Technische Hochschule Rosenheim

Computer Science Fundamentals

Source Coding – Fano- & Huffman-Algorithms

Technische Hochschule Rosenheim
Winter 2021/22
Prof. Dr. Jochen Schmidt

Overview

- Code trees
- Fano algorithm
- Huffman algorithm

Code Trees

- Objective: Creation of a code with variable word length
  - for a given alphabet of symbols
  - with known probabilities of occurrence

- Example for a simple procedure
  - Determine the word length from the symbol information contents rounded up to nearest integer
  - Arrange code words as end nodes (leaves) of a code tree
  - Note: This is in general suboptimal and NOT how we'll do it!

Code Trees – Example Simple Code Creation

- 6 letters {c, v, w, u, r, z} are to be binary encoded with as little redundancy as possible
- Probabilities of occurrence

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

Code Trees – Example Simple Code Creation

- Calculation of the information content of the respective letters

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

$$\begin{aligned}
p(c) &= 0.1643 & I(c) &= 2.6056 \text{ Bit} \\
p(v) &= 0.0455 & I(v) &= 4.4580 \text{ Bit} \\
p(w) &= 0.0874 & I(w) &= 3.5162 \text{ Bit} \\
p(u) &= 0.1963 & I(u) &= 2.3489 \text{ Bit} \\
p(r) &= 0.4191 & I(r) &= 1.2546 \text{ Bit} \\
p(z) &= 0.0874 & I(z) &= 3.5162 \text{ Bit}
\end{aligned}$$

Code Trees – Example Simple Code Creation

- Calculation of the information content of the respective letters and derivation of the word length

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

Code Trees – Example Simple Code Creation

- Calculation of the information content of the respective letters and derivation of the word length with encoding

$$
\begin{array}{lclcrcll}
I(c) & = & 2.6056\text{ Bit} & \longrightarrow & l(c) & = & 3\text{ Bit} & \texttt{001} \\
I(v) & = & 4.4580\text{ Bit} & \longrightarrow & l(v) & = & 5\text{ Bit} & \texttt{10111} \\
I(w) & = & 3.5162\text{ Bit} & \longrightarrow & l(w) & = & 4\text{ Bit} & \texttt{0001} \\
I(u) & = & 2.3489\text{ Bit} & \longrightarrow & l(u) & = & 3\text{ Bit} & \texttt{011} \\
I(r) & = & 1.2546\text{ Bit} & \longrightarrow & l(r) & = & 2\text{ Bit} & \texttt{11} \\
I(z) & = & 3.5162\text{ Bit} & \longrightarrow & l(z) & = & 4\text{ Bit} & \texttt{0000}
\end{array}
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\\1\\1\end{array}}\right\}\text{Exemplary Code}
$$

Code Trees – Example Simple Code Creation

Technische Hochschule Rosenheim

Symbol	$p(c)$	$I(c)$	$l(c)$
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

- Entropy of this data source

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

- maximum entropy possible with 6 symbols

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

- Average word length

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

- Redundancy

Code

$$\begin{aligned} R &= L - H \\ &\approx 2.8467 - 2.2327 \\ &= 0.6140 \text{ bits/symbol} \end{aligned}$$

