Fachhochschule Wedel

Seminararbeit

Algorithmus zur komprimierten Übertragung von Textdaten an mobile Endgeräte

Sven Reinck

17. Januar 2007

Inhaltsverzeichnis

1 Motivation **1**

2 Wörterbuch **1**
2.1 Beispiel Codieren 2
2.2 Einsparung 3

3 Huffman-Codierung **4**
3.1 Baum erstellen 4
3.2 Beispiel Codieren 5
3.3 Beispiel Decodieren 6

4 Kanonische Huffman-Codierung **7**
4.1 Tabelle Erstellen 7
4.2 Beispiel Codieren 7
4.3 Speichern der Tabelle 8
4.4 Laden der Tabelle 8
4.5 Beispiel Decodieren 8
4.6 Einsparung 9

5 Vorhersagendes Codieren **10**
5.1 Beispiel ohne erstellen der Bäume 10
5.2 Einsparung 11

6 Literatur **11**

1 Motivation

Wenn wir größere Textmengen an ein mobiles Endgerät (z.B. ein Handy) übertragen wollen, haben wir das Problem, dass der Übertragungskanal besonders langsam und für den Kunden auch sehr teuer ist. Daher lohnt es sich den Text komprimiert zu übertragen. Allerdings hat man in der Regel auf der Empfängerseite nicht besonders viel Ressourcen zur Verfügung um die Daten zu dekomprimieren. In dieser Arbeit stelle ich einen Algorithmus vor, der auf das Komprimieren von Text-Daten spezialisiert ist und dadurch trotz seiner geringen Anforderungen an der Hardware des Empfängers gute Komprimierungsraten erzielt.

2 Wörterbuch

Der erste Ansatz zur Komprimierung bildet ein Wörterbuch. Dabei werden häufig vorkommende Zeichengruppen im Wörterbuch gespeichert und im Text steht nur noch eine Referenz. Es ist leicht einzusehen, dass die Einsparung proportional sowohl zu der Länger dieser Gruppe also auch zu der Häufigkeit ist. Tatsächlich gilt folgende Formel: Ersparnis = (Vorkommen - 1) · (Länge - 1).

Bei einer Analyse üblicher Texte stellt sich heraus, dass Gruppen der Länge 2 und 3 die größte Ersparnis bei dieser Technik bringen. Da die Datenstruktur aber möglichst einfach sein soll, wird das Wörterbuch nur Einträge der Länge 2 aufnehmen können. Dafür wird es aber explizit gestattet, dass die Einträge im Wörterbuch wiederum Referenzen auf andere Einträge beinhalten können. Eine Gruppe der Länge 3 ließe sich somit also auch durch zwei Gruppen der Länge 2 darstellen. Dies verschlechtert zwar die Komprimierung des Wörterbuches, ist aber einer der notwendigen Kompromisse um die Dekomprimierung so einfach wie möglich zu halten.

Um im Text die einzelnen Zeichen von den Referenzen auf das Wörterbuch unterscheiden zu können, wird eine Liste mit allen vorkommenden Zeichen erstellt. Steht im Text ein einzelnes Zeichen, so wird es durch seine Position innerhalb dieser Liste repräsentiert. Ein größerer Wert stellt eine Referenz auf das Wörterbuch dar. Damit alle Werte als Bytes gespeichert werden können, haben die Liste und das Wörterbuch zusammen ein Läge von 256.

2.1 Beispiel Codieren

Im folgenden soll das Verfahren an einem kleinen Beispiel demonstriert werden:

`fischers fritz fischt frische fische`

Zunächst wird für jedes Paar von Zeichen bestimmt, wie viel es einsparen würde dieses in Wöterbuch zu übernehmen.

Paar	Vorkommen	Ersparnis
'sc'	4	3
'is'	4	3
'ch'	4	3
' f'	4	3
'he'	3	2
'fi'	3	2
'ri'	2	1
'fr'	2	1

Das beste Vorkommen wird dann tatsächlich ins Wörterbuch übernommen und im Text entsprechend ersetzt.

