05

Symbol Codes

Notice

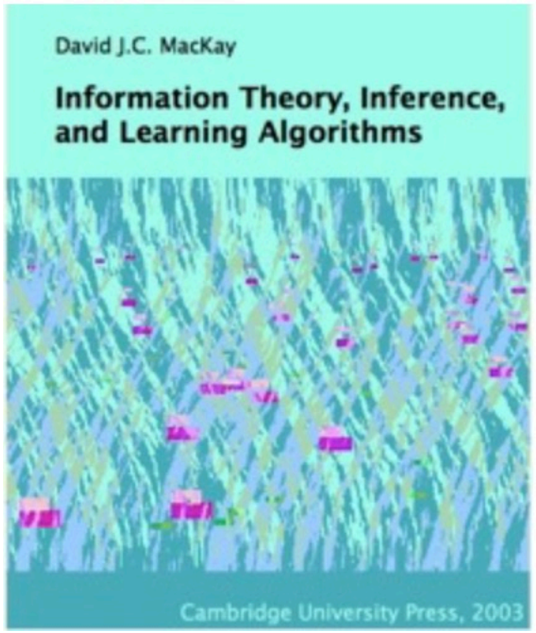
- **Author**
  - **João Moura Pires (jmp@fct.unl.pt)**

- **This material can be freely used for personal or academic purposes without any previous authorization from the author, provided that this notice is maintained/kept.**

- **For commercial purposes the use of any part of this material requires the previous authorization from the author.**

Bibliography

- **Many examples are extracted and adapted from:**



Information Theory, Inference, and Learning Algorithms
David J.C. MacKay
2005, Version 7.2

- **And some slides were based on Iain Murray course**
  - **http://www.inf.ed.ac.uk/teaching/courses/it/2014/**

Table of Contents

Symbol codes

Symbol codes

- **Variable-length symbol codes**,
  - Encode **one source symbol at a time** (instead of encoding huge strings of *N* source symbols);
  - These codes are **lossless**: they are guaranteed to compress and decompress without any errors;
  - There is a chance that the codes may **sometimes produce encoded strings longer than the original** source string.

The idea is that we can achieve compression, **on average**, by **assigning shorter encodings** to the **more probable outcomes** and longer encodings to the less probable

Key issues

- **What are the implications if a symbol code is lossless?**
  - If some codewords are shortened, by how much do other codewords have to be lengthened?
- **Making compression practical**.
  - How can we ensure that a symbol code is easy to decode?
- **Optimal symbol codes**.
  - How should we assign codelengths to achieve the best compression,
  - What is the best achievable compression?

Source coding theorem (symbol codes)

A (binary) symbol code

- **Notation on Alpahabets**
  - A^N . denotes the set of ordered N-tuples of elements from the set A, i.e., **all strings of length N**.
  - A^+ will denote the set of **all strings of finite length** composed of elements from the set A.
  - Examples:
    - $\{0, 1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$.
    - $\{0, 1\}^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$.

A (binary) symbol code

- **A (binary) symbol code** C for an ensemble X is a **mapping** from the range of x,

  $A_X = \{a_1, ..., a_I\}$, to $\{0,1\}^+$.

  - $c(x)$ will denote the codeword corresponding to x
  - $l(x)$ will denote its length, with $l_i = l(a_i)$.

- The *extended code* C^+ is a **mapping** from A_X^+ to $\{0,1\}^+$ obtained by concatenation, without punctuation, of the corresponding codewords: $c^+(x_1x_2...x_N) = c(x_1)c(x_2)...c(x_N)$

- Basic requirements for a **useful symbol code**

  - *Any encoded string must have a **unique decoding***
  - *The symbol code must be **easy to decode***
  - *The code should achieve **as much compression as possible**.*

A (binary) symbol code - **example**

- **A (binary) symbol code** C_0 for an ensemble X

  A_X ={a, b, c, d}

  P_X = {1/2, 1/4, 1/8, 1/8}

  - $c(x)$ will denote the codeword corresponding to x
  - $l(x)$ will denote its length, with $l_i = l(a_i)$.

C_0:

a_i	$c(a_i)$	l_i
a	1000	4
b	0100	4
c	0010	4
d	0001	4

- Using the extended code, we may encode **acdbac** as

  c^+(**acdbac**) = **1000001000010100100000010**

  a c d b a c

Any encoded string must have a unique decoding

- A code $C(X)$ is **uniquely decodeable**

  If, under the extended code C^+, **no two distinct strings have the same encoding**, i.e.,

$$\forall x, y \in A_X^+, x \neq y \Rightarrow c^+(x) \neq c^+(y)$$

- The example is a uniquely decodeable code

  c^+(**acdbac**) = **100000100001010010000010**

  a c d b a c

  How to prove for this example?

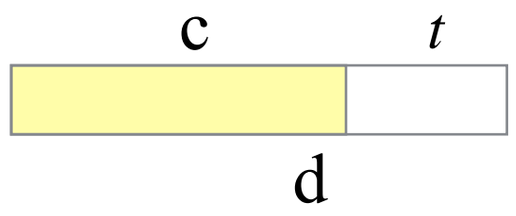
C_0:

a_i	$c(a_i)$	l_i
a	1000	4
b	0100	4
c	0010	4
d	0001	4

The symbol code must be easy to decode

- A symbol code is easiest to decode if it **is possible to identify the end of a codeword as soon as it arrives**,
  - which means that **no codeword can be a *prefix* of another codeword**

- A word *c* is a prefix of another word *d*
  - if there exists a tail string *t* such that the concatenation *ct* is identical to *d*.
  - Example
    - 1 is prefix of 100
    - 10 is prefix of 100



The symbol code must be easy to decode

- **A symbol code is called a prefix code**

  if **no codeword is a prefix of any other codeword**.

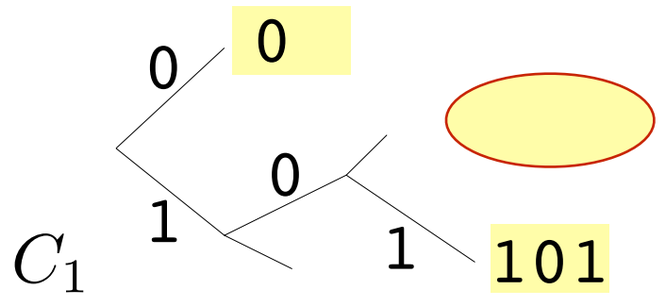
- A prefix code is also known as an ***instantaneous*** or *self-punctuating* code, because an encoded string can be decoded from left to right without looking ahead to subsequent codewords.

- The **end of a codeword is immediately recognizable**.

- A **prefix code is uniquely decodeable**.

