Three Concepts: Information

Lecture 4: Source Coding: Practice

Teemu Roos

Complex Systems Computation Group
Department of Computer Science, University of Helsinki

Fall 2007

Lecture 4: Source Coding: Practice

Concentric Circular Tower
(David Huffman)

[Photo: Tony Grant. Courtesy of the Huffman family.]

```
"Design with the help
of binary code (0 and
1) the most efficient
method to represent
characters, figures
and symbols."
```

(Assignment at Prof. R.M. Fano's 1952 MIT Information Theory course.)

1. Codes
   - Decodable Codes
   - Prefix Codes
   - Kraft-McMillan Theorem

Extension Code

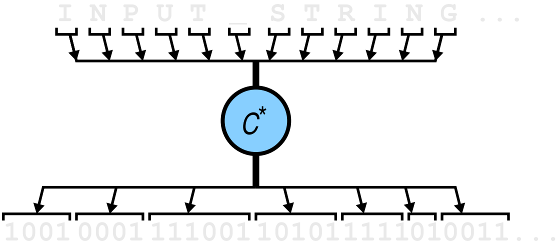
A (binary) **symbol code** $C : \mathcal{X} \to \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

Extension Code

A (binary) **symbol code** $C : \mathcal{X} \rightarrow \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

The **extension** of code C is the mapping $C^* : \mathcal{X}^* \rightarrow \{0,1\}^*$ obtained by concatenating the codewords $C(x_i)$ for each input symbol x_i:

$$C^*(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n) \ .$$

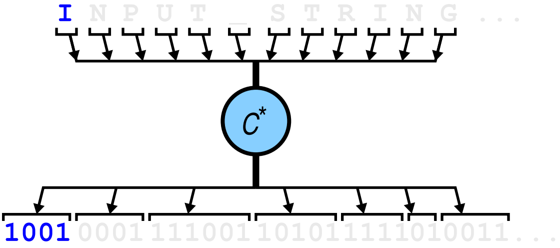


Extension Code

A (binary) **symbol code** $C : \mathcal{X} \rightarrow \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

The **extension** of code C is the mapping $C^* : \mathcal{X}^* \rightarrow \{0,1\}^*$ obtained by concatenating the codewords $C(x_i)$ for each input symbol x_i:

$$C^*(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n) \quad .$$

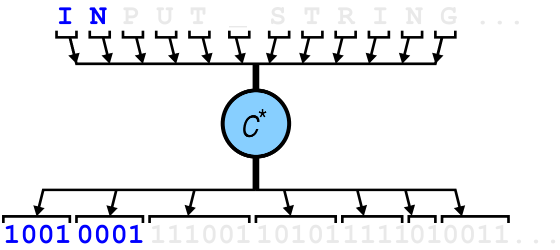


Extension Code

A (binary) **symbol code** $C : \mathcal{X} \to \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

The **extension** of code C is the mapping $C^* : \mathcal{X}^* \to \{0,1\}^*$ obtained by concatenating the codewords $C(x_i)$ for each input symbol x_i:

$$C^*(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n) \ .$$

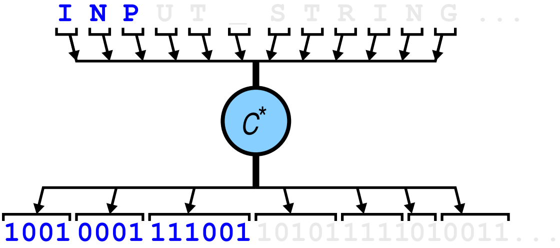


Extension Code

A (binary) **symbol code** $C : \mathcal{X} \to \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

The **extension** of code C is the mapping $C^* : \mathcal{X}^* \to \{0,1\}^*$ obtained by concatenating the codewords $C(x_i)$ for each input symbol x_i:

$$C^*(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n) \ .$$

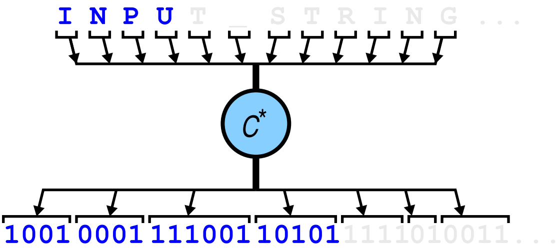


Extension Code

A (binary) **symbol code** $C : \mathcal{X} \rightarrow \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

The **extension** of code C is the mapping $C^* : \mathcal{X}^* \rightarrow \{0,1\}^*$ obtained by concatenating the codewords $C(x_i)$ for each input symbol x_i:

$$C^*(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n) \ .$$

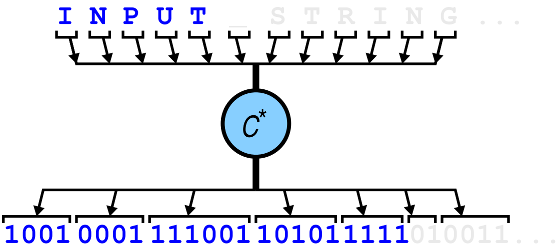


Extension Code

A (binary) **symbol code** $C : \mathcal{X} \to \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

