Übungsblatt 1

Florian

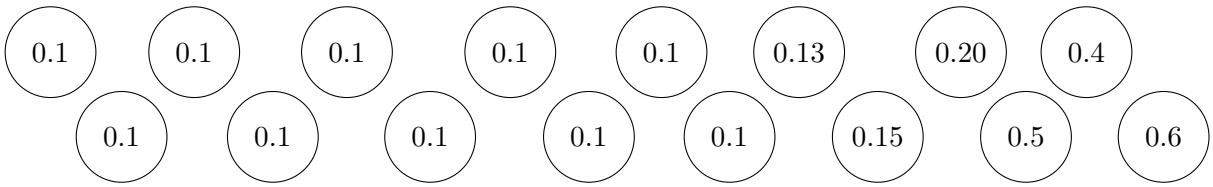
September 26, 2012

Aufgabe 1

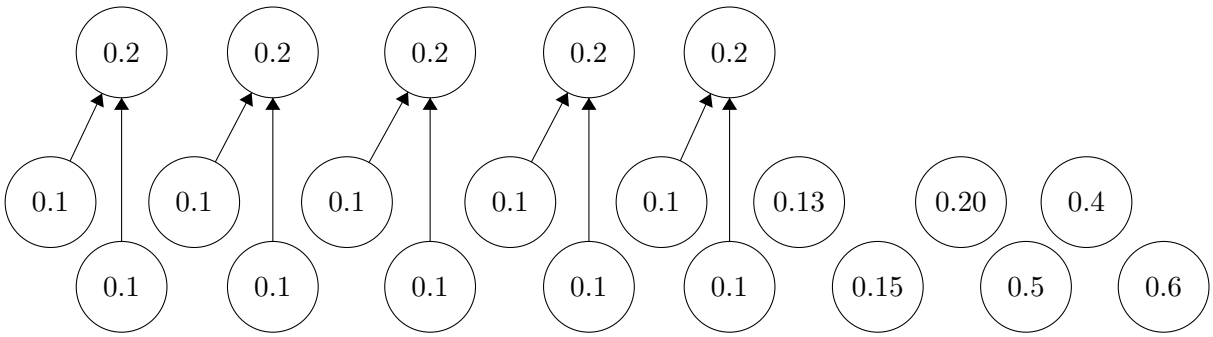
a) Die Auftretenswahrscheinlichkeiten $p(x)$ (wobei $\sum_{x \in A} p(x) = 2.98$) werden normiert auf 1.0.

$$\begin{aligned} E(A) &= \sum_{x \in A} p(x) \cdot \log_2 \frac{1}{p(x)} \\ &= 10 \cdot \frac{0.1}{2.98} \cdot \log_2 \frac{2.98}{0.1} + \frac{0.13}{2.98} \cdot \log_2 \frac{2.98}{0.13} + \frac{0.15}{2.98} \cdot \log_2 \frac{2.98}{0.15} \\ &\quad + \frac{0.20}{2.98} \cdot \log_2 \frac{2.98}{0.20} + \frac{0.5}{2.98} \cdot \log_2 \frac{2.98}{0.5} + \frac{0.4}{2.98} \cdot \log_2 \frac{2.98}{0.4} + \frac{0.6}{2.98} \cdot \log_2 \frac{2.98}{0.6} \\ &= 3.60565808338531\ldots \end{aligned}$$

