

Grundlagen der Informatik

Prof. Dr. J. Schmidt

Fakultät für Informatik

GDI – WS 2018/19

Information und Quellencodierung
Huffman-/Fano-Algorithmus



- Code-Bäume
- Huffman-Algorithmus
- Fano-Algorithmus



- Zielsetzung
 - Erzeugung eines Codes mit **variabler** Wortlänge
 - bei einem gegebenen Alphabet von Zeichen
 - mit bekannten Auftrittswahrscheinlichkeiten
- Mögliche Vorgehensweise
 - Bestimmung der Wortlänge aus den ganzzahlig aufgerundeten Informationsgehalten
 - Anordnung der Code-Wörter als Endknoten (Blätter) eines Code-Baums



● Beispiel Code-Erzeugung

- 6 Buchstaben {c,v,w,u,r,z} sollen mit möglichst geringer Redundanz binär kodiert werden

- Auftrittswahrscheinlichkeiten

$$p(c) = 0.1643$$

$$p(v) = 0.0455$$

$$p(w) = 0.0874$$

$$p(u) = 0.1963$$

$$p(r) = 0.4191$$

$$p(z) = 0.0874$$



● Beispiel Code-Erzeugung

● Berechnung der Informationsgehalte der jeweiligen Zeichen

$$p(c) = 0.1643 \quad I(c) = 2.6056 \text{ Bit}$$

$$p(v) = 0.0455 \quad I(v) = 4.4580 \text{ Bit}$$

$$p(w) = 0.0874 \quad I(w) = 3.5162 \text{ Bit}$$

$$p(u) = 0.1963 \quad I(u) = 2.3489 \text{ Bit}$$

$$p(r) = 0.4191 \quad I(r) = 1.2546 \text{ Bit}$$

$$p(z) = 0.0874 \quad I(z) = 3.5162 \text{ Bit}$$



● Beispiel Code-Erzeugung

- Berechnung der Informationsgehalte der jeweiligen Zeichen und Ableitung der Wortlänge

$$I(c) = 2.6056 \text{ Bit} \rightarrow l(c) = 3 \text{ Bit}$$

$$I(v) = 4.4580 \text{ Bit} \rightarrow l(v) = 5 \text{ Bit}$$

$$I(w) = 3.5162 \text{ Bit} \rightarrow l(w) = 4 \text{ Bit}$$

$$I(u) = 2.3489 \text{ Bit} \rightarrow l(u) = 3 \text{ Bit}$$

$$I(r) = 1.2546 \text{ Bit} \rightarrow l(r) = 2 \text{ Bit}$$

$$I(z) = 3.5162 \text{ Bit} \rightarrow l(z) = 4 \text{ Bit}$$



● Beispiel Code-Erzeugung

- Berechnung der Informationsgehalte der jeweiligen Zeichen und Ableitung der Wortlänge mit anschließender Kodierung

$I(c)$	=	2.6056 Bit	➔	$l(c)$	=	3 Bit	001	} Exemplarischer Code
$I(v)$	=	4.4580 Bit	➔	$l(v)$	=	5 Bit	10111	
$I(w)$	=	3.5162 Bit	➔	$l(w)$	=	4 Bit	0001	
$I(u)$	=	2.3489 Bit	➔	$l(u)$	=	3 Bit	011	
$I(r)$	=	1.2546 Bit	➔	$l(r)$	=	2 Bit	11	
$I(z)$	=	3.5162 Bit	➔	$l(z)$	=	4 Bit	0000	



- Nun: Bestimmung der Redundanz dieses Codes

Zeichen	$p(c)$	$I(c)$	$l(c)$
c	0.1643	2.6056	3
v	0.0455	4.4580	5
w	0.0874	3.5162	4
u	0.1963	2.3489	3
r	0.4191	1.2546	2
z	0.0874	3.5162	4



- **Entropie** dieser Nachrichtenquelle

$$\begin{aligned} H &= p(c)I(c) + p(v)I(v) + p(w)I(w) + p(u)I(u) + p(r)I(r) + p(z)I(z) \\ &\approx 0.4282 + 0.2028 + 0.3073 + 0.4611 + 0.5260 + 0.3073 \\ &= 2.2327 \text{ Bit/Zeichen} \end{aligned}$$

- maximale **Entropie** dieser Nachrichtenquelle

$$H_0 = \lg 6 = 2.5850$$

- **Mittlere Wortlänge**

$$\begin{aligned} L &= p(c)l(c) + p(v)l(v) + p(w)l(w) + p(u)l(u) + p(r)l(r) + p(z)l(z) \\ &= 0.4929 + 0.2275 + 0.3496 + 0.5889 + 0.8382 + 0.3496 \\ &= 2.8467 \text{ Bit/Zeichen} \end{aligned}$$