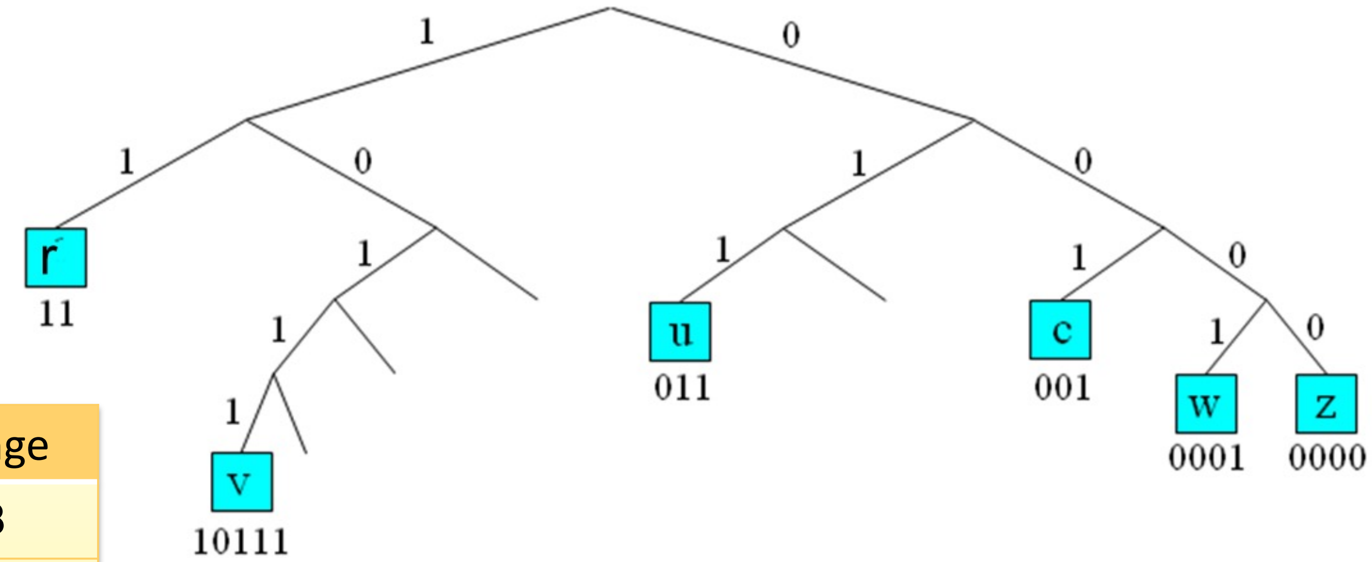
Source

$$\begin{aligned} R_S &= H_0 - H \\ &\approx 2.5850 - 2.2327 \\ &= 0.3523 \text{ bits/symbol} \end{aligned}$$