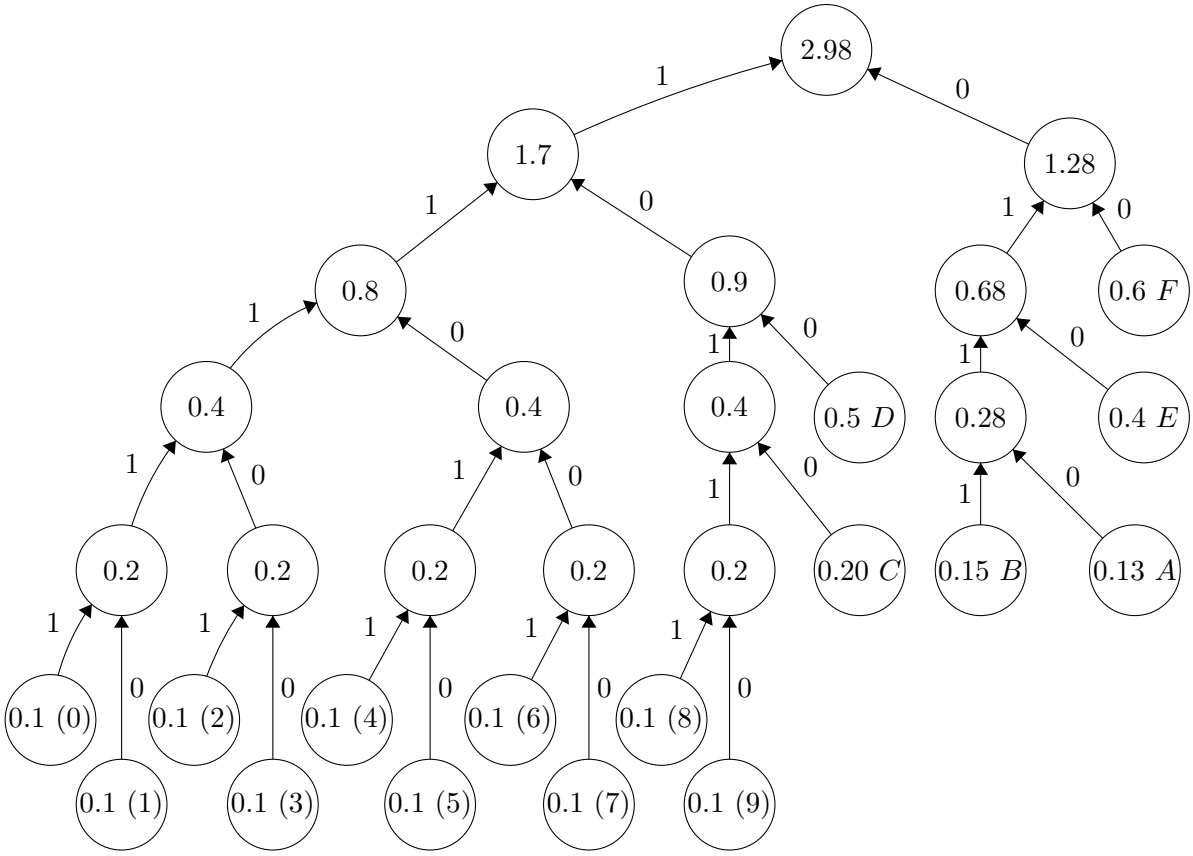
b) Nur Blätter:



Die 10 0.1-er zusammengefasst:



Und so weiter, und so fort bis zu:



Daraus folgt entlang der Pfeile der Code:

0 11111
1 11110
2 11101
3 11100
4 11011
5 11010
6 11001
7 11000
8 10111
9 10110
A 0110

B 0111

C 1010

D 100

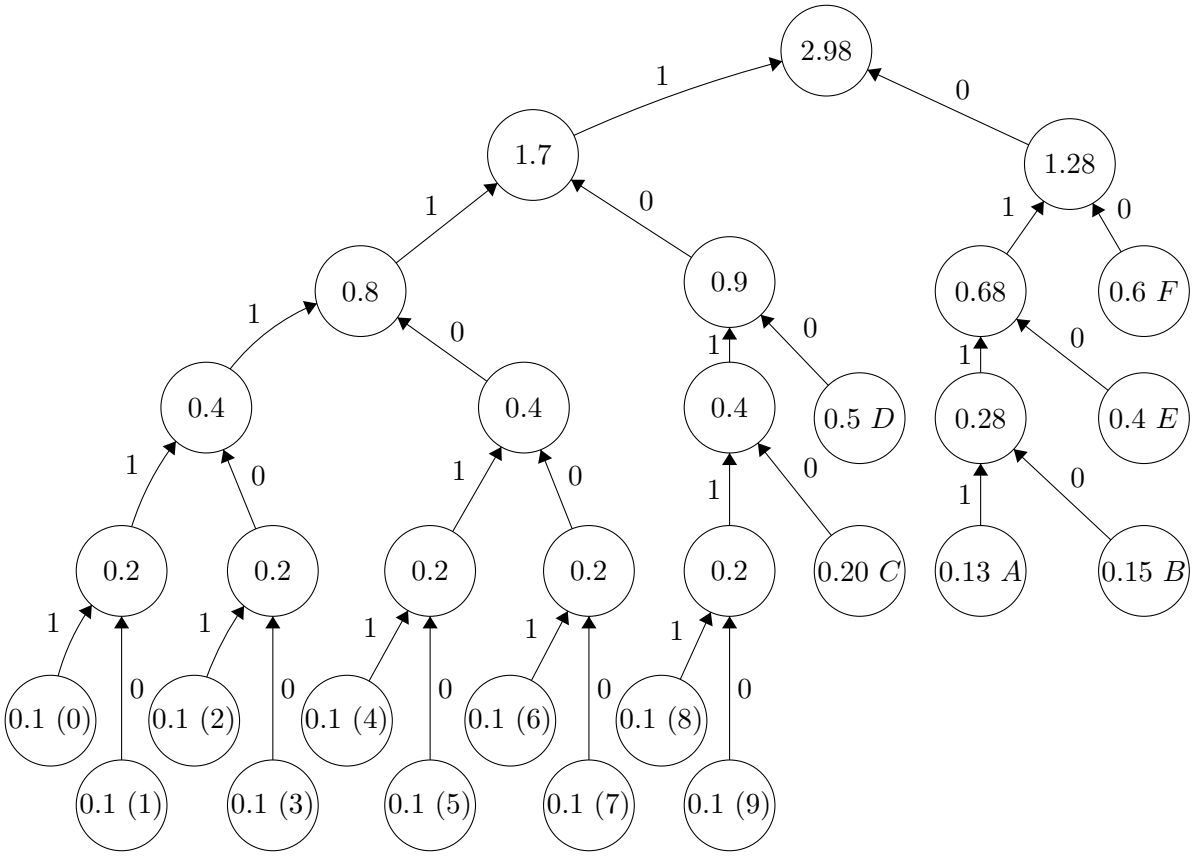
E 010

F 00

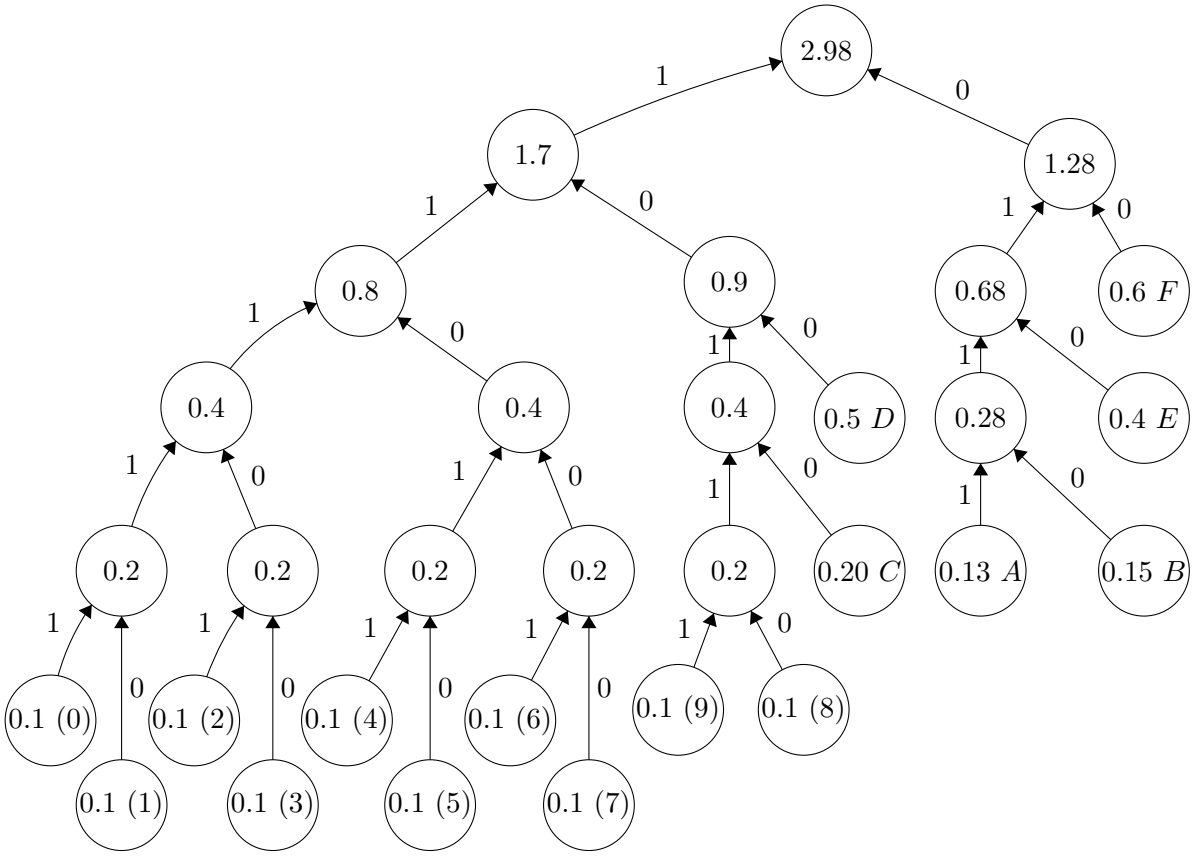
Die Code-Länge ist 5 (entspricht der Tiefe des Baums).

