Technische Hochschule **Rosenheim**

Grundlagen der Informatik

Prof. Dr. J. Schmidt
Fakultät für Informatik

GDI – WS 2020/21

Information und Quellencodierung
Fano- & Huffman-Algorithmus

Überblick

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

Code-Bäume (1)

- Zielsetzung
  - Erzeugung eines Codes mit variabler Wortlänge
    - bei einem gegebenen Alphabet von Zeichen
    - mit bekannten Auftrittswahrscheinlichkeiten
- Einfache (suboptimale!) 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

$$
\begin{aligned}
p(c) &= 0.1643 \\
p(v) &= 0.0455 \\
p(w) &= 0.0874 \\
p(u) &= 0.1963 \\
p(r) &= 0.4191 \\
p(z) &= 0.0874
\end{aligned}
$$

Beispiel Code-Erzeugung

- Berechnung der Informationsgehalte der jeweiligen Zeichen

$$I(x) = \mathrm{ld}\frac{1}{p(x)} \; [Bit]$$

$p(c)$	=	0.1643	$I(c)$	=	2.6056 Bit
$p(v)$	=	0.0455	$I(v)$	=	4.4580 Bit
$p(w)$	=	0.0874	$I(w)$	=	3.5162 Bit
$p(u)$	=	0.1963	$I(u)$	=	2.3489 Bit
$p(r)$	=	0.4191	$I(r)$	=	1.2546 Bit
$p(z)$	=	0.0874	$I(z)$	=	3.5162 Bit

Beispiel Code-Erzeugung

- Berechnung der Informationsgehalte der jeweiligen Zeichen und Ableitung der Wortlänge

$$
\begin{aligned}
I(c) &= 2.6056 \text{ Bit} \rightarrow l(c) = 3 \text{ Bit} \\
I(v) &= 4.4580 \text{ Bit} \rightarrow l(v) = 5 \text{ Bit} \\
I(w) &= 3.5162 \text{ Bit} \rightarrow l(w) = 4 \text{ Bit} \\
I(u) &= 2.3489 \text{ Bit} \rightarrow l(u) = 3 \text{ Bit} \\
I(r) &= 1.2546 \text{ Bit} \rightarrow l(r) = 2 \text{ Bit} \\
I(z) &= 3.5162 \text{ Bit} \rightarrow l(z) = 4 \text{ Bit}
\end{aligned}
$$

Beispiel Code-Erzeugung

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

$I(c)$	=	2.6056 Bit	→	$l(c)$	=	3 Bit	001
$I(v)$	=	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

Exemplarischer Code

Code-Bäume (6)

- 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

Code-Bäume (7)