Code Trees – Example Simple Code Creation

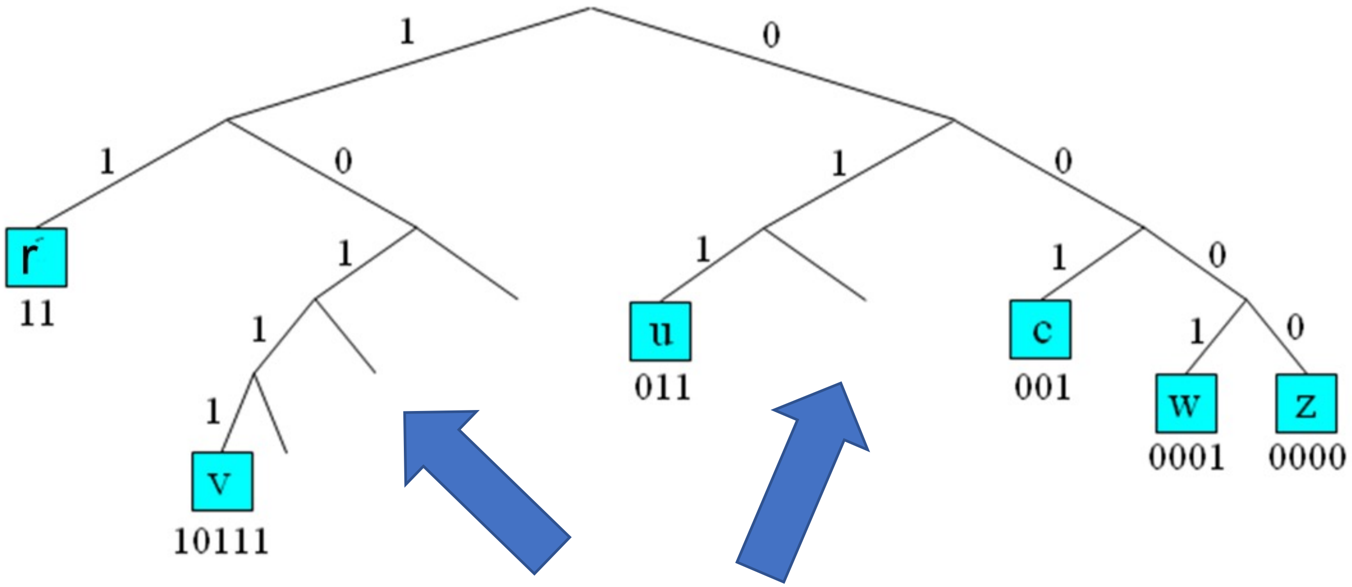
Code visualization

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



Code Trees – Example Simple Code Creation

Code visualization



- Unoccupied leaves (end nodes) are present that are closer to the root
- Not an optimal code: shorter code words would be possible

Code Trees – Example Simple Code Creation

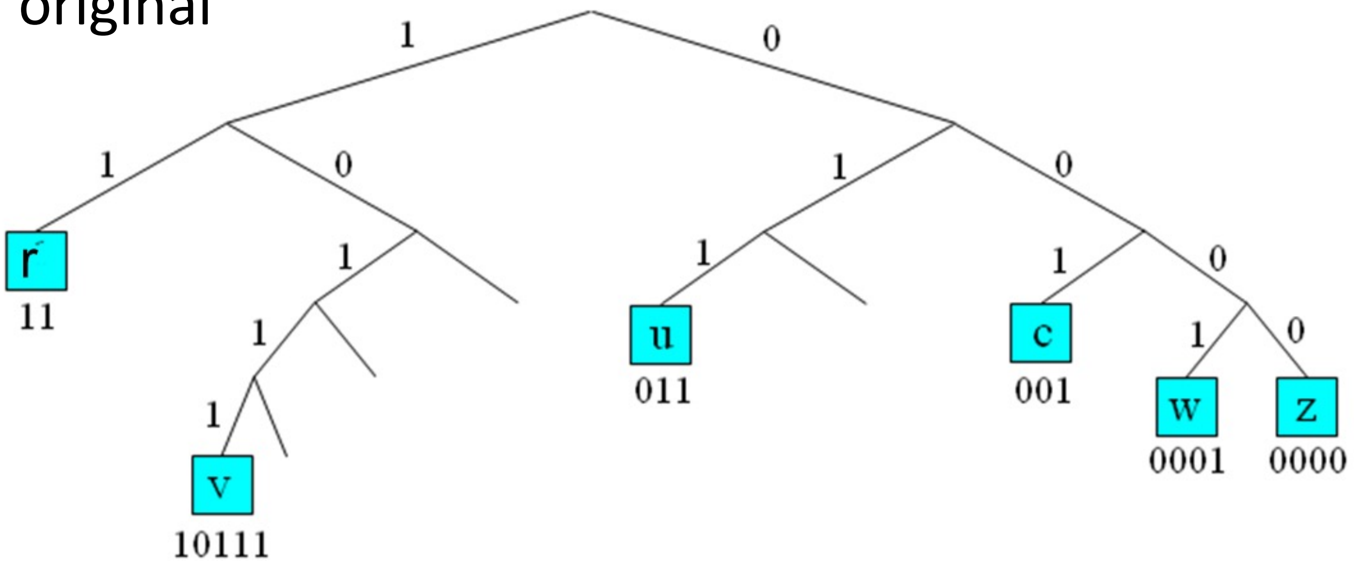
Improvements

- Move symbols such that we get rid of unoccupied leaves

- Caution
  - A code for a symbol with a lower probability of occurrence must be longer than the code for a symbol with a higher probability of occurrence

