haben wir uns dafür entschieden das Wörterbuch als Hash abzuspeichern. Als Hashfunktion haben wir den SDBM-Algorithmus[1] verwendet. Die Größe der Hashtabelle haben wir auf 2000003 (Primzahl) gesetzt. Bei der Kollisionsauflösung haben wir uns für das Hashing mit Verkettung[2] entschieden. Als dynamische Strukturen haben wir AVL-Bäume[3] gewählt.

Zum Schluss werden die bewerteten Sätze mit Hilfe der Funktion **`sort`** nach ihrer Bewertung sortiert. Der Satz mit der besten Bewertung wird dann auf dem Bildschirm ausgegeben.

Beträgt die Wahrscheinlichkeit eines Satzes 100 %, ist es mit Hilfe des Wörterbuches möglich Leerzeichen zu setzen, um die einzelnen gefundenen Wörter kenntlich zu machen.

Die Funktion **`print_plaintext`** gibt alle möglichen Klartexte eines Lösungssatzes auf dem Bildschirm aus. Ist in einem Lösungssatz beispielsweise „wer den“ enthalten, gibt die Funktion zwei Klartextversionen aus: einen mit „wer den“ und den anderen mit „werden“.

1 http://www.ntecs.de/projects/guugelhupf/doc/html/x435.html#AEN466
2 http://de.wikipedia.org/wiki/Hashtabelle#Hashing_mit_Verkettung
3 http://de.wikipedia.org/wiki/AVL-Baum

Programm-Ablaufprotokoll

Das Programm bekommt den Namen der Datei als Parameter übergeben, in der die Zahlenfolge enthalten ist. Der Klartext wird dann auf dem Bildschirm ausgegeben:

```
$ ./aufgabe4 zahlenfolge0.txt
entschluesselt: ohneliebekeinewahrheit
Klartext(e):
ohne liebe keine wahrheit
```

Soll zu jedem Satz die Bewertung (Wahrscheinlichkeit) ausgegeben werden, gibt man z.B. **`-p 3`** als Parameter an, um sich die „besten“ drei Sätze anzeigen zu lassen:

```
$ ./aufgabe4 zahlenfolge1.txt -p 3
 100,0 %    dasdenkensollmandenpferdenüberlassendiehabendengrößerenkopf
  86,4 %    irewarinnraredesmnelkerhnahawimuoauenttalfolgelanereildniec
  83,1 %    giunmmevpnehainirenseichimilnamweilosttintsindnndundddrtaud
```

Der zweite Satz ist also mit einer Wahrscheinlichkeit von 86,4% ein deutscher Satz.

Bei der Zahlenfolge 4 gibt es mehrere mögliche Klartext-Versionen:

```
$ ./aufgabe4 zahlenfolge4.txt
entschluesselt: wovondasherzvollistdavongehtdermundüber
Klartext(e):
wo von das herz voll ist da von geht der mund über
wo von das herz voll ist davon geht der mund über
wovon das herz voll ist da von geht der mund über
wovon das herz voll ist davon geht der mund über
```

Außerdem ist es möglich dem Programm mehrere Dateinamen als Parameter zu übergeben:

```
$ /aufgabe4 zahlenfolge9.txt zahlenfolgeA.txt
zahlenfolge9.txt
entschluesselt: wennmandurchzweifelläuftistderwegzumhimmellang
Klartext(e):
wenn man durch zweifel läuft ist der weg zum himmel lang

zahlenfolgeA.txt
entschlusselt: jededummheitleidetamekelvorsichselbst
Klartext(e):
je de dummheit leid et am ekel vor sich selbst
je de dummheit leidet am ekel vor sich selbst
jede dummheit leid et am ekel vor sich selbst
jede dummheit leidet am ekel vor sich selbst
```

Als Standard-Wörterbuch haben wir uns ein Wörterbuch aus dem neusten igerman98-Paket[4] zusammengestellt. Möchte man ein anderes Wörterbuch verwenden, kann man dies mit dem Parameter **`-d`** ***`Dateiname`*** ändern.

Möchte man ein anderes Buch als **`EffiBriest.txt`** verwenden, kann man das Buch mit dem Parameter **`-b`** ***`Dateiname`*** ändern.

4 Homepage von igerman98: http://www.j3e.de/ispell/igerman98/
benutztes Paket: http://www.j3e.de/ispell/igerman98/dict/igerman98-20110609.tar.bz2

Mit dem Parameter **-h** wird dem Benutzer eine Hilfe zur Benutzung des Programms ausgegeben:

```
$ ./aufgabe4 -h
Verwendung: aufgabe4 [Optionen] Dateiname...

moegliche Optionen:
    -p Anzahl         zusaetzliche Informationen ausgeben
    -b Buch           Buch festlegen
    -d Woerterbuch    Woerterbuch festlegen
    -h                diese Hilfe anzeigen
```

Die gefundenen Lösungen zu den Zahlenfolgen lauten:

Zahlenfolge 0: ohne liebe keine wahrheit

Zahlenfolge 1: das denken soll man den pferden überlassen die haben den größeren kopf

Zahlenfolge 2: wenn du irgendetwas verloren hast nimm an du hättest es einem armen gegeben

Zahlenfolge 3: der ellenbogen ist dem munde nahe dennoch kann man nicht selbst hinein beißen

Zahlenfolge 4: wo von das herz voll ist davon geht der mund über

Zahlenfolge 5: je verdorbener der staat desto mehr gesetze hat er

Zahlenfolge 6: erinnere dich mensch dass du aus staub bist und zu staub zurückkehren wirst

Zahlenfolge 7: wer den aal hält bei dem schwanz dem bleibt er weder halb noch ganz

Zahlenfolge 8: schlechte handwerker klagen über ihr werkzeug

Zahlenfolge 9: wenn man durch zweifel läuft ist der weg zum himmel lang

Zahlenfolge A: jede dummheit leidet am ekel vor sich selbst

Zahlenfolge B: ein guter spruch ist die wahrheit eines ganzen buches in einem einzigen satz