Zeichen	$p(c)$	$I(c)$	$l(c)$
c	0.1643	2.6056	3
v	0.0455	4.4580	5
w	0.0874	3.5162	4
u	0.1963	2.3489	3
r	0.4191	1.2546	2
z	0.0874	3.5162	4

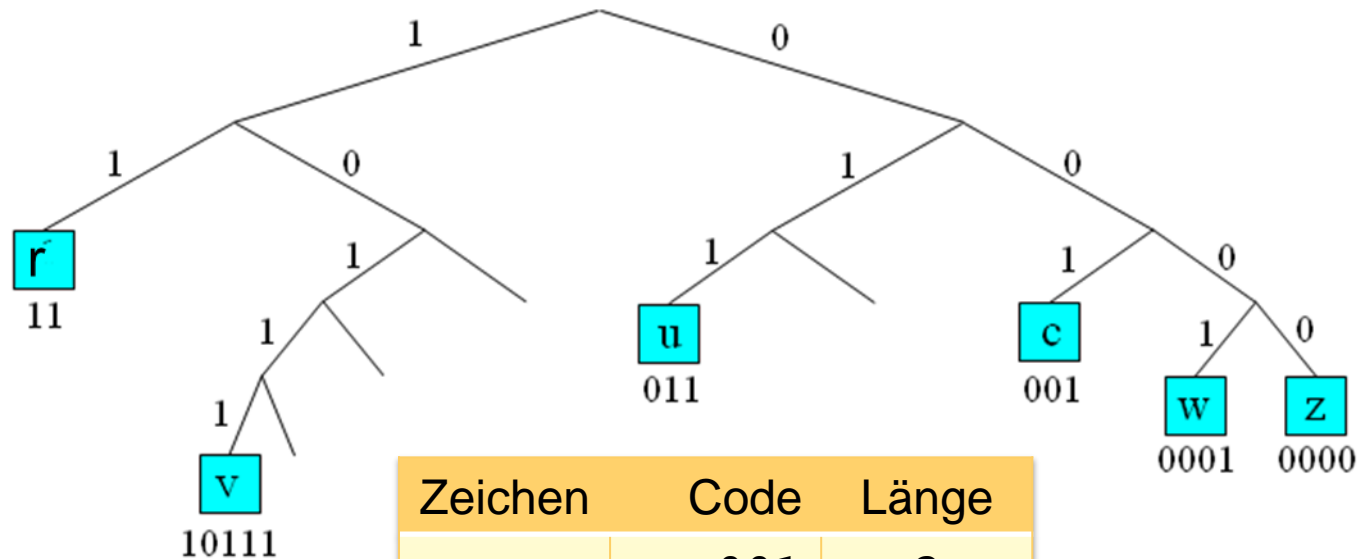
- **Redundanz**

$$\begin{aligned} R_C &= L - H \\ &\approx 2.8467 - 2.2327 \\ &= 0.6140 \text{ Bit/Zeichen} \end{aligned}$$

$$\begin{aligned} R_Q &= H_0 - H \\ &\approx 2.5850 - 2.2327 \\ &= 0.3523 \text{ Bit/Zeichen} \end{aligned}$$