c) Der Baum ist nicht eindeutig.

Alternative Darstellungen:



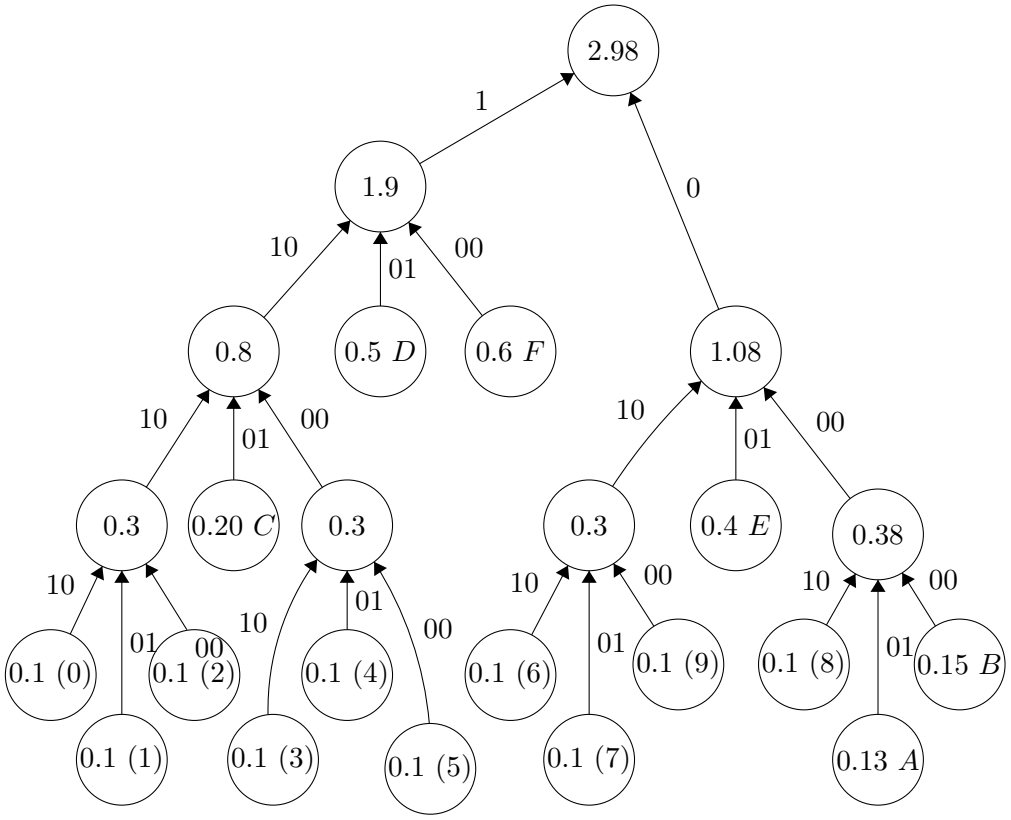
(A und B vertauscht)



(8 und 9 vertauscht)

d) Indem kein Binär-, sondern ein regulärer Baum mit beispielsweise 3 Kindkoten pro Knoten verwendet wird.

e) Regulärer 3-kindknotiger Baum:



f) Nein, weil auch die einzelnen Zeichen von Binärdaten wiederkehrende Häufigkeiten aufweisen (0 wird beispielsweise relativ häufig auftauchen).