Prefix codes - examples

- The code $C_1 = \{0, 101\}$ is a prefix code because 0 is not a prefix of 101, nor is 101 a prefix of 0.

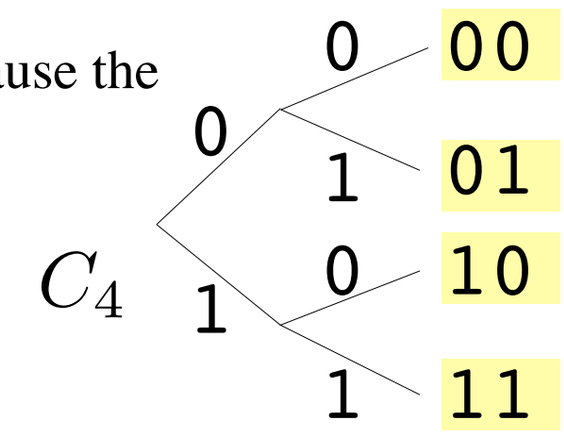


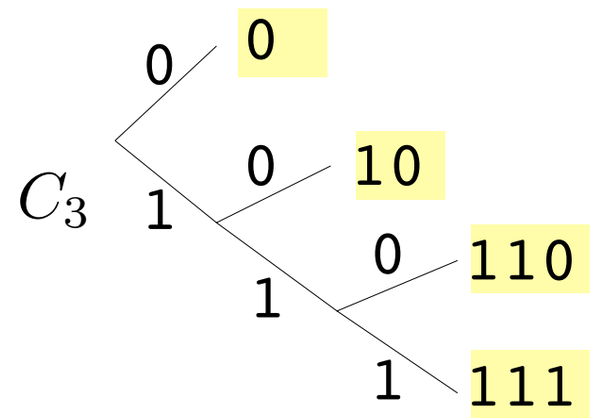
C_1 is an *incomplete* code

- The code $C_2 = \{1, 101\}$ is not a prefix code **because 1 is a prefix of 101**.

- The code $C_3 = \{0, 10, 110, 111\}$ is a prefix code

- The code $C_4 = \{00, 01, 10, 11\}$ is a prefix code.
  - C_4 is said to be *complete*, because the corresponding binary tree has no unused branch





The code should achieve as much compression as possible

- The **expected length** $L(C, X)$ of a symbol code C for an ensemble X is

$$L(C,X) = \sum_{x \in A_X} P(x) l(x)$$

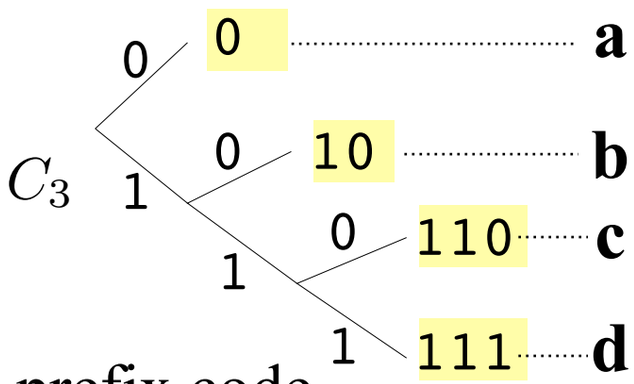
$$L(C,X) = \sum_{i=1}^{I} p_i l_i \qquad \text{where} \quad I = \left| A_X \right|$$

- **A (binary) symbol code** C_3 for an ensemble X

$$l_i = h_i = \log_2(1/p_i)$$

$A_X = \{a, b, c, d\}$ $\quad H(X) = 1.75 bits$

$P_X = \{1/2, 1/4, 1/8, 1/8\}$



C_3 is prefix code

$L(C,X) = 1.75 bits$

C_3:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	1/2	1.0	1
b	10	1/4	2.0	2
c	110	1/8	3.0	3
d	111	1/8	3.0	3

The code should achieve as much compression as possible

- **A (binary) symbol code** C_6 for an ensemble X

$A_X = \{a, b, c, d\}$; $P_X = \{1/2, 1/4, 1/8, 1/8\}$ $\quad H(X) = 1.75 bits$

C_6:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	01	$1/4$	2.0	2
c	011	$1/8$	3.0	3
d	111	$1/8$	3.0	3

$\boxed{L(C,X) = 1.75 bits}$

Is C_6 a prefix code? No

Is C_6 a uniquely decodeable? Yes

Certainly isn't easy to decode

The code should achieve as much compression as possible

- **A (binary) symbol code** C_6 for an ensemble X

$A_X = \{a, b, c, d\}$; $P_X = \{1/2, 1/4, 1/8, 1/8\}$ $\quad H(X) = 1.75 bits$

C_6:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	01	$1/4$	2.0	2
c	011	$1/8$	3.0	3
d	111	$1/8$	3.0	3

C_3:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	10	$1/4$	2.0	2
c	110	$1/8$	3.0	3
d	111	$1/8$	3.0	3

the codewords of C_6 are the reverse of C_3

What limit is imposed by unique decodeability?

What limit is imposed by unique decodeability?

- Given a list of positive integers $\{l_i\}$, does there exist a uniquely decodeable code with those integers as its codeword lengths?
- We have observed that if we take a code such as $\{00, 01, 10, 11\}$,
  - shorten one of its codewords, for example $00 \rightarrow 0$,
  - then we can retain unique decodeability only if we lengthen other codewords !
- Thus there seems to be a **constrained budget** that we can spend on codewords, with **shorter codewords being more expensive**.
  - In general for a code with l (constant bits) we have 2^l possible codewords.
  - **So the "cost" of a codeword of length l is $1/2^l$**

Kraft inequality

- **Kraft inequality**. For **any uniquely decodeable** code $C(X)$ over the binary alphabet $\{0, 1\}$, the codeword lengths must satisfy:

$$\sum_{i=1}^{I} 2^{-l_i} \leq 1 \quad \text{where} \quad I = \left| A_X \right|$$

- **Completeness**. If a uniquely dcodeable codes **satisfies the Kraft inequality with equality** then the code it is called **complete code**.

