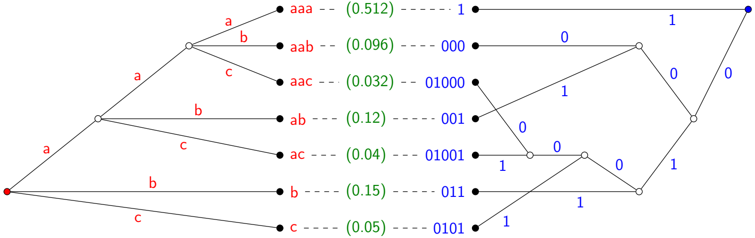
Universal Codes, V2V Codes, and Shannon-Fano-Elias Codes



Last Lecture: Entropy Measures

Entropy Measures

- Scalar/marginal entropy: $H(S_n) \;=\; \mathrm{E}\{-\log_2 p(S_n)\}$
- Conditional entropy: $H(S_n \,|\, S_{n-1}) \;=\; \mathrm{E}\{-\log_2 p(S_n \,|\, S_{n-1})\}$
- N-th order block entropy: $H_N(\boldsymbol{S}) = H(S_n, \cdots, S_{n+N-1}) \;=\; \mathrm{E}\{-\log_2 p(S_n, \cdots, S_{n+N-1})\}$

Important Relations for Entropy Measures

- Conditioning never increases entropy: $H(S_n \,|\, S_{n-1}, S_{n-2}) \;\le\; H(S_n \,|\, S_{n-1}) \;\le\; H(S_n)$
- Increasing block size never increases lower bound: $\frac{1}{N}\, H_N(\boldsymbol{S}) \;\le\; \frac{1}{N-1}\, H_{N-1}(\boldsymbol{S}) \;\le\; H(S_n)$

Entropy Rate and Relations for Stationary Sources

- Entropy rate $\bar{H}(\boldsymbol{S})$ with

$$\boxed{\bar{H}(\boldsymbol{S}) \;=\; \lim_{N\to\infty} \frac{1}{N}\, H_N(\boldsymbol{S})}$$

$\bar{H}(\boldsymbol{S}) \;\le\; H(S_n)$ (equality iff $\boldsymbol{S}$ is iid)

$\bar{H}(\boldsymbol{S}) \;\le\; \frac{1}{N}\, H_N(\boldsymbol{S})$ (equality iff $\boldsymbol{S}$ is iid)

$\bar{H}(\boldsymbol{S}) \;\le\; H(S_n \,|\, S_{n-1})$ (equality iff $\boldsymbol{S}$ is Markov)

Last Lecture: Huffman Codes

Types of Variable-Length Codes

- Scalar codes: Assign one codeword to each possible symbol
- Blocks codes: Assign one codeword to each possible block of N successive symbols
- Conditional codes: Multiple scalar codeword tables: Table for current symbol s_n is selected based on the value of a condition $c_n = f(s_{n-1}, \cdots)$

Huffman Algorithm

- Generates one optimal prefix code (minimum redundancy) for any given finite pmf
- Can be used for all types of variable-length codes: scalar, conditional, block codes, etc.

Bounds on Average Codeword Length (note: lower bound applies to all codes of a type)

- Scalar Huffman codes: $H(S_n) \leq \bar{\ell} < H(S_n) + 1$
- Block Huffman codes of size N: $\frac{1}{N} H_N(\boldsymbol{S}) \leq \bar{\ell} < \frac{1}{N} H_N(\boldsymbol{S}) + \frac{1}{N}$
- Conditional Huffman codes: $H(S_n \,|\, C) \leq \bar{\ell} < H(S_n \,|\, C) + 1$ with $C = f(S_{n-1}, S_{n-2}, \cdots)$
- **All lossless codes:** $\bar{H}(\boldsymbol{S}) \leq \bar{\ell}$

Unary Code

- ➡ **Structured code** for non-negative inters n
- ➡ Very simple encoding and decoding
- ➡ Optimal for geometric pmf $p(n) = p(1-p)^n$ with $p = 0.5$

n	codewords
0	1
1	01
2	001
3	0001
4	0000 1
5	0000 01
6	0000 001
7	0000 0001
8	0000 0000 1
9	0000 0000 01
10	0000 0000 001
11	0000 0000 0001
12	0000 0000 0000 1
13	0000 0000 0000 01
14	0000 0000 0000 001
15	0000 0000 0000 0001
...	...

```
// encoding
void encUnary( int n )
{
  while( n-- )
  {
    bitstream.put( 0 );
  }
  bitstream.put( 1 );
};
```

```
// decoding
int decUnary()
{
  int n = 0;
  while( !bitstream.get() )
  {
    n++;
  }
  return n;
};
```

➡ **often used as part of other codes**

Rice Codes

- Family of codes parameterized by Rice parameter $R \geq 0$
- Represents non-negative integers n using a prefix and a suffix part

$$\text{prefix} = (\, n \gg R \,) \qquad \rightarrow \text{ unary code}$$
$$\text{suffix} = n - (\, \text{prefix} \ll R \,) \qquad \rightarrow \text{ fixed-length code with } R \text{ bits}$$

n	$R = 0$ (*unary*)	$R = 1$	$R = 2$	$R = 3$
0	1	10	100	1000
1	01	11	101	1001
2	001	010	110	1010
3	0001	011	111	1011
4	0000 1	0010	0100	1100
5	0000 01	0011	0101	1101
6	0000 001	0001 0	0110	1110
7	0000 0001	0001 1	0111	1111
8	0000 0000 1	0000 010	0010 0	0100 0
9	0000 0000 01	0000 011	0010 1	0100 1
10	0000 0000 001	0000 0010	0011 0	0101 0
11	0000 0000 0001	0000 0011	0011 1	0101 1
12	0000 0000 0000 1	0000 0001 0	0001 00	0110 0
13	0000 0000 0000 01	0000 0001 1	0001 01	0110 1
14	0000 0000 0000 001	0000 0000 10	0001 10	0111 0
15	0000 0000 0000 0001	0000 0000 11	0001 11	0111 1
...	...	...	...	...

```
// encoding
void encRice( int n, int r )
{
  int pre = n >> r;
  int suf = n - ( pre << r );
  encUnary( pre );
  encFixed( suf , r ); // r bits
};
```

```
// decoding
int decRice( int r )
{
  int pre = decUnary();
  int suf = decFixed( r );
  int n   = ( pre << r ) + suf;
  return n;
};
```

used in:

- FLAC, Apple Lossless
- JPEG-LS, HEVC, VVC

Exponential-Golomb Codes

- Another family of parameterized codes (order $K \geq 0$)
- Exponentially growing "classes"

  class index p → unary code

  index inside class → fixed-length code with $K+p$ bits

n	$K=0$	$K=1$	$K=2$	$K=3$
0	1	10	100	1000
1	010	11	101	1001
2	011	0100	110	1010
3	0010 0	0101	111	1011
4	0010 1	0110	0100 0	1100
5	0011 0	0111	0100 1	1101
6	0011 1	0010 00	0101 0	1110
7	0001 000	0010 01	0101 1	1111
8	0001 001	0010 10	0110 0	0100 00
9	0001 010	0010 11	0110 1	0100 01
10	0001 011	0011 00	0111 0	0100 10
11	0001 100	0011 01	0111 1	0100 11
12	0001 101	0011 10	0010 000	0101 00
13	0001 110	0011 11	0010 001	0101 01
14	0001 111	0001 0000	0010 010	0101 10
15	0001 0000	0001 0001	0010 011	0101 11
...	...	...	...	...

```
// encoding
void encExpGolomb( int n, int k )
{
  // good implementation for first line
  // should be based on finding the
  // most significant bit in an integer
  int p = floor( log2(n+(1<<k)) ) - k;
  int m = (1 << (k+p)) - (1<<k);
  encUnary( p );
  encFixed( n-m, k+p ); // k+p bits
};
```

```
// decoding
int decExpGolomb( int k )
{
  int p = decUnary();
  int s = decFixed( k+p );
  int m = (1 << (k+p)) - (1<<k);
  return m+s;
};
```

- **Exp-Golomb order 0 used in**:
  - H.264 | AVC
  - H.265 | HEVC, VVC

V2V Codes

Generalization of Block Codes

- Assign variable-length codewords to **symbol sequences of variable-length** (V2V)
- How to select symbol sequences?
  - ➡ All messages must be representable by symbol sequences
  - ➡ Desirable: Redundancy-free set of symbol sequences

Examples: Binary symbol alphabet $\mathcal{A} = \{a, b\}$

code A		code B		code C		code D	
aaaa	0	aaa	000	aaaa	0	aa	00
aaab	10	aa	01	aaab	10	ab	0100
aab	110	a	1	aab	110	ba	0101
bba	1110	b	0010	ab	1110	bba	011
ba	1111	bb	0011	b	1111	bbb	1
abbbb... ?		redundant!		suitable		suitable	

Suitable Set of Variable-Length Symbol Sequences

Suitable Sets of Symbol Sequences ?

- Consider m-ary alphabet $\mathcal{A} = \{a_1, a_2, \cdots, a_m\}$

➡ **All sets of symbol sequences that are representable by a full m-ary tree**

- ➡ All messages are representable by a concatenation of the individual symbol sequences

  Note: Fill symbol sequence at end of message (as for block codes)

- ➡ Redundancy-free set of symbol sequences
- ➡ Instantaneous encodable codes

Special Cases

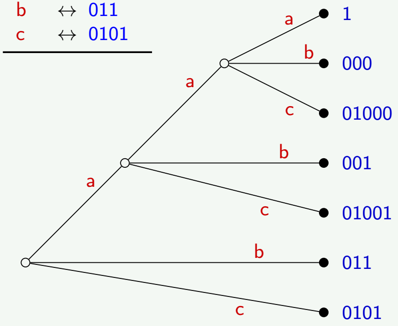
- Scalar code: Full m-ary tree of depth 1
- Block code of size N: Perfect m-ary tree of depth N

Example: Ternary alphabet

aaa	$\leftrightarrow$	1
aab	$\leftrightarrow$	000
aac	$\leftrightarrow$	01000
ab	$\leftrightarrow$	001
ac	$\leftrightarrow$	01001
b	$\leftrightarrow$	011
c	$\leftrightarrow$	0101

$\mathcal{A} = \{a, b, c\}$

$\Rightarrow \quad m = 3$

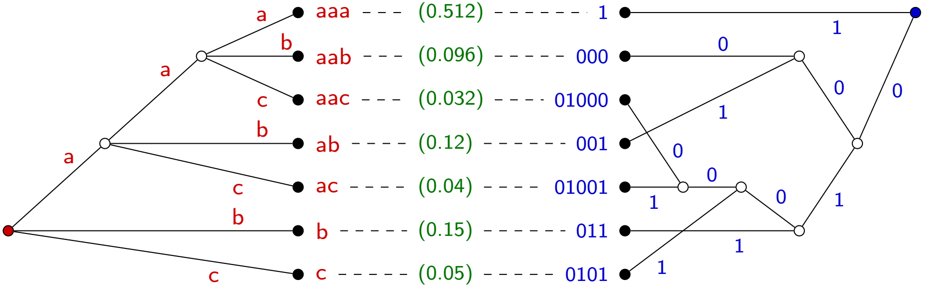


Prefix Codes for Symbol Sequences: V2V Codes as Double Tree

iid source ($m = 3$)

symbol	probability
a	0.80
b	0.15
c	0.05

entropy rate:	$\bar{H}(\boldsymbol{S}) = 0.88418$	(equal to $H(S)$)
scalar Huffman:	$\bar{\ell} = 1.2$	(3 codewords)
2-symbol blocks:	$\bar{\ell} = 0.93375$	(9 codewords)
V2V code:	$\bar{\ell} = 0.88934$	(7 codewords)
	$\varrho = 0.00516$	(0.58 %)



Average Codeword Length of V2V Codes

What we know

- Structure of V2V code:
  - ➡ N : number of symbols sequences (leaf nodes)
  - ➡ n_k : number of symbols in k-th symbol sequence
  - ➡ ℓ_k : length of codeword for k-th symbol sequence
- Statistical properties of source:
  - ➡ p_k : probability of k-th symbol sequence

Average Codeword Length of V2V Codes

$$\bar{\ell} = \frac{\textit{average codeword length per sequence}}{\textit{average number of symbols per sequence}} = \frac{\sum_{k=1}^{N} p_k \cdot \ell_k}{\sum_{k=1}^{N} p_k \cdot n_k}$$

How to Determine the PMF for Symbol Sequences ?

How can we determine the pmf $p(\boldsymbol{a}_k)$ *for the selected symbol sequences* $\boldsymbol{a}_k = (a_{k1}, a_{k2}, \cdots, a_{kn_k})$ *?*

IID Sources

- No dependencies on previous symbols ➡ $p(\boldsymbol{a}_k) = p(a_{k1}) \cdot p(a_{k2}) \cdot \ldots \cdot p(a_{kn_k})$

General Stationary Sources

- Probability that a symbol sequence starts with any particular letter depends on preceding symbols

➡ Probabilities $p(\boldsymbol{a}_k)$ of symbol sequences $\boldsymbol{a}_k$ can be determined by solving linear equation system

- Further details can be found in [Wiegand, Schwarz: "Source Coding"]

In Practice: Estimate Pmf based on Training Set

- Use (large) training set of typical messages for the considered source

➡ Count occurences $N(\boldsymbol{a}_k)$ of variable-length symbol sequences $\boldsymbol{a}_k$ in the messages of the training set

➡ Estimate probability $p(\boldsymbol{a}_k)$ according to

$$p(\boldsymbol{a}_k) = \frac{N(\boldsymbol{a}_k)}{\sum_k N(\boldsymbol{a}_k)}$$

Optimal V2V Codes for Given Set of Symbol Sequence

What is given?