Code Trees – Example Simple Code Creation

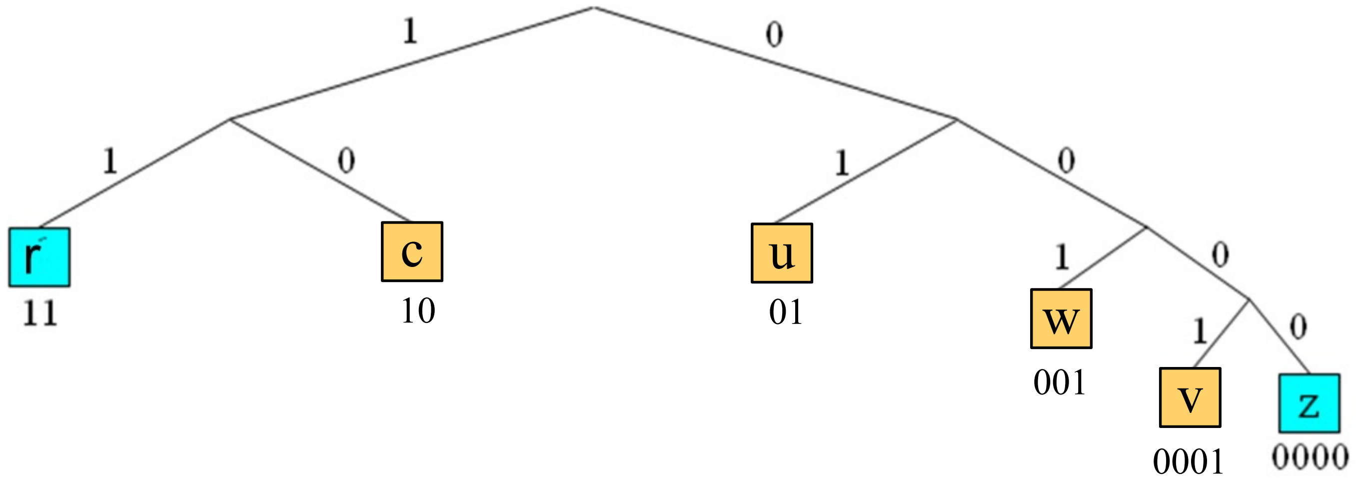
original



Symbol	Code	Length
c	**001**	**3**
v	**10111**	**5**
w	**0001**	**4**
u	**011**	**3**
r	**11**	**2**
z	**0000**	**4**

improved

Symbol	Code	Length
c	**10**	**2**
v	**0001**	**4**
w	**001**	**3**
u	**01**	**2**
r	**11**	**2**
z	**0000**	**4**



Code Trees – Example Simple Code Creation

Technische Hochschule Rosenheim

Results of optimization

Symbol	$p(c)$	$I(c)$	$l(c)$
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

- Average word length of the improved code

  $L =$

  $= 2.3532$ bits/symbol

- Redundancy

  $R \approx$ $= 0.1205$ bits/symbol

- Redundancy was reduced from 0.6140 bits/symbol to 0.1205 bits/symbol

- However:
  - This was tedious – it has to be done automatically by a computer. How would we implement this?
  - Is this the best we can do? Or can we get an even better code by re-arranging the code words?

Fano Algorithm

- 1949 by Robert Fano
- general method for generating codes with variable word length
- Objective: Redundancy minimization
- works recursively, by recurring division of the code table (top-down)

Fano Algorithm – Example

Starting point: 6 letters {c, v, w, u, r, z} with the probabilities of occurrence

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

Fano Algorithm – Example

Step 1

Arrange symbols c to be encoded and the associated probabilities of occurrence $p(c)$ in a table, sorted by descending probability values

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

Fano Algorithm – Example

Step 2

Enter the subtotals of the probabilities (starting with the smallest one) in a third column

Symbol 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 Algorithm – Example

Subdivide table into two parts, as close as possible to half of the respective interval

Step 3

Symbol 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 Algorithm – Example

Step 4

Assign a 0 for all symbols above the division, and a 1 for all symbols below (or vice versa); this will form the code words from left to right

Symbol 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 Algorithm – Example

Step 3, 4 (repeat)

All resulting parts are split again analogously to step 3, and the next binary digit is assigned analogously to step 4.

Symbol 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 Algorithm – Example

Step 3, 4 (repeat)

All resulting parts are split again analogously to step 3, and the next binary digit is assigned analogously to step 4.

Symbol 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 Algorithm – Example

Technische Hochschule Rosenheim

Step 3, 4 (repeat)

All resulting parts are split again analogously to step 3, and the next binary digit is assigned analogously to step 4.