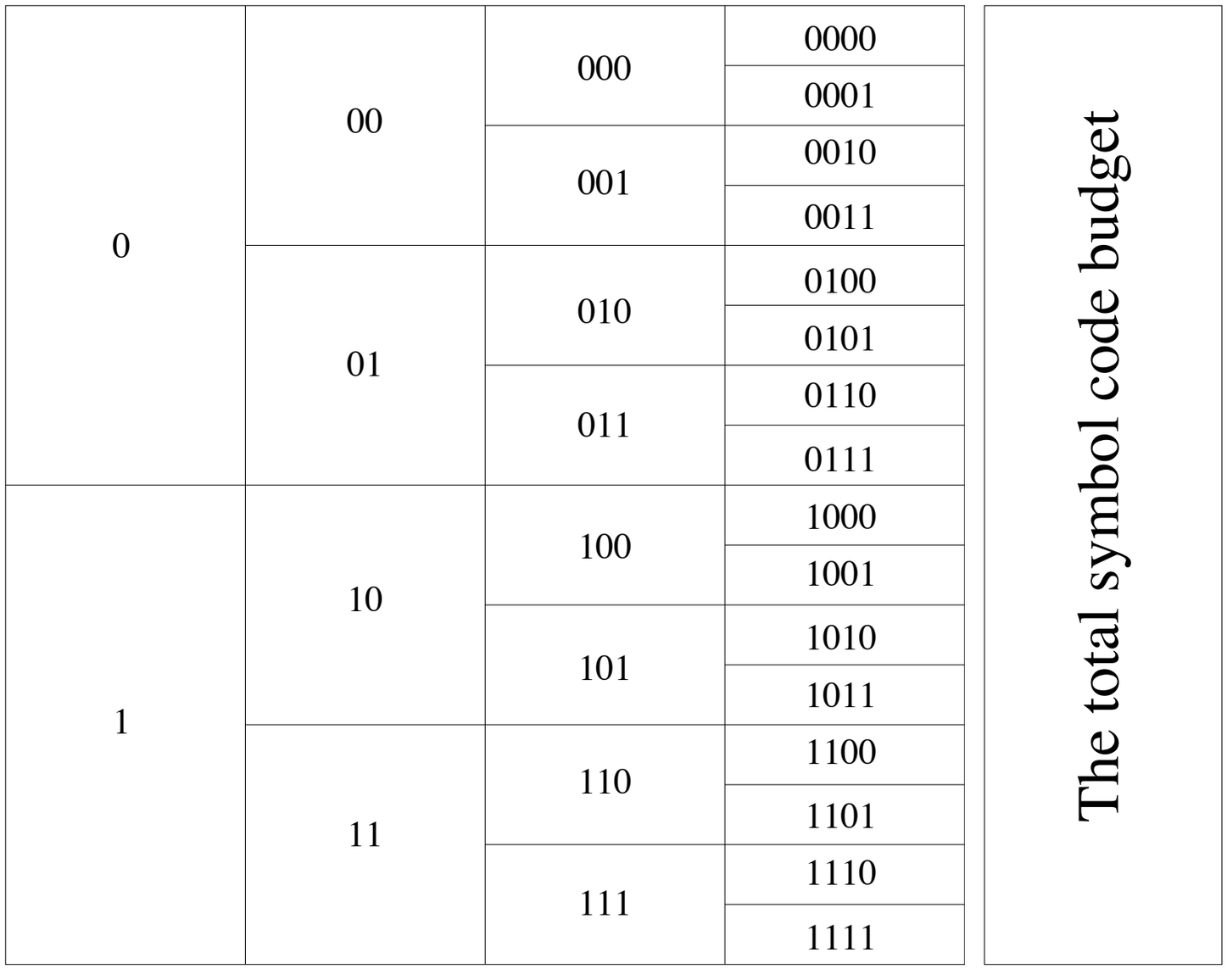
$$\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^3} = \frac{1}{2} + \frac{1}{4} + \frac{1}{4} = 1$$

C_3:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	10	$1/4$	2.0	2
c	110	$1/8$	3.0	3
d	111	$1/8$	3.0	3

The symbol coding budget

- The "**cost**" of 2^{-l} of each codeword with length l is indicated by the **size of the box** is written in.



The **total budget available** when making a uniquely decodeable code is 1.

$$\sum_{i=1}^{I} 2^{-l_i} \leq 1$$

If the cost of the codewords that you take exceeds the budget then your code will not be uniquely decodeable.

The symbol coding budget - Code C_0

- Code C_0

0	00	000	0000	
			0001	**d**
		001	0010	**c**
			0011	
	01	010	0100	**b**
			0101	
		011	0110	
			0111	
1	10	100	1000	**a**
			1001	
		101	1010	
			1011	
	11	110	1100	
			1101	
		111	1110	
			1111	

$$\sum_{i=1}^{I} 2^{-l_i} = 4\frac{1}{2^4} = 4\frac{1}{16} = \frac{1}{4} \leq 1$$

C_0:

a_i	$c(a_i)$	l_i
a	1000	4
b	0100	4
c	0010	4
d	0001	4

The symbol coding budget - Code C_3

- Code C_3

0 **a**	00	000	0000
			0001
		001	0010
			0011
	01	010	0100
			0101
		011	0110
			0111
1	10 **b**	100	1000
			1001
		101	1010
			1011
	11	**c** 110	1100
			1101
		d 111	1110
			1111

$$\sum_{i=1}^{I} 2^{-l_i} = 1$$

C_3:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	10	$1/4$	2.0	2
c	110	$1/8$	3.0	3
d	111	$1/8$	3.0	3