The **extension** of code C is the mapping $C^* : \mathcal{X}^* \to \{0,1\}^*$ obtained by concatenating the codewords $C(x_i)$ for each input symbol x_i:

$$C^*(x_1, x_2, \dots, x_n) = C(x_1)C(x_2)\dots C(x_n) \ .$$

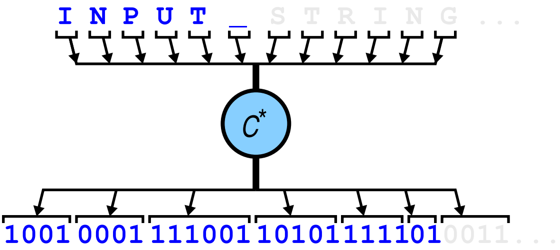


Extension Code

A (binary) **symbol code** $C : \mathcal{X} \to \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

The **extension** of code C is the mapping $C^* : \mathcal{X}^* \to \{0,1\}^*$ obtained by concatenating the codewords $C(x_i)$ for each input symbol x_i:

$$C^*(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n) \quad .$$

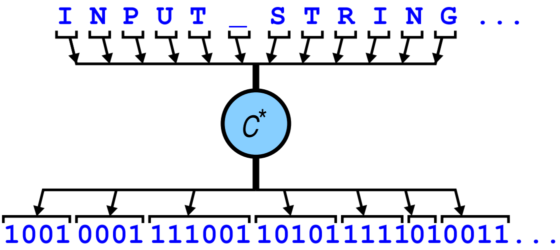


Extension Code

A (binary) **symbol code** $C : \mathcal{X} \to \{0,1\}^*$ is a mapping from the alphabet $\mathcal{X}$ to the set of finite binary sequences.

The **extension** of code C is the mapping $C^* : \mathcal{X}^* \to \{0,1\}^*$ obtained by concatenating the codewords $C(x_i)$ for each input symbol x_i:

$$C^*(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n) \ .$$



Decodable Codes

Decodable Code

Code C is (uniquely) **decodable** iff its extension C^* is a one-to-one mapping, i.e., iff

$$(x_1, \ldots, x_n) \neq (y_1, \ldots, y_n) \Rightarrow C^*(x_1, \ldots, x_n) \neq C^*(y_1, \ldots, y_n) \ .$$

Decodable Codes

Decodable Code

Code C is (uniquely) **decodable** iff its extension C^* is a one-to-one mapping, i.e., iff

$$(x_1, \ldots, x_n) \neq (y_1, \ldots, y_n) \;\Rightarrow\; C^*(x_1, \ldots, x_n) \neq C^*(y_1, \ldots, y_n) \;.$$

X A code with codewords $\{0, 1, 10, 11\}$ is *not* uniquely decodable: What does 10 mean?

Decodable Codes

Decodable Code

Code C is (uniquely) **decodable** iff its extension C^* is a one-to-one mapping, i.e., iff

$$(x_1, \ldots, x_n) \neq (y_1, \ldots, y_n) \Rightarrow C^*(x_1, \ldots, x_n) \neq C^*(y_1, \ldots, y_n) .$$

- X A code with codewords $\{0, 1, 10, 11\}$ is *not* uniquely decodable: What does 10 mean?
- √ A code with codewords $\{00, 01, 10, 11\}$ *is* uniquely decodable: Each pair of bits can be decoded individually.

Decodable Codes

Decodable Code

Code C is (uniquely) **decodable** iff its extension C^* is a one-to-one mapping, i.e., iff

$$(x_1, \ldots, x_n) \neq (y_1, \ldots, y_n) \Rightarrow C^*(x_1, \ldots, x_n) \neq C^*(y_1, \ldots, y_n) \ .$$

- ✗ A code with codewords $\{0, 1, 10, 11\}$ is *not* uniquely decodable: What does 10 mean?
- √ A code with codewords $\{00, 01, 10, 11\}$ *is* uniquely decodable: Each pair of bits can be decoded individually.
- √ A code with codewords $\{0, 01, 011, 0111\}$ is also uniquely decodable: What does 0011 mean?

Prefix Codes

An important subset of decodable codes is the set of **prefix(-free) codes**.

Prefix Code

A code $C : \mathcal{X} \to \{0,1\}^*$ is called a **prefix code** iff no codeword is a prefix of another.

It is easily seen that all prefix codes are uniquely decodable: each symbol can be decoded as soon as its codeword is read. Therefore, prefix codes are also called *instantaneous* codes.

Prefix Codes

An important subset of decodable codes is the set of **prefix(-free) codes**.

Prefix Code

A code $C : \mathcal{X} \to \{0,1\}^*$ is called a **prefix code** iff no codeword is a prefix of another.

It is easily seen that all prefix codes are uniquely decodable: each symbol can be decoded as soon as its codeword is read. Therefore, prefix codes are also called *instantaneous* codes.

X A code with codewords $\{0, 01, 011, 0111\}$ is uniquely decodable *but not prefix-free*: e.g., 0 is a prefix of 01.

Prefix Codes

An important subset of decodable codes is the set of **prefix(-free) codes**.