- 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

$$H_0 = \mathrm{ld}\ 6 = 2.5850$$

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

- Mittlere Wortlänge

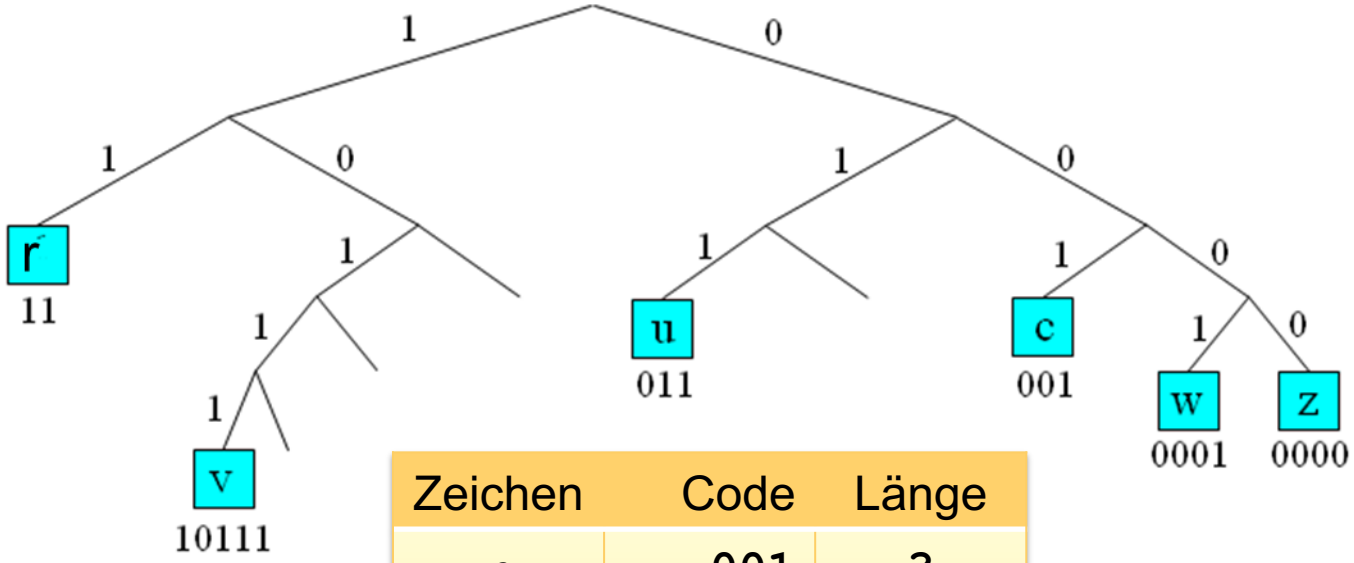
$$\begin{aligned} L &= p(c)l(c) + p(v)l(v) + p(w)l(w) + p(u)l(u) + p(r)l(r) + p(z)l(z) \\ &= 0.4929 + 0.2275 + 0.3496 + 0.5889 + 0.8382 + 0.3496 \\ &= 2.8467 \text{ Bit/Zeichen} \end{aligned}$$

- **Redundanz**

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

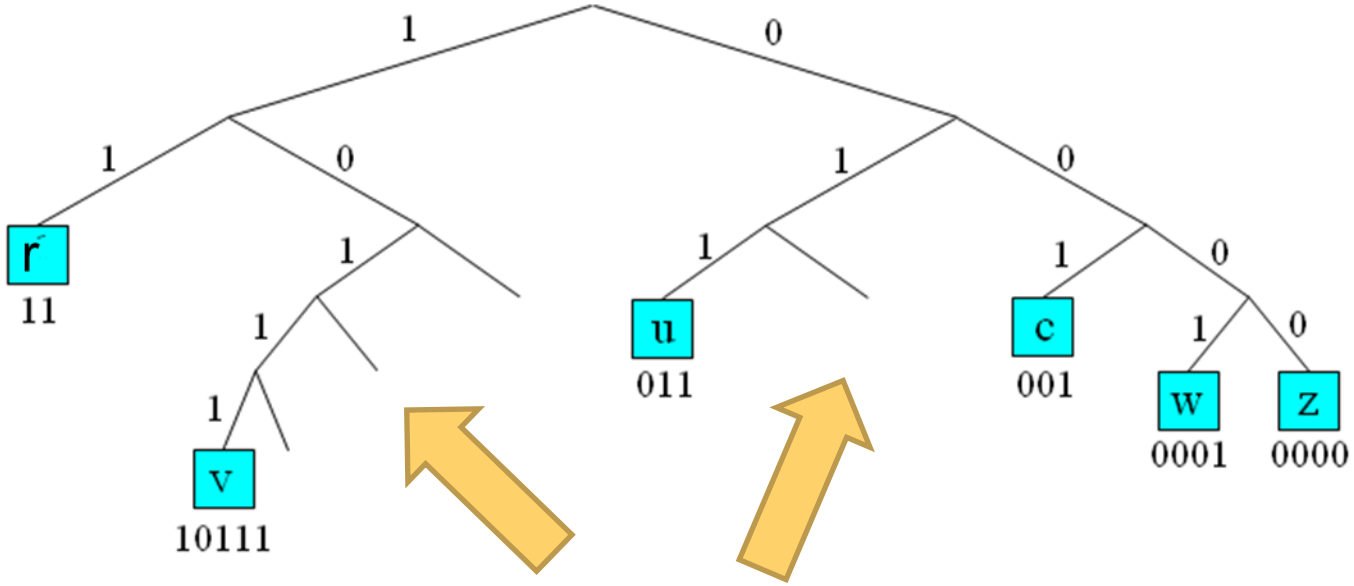
$$\begin{aligned} R_Q &= H_0 - H \\ &\approx 2.5850 - 2.2327 \\ &= 0.3523 \text{ Bit/Zeichen} \end{aligned}$$