Aufgabe 2

1. Man kann Redundanz in die übertragenen Daten einbauen und dann die überschüssige Information verwenden, um Fehlererkennung bzw. -korrektur zu betreiben.

2. Die Kompression sollte vor der Chiffrierung erfolgen, weil das Ziel der Chiffrierung eine möglichst zufällige Verteilung der Zeichen ist. Dies hat negative Auswirkungen auf die Kompression, weil diese sich normalerweise wiederauftretende Muster bzw. ungleiche Auftretenswahrscheinlichkeiten zu Nutze macht. Die Fehlerkorrektur kommt sinnvollerweise erst am Ende, damit während dem Empfangen gleich als erstes die Fehleranalyse gemacht werden kann und nicht erst die Daten dechiffriert und dekomprimiert werden müssen.

   Also: Kompression $\Rightarrow$ Chiffrierung $\Rightarrow$ Fehlerkorrektur beim Senden, Fehlerkorrektur $\Rightarrow$ Dechiffrierung $\Rightarrow$ Dekompression beim Empfangen.

3. • Generatormatrix:

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

• Allgemein:

$$\begin{bmatrix} p_1 & p_2 & d_3 & p_4 & d_5 & d_6 & d_7 \end{bmatrix} = \begin{bmatrix} d_3 & d_5 & d_6 & d_7 \end{bmatrix} G$$

Mögliche Codewörter:

$d_3d_5d_6d_7$	$p_1 = d_3 + d_5 + d_7$	$p_2 = d_3 + d_6 + d_7$	$p_4 = d_5 + d_6 + d_7$	$p_1p_2d_3p_4d_5d_6d_7$
0000	0	0	0	0000000
0001	1	1	1	1101001
0010	0	1	1	0101010
0011	1	2=0	2=0	1000011
0100	1	0	1	1001100
0101	2=0	1	2=0	0100101
0110	1	1	2=0	1100110
0111	2=0	2=0	3=1	0001111
1000	1	1	0	1110000
1001	2=0	2=0	1	0011001
1010	1	2=0	1	1011010
1011	2=0	3=1	2=0	0110011
1100	2=0	1	1	0111100
1101	3=1	2=0	2=0	1010101
1110	2=0	2=0	2=0	0010110
1111	3=1	3=1	3=1	1111111

• Indem wir die übertragenen Codewörter mit der Checkmatrix H multiplizieren und 0 als Resultat erwarten.

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Beispielsweise für das Datenwort 0010:

$$H \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}^T = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Wenn hingegen ein Übertragungsfehler stattfindet, beispielsweise das dritte Bit gedreht ist:

$$H \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}^T = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

- Der resultierende Vektor ist die binäre Repräsentation der Fehlerstelle: $011_b = 3$. Dementsprechend können wir diese Stelle korrigieren und erhalten das gültige, beabsichtigte Codewort.

Aufgabe 3

Siehe `code/Aufgabe3.py`. Zwar ein sehr naiver Ansatz, scheint aber zu funktionieren.

Aufgabe 4

(1) Nun denn:

```
florian@debian:~$ python Huffman3.py
Reading in frequencies (and optional symbol names) from stdin
0.1 1
0.1 2
0.1 3
0.1 4
0.1 5
0.1 6
0.1 7
0.1 8
0.1 9
0.13 A
0.15 B
0.2 C
0.5 D
0.4 E
0.6 F
#Symbol Count Codeword
1 (0.1) 10000
2 (0.1) 10001
3 (0.1) 10110
4 (0.1) 10111
5 (0.1) 10100
6 (0.1) 10101
7 (0.1) 00110
8 (0.1) 00111
9 (0.1) 0000
A (0.13) 0001
B (0.15) 0010
C (0.2) 1001
D (0.5) 111
```

```
E (0.4) 110
F (0.6) 01
```

(2) Ich verstehe leider die Aufgabe nicht, werde diese aber noch nachliefern, sobald ich sie verstanden habe. Das Ergebnis von (1) ist ein Huffman-Code über das Alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$. Inwiefern kann dieser Code runlength-encoded werden?