- Set of N variable-length symbol sequences $\boldsymbol{a}_k$ with n_k symbols (full m-ary tree with N leafs)
- Associated probabilities masses $p_k = \mathrm{P}(\boldsymbol{S}_{\mathrm{next}} = \boldsymbol{a}_k)$ (calculated or experimentally determined)

Optimal V2V Codes

- Assign codewords of length ℓ_k to symbol sequences $\boldsymbol{a}_k$
- Goal: Minimize average codeword length per symbol

$$\bar{\ell} = \frac{\bar{\ell}_{\mathrm{seq}}}{\bar{n}} = \frac{\sum_{k=1}^{N} p_k \, \ell_k}{\sum_{k=1}^{N} p_k \, n_k} \quad \begin{array}{l} \leftarrow \text{ optimization problem: best prefix code for given pmf } \{p_k\} \\ \leftarrow \text{ fixed value for given set } \{\boldsymbol{a}_k\} \text{ and given pmf } \{p_k\} \end{array}$$

➡ **Optimal code can be obtained by Huffman algorithm for pmf $\{p_k\}$**

➡ Resulting average codeword length per symbol is bounded by

$$\left(\frac{-\sum_{k=1}^{N} p_k \log_2 p_k}{\sum_{k=1}^{N} p_k \, n_k} \right) \le \bar{\ell} < \left(\frac{-\sum_{k=1}^{N} p_k \log_2 p_k}{\sum_{k=1}^{N} p_k \, n_k} \right) + \left(\frac{1}{\sum_{k=1}^{N} p_k \, n_k} \right)$$

Example: Coding of Black and White Document Scans (300 dpi)

Code design:
- Select set of symbol sequences $\boldsymbol{a}_k$ (full m-ary tree)
- Experimentally determine pmf $p_k = p(\boldsymbol{a}_k)$ using actual document scans
- Apply Huffman algorithm for determining codewords

block Huffman code

$\boldsymbol{a}_k$	p_k	codewords
000	0.8833	1
001	0.0161	0110
010	0.0006	011101
011	0.0159	01111
100	0.0160	0101
101	0.0005	011100
110	0.0160	0100
111	0.0516	00

$$\left.\begin{array}{r}\bar{\ell}_{\text{seq}} = 1.265 \\ n = 3\end{array}\right\} \rightarrow \bar{\ell} = 0.42$$

V2V code (Huffman design)

$\boldsymbol{a}_k$	p_k	codewords
0000000	0.7074	1
0000001	0.0141	0001
000001	0.0132	0000
00001	0.0116	00100
0001	0.0121	00101
001	0.0128	00110
01	0.0131	00111
1	0.2157	01

$$\left.\begin{array}{r}\bar{\ell}_{\text{seq}} = 1.496 \\ \bar{n} = 5.516\end{array}\right\} \rightarrow \bar{\ell} = 0.27$$

➡ **V2V code is better than block Huffman code with same table size (36 % bit savings)**

Optimal V2V Codes for Given Maximum Table Size

Optimal Code for Maximum Number N of Codewords ?

- ➡ No known design algorithm
- ➡ Exhaustive search over all possible full m-ary trees with up to N leaf nodes
- ➡ Extremely complex

Example: Stationary Markov Source

alphabet $\mathcal{A} = \{a, b, c\}$

x	$p(x \mid a)$	$p(x \mid b)$	$p(x \mid b)$
a	0.90	0.15	0.25
b	0.05	0.80	0.15
c	0.05	0.05	0.60

entropy rate $\bar{H} = 0.7331$

optimal codes: $\bar{\ell}$ for selected table sizes N

N	scalar	cond.	block	V2V
3	1.3556		1.3556	1.3556
9		1.1578	1.0094	1.0051
13				0.9412
17				0.9074
21				0.8891
27			0.9150	?
81			0.8690	?

V2V Codes in Practice

Only Structured V2V Codes

- Set of symbol sequences follow a certain structure

➡ Examples:
- run-length coding
- run-level coding

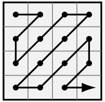
binary run-length coding

type 1	type 2
1	1 1 1 1 1
0 1	1 1 1 1 0
0 0 1	1 1 1 0
0 0 0 1	1 1 0
0 0 0 0 1	1 0
0 0 0 0 0 1	0 1
0 0 0 0 0 0 1	0 0 1
0 0 0 0 0 0 0 1	0 0 0 1
0 0 0 0 0 0 0 0 1	0 0 0 0 1
0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

run-level coding	
1	
⋮	
x	(x: max. value)
0 1	
0 ⋮	
0 x	
0 0 1	
0 0 ⋮	
0 0 x	
0 0 0 1	
0 0 0 ⋮	
0 0 0 x	
0 0 0 0 1	
0 0 0 0 ⋮	
0 0 0 0 x	
0 0 0 0 0 1	
0 0 0 0 0 ⋮	
0 0 0 0 0 x	
0 0 0 0 0 0 1	
0 0 0 0 0 0 ⋮	
0 0 0 0 0 0 x	
⋮	
0 0 0 0 0 0 0 ··· 0	

Run-Level Coding (JPEG, MPEG-2 Video, ...)