Visualisierung der Codierung



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

Visualisierung der Codierung



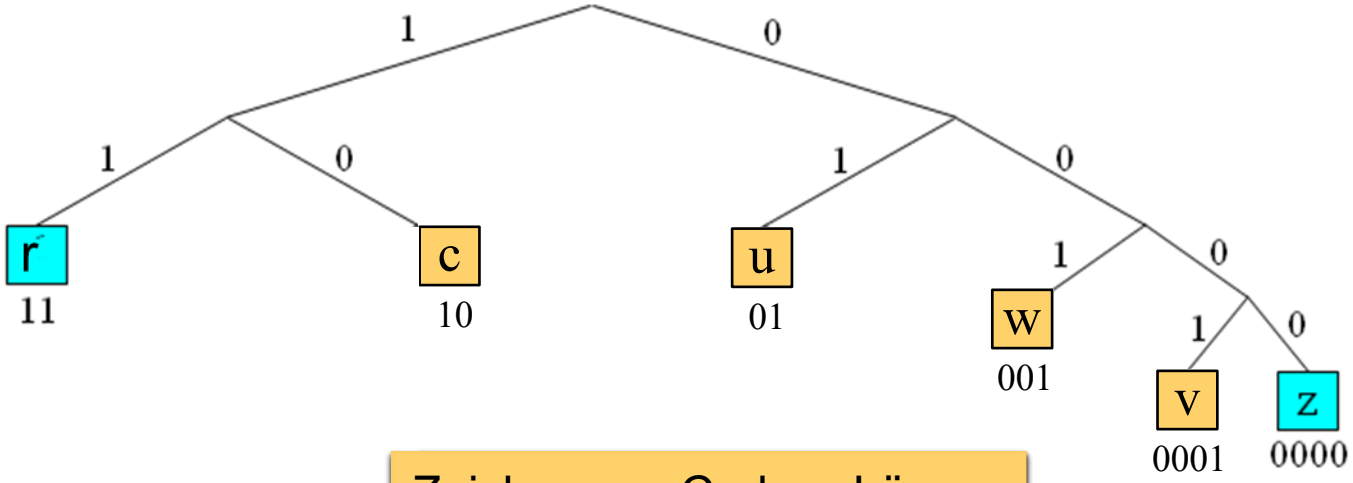
- Unbesetzte Blätter (Endknoten) sind vorhanden, die näher an der Wurzel liegen
- Kein optimaler Code: kürzere Codewörter wären möglich

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
c	10	2
v	0001	4
w	001	3
u	01	2
r	11	2
z	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

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

- Mittlere Wortlänge des verbesserten Codes

  $L =$

  $= 2.3532$ Bit/Zeichen

- Redundanz

  $R \approx$

  $= 0.1205$ Bit/Zeichen

- Redundanz wurde reduziert
  - von 0.6140 Bit/Zeichen auf 0.1205 Bit/Zeichen

Fano-Algorithmus

- 1949 von **Robert Fano** formuliert
- allgemeines Verfahren zur Erzeugung von Codes mit variabler Wortlänge
- Ziel: **Redundanzminimierung**
- arbeitet rekursiv, mit fortlaufender Teilung

Fano-Algorithmus – Vorgehen

1. Anordnung der zu codierenden Zeichen x_i und der zugehörigen Auftrittswahrscheinlichkeiten $p(x_i)$ in einer Tabelle nach fallenden Auftrittswahrscheinlichkeiten