Symbol 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 Algorithm – Example

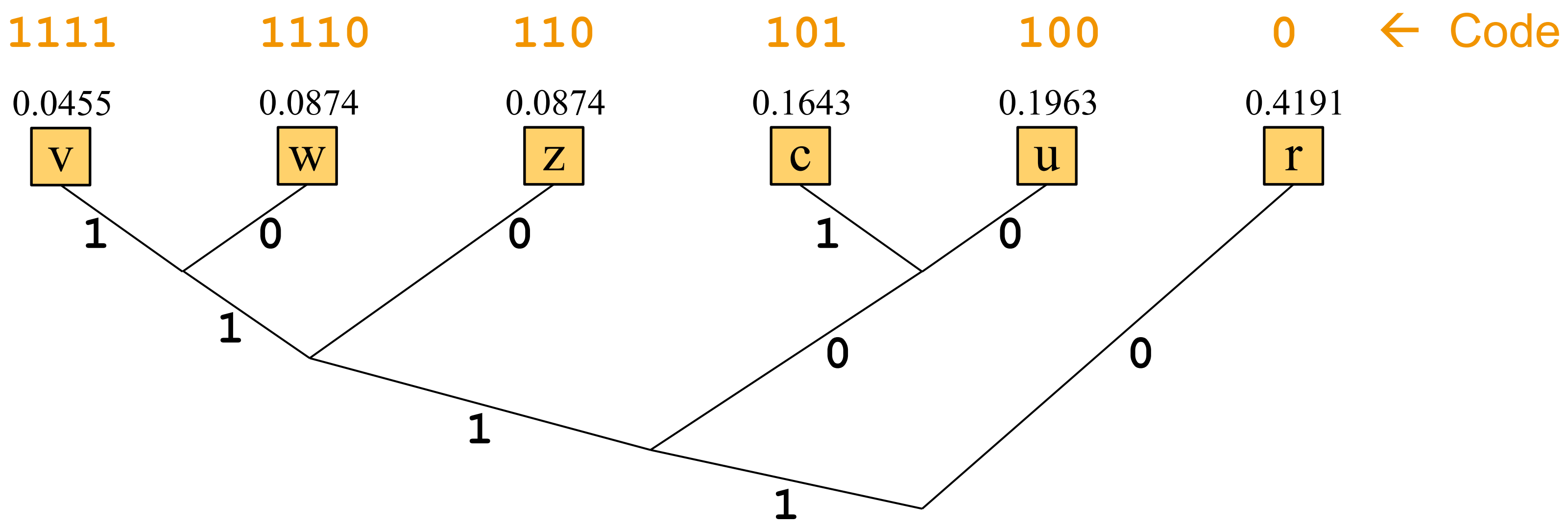
Step 3, 4 (repeat)

All resulting parts are split again analogously to step 3, and the next binary digit is assigned analogously to step 4.

Symbol 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 Algorithm – Example

resulting code tree



Fano Algorithm – Example

- Average code word length
  $$L = 0.4191 + (0.1963 + 0.1643 + 0.0874)\cdot 3 + (0.0874 + 0.0455)\cdot 4 = 2.2947 \text{ bits/symbol}$$
- Redundancy
  $$R = 2.2947 - 2.2327 = 0.0620 \text{ bits/symbol}$$

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

- Comparison to previous codes
  - First, redundancy was reduced from 0.6140 bits/symbol to 0.1205 bits/symbol
  - Now further reduction of redundancy to 0.062 bits/symbol
- Is this the best code we can get?
  - Not necessarily, Fano algorithm does not guarantee this

Fano Algorithm – Summary

1. Arrange symbols c to be encoded and the associated probabilities of occurrence $p(c)$ in a table, sorted by descending probability values.
2. Enter the subtotals of the probabilities (starting with the smallest one) in a third column.
3. Subdivide table into two parts, as close as possible to half of the respective interval.
4. Assign a 0 for all symbols above the division, and a 1 for all symbols below (or vice versa); this will form the code words from left to right.
5. Continue this procedure recursively for all partitions.
6. End when division is no longer possible.