The symbol coding budget - Code C_4

- Code C_4

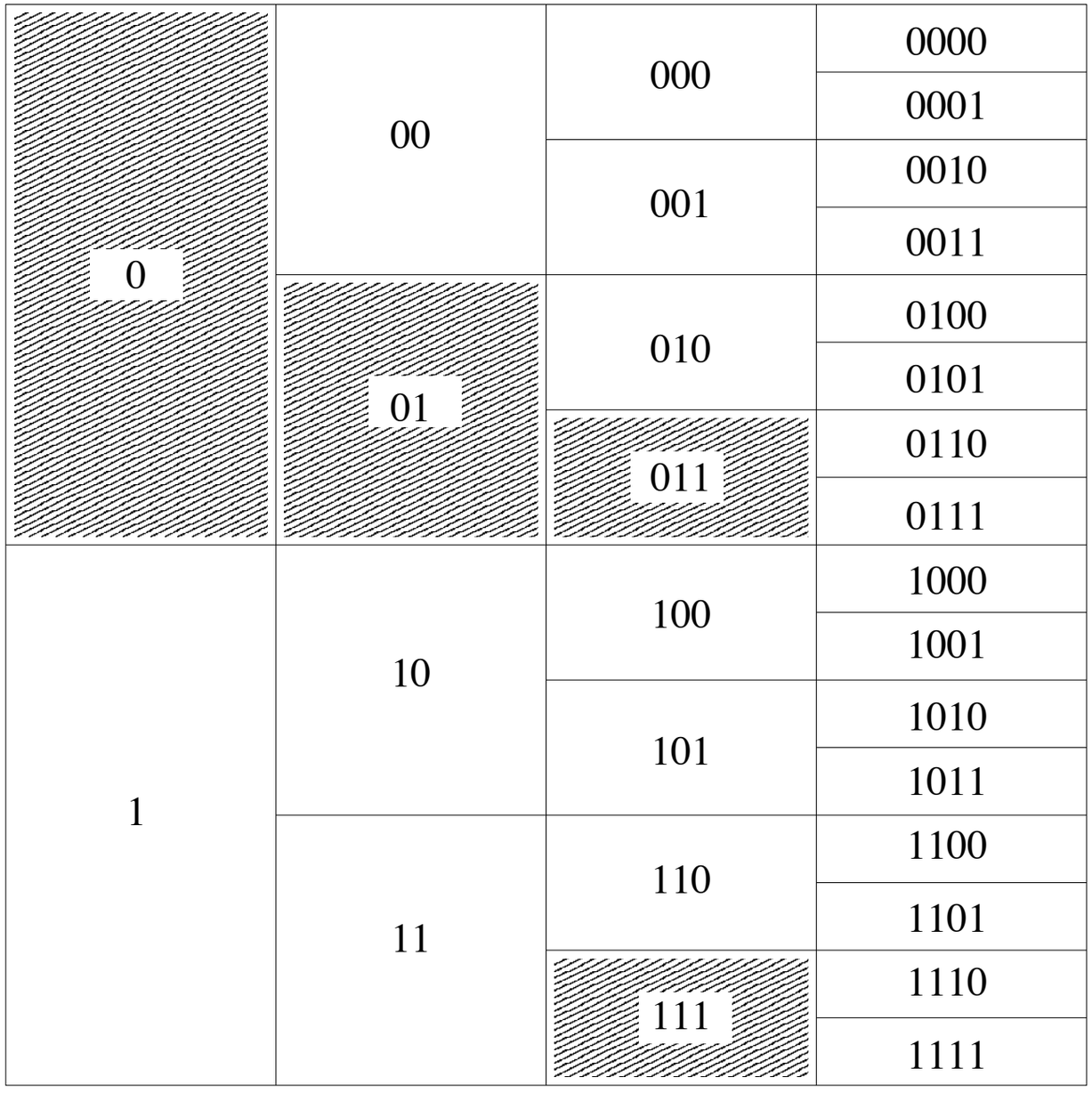
0	00	000	0000
			0001
		001	0010
			0011
	01	010	0100
			0101
		011	0110
			0111
1	10	100	1000
			1001
		101	1010
			1011
	11	110	1100
			1101
		111	1110
			1111

$$\sum_{i=1}^{I} 2^{-l_i} = 1$$

	C_4	C_5
a	00	0
b	01	1
c	10	00
d	11	11

The symbol coding budget - Code C_6

- Code C_6



$$\sum_{i=1}^{I} 2^{-l_i} = 1$$

C_6:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	01	$1/4$	2.0	2
c	011	$1/8$	3.0	3
d	111	$1/8$	3.0	3

Kraft inequality and prefix codes

- We want codes that are **uniquely decodeable.**
- Prefix codes are uniquely decodeable, and are easy to decode.
- **For any source there is an optimal symbol code that is also a prefix code.**

- **Kraft inequality and prefix codes.**
  - **Given a set of codeword lengths that satisfy the Kraft inequality, there exists a uniquely decodeable prefix code with these codeword lengths.**

What's the most compression that we can hope for?

What's the most compression that we can hope for?

- We wish to minimize the expected length of a code

$$L(C,X)=\sum_i p_i l_i$$ ***P***

- **Lower bound on expected length**. The expected length $L(C, X)$ of a uniquely decodeable code is bounded below by $H(X)$.

- Let define the *implicit* probabilities $q_i \equiv 2^{-l_i} / z$, where $z=\sum_{i'} 2^{-l_{i'}}$ ***Q***

- So $l_i = \log {}^{1}\!/_{q_i} - \log z$

- Now, consider the relative entropy between P and Q and the Gibb's inequality

$$D_{KL}(P \| Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \qquad D_{KL}(P \| Q) \geq 0$$

$$= \sum_x P(x) \log P(x) - \sum_x P(x) \log Q(x)$$

$$= \sum_x P(x) \log {}^{1}\!/_{Q(x)} - \sum_x P(x) \log {}^{1}\!/_{P(x)}$$

What's the most compression that we can hope for?

- Let define the *implicit* probabilities $q_i \equiv 2^{-l_i} / z$, where $z = \sum_{i'} 2^{-l_{i'}}$ $\qquad L(C,X) = \sum_i p_i l_i$
- So $l_i = \log 1/q_i - \log z$
- Now, consider the relative entropy between P and Q and the Gibb's inequality

$$D_{KL}(P \| Q) = \sum_x P(x) \log 1/Q(x) - \sum_x P(x) \log 1/P(x) \qquad D_{KL}(P \| Q) \geq 0$$

$$\sum_i p_i \log 1/q_i \geq \sum_i p_i \log 1/p_i$$

$$L(C,X) = \sum_i p_i l_i = \sum_i p_i \log 1/q_i - \log z$$

$$\geq \sum_i p_i \log 1/p_i - \log z$$

$$\geq H(X)$$

The equality $L(C, X) = H(X)$ is achieved only if the Kraft equality $z = 1$ is satisfied, and if the codelengths satisfy $l_i = \log(1/p_i)$.

What's the most compression that we can hope for?

- **Optimal source codelengths**. The **expected length is minimized** and is equal to $H(X)$ only if the **codelengths are equal to the Shannon information contents**:

$$l_i = \log_2(1/p_i)$$

- **Implicit probabilities defined by codelengths**. Conversely, any choice of codelengths $\{l_i\}$ implicitly defines a probability distribution $\{q_i\}$,

$$q_i \equiv 2^{-l_i} / z$$

for which those codelengths would be the optimal codelengths.

If the code is complete then $z = 1$ and the implicit probabilities are given by $q_i = 2^{-l_i}$

How much can we compress?

How much can we compress?

- So, we can't compress below the entropy. **How close can we expect to get to the entropy?**
- **Theorem - Source coding theorem for symbol codes**.