Prefix Code

A code $C : \mathcal{X} \rightarrow \{0,1\}^*$ is called a **prefix code** iff no codeword is a prefix of another.

It is easily seen that all prefix codes are uniquely decodable: each symbol can be decoded as soon as its codeword is read. Therefore, prefix codes are also called *instantaneous* codes.

- ✗ A code with codewords $\{0, 01, 011, 0111\}$ is uniquely decodable *but not prefix-free*: e.g., 0 is a prefix of 01.
- ✓ A code with codewords $\{0, 10, 110, 111\}$ *is* prefix-free.

Kraft Inequality

The codeword lengths of a prefix codes satisfy the following important property.

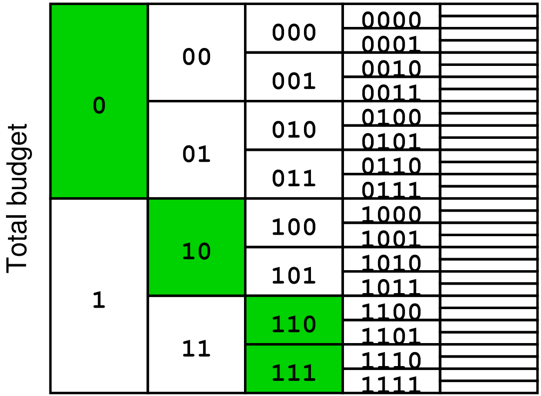
Kraft Inequality

The codeword lengths $\ell_1, \ldots, \ell_m$ of any (binary) prefix code satisfy

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1 \ .$$

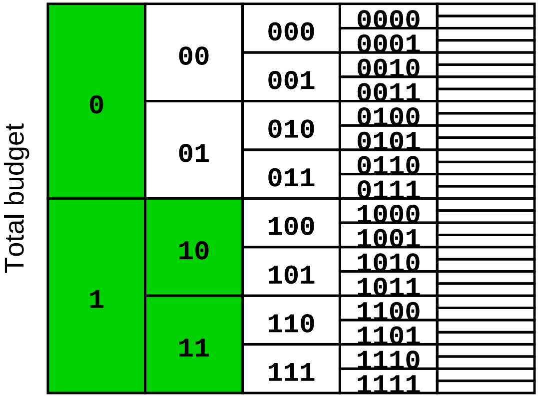
Conversely, given a set of codeword lengths that satisfy this inequality, there is a prefix code with these codeword lengths.

Kraft Inequality



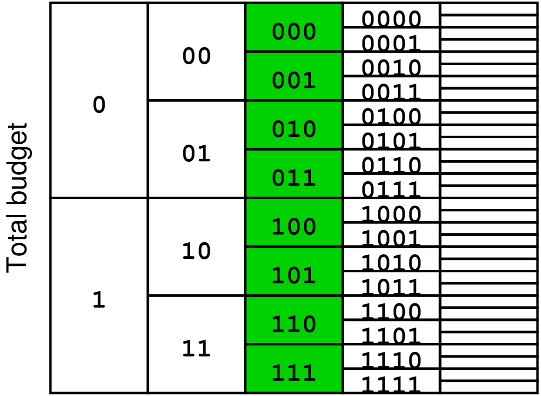
√ Codewords $\{0, 10, 110, 111\}$

Kraft Inequality



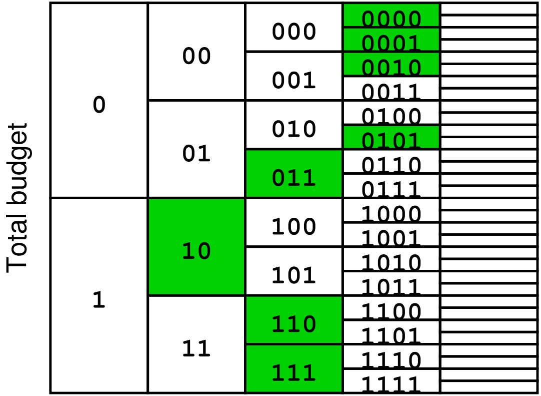
✗ Kraft inequality violated. $\Rightarrow$ Not decodable.