4	0	1	0
0	0	0	0
1	0	0	0
0	0	0	0



4, 0, 0, 1, 0, 1, 0, $\cdots$, 0

Coding of Block Quantization Indexes (absolute values)

1. Convert block into sequence of indexes (zig-zag scan)
2. Convert sequence of indexes into **(run, level) pairs** and a special **end-of-block (eob) symbol**

   run: number of zeros that precede next non-zero index
   level: value of next non-zero index
   eob: all following indexes are equal to zero (end-of-block)

   ➜ **Example:**

sequence of indexes:	**4 0 0 1 0 1 0 0 $\cdots$ 0**
(run, level) *pairs*:	(0, **4**) (2, **1**) (1, **1**) (**eob**)

3. **Codewords** are assigned to **(run, level) pairs**

MPEG-2 Video: 112 typical symbol sequences + *escape*

codeword	(run, level)	symbol sequence
10	(eob)	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, $\cdots$
11	(0,1)	1
011	(1,1)	0, 1
0100	(0,2)	2
0101	(2,1)	0, 0, 1
0010 1	(0,3)	3
0011 1	(3,1)	0, 0, 0, 1
0011 0	(4,1)	0, 0, 0, 0, 1
0001 10	(1,2)	0, 2
0001 11	(5,1)	0, 0, 0, 0, 0, 1
0001 01	(6,1)	0, 0, 0, 0, 0, 0, 1
0001 00	(7,1)	0, 0, 0, 0, 0, 0, 0, 1
0000 110	(0,4)	4
0000 100	(2,2)	0, 0, 2
0000 111	(8,1)	0, 0, 0, 0, 0, 0, 0, 0, 1
0000 101	(9,1)	0, 0, 0, 0, 0, 0, 0, 0, 0, 1
0000 01	*escape*	*< followed by fixed-length codes >*
0010 0110	(0,5)	5
0010 0001	(0,6)	6
0010 0101	(1,3)	0, 3
0010 0100	(3,2)	0, 0, 0, 2
0010 0111	(10,1)	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
0010 0011	(11,1)	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
0010 0010	(12,1)	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
...	...	...

Shannon-Fano-Elias Coding and Arithmetic Coding

Our Findings

- Achievable coding efficiency is limited by entropy rate: $\bar{\ell} \geq \bar{H}$
- Block Huffman codes approach entropy rate $\bar{H}$ for large block sizes N

$$\left(\frac{1}{N} H_N \right) \leq \bar{\ell} < \left(\frac{1}{N} H_N \right) + \left(\frac{1}{N} \right), \qquad \bar{H} = \lim_{N \to \infty} \frac{1}{N} H_N$$

➡ Not implementable due to extreme memory requirements for large N

Basic Idea of Arithmetic Coding

- Block codes for large N stay efficient even if they are slightly suboptimal, e.g.,

$$\left(\frac{1}{N} H_N \right) + \left(\frac{A}{N} \right) \leq \bar{\ell} < \left(\frac{1}{N} H_N \right) + \left(\frac{1+A}{N} \right) \quad \text{with} \quad A \ll N$$

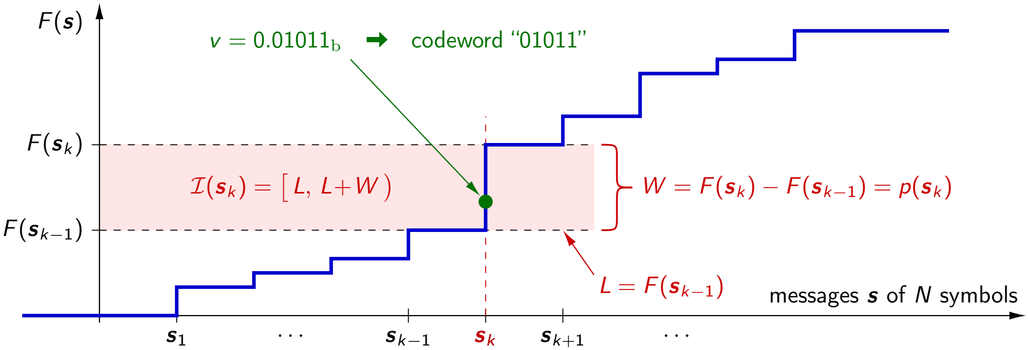
- If accepting suboptimality, can we construct codewords on-the-fly (i.e., without storing a large table) ?
  - ➡ **Arithmetic coding**
  - ➡ First: **Shannon-Fano-Elias coding** (idealized variant of arithmetic coding)

Basic Idea of Shannon-Fano-Elias Coding

Special Block Code for N symbols

- Order all possible symbol sequences with N symbols: $\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, \cdots$
- Each symbol sequence $\boldsymbol{s}_k$ is associated with a half-open interval $\mathcal{I}(\boldsymbol{s}_k) = \big[L, L+W\big)$ of the cdf $F(\boldsymbol{s})$

➡ Transmit any number v inside the interval $\mathcal{I}(\boldsymbol{s}_k)$ as binary fraction



Unique Identification of Probability Intervals

Probability Intervals

- All half-open intervals $\mathcal{I}(\boldsymbol{s}_k) \subset \big[0,1\big)$ are disjoint by definition
- Each interval $\mathcal{I}(\boldsymbol{s}_k) = \big[L,\, L+W\big)$ is characterized by

➡ **interval width**: $W \;=\; F(\boldsymbol{s}_k) - F(\boldsymbol{s}_{k-1}) \;=\; \mathrm{P}(\boldsymbol{S} = \boldsymbol{s}_k) \;=\; p(\boldsymbol{s}_k)$

➡ **lower boundary**: $L \;=\; F(\boldsymbol{s}_{k-1}) \;=\; \mathrm{P}(\boldsymbol{S} < \boldsymbol{s}_k) \;=\; \sum_{\forall i<k} p(\boldsymbol{s}_i)$

Identification of Intervals by Binary Fraction

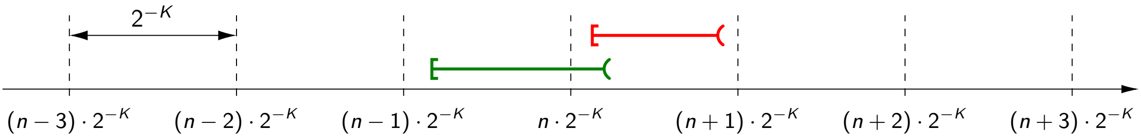
- All real numbers $v \in \big[0,1\big)$ belong to exactly one interval
- Represent number $v \in \mathcal{I}(\boldsymbol{s}_k)$ as binary fraction with K bits of precision

$$v = (0.b_1 b_2 b_3 \cdots b_K)_{\mathrm{b}} = \sum_{i=1}^{K} b_i \cdot 2^{-i} = z \cdot 2^{-K} \qquad (z \text{ is an integer})$$

➡ **codeword**: ➡ Bit sequence $\{b_1, b_2, b_3, \cdots, b_K\}$

➡ Binary representation of integer z with K bits

How Many Bits for Identifying an Interval ?