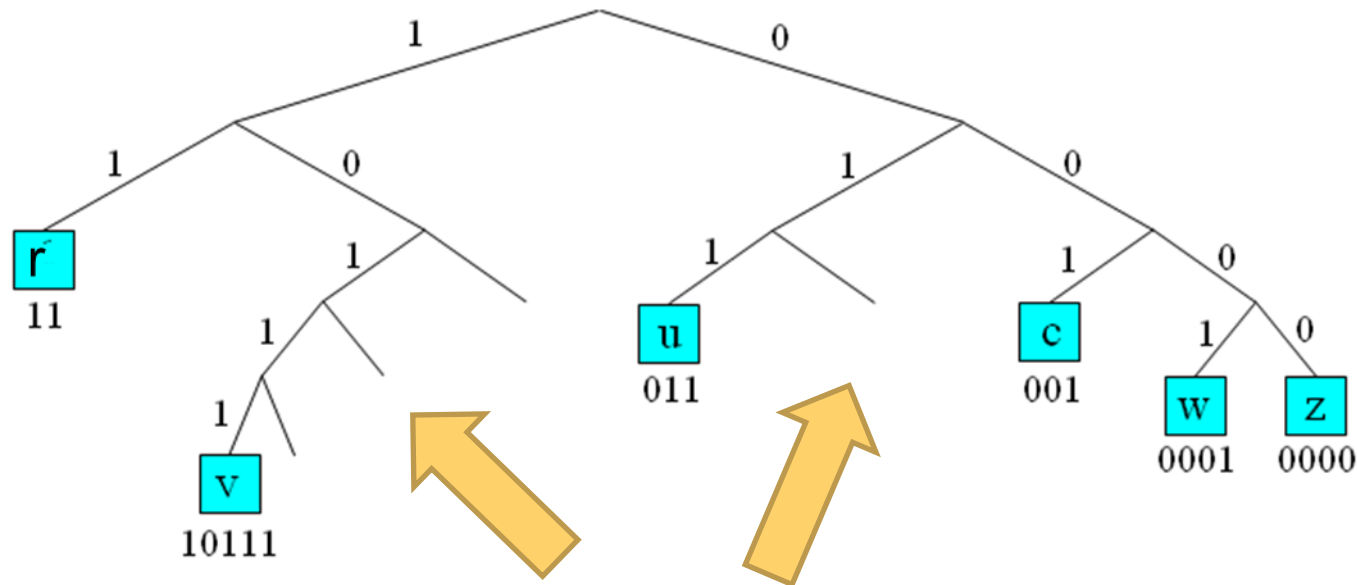


● Visualisierung der Kodierung



Zeichen	Code	Länge
<i>c</i>	001	3
<i>v</i>	10111	5
<i>w</i>	0001	4
<i>u</i>	011	3
<i>r</i>	11	2
<i>z</i>	0000	4

● Visualisierung der Kodierung

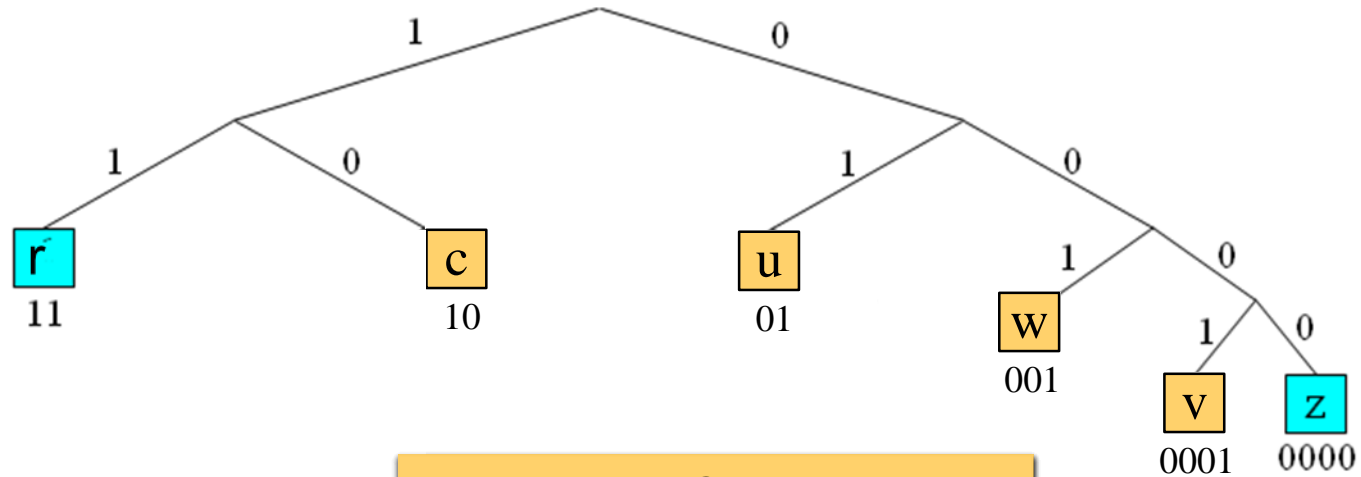


- Unbesetzte Blätter (Endknoten) sind vorhanden, die näher an der Wurzel liegen
- **Kein optimaler Code:** kürzere Codewörter wären möglich

- Verbesserungen im Code-Baum
 - Freie Blätter werden mit Zeichen besetzt,
 - deren Wortlänge größer ist als die zu dem freien Blatt gehörende Wortlänge
 - Beachte!
 - Ein Code für ein Zeichen mit geringerer Auftrittswahrscheinlichkeit muss länger sein als der Code für ein Zeichen mit höherer Auftrittswahrscheinlichkeit



● Verbesserter Code-Baum



Zeichen	Code	Länge
<i>c</i>	10	2
<i>v</i>	0001	4
<i>w</i>	001	3
<i>u</i>	01	2
<i>r</i>	11	2
<i>z</i>	0000	4

● Aufgabe

- Bestimmen Sie die Redundanz des verbesserten Codes!