Kraft Inequality



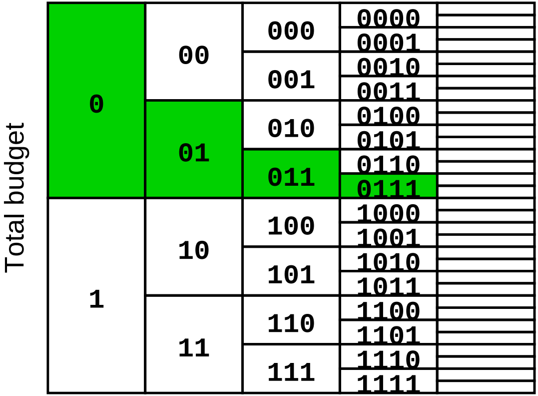
√ Fixed-length code

Kraft Inequality



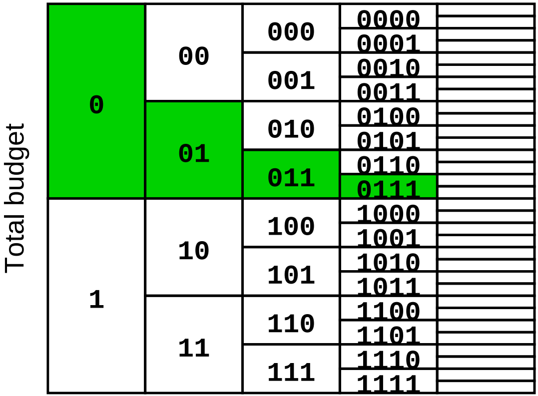
√ Decodable & prefix-free

Kraft Inequality



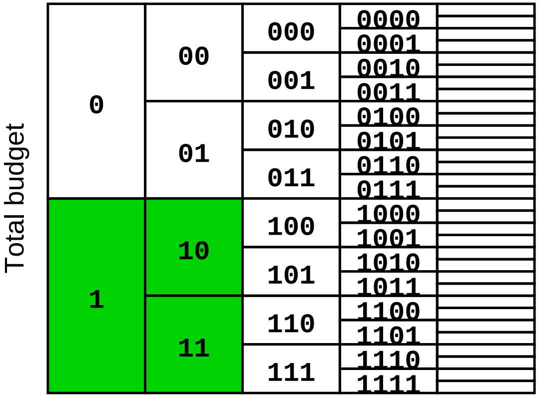
Kraft? Decodable? Prefix-free?

Kraft Inequality



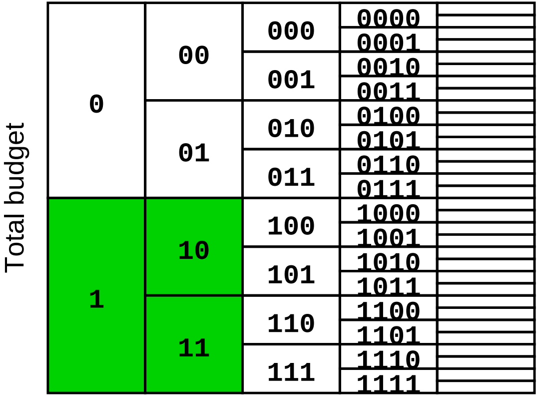
Kraft? √ Decodable? √ Prefix-free? X

Kraft Inequality



Kraft? Decodable? Prefix-free?

Kraft Inequality



Kraft? √ Decodable? X Prefix-free? X

Kraft Inequality

Question: What if the inequality is satisfied strictly, i.e., the sum of the terms in the sum equals *less* than one:

$$\sum_{i=1}^{m} 2^{-\ell_i} < 1 \ .$$

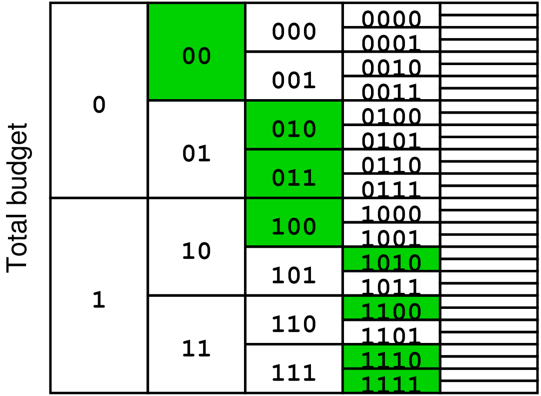
Kraft Inequality

Question: What if the inequality is satisfied strictly, i.e., the sum of the terms in the sum equals *less* than one:

$$\sum_{i=1}^{m} 2^{-\ell_i} < 1 \ .$$

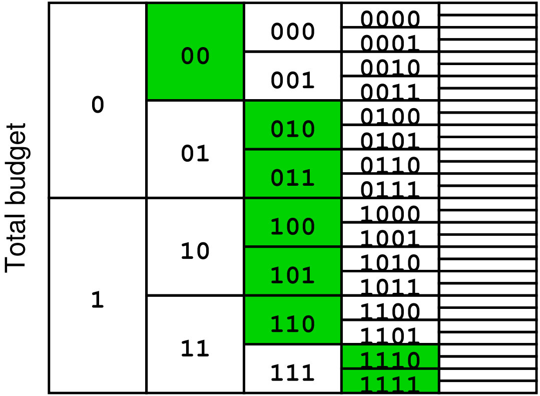
Then it is possible to make the codewords shorter and still have a decodable (prefix) code.

Kraft Inequality



Not all of budget used. $\Rightarrow$ Some codewords can be made shorter.

Kraft Inequality



"Kraft tight" / complete code.

Kraft–McMillan Theorem

The Kraft inequality restricts the codeword lengths of prefix codes. Could we do much better if we would only require decodability?

Kraft–McMillan Theorem

The Kraft inequality restricts the codeword lengths of prefix codes. Could we do much better if we would only require decodability?

In fact it can be shown that we do not lose anything at all!

Kraft–McMillan Theorem

The Kraft inequality restricts the codeword lengths of prefix codes. Could we do much better if we would only require decodability?

In fact it can be shown that we do not lose anything at all!