Required Number of Bits

- Distance between successive binary fractions of K bits is 2^{-K}

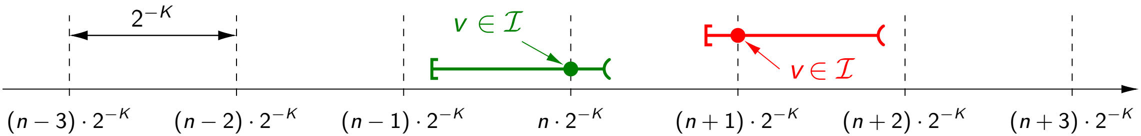
➡ For guaranteeing that a binary fraction of K bits falls inside an interval $\mathcal{I}(\boldsymbol{s}_k)$ of width W, we require

$$\begin{aligned} 2^{-K} &\leq W \\ K &\geq -\log_2 W \end{aligned}$$

➡ Hence, we choose

$$K = \left\lceil -\log_2 W \right\rceil = \left\lceil -\log_2 p(\boldsymbol{s}_k) \right\rceil$$

How To Select Codeword ?



Interval Representative (number $v \in \mathcal{I}$)

➡ Round up lower interval boundary L to next binary fraction of K bits

$$\mathcal{I} = [L, W) : \quad \Rightarrow \quad v = \lceil L \cdot 2^K \rceil \cdot 2^{-K} \quad \text{with} \quad K = \lceil -\log_2 W \rceil$$

Codeword

- K fractional bits of interval representative $v = (0.b_1 b_2 b_3 \cdots b_K)_\mathrm{b}$

➡ Binary representation $[b_1 b_2 \cdots b_K]$ with K bits of integer number

$$z = \lceil L \cdot 2^K \rceil = v \cdot 2^K$$

Shannon-Fano-Elias Encoding

given:
- ordered set of sequences $\{\boldsymbol{s}_k\}$
- associated pmf $p_k = p(\boldsymbol{s}_k)$

Codeword construction for $\boldsymbol{s}_k$

1 determine interval $\mathcal{I} = [L, L+W)$

$$W = p(\boldsymbol{s}_k) \quad \text{and} \quad L = \sum_{i<k} p(\boldsymbol{s}_i)$$

2 determine codeword length K

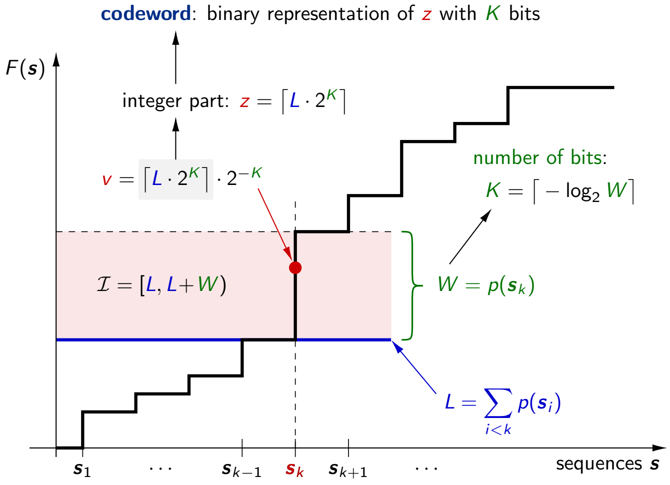
$$K = \lceil -\log_2 W \rceil$$

3 determine representative integer z

$$z = \lceil L \cdot 2^K \rceil$$

4 determine **codeword**

➡ K-bit representation of integer z



Shannon-Fano-Elias Decoding

given: • ordered set of sequences $\{\boldsymbol{s}_k\}$
• associated pmf $p_k = p(\boldsymbol{s}_k)$

Decode given codeword

1 read **codeword** ➡ integer z of K bits

2 initialization:

$$v = z \cdot 2^{-K}$$
$$k = 1 \qquad (\textit{message index})$$
$$U_k = L(\boldsymbol{s}_1) + W(\boldsymbol{s}_1) = p(\boldsymbol{s}_1)$$

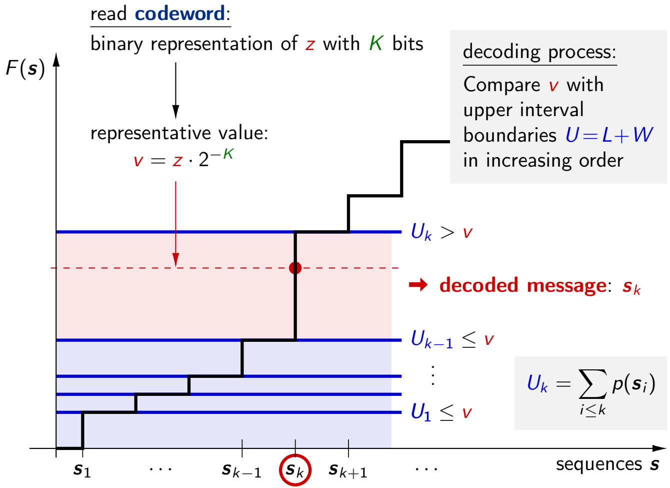
3 if ($v < U_k$)

➡ output $\boldsymbol{s}_k$ (*decoded message*)

else

➡ update: $k = k + 1$
$U_k = U_{k-1} + p(\boldsymbol{s}_k)$

➡ goto step **3**



Example: Shannon-Fano-Elias Code

Blocks of 3 Symbols for a Binary IID Source

➡ **Binary iid source with alphabet $\mathcal{A} = \{a, b\}$ and pmf $p = \{0.8, 0.2\}$**

s_k	p_k	W_k	L_k	K_k	z_k	codeword
aaa	0.512	0.512	0.000	1	0	0
aab	0.128	0.128	0.512	3	5	101
aba	0.128	0.128	0.640	3	6	110
abb	0.032	0.032	0.768	5	25	11001
baa	0.128	0.128	0.800	3	7	111
bab	0.032	0.032	0.928	5	30	11110
bba	0.032	0.032	0.960	5	31	11111
bbb	0.008	0.008	0.992	7	127	1111111

average codeword length: $\bar{\ell} = 0.733$

block Huffman code: $\bar{\ell} = 0.728$

$$W_k = p_k$$

$$L_k = \sum_{i<k} p_i$$

$$K_k = \lceil -\log_2 W_k \rceil$$

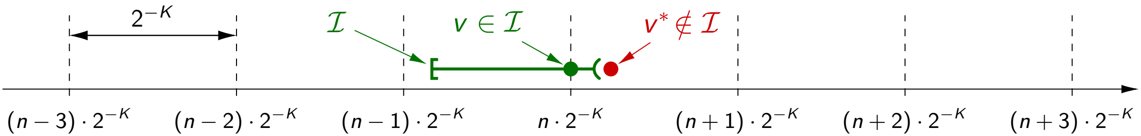
$$z_k = \lceil L_k \cdot 2^{K_k} \rceil$$

codeword :
z_k with K_k bits

➡ Worse than block Huffman code for same block size ($N = 3$)

➡ **Code is not prefix-free !** ➡ Can be a problem (depends on application) !

Why Is The Code Not Prefix-Free ?