(3) web.tar:

```
florian@debian:~/njhuffman$ python huffman.py compress web.tar
Done. Your compressed file is saved as: webCompressed.txt
Original: 4510.0 KB; Compressed: 2651.79101562 KB
Compression rate of 0.58798026954
```

$$\begin{aligned} \text{Mass} &= \frac{\text{Grösse Ausgabedatei}}{\text{Grösse Eingabedatei}} = \frac{2651.8 \text{ kB}}{4510.0 \text{ kB}} = 0.588 \\ \text{Faktor} &= \frac{1}{\text{Mass}} = 1.701 \end{aligned}$$

dkbibel.txt:

```
florian@debian:~/njhuffman$ python huffman.py compress dkbibel.txt
Done. Your compressed file is saved as: dkbibelCompressed.txt
Original: 3913.73632812 KB; Compressed: 2234.33105469 KB
Compression rate of 0.570894630441
```

$$\begin{aligned} \text{Mass} &= \frac{\text{Grösse Ausgabedatei}}{\text{Grösse Eingabedatei}} = \frac{2234.3 \text{ kB}}{3913.7 \text{ kB}} = 0.570 \\ \text{Faktor} &= \frac{1}{\text{Mass}} = 1.752 \end{aligned}$$

ylt.tar:

```
florian@debian:~/njhuffman$ python huffman.py compress ylt.tar
Done. Your compressed file is saved as: yltCompressed.txt
Original: 4430.0 KB; Compressed: 2571.31835938 KB
Compression rate of 0.580433038234
```

$$\text{Mass} = \frac{\text{Grösse Ausgabedatei}}{\text{Grösse Eingabedatei}} = \frac{2571.3\text{ kB}}{4430.0\text{ kB}} = 0.580$$
$$\text{Faktor} = \frac{1}{\text{Mass}} = 1.724$$

Grundsätzlich kann man sagen, dass Mass und Faktor immer ähnlich sind. Wahrscheinlich liegt das daran, dass erstens alles Text ist und zweitens Dänisch und Englisch sehr ähnliche Sprachen sind.

Aufgabe 5

Siehe `code/Aufgabe5.py`. Mein erstes sinnvolles Python-Programm! Woohoo!