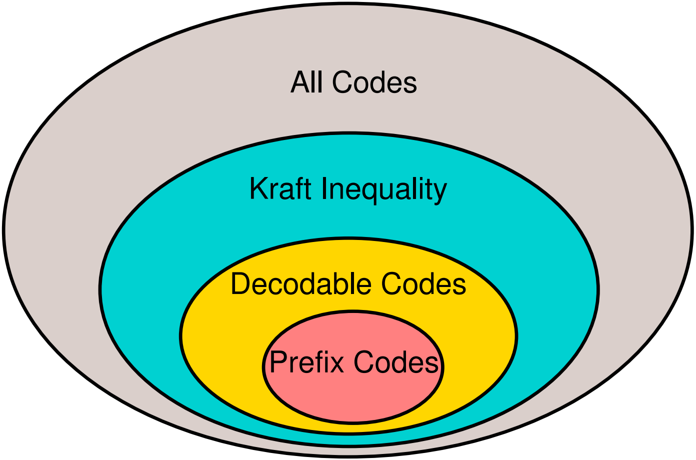
Kraft-McMillan Theorem

The codeword lengths $\ell_1, \ldots, \ell_m$ of any **uniquely decodable** (binary) code satisfy

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1 \ .$$

Conversely, given a set of codeword lengths that satisfy this inequality, there is a uniquely decodable (prefix) code with these codeword lengths.

Kraft-McMillan Theorem & Codes



1. Codes
   - Decodable Codes
   - Prefix Codes
   - Kraft-McMillan Theorem

2. Optimal Codes
   - Entropy Lower Bound
   - Shannon-Fano
   - Huffman

3. Below Entropy
   - Problems with Symbol Codes
   - Two-Part Codes
   - Block Codes

Codelengths and Probabilities

Let $\ell_1, \ldots, \ell_m$ be the codeword lengths of a uniquely decodable code $C : \mathcal{X} \to \{0,1\}^*$. By the Kraft-McMillan theorem we have

$$c = \sum_{i=1}^{m} 2^{-\ell_i} \leq 1 \ .$$

Codelengths and Probabilities

Let $\ell_1, \ldots, \ell_m$ be the codeword lengths of a uniquely decodable code $C : \mathcal{X} \rightarrow \{0,1\}^*$. By the Kraft-McMillan theorem we have

$$c = \sum_{i=1}^{m} 2^{-\ell_i} \leq 1 \ .$$

Define a probability mass function $p : \mathcal{X} \rightarrow [0,1]$ as follows:

$$p_i = \frac{2^{-\ell_i}}{c}$$

where c is given above.

Codelengths and Probabilities

Let $\ell_1, \ldots, \ell_m$ be the codeword lengths of a uniquely decodable code $C : \mathcal{X} \to \{0,1\}^*$. By the Kraft-McMillan theorem we have

$$c = \sum_{i=1}^{m} 2^{-\ell_i} \leq 1 \ .$$

Define a probability mass function $p : \mathcal{X} \to [0,1]$ as follows:

$$p_i = \frac{2^{-\ell_i}}{c} \quad \Leftrightarrow \quad \ell_i = \log_2 \frac{c}{p_i} \ ,$$

where c is given above.

Codelengths and Probabilities

Let $\ell_1, \ldots, \ell_m$ be the codeword lengths of a uniquely decodable code $C : \mathcal{X} \to \{0,1\}^*$. By the Kraft-McMillan theorem we have

$$c = \sum_{i=1}^{m} 2^{-\ell_i} \leq 1 \ .$$

Define a probability mass function $p : \mathcal{X} \to [0,1]$ as follows:

$$p_i = \frac{2^{-\ell_i}}{c} \quad \Leftrightarrow \quad \ell_i = \log_2 \frac{c}{p_i} \ ,$$

where c is given above.

Function p is indeed a pmf:

1. Non-negative: $p(x) \geq 0$ for all $x \in \mathcal{X}$.

Codelengths and Probabilities

Let $\ell_1, \ldots, \ell_m$ be the codeword lengths of a uniquely decodable code $C : \mathcal{X} \to \{0,1\}^*$. By the Kraft-McMillan theorem we have

$$c = \sum_{i=1}^{m} 2^{-\ell_i} \leq 1 \ .$$

Define a probability mass function $p : \mathcal{X} \to [0,1]$ as follows:

$$p_i = \frac{2^{-\ell_i}}{c} \quad \Leftrightarrow \quad \ell_i = \log_2 \frac{c}{p_i} \ ,$$

where c is given above.

Function p is indeed a pmf:

1. Non-negative: $p(x) \geq 0$ for all $x \in \mathcal{X}$.
2. Sums to one: $\sum_{x \in \mathcal{X}} p(x) = \sum_{i=1}^{m} \frac{1}{c} 2^{-\ell_i} = \frac{c}{c} = 1$.