- Encoder transmits codeword of K bits, signaling the binary fraction $v \in \mathcal{I}$

$$v = (0.b_1 b_2 b_3 \cdots b_K)_\mathrm{b}$$

- Decoder sees a modified binary fraction v^* given by

$$v^* = (0.b_1 b_2 b_3 \cdots b_K b_{K+2} b_{K+3} \cdots)_\mathrm{b}$$

where $\{b_{K+1} b_{K+2} \cdots\}$ are the bits of following codewords

➡ **Value v^* seen by decoder can lay outside the interval $\mathcal{I}$**

Prefix-Free Variant: How Can We Fix That ?

- Need to ensure that $v^* < L + W$

$$
\begin{array}{rrcl}
\text{worst case}: & v^* = v + \sum_{i=K+1}^{\infty} 2^{-i} & < & L + W \\
\text{sufficient}: & v + 2^{-K} & \leq & L + W \\
v = \lceil L \cdot 2^K \rceil \, 2^{-K}: & \lceil L \cdot 2^K \rceil \cdot 2^{-K} + 2^{-K} & \leq & L + W \\
\lceil x \rceil < x + 1: & (L \cdot 2^K + 1) \cdot 2^{-K} + 2^{-K} & \leq & L + W \\
 & L + 2 \cdot 2^{-K} & \leq & L + W \\
 & 2^{1-K} & \leq & W
\end{array}
$$

→ need : $\boxed{K \geq 1 - \log_2 W}$

- Unique decodability is guaranteed, if we choose

→ prefix-free : $\boxed{K = \lceil 1 - \log_2 W \rceil}$

- **Require one additional bit per codeword** (i.e., per N symbols)

Example: Prefix-Free Shannon-Fano-Elias Code

Repeated: Blocks of 3 Symbols for a Binary IID Source

➡ **Binary iid source with alphabet $\mathcal{A} = \{a, b\}$ and pmf $p = \{0.8, 0.2\}$**

s_k	p_k	W_k	L_k	K_k	z_k	codeword
aaa	0.512	0.512	0.000	2	0	00
aab	0.128	0.128	0.512	4	9	1001
aba	0.128	0.128	0.640	4	11	1011
abb	0.032	0.032	0.768	6	50	110010
baa	0.128	0.128	0.800	4	13	1101
bab	0.032	0.032	0.928	6	60	111100
bba	0.032	0.032	0.960	6	62	111110
bbb	0.008	0.008	0.992	8	254	11111110

average codeword length: $\bar{\ell} = 1.067$

block Huffman code: $\bar{\ell} = 0.728$

$$W_k = p_k$$

$$L_k = \sum_{i<k} p_i$$

$$K_k = \lceil 1 - \log_2 W_k \rceil$$

$$z_k = \lceil L_k \cdot 2^{K_k} \rceil$$

codeword :

z_k with K_k bits

➡ Additional bit ensures that code becomes a **prefix code**

➡ Worse than block Huffman code (several redundant bits)

Efficiency of Shannon-Fano-Elias Codes

Average Codeword Length

- Average codeword length $\bar{\ell}$ per symbol (for N-symbol messages $\boldsymbol{S}$)

$$\bar{\ell} = \frac{\mathrm{E}\Big\{ K(\boldsymbol{S}) \Big\}}{N} = \frac{\mathrm{E}\Big\{ \lceil A - \log_2 p_N(\boldsymbol{S}) \rceil \Big\}}{N} \qquad \text{with} \qquad A = \begin{cases} 1 & : \text{ prefix-free} \\ 0 & : \text{ otherwise} \end{cases}$$

Bounds on Average Codeword Length

- Using the inequality $x \le \lceil x \rceil < x + 1$, we obtain

$$\frac{\mathrm{E}\{ -\log_2 p_N(\boldsymbol{S}) \}}{N} + \frac{A}{N} \le \bar{\ell} < \frac{\mathrm{E}\{ -\log_2 p_N(\boldsymbol{S}) \}}{N} + \frac{1+A}{N}$$

$$\frac{H_N(\boldsymbol{S})}{N} + \frac{A}{N} \le \bar{\ell} < \frac{H_N(\boldsymbol{S})}{N} + \frac{1+A}{N}$$

➡ Non-prefix-free version ($A = 0$): Same bounds as for block Huffman coding

➡ Both versions: **Close to entropy rate for $N \gg 1$** (for typical sources)

Shannon-Fano-Elias Coding: Intermediate Results

Shannon-Fano-Elias Code

- Special block code (for given number of symbols N)
- Worse than block Huffman code of same size N
- Still close to entropy bound (H_N/N) for $N \gg 1$

➡ **No need to store codeword table !**

➡ **Have to store N-th order joint pmf (or N-th order joint cdf) !**

What is the advantage ?

Iterative Coding

- Can define a suitable order for sequences of N symbols

➡ Iterative calculation of interval boundaries

➡ Iterative codeword construction

Lexicographical Order: Nested Probability Intervals

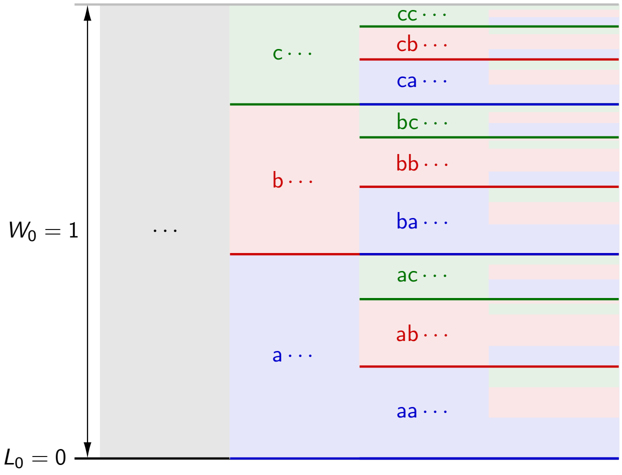
Lexicographical Order

- Sorted alphabet $\mathcal{A} = \{a_1, a_2, a_3, \cdots\}$
- Two symbol sequences: $\boldsymbol{x} < \boldsymbol{y}$ iff
  $\exists n: \left(\forall k < n: x_k = y_k\right) \wedge \left(x_n < y_n\right)$

Example: $\mathcal{A} = \{a, b, c\}$

$N = 4$:

a a a a
a a a b
a a a c
a a b a
a a b b
a a b c
a a c a
a a c b
a a c c
a b a a
a b a b
a b a c
$\cdots$