Zeichen	$p(c)$	$I(c)$	$l(c)$
c	0.1643	2.6056	2
v	0.0455	4.4580	4
w	0.0874	3.5162	3
u	0.1963	2.3489	2
r	0.4191	1.2546	2
z	0.0874	3.5162	4

● Ergebnisse der Optimierung

● Mittlere Wortlänge des verbesserten Codes

$$L = (0.4191 + 0.1963 + 0.1643) \cdot 2 + \\ 0.0874 \cdot 3 + \\ (0.0874 + 0.0455) \cdot 4$$

$$= 2.3532 \text{ Bit/Zeichen}$$

● Redundanz

$$R \approx 2.3523 - 2.2327$$

$$= 0.1205 \text{ Bit/Zeichen}$$

● Redundanz wurde reduziert

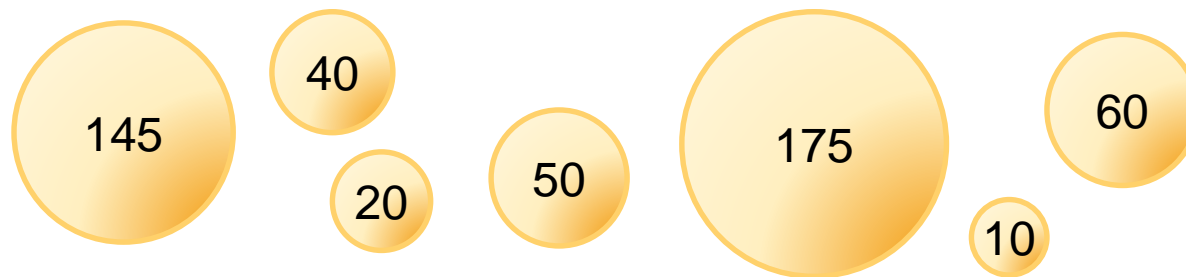
- von 0.6140 Bit/Zeichen
auf 0.1205 Bit/Zeichen

Zeichen	$p(c)$	$I(c)$	$l(c)$
c	0.1643	2.6056	2
v	0.0455	4.4580	4
w	0.0874	3.5162	3
u	0.1963	2.3489	2
r	0.4191	1.2546	2
z	0.0874	3.5162	4



- Konstruieren Sie ein Mobile!

- Gegeben: 7 verschiedene Weihnachtskugeln mit jeweils folgenden Gewichten in Gramm



- Gesucht: Bestmöglich ausbalanciertes Mobile
(also mit möglichst kleiner mittlerer Differenz zwischen linker und rechter Seite)

Aufgabe

Kapitel 3: Information und Quellencodierung – Huffman/Fano



- Allgemeines Verfahren
 - zur Erzeugung von optimalen Codes
 - bzgl. des Kriteriums **Redundanzminimierung**
- 1952 vom amerikanischen Mathematiker **David A. Huffman** formuliert



● Vorgehen

1. Ordne alle Zeichen nach ihren Auftrittswahrscheinlichkeiten (Knoten)
2. Fasse die beiden Zeichen mit den geringsten Wahrscheinlichkeiten p_1 und p_2 zu einem Knoten zusammen
 - Wahrscheinlichkeit = $p_1 + p_2$
 - Ergebnis: Neue Folge von Wahrscheinlichkeiten
3. Fasse die beiden Elemente (Zeichen oder Knoten) mit den geringsten Wahrscheinlichkeiten zu einem neuen Knoten zusammen
4. Wiederhole 3. solange bis nur noch ein Baum übrig ist (Huffman-Baum)