Codelengths and Probabilities

Assuming that the code is "Kraft tight", $c = 1$, then under the pmf p corresponding to the codeword lengths $\ell_1, \ldots, \ell_m$, the expected codeword length is

$$E[\ell(X)] = \sum_{i=1}^{m} 2^{-\ell_i} \, \ell_i$$

Codelengths and Probabilities

Assuming that the code is "Kraft tight", $c = 1$, then under the pmf p corresponding to the codeword lengths $\ell_1, \ldots, \ell_m$, the expected codeword length is

$$\begin{aligned} E[\ell(X)] &= \sum_{i=1}^{m} 2^{-\ell_i} \, \ell_i \\ &= \sum_{i=1}^{m} p_i \, \log_2 \frac{1}{p_i} \end{aligned}$$

Codelengths and Probabilities

Assuming that the code is "Kraft tight", $c = 1$, then under the pmf p corresponding to the codeword lengths $\ell_1, \ldots, \ell_m$, the expected codeword length is

$$E[\ell(X)] = \sum_{i=1}^{m} 2^{-\ell_i} \, \ell_i$$
$$= \sum_{i=1}^{m} p_i \, \log_2 \frac{1}{p_i} = H(X) \ .$$

Codelengths and Probabilities

Assuming that the code is "Kraft tight", $c = 1$, then under the pmf p corresponding to the codeword lengths $\ell_1, \ldots, \ell_m$, the expected codeword length is

$$E[\ell(X)] = \sum_{i=1}^{m} 2^{-\ell_i} \, \ell_i$$

$$= \sum_{i=1}^{m} p_i \, \log_2 \frac{1}{p_i} = H(X) \ .$$

This is the best we can hope for:

The expected codelength of any uniquely decodable code is at least the entropy:

$$E[\ell(X)] \geq H(X) \ .$$

Entropy Lower Bound

$$E[\ell(X)] \geq H(X) \quad .$$

Entropy Lower Bound

$$E[\ell(X)] \geq H(X) \quad .$$

Proof.

$$E[\ell(X)] - H(X) = \sum_{x \in \mathcal{X}} p(x)\, \ell(x) - \sum_{x \in \mathcal{X}} p(x) \, \log_2 \frac{1}{p(x)}$$

Entropy Lower Bound

$$E[\ell(X)] \geq H(X) \ .$$

Proof.

$$E[\ell(X)] - H(X) = \sum_{x \in \mathcal{X}} p(x)\, \ell(x) - \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{p(x)}$$

$$= \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{2^{-\ell_x}} - \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{p(x)}$$

Entropy Lower Bound

$$E[\ell(X)] \geq H(X) \ .$$

Proof.

$$E[\ell(X)] - H(X) = \sum_{x \in \mathcal{X}} p(x)\, \ell(x) - \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{p(x)}$$

$$= \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{2^{-\ell_x}} - \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{p(x)}$$

$$= \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{p(x)}{2^{-\ell_x}}$$

Entropy Lower Bound

$$E[\ell(X)] \geq H(X) \ .$$

Proof.

$$\begin{aligned}
E[\ell(X)] - H(X) &= \sum_{x \in \mathcal{X}} p(x)\, \ell(x) - \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{p(x)} \\
&= \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{2^{-\ell_x}} - \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{p(x)} \\
&= \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{p(x)}{2^{-\ell_x}} \\
&= \sum_{x \in \mathcal{X}} p(x) \left[\log_2 \frac{p(x)}{q(x)} + \log_2 \frac{1}{c} \right]
\end{aligned}$$

$$\boxed{q(x) = \frac{2^{-\ell(x)}}{c}}$$

Entropy Lower Bound

$$E[\ell(X)] \geq H(X) \quad .$$

Proof.

$$E[\ell(X)] - H(X) = \sum_{x \in \mathcal{X}} p(x)\,\ell(x) - \sum_{x \in \mathcal{X}} p(x)\,\log_2 \frac{1}{p(x)}$$

$$= \sum_{x \in \mathcal{X}} p(x)\,\log_2 \frac{1}{2^{-\ell_x}} - \sum_{x \in \mathcal{X}} p(x)\,\log_2 \frac{1}{p(x)}$$

$$= \sum_{x \in \mathcal{X}} p(x)\,\log_2 \frac{p(x)}{2^{-\ell_x}}$$

$$= \sum_{x \in \mathcal{X}} p(x)\left[\log_2 \frac{p(x)}{q(x)} + \log_2 \frac{1}{c}\right] \qquad \boxed{q(x) = \frac{2^{-\ell(x)}}{c}}$$

$$= D(p \parallel q) + \log_2 \frac{1}{c} \geq 0 \quad .$$

Entropy Lower Bound

So what have we learned?

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right]$$

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right] = \sum_{i=1}^{n} E[\ell(X_i)]$$

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right] = \sum_{i=1}^{n} E[\ell(X_i)] = nH(X) \ .$$

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right] = \sum_{i=1}^{n} E[\ell(X_i)] = nH(X) \quad .$$

! By Shannon's Noiseless Channel Coding Theorem, this is optimal among all codes, **not only symbol codes.**

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right] = \sum_{i=1}^{n} E[\ell(X_i)] = nH(X) \ .$$

! By Shannon's Noiseless Channel Coding Theorem, this is optimal among all codes, **not only symbol codes.** Fine print: only if X_i i.i.d.!