➡ **probability intervals are nested !**

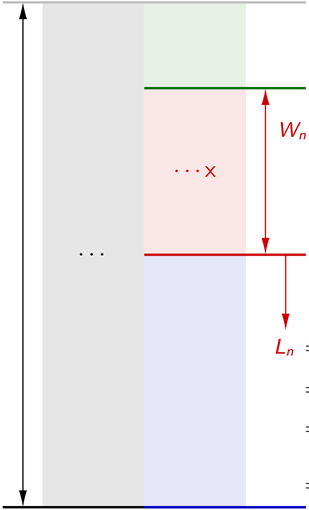
Iterative Interval Refinement

$$
\begin{aligned}
W_0 &= 1 \\
L_0 &= 0 \\
W_n &= W_{n-1} \cdot p(s_n \mid \cdots) \\
L_n &= L_{n-1} + W_{n-1} \cdot c(s_n \mid \cdots)
\end{aligned}
$$

$$
c(x \mid \cdots) = \sum_{\forall a < x} p(a \mid \cdots)
$$

$$
\mathrm{P}\big(\boldsymbol{S}^{n-1} = \cdots\big) = W_{n-1}
$$

$$
\mathrm{P}\big(\boldsymbol{S}^{n-1} < \cdots\big) = L_{n-1}
$$



$$
\begin{aligned}
W_n &= \mathrm{P}(\boldsymbol{S}^n = \cdots x) \\
&= \mathrm{P}\big(\boldsymbol{S}^{n-1} = \cdots\big) \cdot \mathrm{P}(S_n = x \mid \cdots) \\
&= W_{n-1} \cdot p(x \mid \cdots)
\end{aligned}
$$

$$
\begin{aligned}
L_n &= \mathrm{P}(\boldsymbol{S}^n < \cdots x) \\
&= \mathrm{P}\big(\boldsymbol{S}^{n-1} < \cdots\big) + \mathrm{P}\big(\boldsymbol{S}^{n-1} = \cdots\big) \cdot \mathrm{P}(S_n < x \mid \cdots) \\
&= L_{n-1} + W_{n-1} \cdot \sum_{\forall a < x} p(a \mid \cdots) \\
&= L_{n-1} + W_{n-1} \cdot c(x \mid \cdots)
\end{aligned}
$$

Iterative Interval Refinement in Practice

Iterative Algorithm for Calculating Interval Boundaries

- Initialization: $W_0 = 1$, $L_0 = 0$
- Iteration Step: $W_n = W_{n-1} \cdot p(s_n \,|\, \cdots)$, $L_n = L_{n-1} + W_{n-1} \cdot c(s_n \,|\, \cdots)$ with $c(x \,|\, \cdots) = \sum_{\forall a<x} p(a \,|\, \cdots)$

➡ Require N-th order conditional pmf instead of N-th order joint pmf

➡ Same amount of data ! ➡ What is the advantage ?

Iterative Refinement in Practice

- Conditional pmfs can be well approximated using simple models

➡ IID model: $p(s_n \,|\, s_{n-1}, \cdots) = p(s_n)$

➡ Markov model: $p(s_n \,|\, s_{n-1}, \cdots) = p(s_n \,|\, s_{n-1})$

➡ Simple function: $p(s_n \,|\, s_{n-1}, \cdots) = p(s_n \,|\, f(s_{n-1}, \cdots))$

Iterative Encoding Algorithm

Given: Sequence $\boldsymbol{s} = \{s_1, s_2, s_3, \cdots, s_N\}$ of N symbols

1. Initialization of probability interval

$$W_0 = 1 \qquad \text{and} \qquad L_0 = 0$$

2. Determine probability interval $\big[L_N, L_N + W_N\big)$:

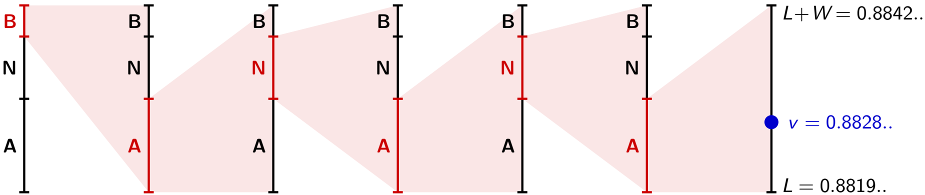
$$\text{for } \; n = 1 \; \text{ to } \; N: \qquad \begin{aligned} W_n &= W_{n-1} \cdot p(\, s_n \,|\, \cdots) \\ L_n &= L_{n-1} + W_{n-1} \cdot c(\, s_n \,|\, \cdots) \end{aligned}$$

3. Determine codeword length and codeword value

$$\begin{aligned} K &= \big\lceil -\log_2 W_N \big\rceil \qquad (\text{for prefix-free variant: } K \rightarrow K+1) \\ z &= \big\lceil L_N \cdot 2^K \big\rceil \end{aligned}$$

4. Transmit codeword: Binary representation of z with K bits

Iterative Encoding Example: IID Source



a	$p(a)$	$c(a)$
A	$\frac{1}{2}$	0
N	$\frac{1}{3}$	$\frac{1}{2}$
B	$\frac{1}{6}$	$\frac{5}{6}$

$$W_{n+1} = W_n \cdot p(s_n)$$

$$L_{n+1} = L_n + W_n \cdot c(s_n)$$

	init	B	A	N	A	N	A
W_n	1	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{36}$	$\frac{1}{72}$	$\frac{1}{216}$	$\frac{1}{432}$
L_n	0	$\frac{5}{6}$	$\frac{5}{6}$	$\frac{21}{24}$	$\frac{21}{24}$	$\frac{127}{144}$	$\frac{127}{144}$

$$K = \lceil -\log_2 W \rceil = \lceil \log_2 432 \rceil = 9$$

$$z = \lceil L \cdot 2^K \rceil = \left\lceil \frac{127}{144} \cdot 512 \right\rceil = 452 \qquad \left(v = \frac{452}{512} \right)$$

$\boldsymbol{b}$ = "**111000100**" ($z = 452$ with $K = 9$ bits)

Iterative Decoding Algorithm

Given:
- Bitstream $\{b_1, b_2, b_3, \cdots, b_M\}$ of $M \geq K$ bits
- Number N of symbols to be decoded