2. Eintragen der Teilsummen (beginnend mit der kleinsten Wahrscheinlichkeit) der Wahrscheinlichkeiten in die dritte Spalte

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 Codewort eine 0 eingetragen; für alle Zeichen unterhalb des Schnitts eine 1 (oder umgekehrt)

5. Setze dieses Vorgehen für alle Teilsummen rekursiv fort

6. Ende, wenn keine Teilung mehr möglich ist

Fano-Algorithmus – Beispiel

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

$$
\begin{aligned}
p(c) &= 0.1643 \\
p(v) &= 0.0455 \\
p(w) &= 0.0874 \\
p(u) &= 0.1963 \\
p(r) &= 0.4191 \\
p(z) &= 0.0874
\end{aligned}
$$

Fano-Algorithmus – Beispiel

Schritt 1

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

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

Fano-Algorithmus – 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

Fano-Algorithmus – 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

Fano-Algorithmus – 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

Fano-Algorithmus – 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

Fano-Algorithmus – 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

Fano-Algorithmus – 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 – Beispiel

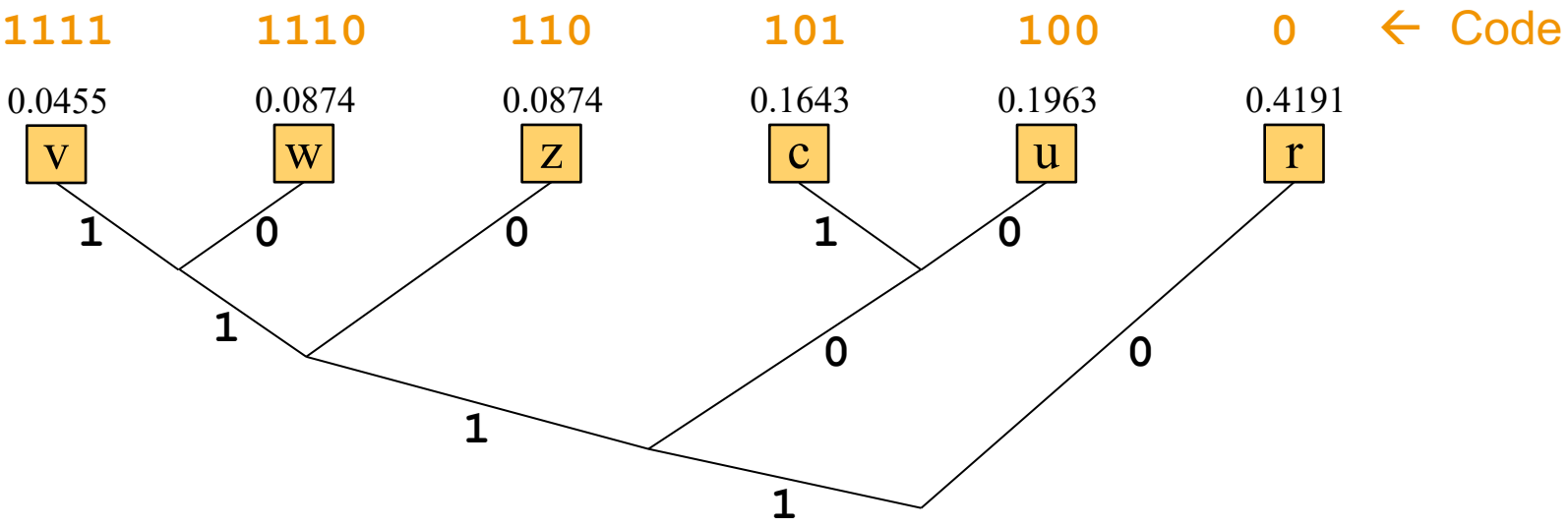
Schritt 3, 4 (Wdh.)

