ETH Zürich
Dept. Informatik

Prof. Markus Gross
WS 06/07

Informationstheorie

Lösung 8

8.1 Arithmetische Codierung

a) Die Codierung geht wie folgt vor:

i. Teile das Intervall $[0, 1)$ in Subintervalle gemäss der Symbol-Wahrscheinlichkeiten. Also hier $[0, 0.5)$, $[0.5, 0.8)$ und $[0.8, 1)$.

ii. Nimm das nächste Symbol aus dem String und ermittle das Subintervall, welches seiner Wahrscheinlichkeit entspricht. Wenn das nächste Symbol ein *a* ist, dann nehmen wir das Intervall $[0, 0.5)$.

iii. Teile dieses Intervall gemäss der Symbolwahrscheinlichkeiten, so wie in Schritt 1. $[0, 0.5)$ wird in $[0, 0.25)$, $[0.25, 0.4)$ und $[0.4, 0.5)$ unterteilt.

iv. Führe die Schritte 2 und 3 aus, bis der String zu Ende ist. Für den String *aacbca* erhalten wir das Intervall $[0.237, 0.2385)$.

v. Weise dem String eine Zahl aus diesem Intervall zu. Nehmen wir zum Beispiel den Mittelpunkt des Intervalls, dann bekommen wir 0.23775 für *aacbca*.

b) Um eine Zahl zu decodieren gehen wir ähnlich vor:

i. Teile das Intervall $[0, 1)$ in Subintervalle gemäss der Symbol-Wahrscheinlichkeiten. Also hier $[0, 0.5)$, $[0.5, 0.8)$ und $[0.8, 1)$.

ii. Nimm das Intervall zu welchem die Zahl gehört und ermittle das entsprechende Symbol. Die Zahl 0.63215699 hat als erstes Symbol *b*.

iii. Teile dieses Intervall gemäss der Symbolwahrscheinlichkeiten, so wie in Schritt 1. $[0.5, 0.8)$ wird in $[0.5, 0.65)$, $[0.65, 0.74)$ und $[0.74, 0.8)$ geteilt.

iv. Führe die Schritte 2 und 3 so oft aus, bis die Länge des Wortes erreicht ist. Für die Zahl 0.63215699 erhalten wir den String *bacacaaaac*.

c) Wir müssen zeigen, dass $\lfloor \overline{T}(x) \rfloor_{l(x)} \in \left[u^{(m)}, o^{(m)} \right)$.
Es gilt $\lfloor \overline{T}(x) \rfloor_{l(x)} < o^{(m)}$, weil $\overline{T}(x) < o^{(m)}$.
Um die zweite Ungleichung zu zeigen, betrachten wir

$$\begin{aligned}
\overline{T}(x) - \lfloor \overline{T}(x) \rfloor_{l(x)} &< 2^{-l(x)} \\
&= \frac{1}{2^{\lceil \log \frac{1}{P(x)} \rceil + 1}} \\
&\leq \frac{1}{2^{\log \frac{1}{P(x)} + 1}} \\
&= \frac{1}{2\frac{1}{P(x)}} = \frac{P(x)}{2}.
\end{aligned}$$

Es ergibt sich also

$$\lfloor \overline{T}(x) \rfloor_{l(x)} > \overline{T}(x) - \tfrac{P(x)}{2} = u^{(m)}.$$

8.2 Das „Perpetuum mobile" der Datenkompression

a) Dies ist nicht möglich, wenn die Entropie der Daten 110 GB beträgt. In diesem Fall beträgt die mittlere Codewortlänge jedes binären Codes mindestens 110 GB, daher gibt es Codewörter, die länger als 100 GB sind und nicht auf die Festplatte passen.

b) Die Wahrscheinlichkeit, dass die (zufälligen) Daten wirklich so speziell sind, dass sie sich kurz beschreiben lassen, ist extrem klein. Wahrscheinlich ist das Kompressionsprogramm über Nacht den Testdaten angepasst worden und muss selbst eine Länge von mindestens 10 GB haben. Ein praktisch brauchbarer Kompressionsalgorithmus sollte aber universell sein, also nicht von den Daten abhängen. Beim zweiten Test sollten die Testdaten erst unmittelbar vor der Codierung erzeugt werden.