Codelengths and Probabilities

The only problem with the $\ell(x) = \log_2 \frac{1}{p(x)}$ codeword choice is the requirement that codeword lengths must be **integers** (try to think about a codeword with length 0.123, for instance), while the so obtained ℓ is not in general an integer.

Codelengths and Probabilities

The only problem with the $\ell(x) = \log_2 \frac{1}{p(x)}$ codeword choice is the requirement that codeword lengths must be **integers** (try to think about a codeword with length 0.123, for instance), while the so obtained ℓ is not in general an integer.

The simplest solution is to round upwards:

Shannon-Fano Code

Given a pmf, the **Shannon-Fano code** has the codeword lengths

$$\ell(x) = \left\lceil \log_2 \frac{1}{p(x)} \right\rceil \quad \text{for all } x \in \mathcal{X}.$$

Alice in Wonderland

Shannon-Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
a	0.0644	3.9	4
b	0.0108	6.5	7
c	0.0178	5.8	6
d	0.0359	4.7	5
e	0.0991	3.3	4
f	0.0147	6.0	7
g	0.0184	5.7	6
h	0.0535	4.2	5
i	0.0551	4.1	5
j	0.0011	9.8	10
k	0.0083	6.8	7
l	0.0343	4.8	5
	⋮		
y	0.0165	5.9	6
z	0.0005	10.7	11
	0.2111	2.2	3

$$H(X) = 4.03$$

Shannon-Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
a	0.0644	3.9	4
b	0.0108	6.5	7
c	0.0178	5.8	6
d	0.0359	4.7	5
e	0.0991	3.3	4
f	0.0147	6.0	7
g	0.0184	5.7	6
h	0.0535	4.2	5
i	0.0551	4.1	5
j	0.0011	9.8	10
k	0.0083	6.8	7
l	0.0343	4.8	5
	$\vdots$		
y	0.0165	5.9	6
z	0.0005	10.7	11
	0.2111	2.2	3

$$H(X) = 4.03$$

Shannon-Fano:

Shannon-Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
a	0.0644	3.9	4
b	0.0108	6.5	7
c	0.0178	5.8	6
d	0.0359	4.7	5
e	0.0991	3.3	4
f	0.0147	6.0	7
g	0.0184	5.7	6
h	0.0535	4.2	5
i	0.0551	4.1	5
j	0.0011	9.8	10
k	0.0083	6.8	7
l	0.0343	4.8	5
	⋮		
y	0.0165	5.9	6
z	0.0005	10.7	11
	0.2111	2.2	3

$H(X) = 4.03$

Shannon-Fano:

1. Sort by probability.

Shannon-Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
	0.2111	2.2	3
e	0.0991	3.3	4
t	0.0781	3.6	4
a	0.0644	3.9	4
o	0.0598	4.0	5
i	0.0551	4.1	5
h	0.0535	4.2	5
n	0.0516	4.2	5
s	0.0475	4.3	5
r	0.0401	4.6	5
d	0.0359	4.7	5
l	0.0343	4.8	5
	⋮		
x	0.0011	9.8	10
j	0.0011	9.8	10
z	0.0005	10.7	11

$H(X) = 4.03$

Shannon-Fano:

1. Sort by probability.

Shannon-Fano: Example

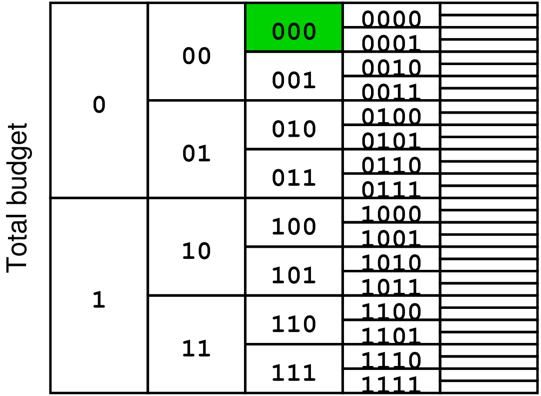
X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
	0.2111	2.2	3
e	0.0991	3.3	4
t	0.0781	3.6	4
a	0.0644	3.9	4
o	0.0598	4.0	5
i	0.0551	4.1	5
h	0.0535	4.2	5
n	0.0516	4.2	5
s	0.0475	4.3	5
r	0.0401	4.6	5
d	0.0359	4.7	5
l	0.0343	4.8	5
	⋮		
x	0.0011	9.8	10
j	0.0011	9.8	10
z	0.0005	10.7	11

$H(X) = 4.03$

Shannon-Fano:

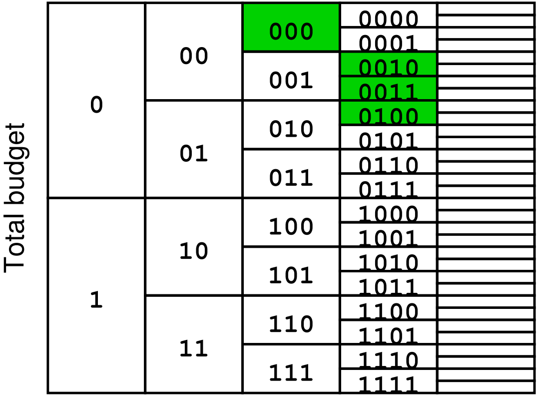
1. Sort by probability.
2. Choose codewords in order, avoiding prefixes. ("Kraft table"!)