1. Determine interval representative: $v = (0.b_1 b_2 b_3 \cdots b_M)_\mathrm{b} = z \cdot 2^{-M}$
2. Initialization of probability interval: $W_0 = 1$ and $L_0 = 0$
3. For $n = 1$ to N: (*iterative decoding*)
   - a. Initialization of upper interval boundary U_1 for first symbol a_1 of sorted alphabet

     $$k = 1, \qquad U_k = L_{n-1} + W_{n-1} \cdot p(\, a_k \mid \cdots)$$

   - b. While ($v \geq U_k$), update upper boundary for next alphabet symbol

     $$k = k + 1, \qquad U_k = U_{k-1} + W_{n-1} \cdot p(\, a_k \mid \cdots)$$

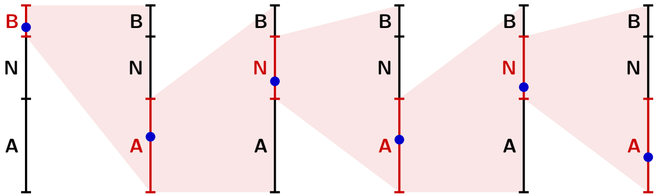
   - c. Output symbol a_k and update probability interval

     $$\begin{aligned} W_n &= W_{n-1} \cdot p(\, a_k \mid \cdots) \\ L_n &= U_k - W_n \end{aligned}$$

Iterative Decoding Example: IID Source

$$L_{n+1} = L_n + W_n \cdot c(.)$$

$$W_{n+1} = W_n \cdot p(.)$$



(L_n, W_n)	$0,1$	$\frac{5}{6},\frac{1}{6}$	$\frac{5}{6},\frac{1}{12}$	$\frac{21}{24},\frac{1}{36}$	$\frac{21}{24},\frac{1}{72}$	$\frac{127}{144},\frac{1}{216}$
(L_{n+1}, W_{n+1}) (A)	$0,\frac{1}{2}$	$\frac{5}{6},\frac{1}{12}$	$\frac{5}{6},\frac{1}{24}$	$\frac{21}{24},\frac{1}{72}$	$\frac{21}{24},\frac{1}{144}$	$\frac{127}{144},\frac{1}{432}$
(L_{n+1}, W_{n+1}) (N)	$\frac{1}{2},\frac{1}{3}$	$\frac{11}{12},\frac{1}{18}$	$\frac{21}{24},\frac{1}{36}$	$\frac{8}{9},\frac{1}{108}$	$\frac{127}{144},\frac{1}{216}$	$\frac{191}{216},\frac{1}{648}$
(L_{n+1}, W_{n+1}) (B)	$\frac{5}{6},\frac{1}{6}$	$\frac{35}{36},\frac{1}{36}$	$\frac{65}{72},\frac{1}{72}$	$\frac{97}{108},\frac{1}{216}$	$\frac{383}{432},\frac{1}{432}$	$\frac{287}{324},\frac{1}{1296}$
symbol s_n	**B**	**A**	**N**	**A**	**N**	**A**

$v = \frac{452}{512}$ $\quad$ $\boldsymbol{b}$ = "111000100" ➡ $\boldsymbol{s}$ = "BANANA"

Summary of Lecture: Universal, V2V, Shannon-Fano-Elias Codes

Universal Codes

- Follow certain structure ➡ No codeword table required
- Examples for coding non-negative integers: Unary code, Rice codes, Exp-Golomb codes

V2V Codes

- Mapping of variable-length symbol sequences to codewords

➡ Typically higher efficiency than block Huffman codes with same number of codewords

Shannon-Fano-Elias Codes

- Sub-optimal block codes (still close to entropy rate for $N \gg 1$)
- No codeword table required

➡ Iterative encoding and decoding procedure

➡ Precursor of **arithmetic coding** (used in most modern codec's)

Exercise 1: V2V Codes for Black and White Document Scans (Part 1/2)

Analyze a structured V2V code for coding 300dpi black and white document scans

1 Write a program that reads all binary samples of a document scan into an array of bits (e.g., of type `vector<bool>` if you use C++)

The original document files are coded in the PBM format, which is a raw data format (see description on the right hand side).

The following files (found on the course web site) should be used as examples:

- "paper300dpi-page00.pbm"
- "paper300dpi-page01.pbm"
- "paper300dpi-page02.pbm"
- "paper300dpi-page03.pbm"

structure of "pbm" files:

```
P4              // ascii (fixed)
width height    // ascii
<binary data> // binary
```

binary data:

- samples in raster-scan order (line by line)
- each sample is represented by one bit
  - ➡ bit 0 $\rightarrow$ white sample
  - ➡ bit 1 $\rightarrow$ black sample
- 8 bits are packet in one byte, where the first sample in scan order is placed in the most significant bit
- the first byte of the binary data contains the first 8 bits in scan order, etc.

Exercise 1: V2V Codes for Black and White Document Scans (Part 2/2)

2 Extend your program as follows:

Experimentally determine the probabilities for the symbol sequences of the two codes (block code and V2V code) shown on the right hand side.

3 Develop optimal codeword tables for both cases (using the Huffman algorithm).

You can do it on paper or implement it.

4 Calculate the average codeword length (per binary sample) for both developed codes.

Which code would yield a better compression efficiency?

block code	V2V code
0000	0000 0000 0000 000
0001	0000 0000 0000 001
0010	0000 0000 0000 01
0011	0000 0000 0000 1
0100	0000 0000 0001
0101	0000 0000 001
0110	0000 0000 01
0111	0000 0000 1
1000	0000 0001
1001	0000 001
1010	0000 01
1011	0000 1
1100	0001
1101	001
1110	01
1111	1

Exercise 2: Audio Coding using Rice Codes

Investigate lossless audio coding with Rice codes. Use the example file "audioData.raw" (from the course web site) for these investigations. The file consists of raw audio data in signed 8-bit format. That means, each byte of the file represents one sample and has to be interpreted as 8-bit signed integer.

1 Write an encoder and decoder for coding the audio data using Rice codes.

- Each sample x_n should be coded as: abs $\rightarrow$ Rice code for abs(x_n)
  if(abs > 0)
  sign $\rightarrow$ single bit indicating the sign
- The Rice parameter should be given as input to the encoder and written at the beginning of the bitstream (e.g., using a fixed-length code of 8 bits or a unary code).
- Check that the decoder decodes the file correctly.
- Try different Rice parameters and measure the size of the generated bitstream.

2 (Optional) Try to improve your lossless audio codec by coding the audio samples using chunks of 1024 successive samples.

- Determine the optimal Rice parameter for each chunk.
- Code the Rice parameter at the beginning of each chunk.

Exercise 3: Iterative Shannon-Fano-Elias Coding

Given is an IID source with the alphabet $\mathcal{A} = \{\ \text{E, R, F}\ \}$ and the pmf

symbol	probability
E	$^5/_8$
R	$^2/_8$
F	$^1/_8$

1. Construct the Shannon-Fano-Elias codeword for the message “REFEREE” using the iterative encoding algorithm.
   - Use the prefix-free variant (only important at the end).
   - Assume that the symbols in the alphabet are ordered as: E, R, F.
2. Verify that the original message can be correctly decoded from the codeword using the iterative decoding algorithm.

Feel free to implement the encoding and decoding (instead of doing it on paper).