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

Fano-Algorithmus – Beispiel

entstandener Code-Baum



Fano-Algorithmus – Beispiel

Charakterisierung des erzeugten Codes:

- $L = 0.4191 + (0.1963 + 0.1643 + 0.0874)\cdot 3 + (0.0874 + 0.0455) \cdot 4 \quad = 2.2947$ [Bit/Zeichen]
- Redundanz
  $R = 2.2947 - 2.2327 \quad = 0.0620$ [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

Decodierung – Präfixcode

- Wesentliche Forderung an einen Code
  - eindeutige Decodierung
- Fano-Bedingung (auch: Präfixcode)
  - Code mit variabler Wortlänge muss so generiert werden, dass kein Codewort eines Zeichens mit dem Anfang des Codewortes irgendeines anderen Zeichens übereinstimmt
  - d.h.: Codewörter dürfen nur an den Blättern (Endknoten) des Code-Baums stehen

Decodierung – Vorgehen

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 Codewort übereinstimmt, wird Decodierung durchgeführt.

3. Dann wird Puffer gelöscht und der Decodier-Vorgang beginnt von neuem für das nächste Zeichen.

Decodierung – Aufgabe

- Dekodiere die Zeichenkette 10110010110001001 10 mithilfe des vorherigen Fano-Codes

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

- Lösung:

Huffman-Algorithmus

- 1952 von **David A. Huffman** formuliert
- Allgemeines Verfahren
  - zur Erzeugung von optimalen Codes
  - bzgl. des Kriteriums **Redundanzminimierung**
- arbeitet genau entgegengesetzt zum Fano-Verfahren: Beginnt bei einzelnen Zeichen und fasst diese fortlaufend zusammen

Huffman-Algorithmus – Vorgehen

1. Ordne alle Zeichen nach ihren Auftrittswahrscheinlichkeiten (Blätter des Codebaums)

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 (Einzelzeichen oder Knoten) mit den geringsten Wahrscheinlichkeiten zu einem neuen Knoten zusammen

4. Wiederhole 3. solange bis alles zusammengefasst wurde und sich die Wahrscheinlichkeit 1 ergibt (Huffman-Baum)

Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

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

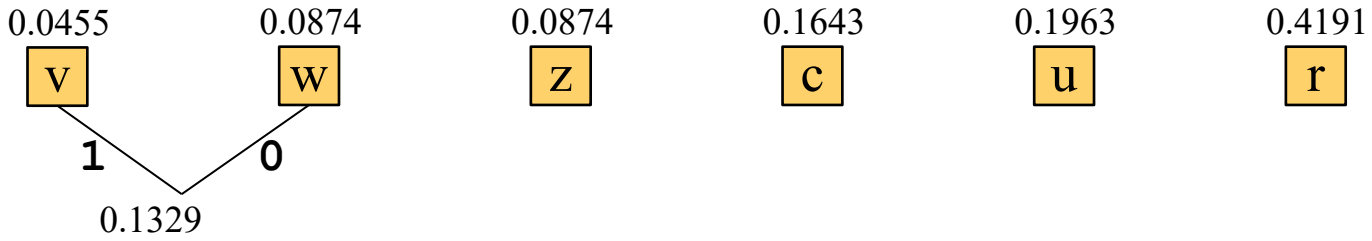
$$
\begin{aligned}
p(c) &= 0.1643 \\
p(v) &= 0.0455 \\
p(w) &= 0.0874 \\
p(u) &= 0.1963 \\
p(r) &= 0.4191 \\
p(z) &= 0.0874
\end{aligned}
$$