(1) web.tar mit Kindknotenanzahl = 2:

```
florian@debian:~/workspace/WissenschaftlichesRechnen/Uebung1/code$ python3 Aufgabe5.p
char: ' ', freq: 811090, code: 111
char: 'e', freq: 409443, code: 000
char: 't', freq: 312296, code: 1010
char: 'h', freq: 279434, code: 1000
char: 'a', freq: 260983, code: 0111
char: 'o', freq: 238223, code: 0101
char: 'n', freq: 216380, code: 0011
char: 's', freq: 190049, code: 11011
char: 'i', freq: 184095, code: 11010
char: 'r', freq: 164784, code: 10111
char: 'd', freq: 151520, code: 10011
char: 'l', freq: 123766, code: 01100
char: '
', freq: 86140, code: 110010
char: 'f', freq: 81601, code: 110000
char: 'm', freq: 78906, code: 101101
char: 'u', freq: 73442, code: 100101
char: ',', freq: 72799, code: 100100
char: 'w', freq: 63950, code: 011011
char: 'y', freq: 60753, code: 010011
char: '', freq: 56563, code: 010010
char: 'c', freq: 53868, code: 001011
char: 'g', freq: 49430, code: 001010
char: 'b', freq: 44684, code: 001000
char: ':', freq: 44077, code: 1100111
char: 'p', freq: 42032, code: 1100011
```

```
char: 'v', freq: 31138, code: 0110100
char: '1', freq: 28209, code: 0100011
char: '.', freq: 28129, code: 0100001
char: 'k', freq: 21765, code: 11001101
char: '[', freq: 21357, code: 11000101
char: ']', freq: 21357, code: 11001100
char: '2', freq: 19689, code: 10110011
char: 'A', freq: 18138, code: 10110001
char: 'I', freq: 13667, code: 01000100
char: '3', freq: 12770, code: 00100111
char: ';', freq: 10180, code: 110001001
char: '4', freq: 9759, code: 101100101
char: 'L', freq: 9307, code: 101100100
char: 'O', freq: 8877, code: 101100000
char: 'D', freq: 8874, code: 011010111
char: 'T', freq: 7959, code: 011010101
char: 'R', freq: 7545, code: 011010100
char: '5', freq: 7406, code: 010001011
char: '6', freq: 6825, code: 010000011
char: 'J', freq: 6508, code: 010000001
char: '7', freq: 6452, code: 010000000
char: 'G', freq: 6031, code: 001001100
char: '8', freq: 5914, code: 001001011
char: '0', freq: 5778, code: 001001010
char: '9', freq: 5762, code: 001001001
char: 'S', freq: 5288, code: 1100010001
char: 'B', freq: 4843, code: 1100010000
char: '-', freq: 4221, code: 1011000010
char: '', freq: 3533, code: 0100010100
char: 'H', freq: 3305, code: 0100000101
char: '?', freq: 3284, code: 0100000100
char: 'M', freq: 3244, code: 0010011011
char: 'E', freq: 2816, code: 0010010001
char: 'j', freq: 2544, code: 0010010000
char: 'W', freq: 2430, code: 10110000111
char: 'F', freq: 2373, code: 10110000110
char: 'z', freq: 2147, code: 01101011011
char: 'P', freq: 1946, code: 01101011001
char: 'N', freq: 1931, code: 01101011000
char: 'C', freq: 1858, code: 01000101011
char: 'x', freq: 1722, code: 01000101010
char: 'q', freq: 977, code: 011010110100
char: 'Z', freq: 927, code: 001001101011
```

```
char: 'K', freq: 585, code: 001001101000
char: 'Y', freq: 549, code: 0110101101011
char: '(', freq: 339, code: 0010011010100
char: ')', freq: 338, code: 0010011010011
char: '!', freq: 306, code: 0010011010010
char: 'U', freq: 287, code: 01101011010101
char: '>', freq: 238, code: 01101011010100
char: 'V', freq: 122, code: 001001101010111
char: '"', freq: 106, code: 001001101010110
char: '&', freq: 49, code: 0010011010101011
char: '(pfeil)', freq: 38, code: 00100110101010101
char: 'X', freq: 23, code: 0010011010101000
char: '/', freq: 10, code: 001001101010100111
char: 'Q', freq: 5, code: 001001101010100100
char: '
       ', freq: 3, code: 0010011010101001011
char: '$', freq: 2, code: 00100110101010011001
char: '{', freq: 2, code: 00100110101010011010
char: '*', freq: 2, code: 00100110101010011011
char: '}', freq: 2, code: 0010011010101001010
char: '@', freq: 1, code: 00100110101010011000
entropy: 4.61644677939
input length: 4532100; output length: 2638807.125 => ratio: 0.582248212749
tree depth: 20; code length: 20
```