Huffman Algorithm

- 1952 by David A. Huffman
- Algorithm for
  - generating optimal codes
  - with regard to the criterion redundancy minimization considering single symbols
- works the opposite way to the Fano method: Start with individual symbols and combine them (bottom-up)

Huffman Algorithm – Example

Step-by-step construction of the Huffman tree and Huffman code

Starting point: 6 letters {c, v, w, u, r, z} with the probabilities of occurrence

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

Huffman Algorithm – Example

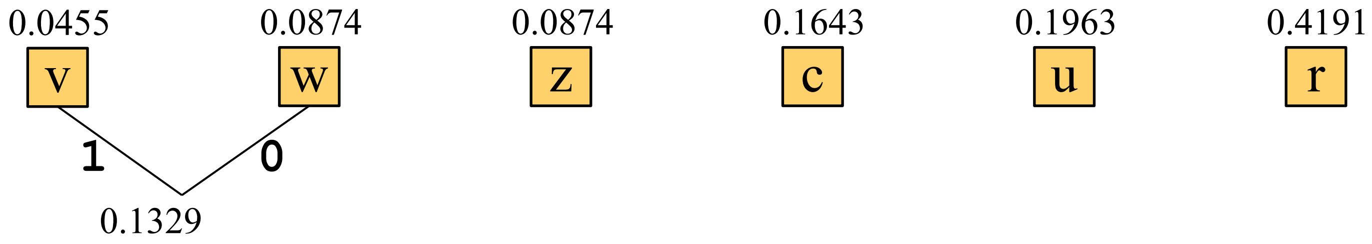
Step 1: Sort symbols by ascending probability