Code	Ersetzung	Rekursiv
0	'sc'	'sc'

`fi0hers fritz fi0ht fri0he fi0he`

Jetzt wird erneut die Tabelle mit den besten Ersetzungen erstellt und die beste durchgeführt. Solange, bis entweder das Wörterbuch voll ist, oder sich keine Ersetzung mehr lohnt. Dann erhalten wir am Ende folgende Daten:

Code	Ersetzung	Rekursiv
0	'sc'	'sc'
1	'i0'	'isc'
2	'1h'	'isch'
3	' f'	' f'
4	'2e'	'ische'
5	'3r'	' fr'

```
f4rs5itz32t5434
```

2.2 Einsparung

1. Abschnitt:

Methode	Overhead	Daten (%)	Gesamt (%)
unkomprimiert	0	2359 (100%)	2359 (100%)
Wörterbuch	449	848 (36%)	1297 (55%)

ganzer Text:

Methode	Overhead	Daten (%)	Gesamt (%)
unkomprimiert	0	16896 (100%)	16896 (100%)
Wörterbuch	437	7899 (47%)	8336 (49%)

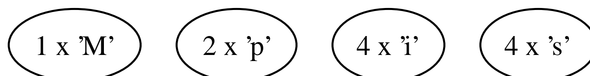
Wie hier zu sehen ist, kann die Wörterbuch-Komprimierung bei längeren Texten ihr Potential noch besser ausspielen. Da der Overhead in der gleichen Größenordnung bleibt, die prozentuale Komprimierung aber nicht wesentlich schlechter wird. Dadurch ist das Gesamtergebnis beim größeren Text besser geworden.

3 Huffman-Codierung

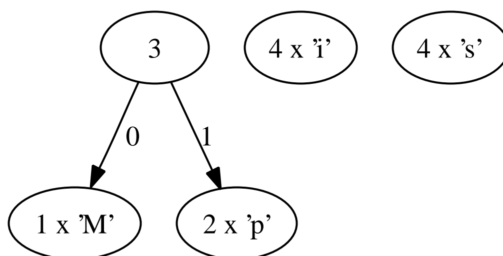
Der zweite Ansatz zur Komprimierung bildet die Huffman-Codierung. Diese ist darauf ausgelegt, die Zeichen möglichst effizient zu speichern, in dem die Häufigkeit der einzelnen Zeichen berücksichtigt wird. Ein häufiges Zeichen wird mit möglichst wenig Bits kodiert. Dadurch können für ein seltenes Zeichen mehr Bits verwendet werden. Diese Codierung kann problemlos mit dem Wörterbuch-Ansatz kombiniert werden. Eine Referenz auf das Wörterbuch wird dabei genau so behandelt wie ein normales Zeichen.

3.1 Baum erstellen

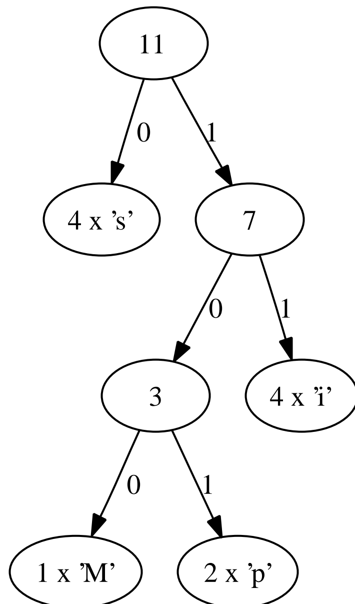
Um herauszufinden, mit welchen Bits ein Zeichen codiert wird, muss der Huffman-Baum aufgestellt werden. Das Verfahren erkläre ich am Wort `Mississippi`. Als erstes stellen wird zunächst eine Liste mit den Zeichen und ihrer jeweiligen Häufigkeit im Text auf. Diese Liste wird dann nach der Häufigkeit sortiert.



Jetzt werden solange jeweils die ersten beiden Elemente der Liste zu einem neuen zusammengefasst und die Liste wieder sortiert, bis nur noch ein Element in der Liste ist. Dabei bekommen die neuen Elemente jeweils die Summe der Häufigkeiten der beiden Zeichen, aus denen sie entstanden sind.



Der entstehende Baum sieht dann so aus:



Aufgrund der Tatsache, dass die Zeichen sich nur an Blättern des Baumes befinden, ist diese Codierung präfixfrei. Das bedeutet, dass die Codierung eines Zeichens nicht der Anfang der Codierung eines anderen Zeichens ist. Dadurch ist es nicht notwendig die einzelnen Codierungen in der Ausgabe voneinander zu trennen, da eindeutig bestimmt ist, wann eine Codierung endet.

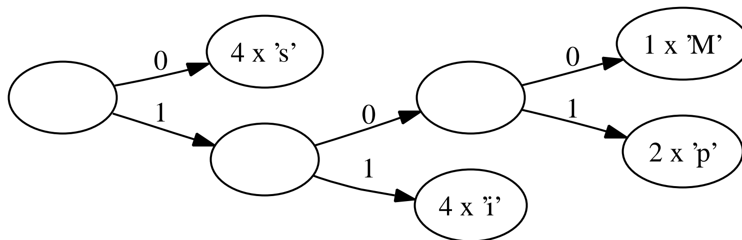
3.2 Beispiel Codieren

Zum Codieren kann der Baum einfach in einer Tabelle umgewandelt werden, in dem der Baum durchgegangen wird und jeweils in den Blättern das jeweilige Zeichen zusammen mit dem Weg zu diesem Blatt in der Tabelle gespeichert wird.