  For an ensemble X there exists a prefix code C with expected length satisfying

$$H(X) \leq L(C,X) \leq H(X)+1$$

- Proof.
  - Let set the codelengths to integers slightly larger than the optimum

$$l_i = \left\lceil \log_2(1/p_i) \right\rceil$$

  - We check that there is a prefix code with these lengths by confirming that the Kraft inequality is satisfied

How much can we compress?

$$H(X) \le L(C,X) \le H(X)+1$$

- Proof.
  - Let set the codelengths to integers slightly larger than the optimum $l_i = \lceil \log_2(1/p_i) \rceil$
  - We check that there is a prefix code with these lengths by confirming that the Kraft inequality is satisfied

$$\sum_i 2^{-l_i} = \sum_i 2^{-\lceil \log_2(1/p_i) \rceil} \le \sum_i 2^{-\log_2(1/p_i)} = \sum_i p_i = 1$$

$$\sum_{i=1}^{I} 2^{-l_i} \le 1$$

Kraft inequality

$$\sum_i 2^{-\lceil \log_2(1/p_i) \rceil} \le 1$$

Then, there is a prefix code with these l_i

  - Then we can confirm

$$L(C,X) = \sum_i p_i \lceil \log_2(1/p_i) \rceil \le \sum_i p_i (\log_2(1/p_i)+1) \le H(X)+1$$

The cost of using the wrong codelengths

- If we use a **code whose lengths are not equal to the optimal codelengths**, the average message length will be larger than the entropy.
- Let $\{p_i\}$ the true probabilities;
- If we use a complete code with lengths l_i, they define an implicit probabilities $q_i = 2^{-l_i}$

$$L(C,X) = \sum_i p_i l_i = \sum_i p_i \log(1/q_i)$$

$$l_i = \log_2(1/q_i)$$

$$= H(X) + \sum_i p_i \log(1/q_i) - H(X)$$

$$H(X) = \sum_i p_i \log(1/p_i)$$

$$= H(X) + \sum_i p_i \log(1/q_i) - \sum_i p_i \log(1/p_i)$$

$$= H(X) + \sum_i p_i \log(p_i/q_i)$$

$$\boxed{L(C,X) = H(X) + D_{KL}(\boldsymbol{p} \,\|\, \boldsymbol{q})}$$

- $L(C,X)$ exceeds the entropy by the relative entropy $D_{\mathrm{KL}}(\mathbf{p} \,\|\, \mathbf{q})$

Huffman coding

Optimal source coding with symbol codes

- **Goal**: Mminimize the expected length $L(C, X)$.
- **How not to do it?**
  - One might try to roughly split the set A_X in two, and continue bisecting the subsets so as to define a binary tree from the root.
  - This construction has the right spirit, as in the weighing problem, but it is not necessarily optimal.
    - It achieves $L(C, X) \leq H(X) + 2$.
- **The Huffman coding algorithm**
  - It is a simple algorithm for finding an optimal prefix code.
  - The ideas is to construct the code backwards starting from the tails of the codewords
  - The algorithm builds the binary tree from its leaves.

Huffman coding

- **The Huffman coding algorithm**
  - It is a simple algorithm for finding an optimal prefix code.
  - The ideas is to construct the code backwards starting from the tails of the codewords
  - The algorithm builds the binary tree from its leaves

1. Take the two least probable symbols in the alphabet. These two symbols will be given the longest codewords, which will have equal length, and differ only in the last digit.
2. Combine these two symbols into a single symbol, and repeat.

- Since each step reduces the size of the alphabet by one, this algorithm will have assigned strings to all the symbols after $|A_X| - 1$ steps.

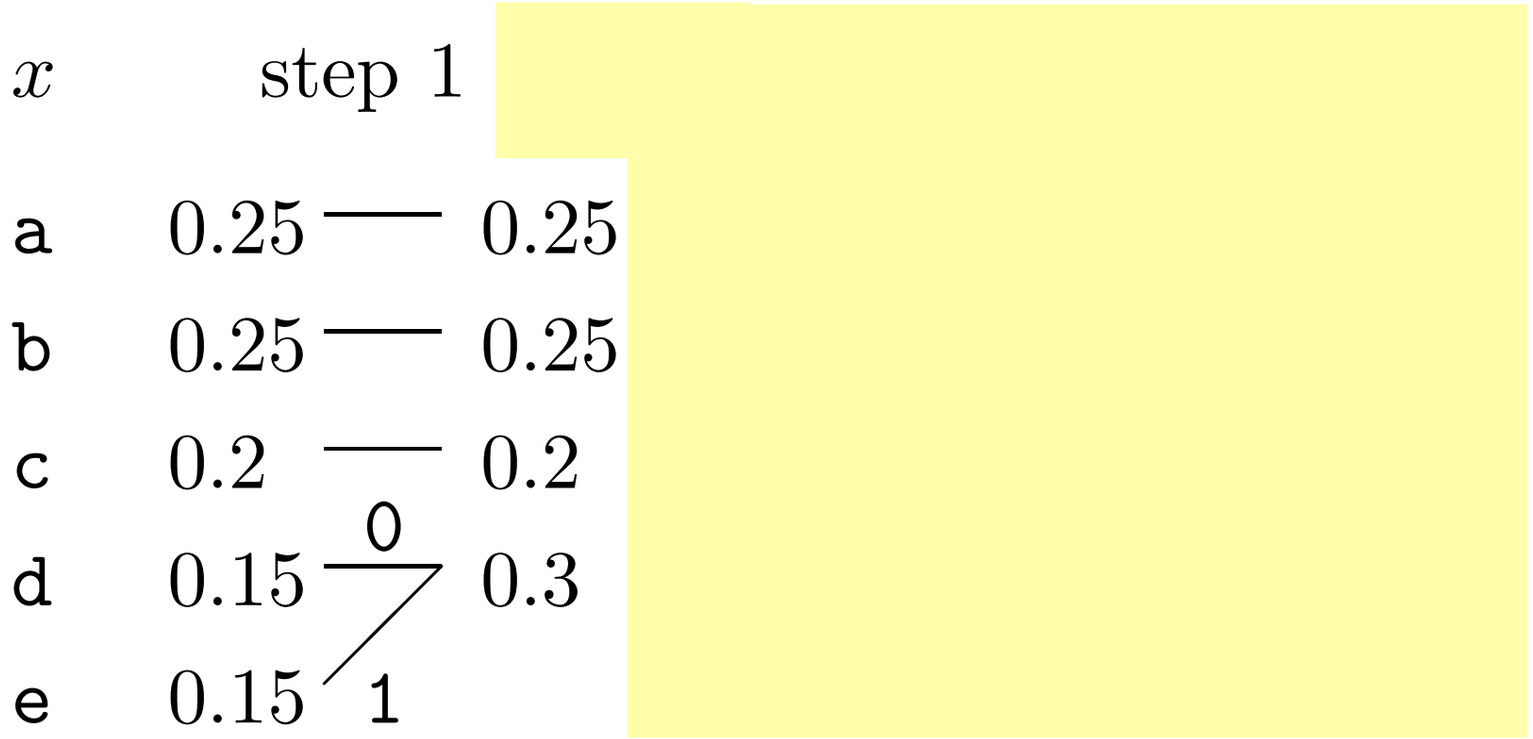
Huffman coding - Example

- $\mathcal{A}_X = \{\, \texttt{a}, \quad \texttt{b}, \quad \texttt{c}, \quad \texttt{d}, \quad \texttt{e} \quad \}$
  $\mathcal{P}_X = \{\, 0.25, 0.25, 0.2, 0.15, 0.15 \,\}.$