Das Beispiel widerspricht weder der Ungleichung $H(X) \leq E[l_C(X)]$ noch $E[l_C(X)] \leq H(X) + 1$ für den optimalen binären Code. Die in der Aufgabenstellung gemachte Aussage bezieht sich nämlich nicht auf die *mittlere* Codewortlänge, sondern auf diejenige Codewortlänge, die bei der Codierung der von Ihnen generierten Testdaten resultiert.

Kommentar: Diese Geschichte beruht auf Tatsachen. Der durchaus ehrliche Erfinder, welcher zwar nichts von Informationstheorie verstand, dafür aber von seinen eigenen Fähigkeiten umso mehr überzeugt war, glaubte sogar, sein Verfahren könne iterativ eingesetzt werden, um die Daten immer stärker zu komprimieren. Er bot sein Verfahren einem der bedeutendsten Hersteller mathematischer Software an.

8.3 Codierung der ganzen Zahlen

a) Die beiden Teile der Codewörter in C_1 und C_2 sind jeweils mit | getrennt.

j	$B(j)$	$L(j)$	$C_1(j)$	$C_2(j)$
1	1	1	|1	1|
2	10	2	0|10	010|0
3	11	2	0|11	010|1
4	100	3	00|100	011|00
5	101	3	00|101	011|01
6	110	3	00|110	011|10
7	111	3	00|111	011|11
8	1000	4	000|1000	00100|000
9	1001	4	000|1001	00100|001
10	1010	4	000|1010	00100|010

$C_1(2^{19}) = 00000000000000000|10000000000000000000$ (Länge 39), und
$C_2(2^{19}) = 000010100|0000000000000000000$ (Länge 28): für grosse Zahlen ist C_2 kürzer als C_1!

b) Weil $C_1(0)$ nicht definiert ist.

c) Für C_2' werden die ersten drei Codewörter abgeändert: $C_2'(1) = 10$, $C_2'(2) = 110$, $C_2'(3) = 111$. Es gilt:

$$E[l_{C_2}(X)] = P(1) + 4P(2) + 4P(3) + \sum_{x>3} P(x) l_{C_2}(x),$$

$$E[l_{C_2'}(X)] = 2P(1) + 3P(2) + 3P(3) + \sum_{x>3} P(x) l_{C_2'}(x).$$

Dann ist für alle Wahrscheinlichkeitsverteilungen mit $P(1) < P(2) + P(3)$:

$$\begin{aligned} E[l_{C_2}(X)] - E[l_{C_2'}(X)] &= (P(1) + 4P(2) + 4P(3)) \; - \; (2P(1) + 3P(2) + 3P(3)) \\ &= -P(1) + P(2) + P(3) \\ &> 0. \end{aligned}$$

8.4 Intervalllängen-Codierung

a) Die Sequenz $s = \langle a, c, c, b, b \rangle$ wird in die Sequenz $z = \langle 3, 2, 1, 5, 1 \rangle$ codiert.

b) In der Intervalllängen-Codierung merkt sich der Sender die letzte Position jedes Symbols in einem Array. Um das n-te Zeichen zu codieren, sucht man die letzte Position des Zeichens, aktualisiert diesen Wert und gibt dann die Differenz zu n aus. Dies geht in $O(1)$. Der Empfänger arbeitet analog.

c) Wäre a das nächste Zeichen, würde es zur 5, entsprechend b zur 1 und c zur 3 codiert werden.

Eine bessere Codierung bekommt man, wenn jedes Zeichen in die Anzahl *verschiedener* Zeichen bis hin zum letzten Auftreten codiert wird. Somit wird b zur 1, c zur 2 und a zur 3 codiert. Dies ist die sogenannte *Move-To-Front-Codierung*.

d) In der Move-To-Front-Codierung kann der Sender zum Beispiel eine lineare Liste der Länge $|\mathcal{X}|$ verwalten, in der die Symbole gemäss ihrem letzten Auftreten geordnet sind.

Um ein Zeichen zu codieren, gibt man die Position des Zeichens in der Liste aus und verschiebt das Zeichen dann an den Anfang der Liste. Das Verschieben geht schnell, das Suchen das Zeichens dauert aber $O(|\mathcal{X}|)$. Der Empfänger arbeitet analog.