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

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

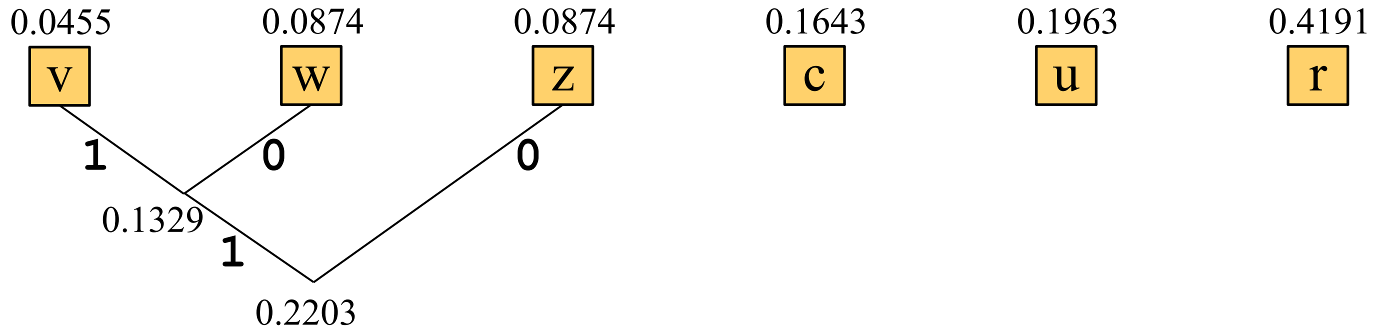
Huffman Algorithm – Example

Step 2: Combine the two symbols with smallest probability



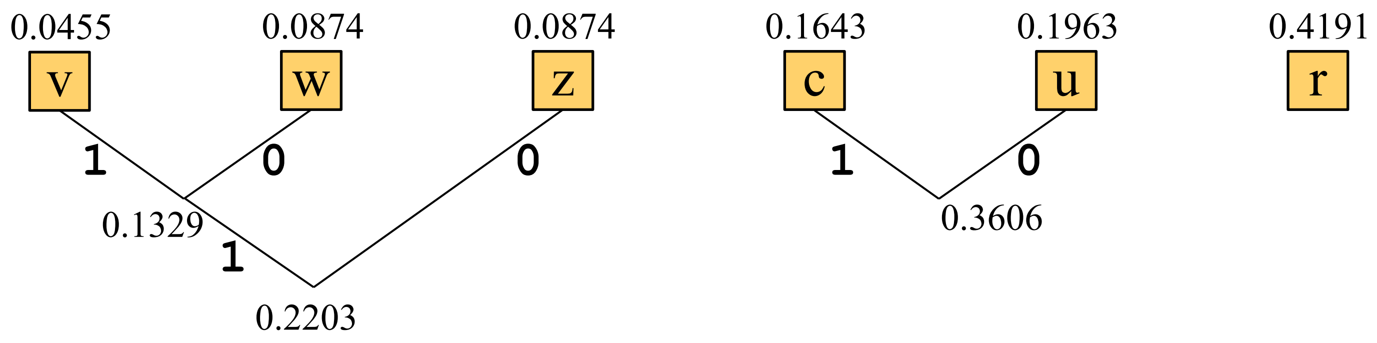
Huffman Algorithm – Example

Step 3: Treat combination as new “symbol”
and again combine the two smallest probabilities (in an algorithm: sort again)