Zeichen	Bits
's'	0
'M'	100
'p'	101
'i'	11

In dieser Tabelle muss dann nur für jedes zu codierende Zeichen nachgesehen werden.

3.3 Beispiel Decodieren



Zum Dekodieren beginnt man an der Wurzel des Baumes. Dann ließt man nach und nach die Bits ein und entscheidet sich entsprechend dem Bit für den linken oder rechten Ast, solange bis man an einem Blatt angekommen ist. In diesem Blatt steht dann das gesuchte Zeichen. Danach beginnt man wieder an der Wurzel, bis alle Bits eingelesen sind.

M	i	s	s	i	s	s	i	p	p	i
100	11	0	0	11	0	0	11	101	101	11

4 Kanonische Huffman-Codierung

Das Problem ist die Codierung zum Empfänger zu übertragen. Der Baum eignet sich nicht besonders gut dafür. Die kanonische Huffman-Codierung ist aber eine sehr geschickte Methode dafür. Zu diesem Zweck wird die Tabelle mit den Codierungs-Bits überarbeitet.

4.1 Tabelle Erstellen

Die Bits, die aus dem Huffman-Baum stammen, können gelöscht werden. Das einzig wichtige ist die Länge der Codierung. Die Tabelle wird nach genau dieser Länge sortiert. Dann fängt man in der ersten Zeile mit einer Bitfolge an, die genau so oft 0 enthält, wie die alte Bitfolge. Hat die nächste Zeile die gleiche Länge wird zu der Bitfolge 1 addiert. Ist die Länge der nächsten Zeile kürzer, so wird die aktuelle Bitfolge erst von rechts entsprechend gekürzt und anschließend mit 1 addiert.

Zeichen	Länge	Bits
'M'	3	000
'p'	3	001
'i'	2	01
's'	1	1

4.2 Beispiel Codieren

Das Codieren verläuft genau so, wie bei der normalen Huffman-Codierung. Es muss nur für jedes Zeichen in der Tabelle nachgesehen zu werden.

M	i	s	s	i	s	s	i	p	p	i
000	01	1	1	01	1	1	01	001	001	01

4.3 Speichern der Tabelle

Die Tabelle wird gespeichert, indem zuerst die Länge der längsten Bitfolge gespeichert wird. Danach wird für jede Länge angegeben, wie oft diese als Bitfolge vorkommt. Als letztes folgen die Zeichen in genau der Reihenfolge, wie sie auch beim Erstellen dieser Codierung verwendet wurden.

3	2 1 1	'M' 'p' 'i' 's'

4.4 Laden der Tabelle

Mit dem ersten Wert kann die Größe der Tabelle bestimmt werden. Wobei diese im Gegensatz zur Tabelle des Senders nur so viele Zeilen enthält, wie es verschieden lange Bitfolgen gibt. Mit den Häufigkeiten der einzelnen Längen kann dann berechnet werden, welches die jeweils erste Bitfolge für eine Länge ist. Diese wird in der Tabelle gespeichert. Zuletzt werden die Zeichen eingelesen. Diese werden entsprechend der Häufigkeiten der einzelnen Bitfolge-Längen auf die Tabelle verteilt.

Länge	Bits	Zeichen
3	000	'M' 'p'
2	01	'i'
1	1	's'

4.5 Beispiel Decodieren

Beim Decodieren werden die Bits nacheinander eingelesen und zusammen gesetzt. Bei jedem Bit wird der aktuelle Wert mit der kleinsten Bitfolge in der Zeile mit der entsprechenden Länge verglichen. Ist der aktuelle Wert kleiner, so müssen weitere Bits eingelesen werden. Ist der Wert größer oder gleich, so sagt die Differenz zwischen der kleinsten Bitfolge und dem aktuellen Wert aus, das wievielte Zeichen in der aktuellen Zeile das gesucht Zeichen ist.