- Schrittweiser Aufbau des Huffman-Baum und Huffman-Code
- Gegeben 6 Buchstaben {c,v,w,u,r,z} mit den Auftrittswahrscheinlichkeiten

$$p(c) = 0.1643$$

$$p(v) = 0.0455$$

$$p(w) = 0.0874$$

$$p(u) = 0.1963$$

$$p(r) = 0.4191$$

$$p(z) = 0.0874$$



- Schrittweiser Aufbau des Huffman-Baums und Huffman-Codes

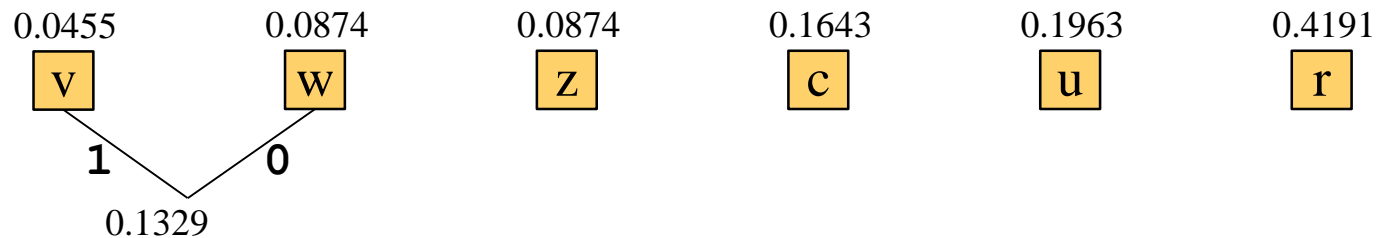
0.1643	0.0455	0.0874	0.1963	0.4191	0.0874
c	v	w	u	r	z

- Schritt 1

0.0455	0.0874	0.0874	0.1643	0.1963	0.4191
v	w	z	c	u	r

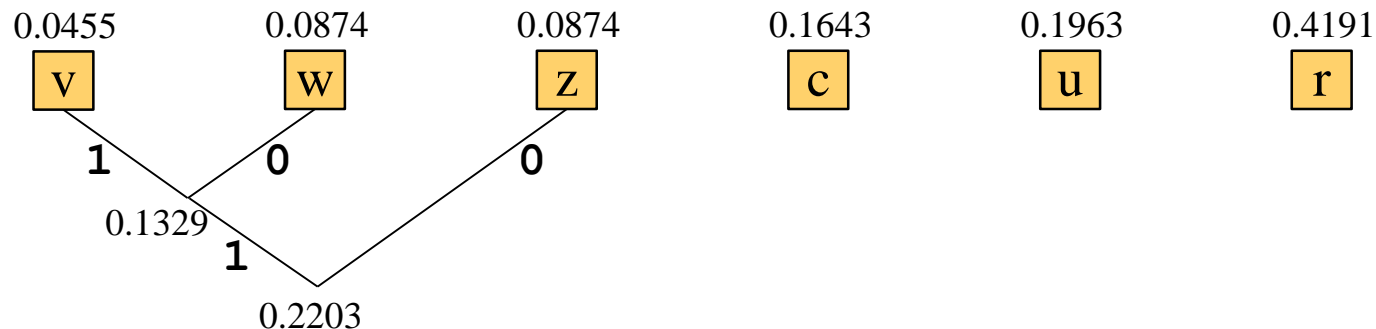
- Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

- Schritt 2



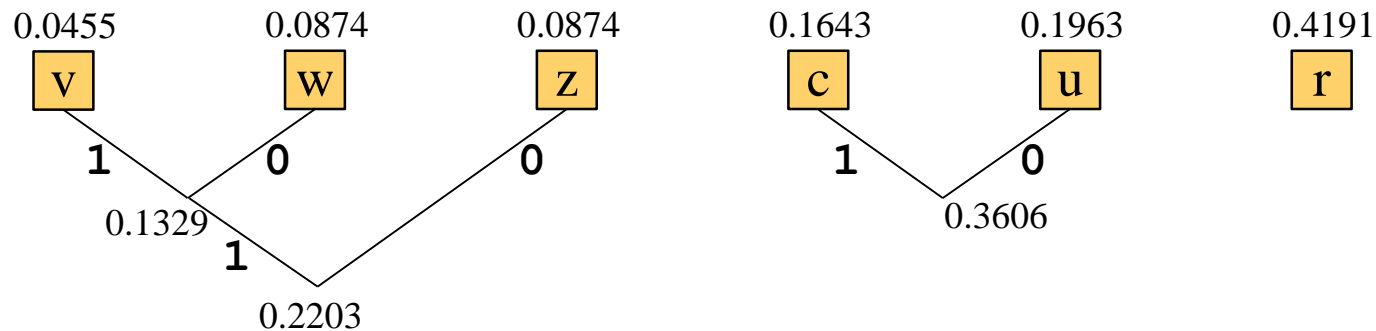
- Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