Schrittweiser Aufbau des Huffman-Baums und Huffman-Codes

- Schritt 1

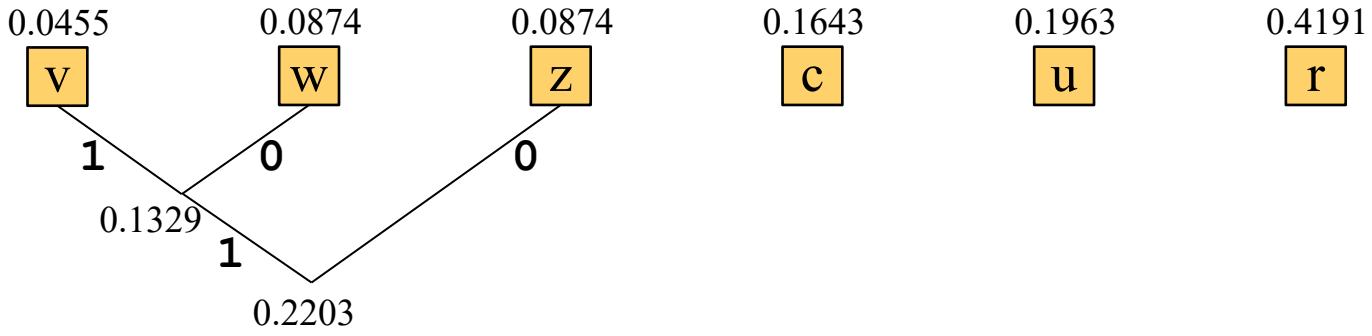
Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

- Schritt 2



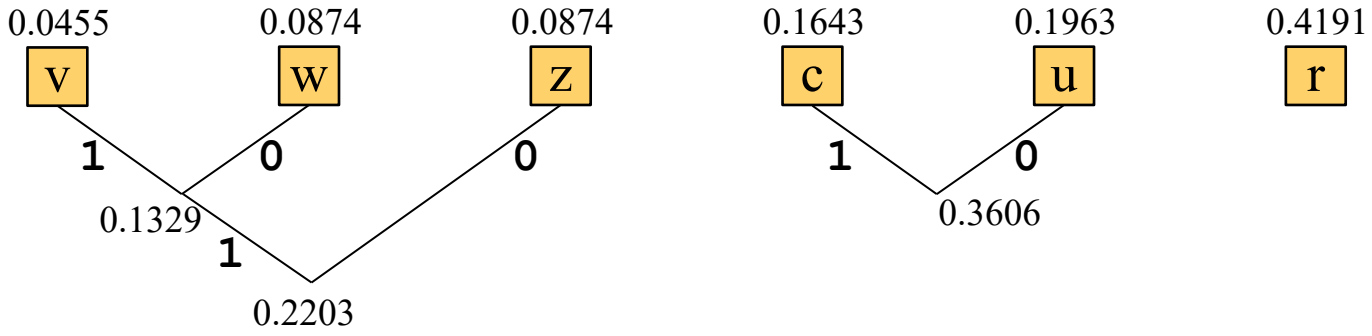
Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

- Schritt 3



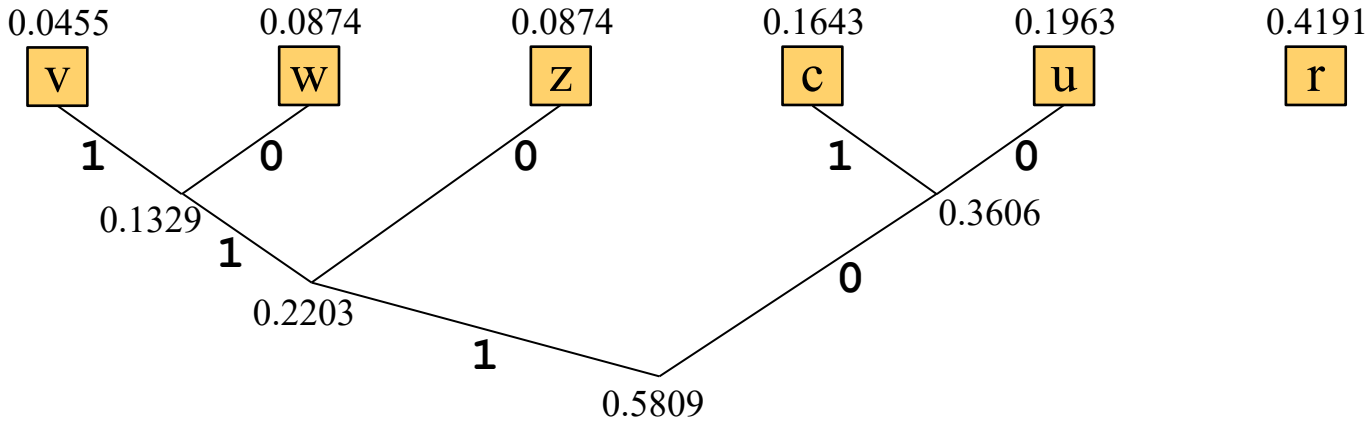
Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