000	01	1	1	01	1	1	01	001	001	01
M	i	s	s	i	s	s	i	p	p	i

4.6 Einsparung

1. Abschnitt:

Methode	Overhead	Daten (%)	Gesamt (%)
unkomprimiert	0	2359 (100%)	2359 (100%)
Wörterbuch	449	848 (36%)	1297 (55%)
Wb. + Huffman	672	773 (33%)	1445 (61%)

ganzer Text:

Methode	Overhead	Daten (%)	Gesamt (%)
unkomprimiert	0	16896 (100%)	16896 (100%)
Wörterbuch	437	7899 (47%)	8336 (49%)
Wb. + Huffman	705	7282 (43%)	7987 (47%)

Durch den weiteren Overhead, der durch die Huffman-Codierung hinzugekommen ist, hat sich die Komprimierung beim kurzen Text verschlechtert. Für den langen Text hat sie eine kleine Verbesserung erreicht. Diese sieht auf den ersten Blick sehr klein aus, aber wir sollten nicht vergessen, dass diese Komprimierung auf Handys und ähnlichen kleinen Geräten zum Einsatz kommen soll. Dort sind auch kleine Einsparungen schon sehr viel Wert, außerdem haben wir wieder weniger Daten, die ans Handy übertragen werden müssen.

5 Vorhersagendes Codieren

Beim vorhersagenden Codieren wird berücksichtigt, dass die Möglichkeiten, welches Zeichen als nächtes kommt, sehr stark eingeschränkt werden können, wenn man das vorhergegangene Zeichen berücksichtigt. Zum Beispiel wird in einem einfachen Text auf einen großen Buchstaben immer nur ein kleiner Buchstabe folgen, was die Anzahl der Möglichkeiten etwa auf die hälfte reduziert. Es gibt aber auch andere Beispiele, wo der Effekt viel stärker auftritt, besonders, wenn man die Statistik für einen konkreten Text aufstellt. Das zuletzt gelesene Zeichen wird als Kontext bezeichnet.

Bei der Kombination mit dem Wörterbuch, kommt dieses mal aber noch eine Kleinigkeit hinzu. Würde man auch einen Wörterbuch-Eintrag als Kontext zulassen, so hätte man 256 Kontexte zu verwalten, was den Speicher bedarf doch sehr erhöhen würde. Daher wird wieder ein Kompromiss eingegangen, indem der Kontext eines Wörterbuch-Eintrags mit dem des letzten Zeichens dieses Eintrags identisch ist. Dadurch müssen nur die Kontexte für die einzelnen Zeichen verwaltet werden.

5.1 Beispiel ohne erstellen der Bäume

Im Falle des Mississippi-Beispiels ergeben sich mit der konischen Huffman-Codierung über die Häufigkeiten der Buchstaben im jeweiligen Kontext folgende Tabelle:

Anfang	
'M'	

M	
'i'	

i	
'p'	0
's'	1

p	
'i'	0
'p'	1

s	
'i'	0
's'	1

Codiert man nun das Wort mit diesen Tabellen, so wird für die ersten beiden Buchstaben nicht mal ein Bit benötigt um diese zu speichern, da im jeweiligen Kontext klar ist, welcher Buchstabe folgen muss. Die restlichen Buchstaben werden mit jeweils mit einem Bit codiert, was gegenüber der letzten Variante ebenfalls eine große Verbesserung ist.

M	i	s	s	i	s	s	i	p	p	i
000	01	1	1	01	1	1	01	001	001	01
x	x	1	1	0	1	1	0	0	1	0

5.2 Einsparung

1. Abschnitt:

Methode	Overhead	Daten (%)	Gesamt (%)
unkomprimiert	0	2359 (100%)	2359 (100%)
Wörterbuch	449	848 (36%)	1297 (55%)
Wb. + Huffman	672	773 (33%)	1445 (61%)
Wb. + Context	1216	464 (20%)	1680 (71%)

ganzer Text:

Methode	Overhead	Daten (%)	Gesamt (%)
unkomprimiert	0	16896 (100%)	16896 (100%)
Wörterbuch	437	7899 (47%)	8336 (49%)
Wb. + Huffman	705	7282 (43%)	7987 (47%)
Wb. + Context	2446	5253 (31%)	7699 (46%)

Auch diese Erweiterung macht das Gesamtergebnis für den kurzen Text schlechter, wobei sich die Größe der eigentlich Daten enorm verkleinert hat. Der lange Text ist insgesamt wieder ein Stück kleiner geworden. Bei einem noch längeren Text wäre das entsprechende Stück sicherlich auch noch etwas größer.

6 Literatur

Wer sich genauer über das Thema Komprimierung oder auch Verschlüsselung und Übertragungssicherheit informieren möchte, dem empfele ich das Buch ,,Grundkurs Codierung, 3. Auflage" von Wilfried Dankmeier aus dem Vieweg Verlag.

Folgende Internetseiten sind für die einzelnen Teile des Algorithmus von ebenfalls von Interesse:

Wörterbuch http://en.wikipedia.org/wiki/LZ77

kanonische Huffman-Codierung http://en.wikipedia.org/wiki/Canonical_Huffman_code

vorhersagendes Codieren http://en.wikipedia.org/wiki/PPM_compression_algorithm