- Schritt 3



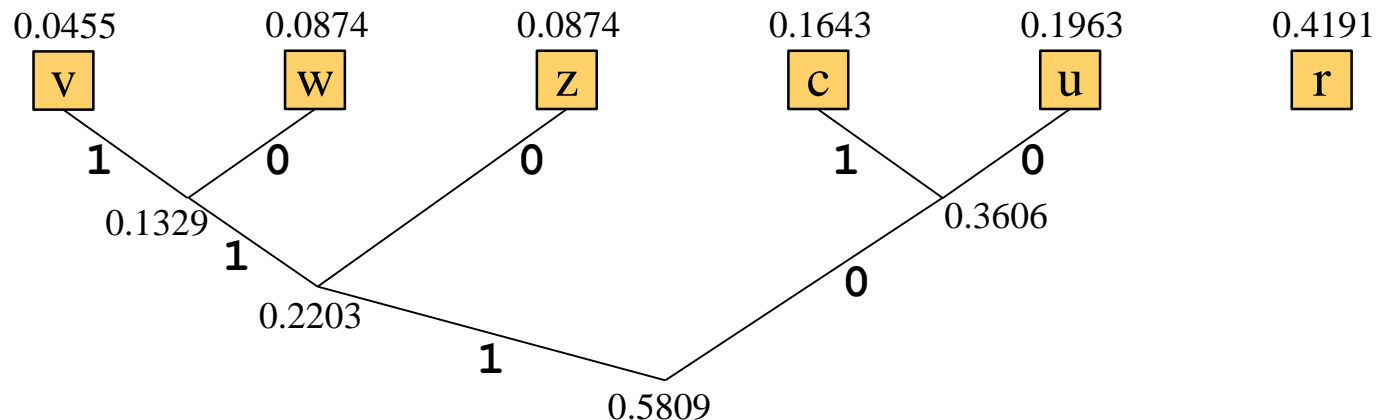
- Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

- Schritt 4



- Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

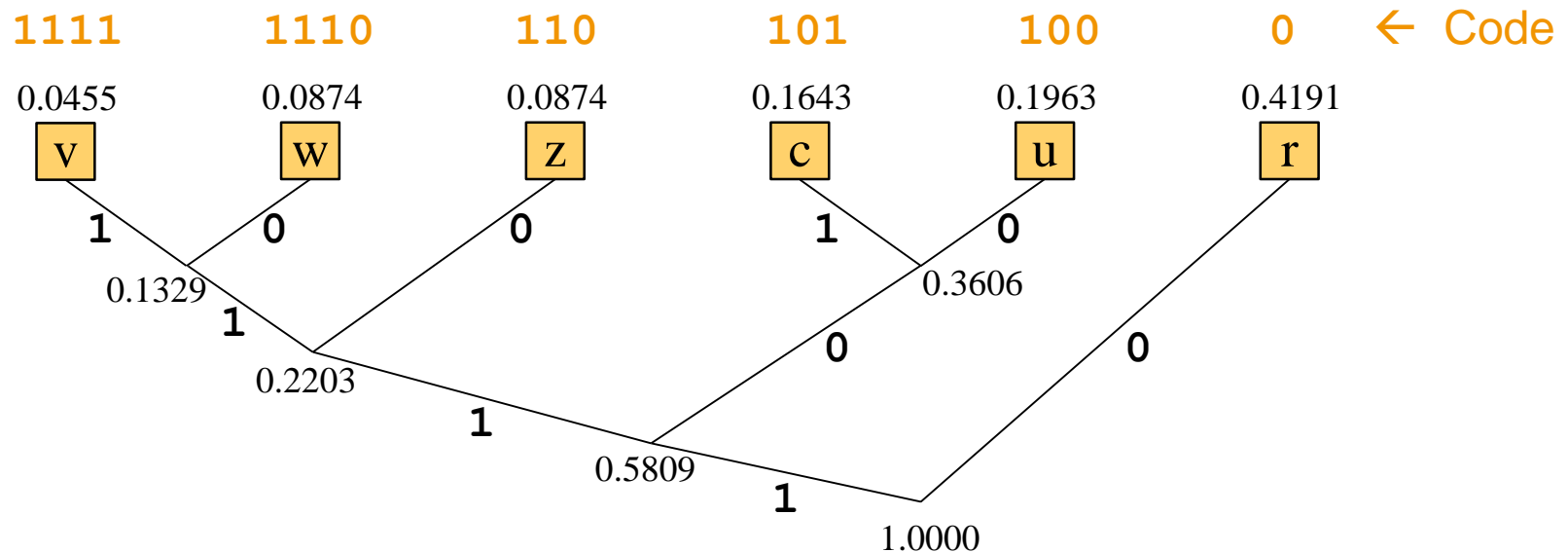
- Schritt 5



Huffman-Algorithmus (9)

● Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

● Schritt 6



- Nun: Bestimmung der Redundanz des Huffman-Codes

Zeichen	$p(c)$	$I(c)$	$l(c)$
c	0.1643	2.6056	3
v	0.0455	4.4580	4
w	0.0874	3.5162	4
u	0.1963	2.3489	3
r	0.4191	1.2546	1
z	0.0874	3.5162	3

- Charakterisierung des erzeugten Codes:

- $$L = 0.4191 + (0.1963 + 0.1643 + 0.0874) \cdot 3 + (0.0874 + 0.0455) \cdot 4 = 2.2947 \text{ Bit/Zeichen}$$

- Redundanz
$$R = 2.2947 - 2.2327 = 0.0620 \text{ Bit/Zeichen}$$

- Vergleich mit vorherigen Codes

- Erst wurde Redundanz von 0.6140 Bit/Zeichen auf 0.1205 Bit/Zeichen reduziert
- Nun weitere Reduzierung der Redundanz auf 0.062 Bit/Zeichen



- Wesentliche Forderung an einen Code
 - Eindeutige Dekodierung
- Fano-Bedingung (auch: Präfixcode)
 - Code mit variabler Wortlänge muss so generiert werden, dass kein Code-Wort eines Zeichens mit dem Anfang des Code-Wortes irgendeines anderen Zeichens übereinstimmt
 - Code-Wörter dürfen nur an den Blättern (Endknoten) des Code-Baums stehen



● Vorgehensweise

1. Die zu interpretierenden Zeichen werden Bit für Bit in einem Puffer gesammelt und laufend mit den tabellierten Codes verglichen
2. Sobald der Pufferinhalt mit einem tabellierten Code-Wort übereinstimmt, wird Dekodierung durchgeführt.
3. Dann wird Puffer gelöscht und der Dekodier-Vorgang beginnt von neuem für das nächste Zeichen.



● Aufgabe

- Dekodiere die Zeichenkette 1011001011000100110 mithilfe des vorherigen Huffman-Codes

Zeichen	Code	Länge
<i>c</i>	101	3
<i>v</i>	1111	4
<i>w</i>	1110	4
<i>u</i>	100	3
<i>r</i>	0	1
<i>z</i>	110	3

- Lösung:

● Vorgehensweise

1. Anordnung der zu kodierenden Zeichen x_i und die zugehörigen Auftrittswahrscheinlichkeiten $p(x_i)$ in einer Tabelle nach fallenden Auftrittswahrscheinlichkeiten
2. Eintragen der Teilsummen (beginnend mit der kleinsten Wahrscheinlichkeit) aus den Auftrittswahrscheinlichkeiten in die dritte Spalte



● Vorgehensweise

3. Unterteilung der Teilsummen in zwei Intervalle (möglichst nahe bei der Hälfte der jeweiligen Teilsumme)
4. Für alle Zeichen oberhalb des Schnitts wird für das Code-Wort eine 0 eingetragen, sowie für alle Zeichen unterhalb des Schnitts eine 1 (oder umgekehrt)



● Vorgehensweise

5. Alle entstandenen Intervalle werden analog zu Schritt 3 wieder halbiert und die nächste Binärstelle wird analog zu Schritt 4 eingetragen
6. Enthält ein Intervall nur noch ein Zeichen, so endet das Verfahren für dieses Zeichen (Code des Zeichens ist komplett)



● Beispiel

- Ausgangssituation: 6 Buchstaben {c,v,w,u,r,z} mit den Auftrittswahrscheinlichkeiten

$$p(c) = 0.1643$$

$$p(v) = 0.0455$$

$$p(w) = 0.0874$$

$$p(u) = 0.1963$$

$$p(r) = 0.4191$$

$$p(z) = 0.0874$$



Fano-Algorithmus (5)

● Beispiel

● Schritt 1

Anordnung der zu kodierenden Zeichen x_i und die zugehörigen Auftretswahrscheinlichkeiten $w(x_i)$ in einer Tabelle nach fallenden Auftretswahrscheinlichkeiten

Zeichen c	$p(c)$
r	0.4191
u	0.1963
c	0.1643
z	0.0874
w	0.0874
v	0.0455



Fano-Algorithmus (6)

● Beispiel

● Schritt 2

Eintragen der Teilsummen
(beginnend mit der kleinsten
Wahrscheinlichkeit) aus den
Auftrittswahrscheinlichkeiten
in die dritte Spalte