- Schritt 4



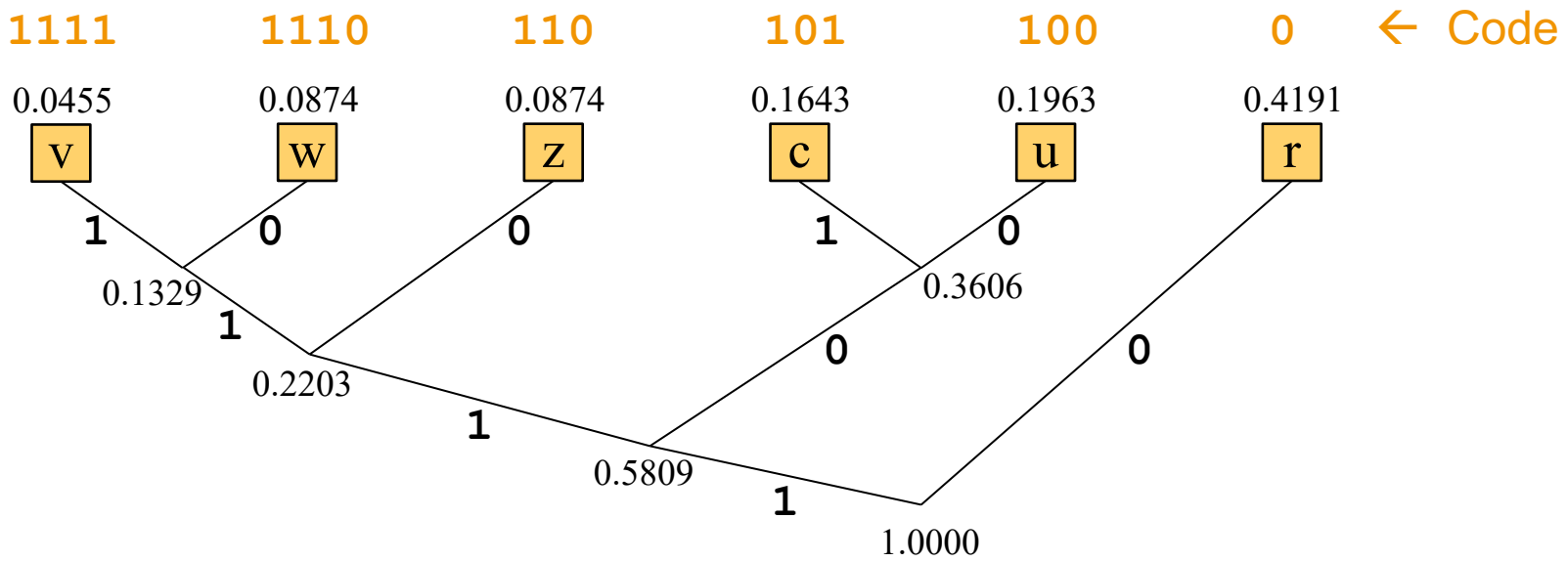
Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

- Schritt 5



Schrittweiser Aufbau des Huffman-Baum und Huffman-Code

- Schritt 6



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

Huffman-Algorithmus – Beispiel

Charakterisierung des erzeugten Codes:

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

Zusammenfassung

- Huffman-Codes sind immer optimal
  - im Sinn einer möglichst kurzen mittleren Wortlänge bei
    - separater Codierung 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