x	
a	0.25
b	0.25
c	0.2
d	0.15
e	0.15

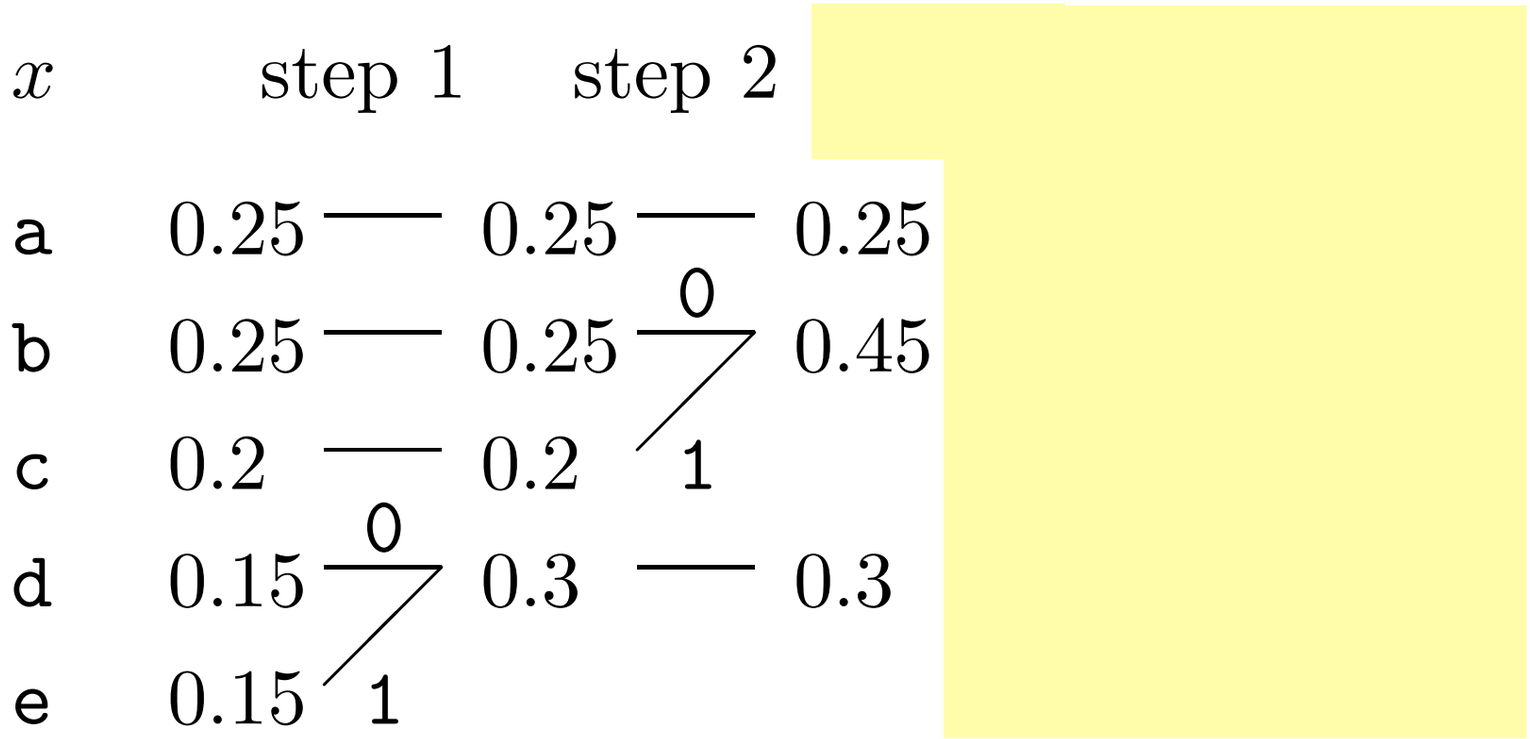
Huffman coding - Example

- $\mathcal{A}_X = \{\,\mathtt{a},\quad \mathtt{b},\quad \mathtt{c},\quad \mathtt{d},\quad \mathtt{e}\quad\}$
  $\mathcal{P}_X = \{\,0.25, 0.25, 0.2, 0.15, 0.15\,\}.$