Zeichen c	$p(c)$	$\sum p(c)$
r	0.4191	1.0000
u	0.1963	0.5809
c	0.1643	0.3847
z	0.0874	0.2203
w	0.0874	0.1329
v	0.0455	0.0455

● Beispiel

● Schritt 3

Unterteilung der Teilsummen in zwei Intervalle (möglichst nahe bei der Hälfte der jeweiligen Teilsumme)

Zeichen c	$p(c)$	$\sum p(c)$
r	0.4191	1.0000
u	0.1963	0.5809
c	0.1643	0.3847
z	0.0874	0.2203
w	0.0874	0.1329
v	0.0455	0.0455

● Beispiel

● Schritt 4

Für alle Zeichen oberhalb des Schnitts wird für das Code-Wort eine 0 eingetragen, sowie für alle Zeichen unterhalb des Schnitts eine 1 (oder umgekehrt)

Zeichen c	$p(c)$	$\sum p(c)$	Code
r	0.4191	1.0000	0
u	0.1963	0.5809	1
c	0.1643	0.3847	1
z	0.0874	0.2203	1
w	0.0874	0.1329	1
v	0.0455	0.0455	1

● Beispiel

● Schritt 3,4 (Wdh.)

Alle entstandenen Intervalle werden analog zu Schritt 3 wieder halbiert und die nächste Binärstelle wird analog zu Schritt 4 eingetragen.

Zeichen c	$p(c)$	$\sum p(c)$	Code
r	0.4191	1.0000	0
u	0.1963	0.5809	10
c	0.1643	0.3847	10
z	0.0874	0.2203	11
w	0.0874	0.1329	11
v	0.0455	0.0455	11

● Beispiel

● Schritt 3,4 (Wdh.)

Alle entstandenen Intervalle werden analog zu Schritt 3 wieder halbiert und die nächste Binärstelle wird analog zu Schritt 4 eingetragen.

Zeichen c	$p(c)$	$\sum p(c)$	Code
r	0.4191	1.0000	0
u	0.1963	0.5809	100
c	0.1643	0.3847	101
z	0.0874	0.2203	11
w	0.0874	0.1329	11
v	0.0455	0.0455	11

● Beispiel

● Schritt 3,4 (Wdh.)

Alle entstandenen Intervalle werden analog zu Schritt 3 wieder halbiert und die nächste Binärstelle wird analog zu Schritt 4 eingetragen.

Zeichen c	$p(c)$	$\sum p(c)$	Code
r	0.4191	1.0000	0
u	0.1963	0.5809	100
c	0.1643	0.3847	101
z	0.0874	0.2203	110
w	0.0874	0.1329	111
v	0.0455	0.0455	111

Fano-Algorithmus (12)

● Beispiel

● Schritt 3,4 (Wdh.)

Alle entstandenen Intervalle werden analog zu Schritt 3 wieder halbiert und die nächste Binärstelle wird analog zu Schritt 4 eingetragen.

Zeichen c	$p(c)$	$\sum p(c)$	Code
r	0.4191	1.0000	0
u	0.1963	0.5809	100
c	0.1643	0.3847	101
z	0.0874	0.2203	110
w	0.0874	0.1329	1110
v	0.0455	0.0455	1111

- Charakterisierung des erzeugten Codes:
 - $L = 0.4191$
 $+ (0.1963 + 0.1643 + 0.0874) \cdot 3$
 $+ (0.0874 + 0.0455) \cdot 4 = 2.2947$ [Bit/Zeichen]
 - Redundanz
 $R = 2.2947 - 2.2327 = 0.0620$ [Bit/Zeichen]
 - Vergleich mit vorherigen Codes
 - Huffman-Code und Fano-Code sind im Beispiel identisch
 - Dies ist aber nicht immer der Fall!
 - Fano-Algorithmus liefert nicht immer optimalen Code bzgl. Redundanzminimierung



- Huffman-Codes sind immer optimal
 - im Sinn einer möglichst kurzen mittleren Wortlänge bei
 - separater Kodierung von Einzelzeichen,
 - ganzzahliger Anzahl Bit pro Zeichen.
 - und damit einer möglichst kleinen Redundanz
 - optimal heißt: **es gibt keinen** besseren Code im o.g. Sinne!
- Fano-Codes sind nicht garantiert optimal
 - in der Praxis ist der Unterschied zu Huffman-Codes jedoch meist gering
 - dennoch werden sie kaum verwendet, da Huffman ähnlich einfach implementiert werden kann