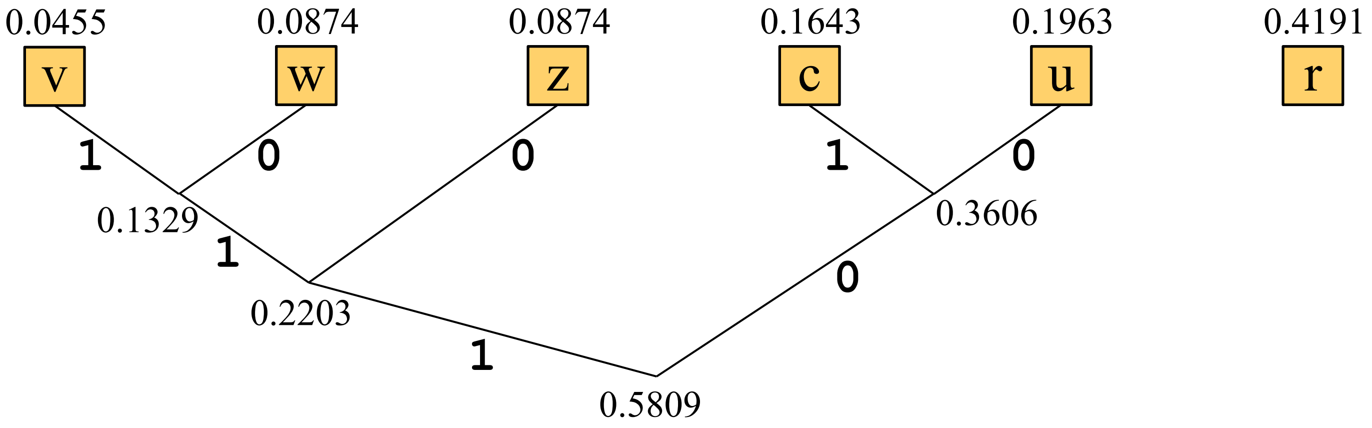
Huffman Algorithm – Example

… and so on…



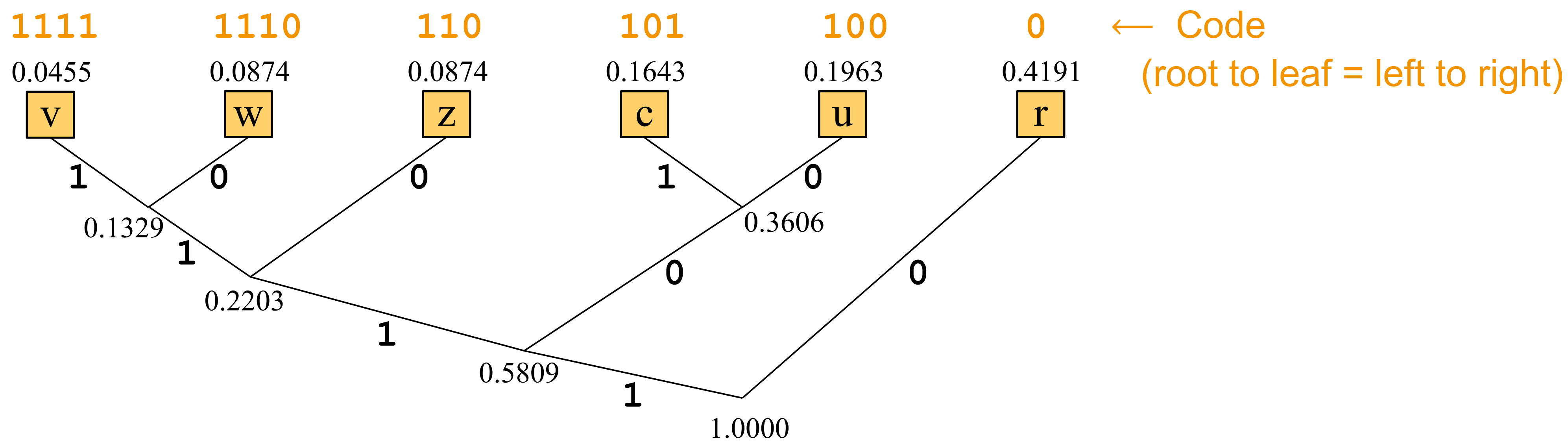
Huffman Algorithm – Example

… and so on…



Huffman Algorithm – Example

… and so on…



Huffman Algorithm – Example

- Average code word length
  $L = \quad 0.4191 +$
  $(0.1963 + 0.1643 + 0.0874)\cdot 3 +$
  $(0.0874 + 0.0455) \cdot 4 \quad = 2.2947$ bits/symbol

- Redundancy
  $R = 2.2947 - 2.2327 \quad = 0.0620$ bits/symbol

Symbol	$p(c)$	$I(c)$	$l(c)$
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

- Comparison to previous codes
  - Huffman code and Fano code are identical in this example
  - But this is not always the case!
  - Fano doesn't always result in the optimal code for redundancy minimization – Huffman does

Huffman Algorithm – Summary

1. Sort all symbols ascending by to their probabilities of occurrence (leaves of the code tree)

2. Combine the two symbols with the lowest probabilities p_1 und p_2 to a node
   - Probability = $p_1 + p_2$
   - Result: New sequence of probabilities

3. Combine the two elements (single symbols and/or nodes) with the lowest probabilities to a new node

4. Repeat (3) until everything has been combined and probability 1 results (result: Huffman Tree)

Decoding – Prefix Code

- Essential requirement for a code: unique decoding
- block codes: easy, each code word has the same length
- variable-length codes: also easy, if the code is a prefix code
  - no code word is prefix of any other code word, i.e.,
  - code words can only be on the leaves of the code tree
- a better term would actually be prefix-free code
- Fano & Huffman automatically generate prefix codes

Decoding Process

1. The code words to be interpreted are collected bit by bit in a buffer and continuously compared with the tabulated code words.
   - Due to the prefix-condition, this is always unique.

2. Once the buffer content matches a tabulated code word, decoding is performed.

3. The buffer is cleared, and the decoding operation starts again for the next code word.

Decoding – Exercise

Technische Hochschule Rosenheim

Decode the message 1011001011000100110 using the Fano code from before

Symbol	Code
c	**101**
v	**1111**
w	**1110**
u	**100**
r	**0**
z	**110**

Solution:

Summary

Technische Hochschule Rosenheim

- Huffman codes are proven to be optimal
  - in the sense of the shortest possible average word length for
    - separate encoding of single symbols,
    - an integral number of bits per symbol (i.e., no fractional bits for a symbol).
  - and therefore in the sense of obtaining the smallest possible redundancy
  - optimal means: **there exists no better coding algorithm in the above sense**!
  - they are widely used in compression algorithms
    - often as a final step (so-called entropy coding) after lossy compression
    - often combined with run-length encoding

- Fano codes are not guaranteed to be optimal
  - in practice, however, the difference to Huffman codes is usually small
  - they are hardly used because Huffman is similarly easy to implement