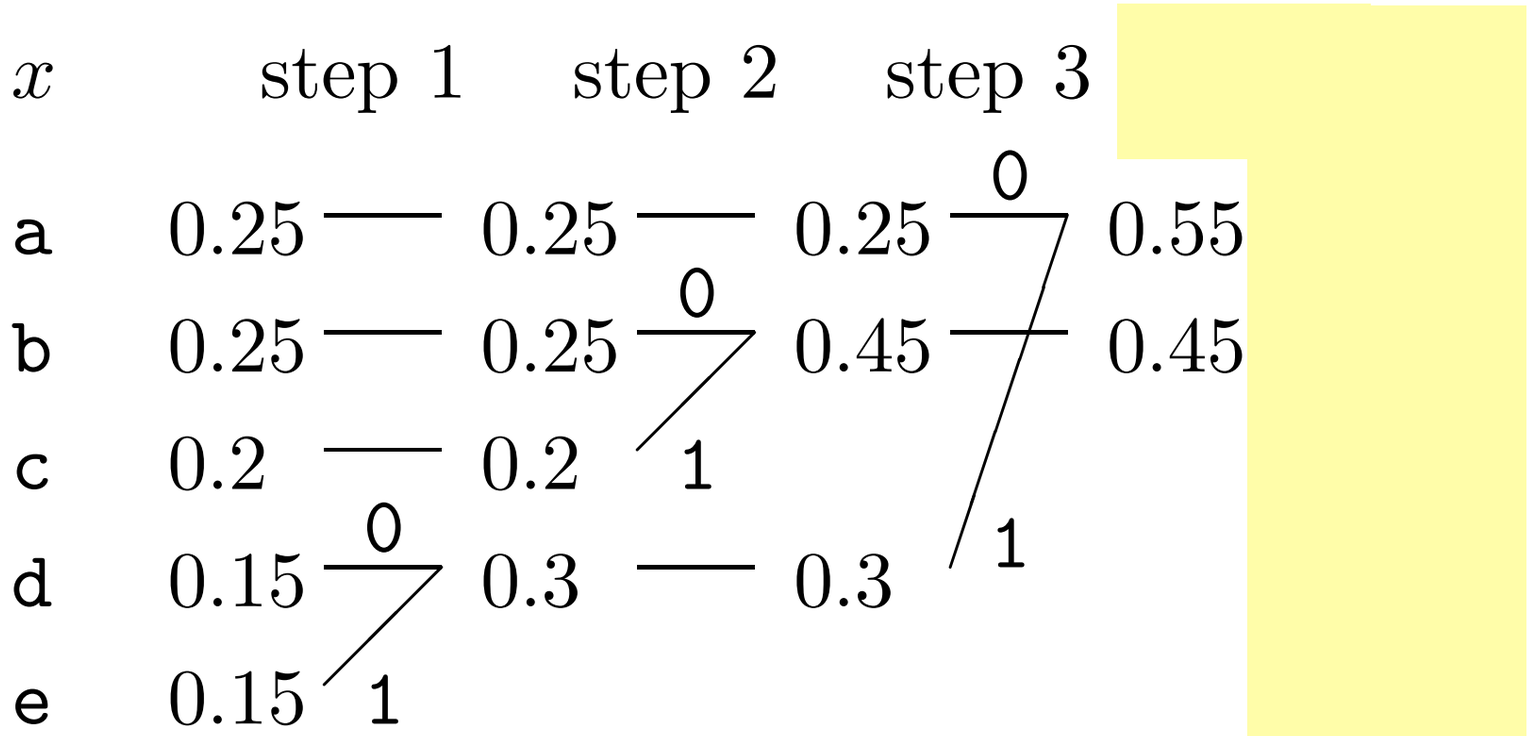
Huffman coding - Example

- $\mathcal{A}_X = \{\,\mathtt{a},\quad \mathtt{b},\quad \mathtt{c},\quad \mathtt{d},\quad \mathtt{e}\quad\}$
  $\mathcal{P}_X = \{\,0.25, 0.25, 0.2, 0.15, 0.15\,\}.$



Huffman coding - Example

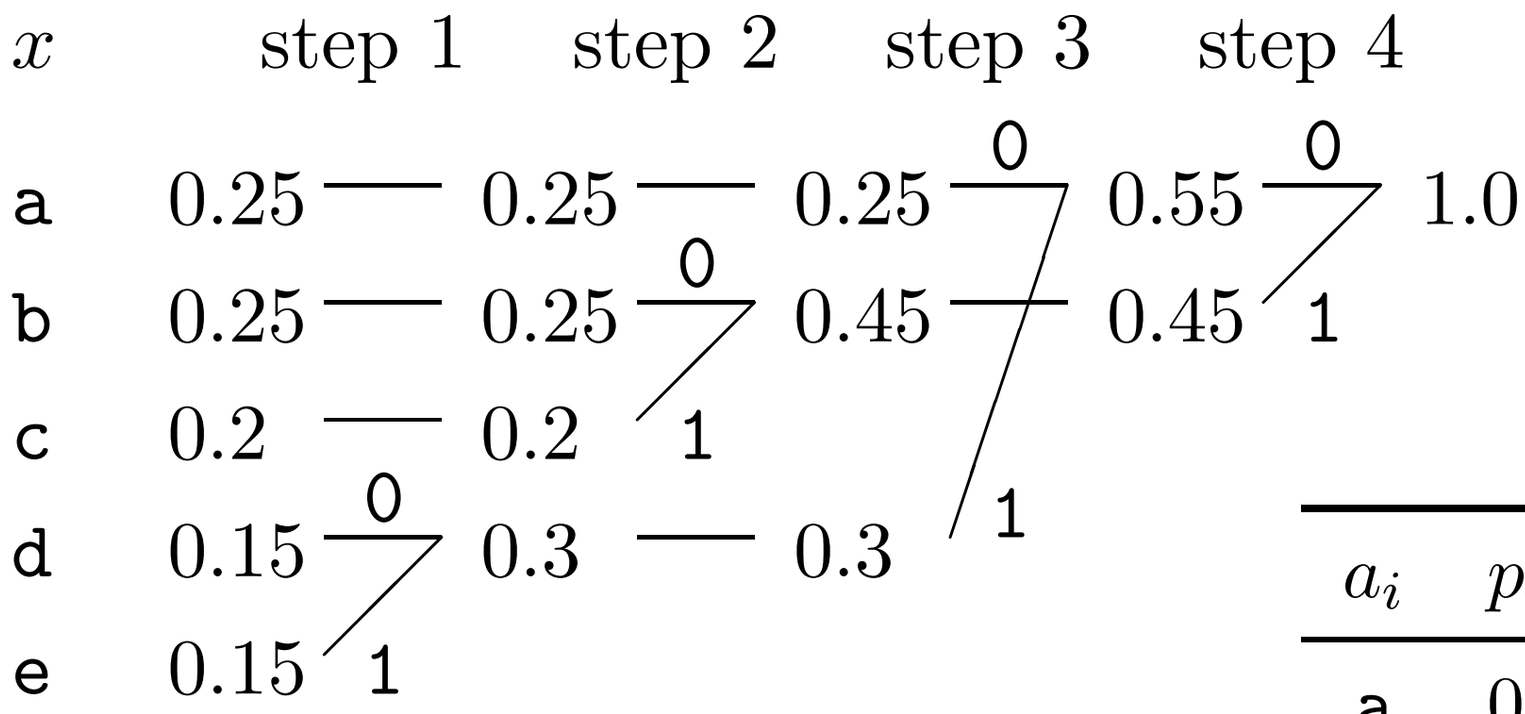
- $\mathcal{A}_X = \{\,\mathtt{a},\ \mathtt{b},\ \mathtt{c},\ \mathtt{d},\ \mathtt{e}\,\}$
  $\mathcal{P}_X = \{\,0.25, 0.25, 0.2, 0.15, 0.15\,\}.$



Huffman coding - Example

- $\mathcal{A}_X = \{\,\texttt{a},\quad \texttt{b},\quad \texttt{c},\quad \texttt{d},\quad \texttt{e}\quad\}$
  $\mathcal{P}_X = \{\,0.25, 0.25, 0.2, 0.15, 0.15\,\}.$