web.tar mit Kindknotenanzahl 3:

```
florian@debian:~/workspace/WissenschaftlichesRechnen/Uebung1/code$ python3 Aufgabe5.p
char: ' ', freq: 811090, code: 0100
char: 'e', freq: 409443, code: 011010
char: 't', freq: 312296, code: 011000
char: 'h', freq: 279434, code: 010110
char: 'a', freq: 260983, code: 010100
char: 'o', freq: 238223, code: 001010
char: 'n', freq: 216380, code: 001000
char: 's', freq: 190049, code: 000101
char: 'i', freq: 184095, code: 000100
char: 'r', freq: 164784, code: 000001
char: 'd', freq: 151520, code: 01100110
char: 'l', freq: 123766, code: 01100100
char: '
', freq: 86140, code: 01010101
char: 'f', freq: 81601, code: 00100110
char: 'm', freq: 78906, code: 00100101
```

```
char: 'u', freq: 73442, code: 00100100
char: ',', freq: 72799, code: 00011010
char: 'w', freq: 63950, code: 00011000
char: 'y', freq: 60753, code: 00001001
char: '', freq: 56563, code: 00001000
char: 'c', freq: 53868, code: 00000001
char: 'g', freq: 49430, code: 00000000
char: 'b', freq: 44684, code: 0110010101
char: ':', freq: 44077, code: 0110010100
char: 'p', freq: 42032, code: 0101011010
char: 'v', freq: 31138, code: 0101011000
char: '1', freq: 28209, code: 0101010010
char: '.', freq: 28129, code: 0101010001
char: 'k', freq: 21765, code: 0001100100
char: '[', freq: 21357, code: 0000101001
char: ']', freq: 21357, code: 0000101010
char: '2', freq: 19689, code: 0000001010
char: 'A', freq: 18138, code: 0000001001
char: 'I', freq: 13667, code: 011001011000
char: '3', freq: 12770, code: 010101100110
char: ';', freq: 10180, code: 010101100101
char: '4', freq: 9759, code: 010101000010
char: 'L', freq: 9307, code: 010101000001
char: 'O', freq: 8877, code: 000110011010
char: 'D', freq: 8874, code: 000110011001
char: 'T', freq: 7959, code: 000110011000
char: 'R', freq: 7545, code: 000110010110
char: '5', freq: 7406, code: 000110010101
char: '6', freq: 6825, code: 000010100010
char: 'J', freq: 6508, code: 000010100001
char: '7', freq: 6452, code: 000010100000
char: 'G', freq: 6031, code: 000000100001
char: '8', freq: 5914, code: 000000100000
char: '0', freq: 5778, code: 01100101101010
char: '9', freq: 5762, code: 01100101101001
char: 'S', freq: 5288, code: 01100101100110
char: 'B', freq: 4843, code: 01100101100101
char: '-', freq: 4221, code: 01100101100100
char: ''', freq: 3533, code: 01010110010010
char: 'H', freq: 3305, code: 01010110010001
char: '?', freq: 3284, code: 01010110010000
char: 'M', freq: 3244, code: 01010100000010
char: 'E', freq: 2816, code: 01010100000000
```

```
char: 'j', freq: 2544, code: 00011001010010
char: 'W', freq: 2430, code: 00011001010001
char: 'F', freq: 2373, code: 00011001010000
char: 'z', freq: 2147, code: 00000010001010
char: 'P', freq: 1946, code: 00000010001000
char: 'N', freq: 1931, code: 0110010110100010
char: 'C', freq: 1858, code: 0110010110100001
char: 'x', freq: 1722, code: 0110010110100000
char: 'q', freq: 977, code: 0101010000000101
char: 'Z', freq: 927, code: 0101010000000100
char: 'K', freq: 585, code: 0000001000100101
char: 'Y', freq: 549, code: 0000001000100100
char: '(', freq: 339, code: 010101000000011001
char: ')', freq: 338, code: 010101000000011000
char: '!', freq: 306, code: 000000100010011010
char: 'U', freq: 287, code: 000000100010011001
char: '>', freq: 238, code: 000000100010011000
char: 'V', freq: 122, code: 01010100000001101001
char: '"', freq: 106, code: 01010100000001101000
char: '&', freq: 49, code: 010101000000110101001
char: '(pfeil)', freq: 38, code: 0101010000000110101000
char: 'X', freq: 23, code: 0101010000000110101010
char: '/', freq: 10, code: 0101010000000110101000
char: 'Q', freq: 5, code: 010101000000011010100100
char: '
       ', freq: 3, code: 0101010000000110101010011010
char: '$', freq: 2, code: 0101010000000110101010010101
char: '{', freq: 2, code: 0101010000000110101010010110
char: '*', freq: 2, code: 0101010000000110101010011000
char: '}', freq: 2, code: 0101010000000110101010011001
char: '@', freq: 1, code: 0101010000000110101010010100
entropy: 4.61644677939
input length: 4532100; output length: 3757985.0 => ratio: 0.829192868648
tree depth: 14; code length: 28
```

Es ist festzustellen, dass die Kompressionswerte in die Nähe des vom Internet geladenen Programms kommen. Allerdings ist auch festzustellen, dass eine Kindknotenanzahl von 2 mit Abstand am besten funktioniert. Ich nehme an, das ist darum, weil der Code schlussendlich binär ist, und darum sowieso nur Kindknotenanzahlen, die Zweierpotenzen sind, Sinn machen, weil ja sonst der Code an sich schon Redundanz enthält. Damit dann möglichst gut auf die diskreten Werte der Häufigkeiten eingegangen werden kann, macht eine kürzere Einzelcodelänge ebenfalls Sinn – das Optimierungspotenzial steckt dann jeweils in der Baumtiefe.

(2) Diese Aufgabe schenke ich mir. Sollte aber keine grosse Sache sein. Die Entropie wird bereits berechnet und ausgewiesen.