$H(X) = 2.2855$ bits



$L(C, X) = 2.30$ bits

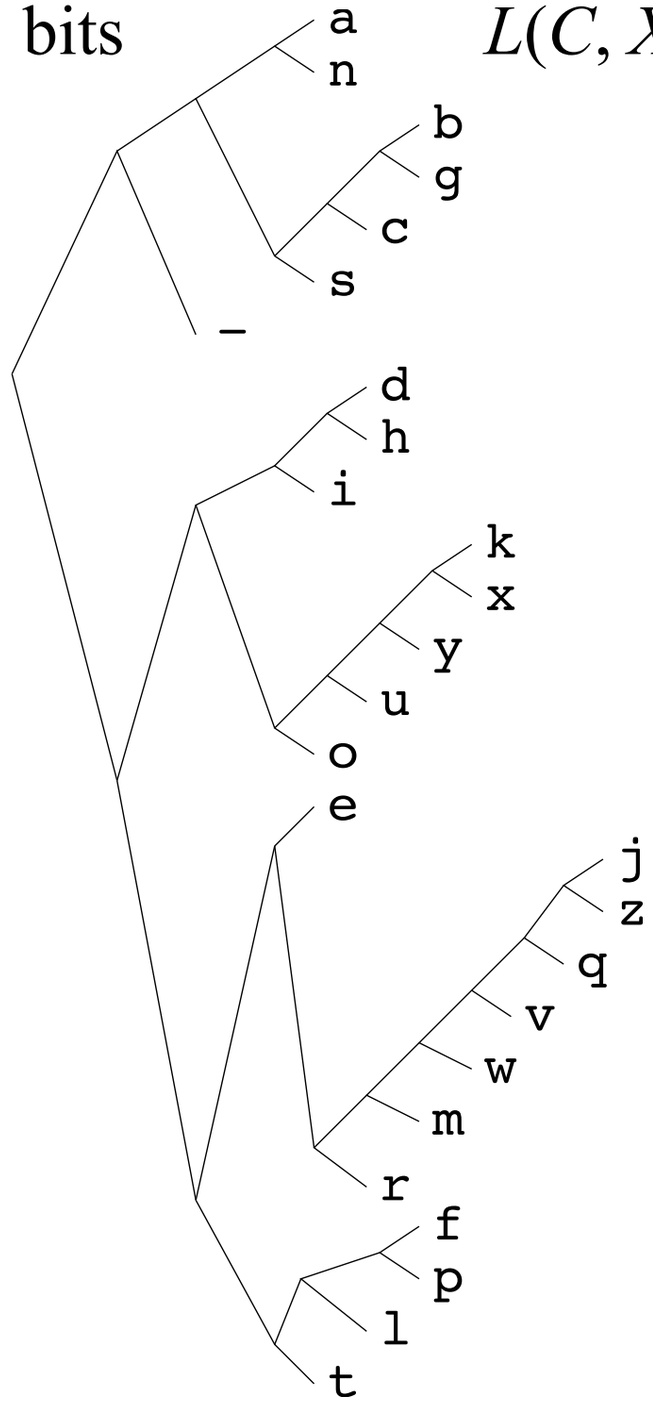
a_i	p_i	$h(p_i)$	l_i	$c(a_i)$
a	0.25	2.0	2	00
b	0.25	2.0	2	10
c	0.2	2.3	2	11
d	0.15	2.7	3	010
e	0.15	2.7	3	011

Huffman code for English language

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
a	0.0575	4.1	4	0000
b	0.0128	6.3	6	001000
c	0.0263	5.2	5	00101
d	0.0285	5.1	5	10000
e	0.0913	3.5	4	1100
f	0.0173	5.9	6	111000
g	0.0133	6.2	6	001001
h	0.0313	5.0	5	10001
i	0.0599	4.1	4	1001
j	0.0006	10.7	10	1101000000
k	0.0084	6.9	7	1010000
l	0.0335	4.9	5	11101
m	0.0235	5.4	6	110101
n	0.0596	4.1	4	0001
o	0.0689	3.9	4	1011
p	0.0192	5.7	6	111001
q	0.0008	10.3	9	110100001
r	0.0508	4.3	5	11011
s	0.0567	4.1	4	0011
t	0.0706	3.8	4	1111
u	0.0334	4.9	5	10101
v	0.0069	7.2	8	11010001
w	0.0119	6.4	7	1101001
x	0.0073	7.1	7	1010001
y	0.0164	5.9	6	101001
z	0.0007	10.4	10	1101000001
–	0.1928	2.4	2	01

$H(X) = 4.11$ bits

$L(C, X) = 4.15$ bits



Disadvantages of the Huffman code

Disadvantages of the Huffman code

- The Huffman algorithm produces an optimal symbol code for an ensemble, but this is not the end of the story. Both the word '**ensemble**' and the phrase '**symbol code**' need careful attention.

- **Changing ensemble**

  - Huffman codes do not handle changing ensemble probabilities with any elegance. One brute-force approach would be to recompute the Huffman code every time the probability over symbols changes.

  - Or, run through the entire file in advance and compute a good probability distribution, which will then remain fixed throughout transmission. The code itself must also be communicated in this scenario

Disadvantages of the Huffman code

$$H(X) \leq L(C,X) \leq H(X)+1$$

- **The extra bit**
  - Huffman code thus incurs an overhead of between 0 and 1 bits per symbol.
  - If $H(X)$ were large, then this overhead would be an unimportant fractional increase. But **for many applications, the entropy may be as low as one bit per symbol**, or even smaller, so the overhead $\boldsymbol{L(C, X) - H(X)}$ **may dominate the encoded file length.**
  - A traditional patch-up of Huffman codes uses them to **compress blocks of symbols**. The overhead per block is at most 1 bit so the overhead per symbol is at most $1/N$ bits.
    - losing the elegant instantaneous decodeability of simple Huffman coding;
    - having to compute the probabilities of all relevant strings and build the associated Huffman tree

Summary

Summary

- **Kraft inequality**. If a code is uniquely decodable its lengths must satisfy

$$\sum_i 2^{-l_i} \leq 1$$

For any lengths **satisfying the Kraft inequality**, there **exists a prefix code** with those lengths.

- **Optimal source code lengths for an ensemble** are equal to the Shannon information contents $l_i = \log_2 {}^1\!/_{p_i}$,

and conversely, any choice of code lengths defines implicit probabilities

$$q_i = \frac{2^{-l_i}}{z}$$

Summary

- **The relative entropy** $D_{KL}(\mathbf{p} \parallel \mathbf{q})$ measures **how many bits per symbol are wasted** by using a code whose implicit probabilities are q, when the ensemble's true probability distribution is p.
- **Source coding theorem for symbol codes.** For an ensemble X, there exists a prefix code whose expected length satisfies

$$H(X) \leq L(C,X) \leq H(X)+1$$

- **The Huffman coding algorithm** generates an optimal symbol code iteratively. At each iteration, the two least probable symbols are combined.

Further Reading and Summary

Q&A

Further Reading

- **Recommend Readings**
  - Information Theory, Inference, and Learning Algorithms from David MacKay, 2015, pages 91 - 102.
- **Supplemental readings:**

What you should know

- What are variable-length symbol codes and what it is the main idea
- What are the implications if a symbol code is lossless?
- What is the meaning of a code being uniquely decodeable?
- What are prefix code?
- What is the relation of prefix codes and Kraft inequality?
- What is the the “cost” of 2-l of each codeword?
- What are the compression limits?
- The Huffman coding algorithm
- What the Disadvantages of the Huffman code?

Further Reading and Summary

Q&A