Shannon-Fano: Example



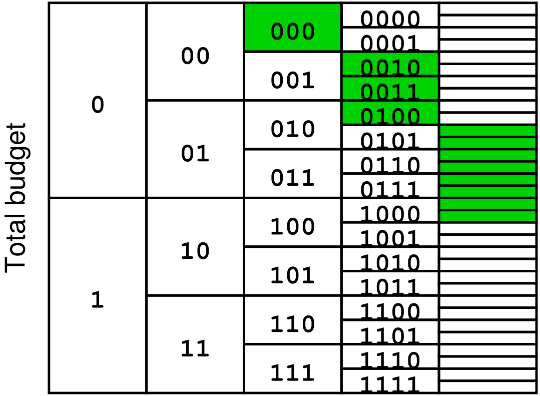
Codeword lengths $(3, 4, 4, 4, 5, 5, 5, 5, \ldots, 10, 10, 11)$

Shannon-Fano: Example



Codeword lengths $(3, 4, 4, 4, 5, 5, 5, 5, \ldots, 10, 10, 11)$

Shannon-Fano: Example



Codeword lengths $(3, 4, 4, 4, 5, 5, 5, 5, \ldots, 10, 10, 11)$

Shannon-Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	3	000
e	0.0991	3.3	4	0010
t	0.0781	3.6	4	0011
a	0.0644	3.9	4	0100
o	0.0598	4.0	5	01010
i	0.0551	4.1	5	01011
h	0.0535	4.2	5	01100
n	0.0516	4.2	5	01101
s	0.0475	4.3	5	01110
r	0.0401	4.6	5	01111
d	0.0359	4.7	5	10000
l	0.0343	4.8	5	10001
	⋮			
x	0.0011	9.8	10	1010111101
j	0.0011	9.8	10	1010111110
z	0.0005	10.7	11	10101111110

Shannon-Fano: Example

	X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
		0.2111	2.2	3	000
	e	0.0991	3.3	4	0010
	t	0.0781	3.6	4	0011
	a	0.0644	3.9	4	0100
	o	0.0598	4.0	5	01010
	i	0.0551	4.1	5	01011
	h	0.0535	4.2	5	01100
	n	0.0516	4.2	5	01101
	s	0.0475	4.3	5	01110
	r	0.0401	4.6	5	01111
	d	0.0359	4.7	5	10000
	l	0.0343	4.8	5	10001
		$\vdots$			
	x	0.0011	9.8	10	1010111101
	j	0.0011	9.8	10	1010111110
	z	0.0005	10.7	11	10101111110

$$H(X) = 4.03$$
$$E[\ell(X)] = 4.60$$
$$E[\ell(X)] - H(X) = 0.57$$

Shannon-Fano Code

The expected codeword length of the Shannon-Fano code is

$$E\left[\ell(X)\right] = E\left[\left\lceil \log_2 \frac{1}{p(X)} \right\rceil\right] \leq E\left[\log_2 \frac{1}{p(X)} + 1\right] = H(X) + 1 \ .$$

Shannon-Fano Code

The expected codeword length of the Shannon-Fano code is

$$E\left[\ell(X)\right] = E\left[\left\lceil \log_2 \frac{1}{p(X)} \right\rceil\right] \leq E\left[\log_2 \frac{1}{p(X)} + 1\right] = H(X) + 1 \ .$$

In the Alice example we had

$$E[\ell(X)] - H(X) = 4.60 - 4.03 = 0.57 \leq 1 \ .$$

Shannon-Fano Code

Consider the Shannon-Fano code for Alice in Wonderland. The longest codewords are as follows:

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
b	0.0108	6.5	7	1010101
k	0.0083	6.8	7	1010110
v	0.0061	7.3	8	10101110
q	0.0015	9.3	10	1010111100
x	0.0011	9.8	10	1010111101
j	0.0011	9.8	10	1010111110
z	0.0005	10.7	11	10101111110

Shannon-Fano Code

Consider the Shannon-Fano code for Alice in Wonderland. The longest codewords are as follows:

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
b	0.0108	6.5	7	1010101
k	0.0083	6.8	7	1010110
v	0.0061	7.3	8	10101110
q	0.0015	9.3	10	1010111100
x	0.0011	9.8	10	1010111101
j	0.0011	9.8	10	1010111110
z	0.0005	10.7	11	10101111110

Can you find a way to improve the code without violating the prefix-free property?

Shannon-Fano Code

Consider the Shannon-Fano code for Alice in Wonderland. The longest codewords are as follows:

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
b	0.0108	6.5	7	1010101
k	0.0083	6.8	7	1010110
v	0.0061	7.3	8	10101110
q	0.0015	9.3	10	1010111100
x	0.0011	9.8	10	1010111101
j	0.0011	9.8	10	1010111110
z	0.0005	10.7	11	10101111110

Can you find a way to improve the code without violating the prefix-free property? *Hint:* zzz...

Huffman Code

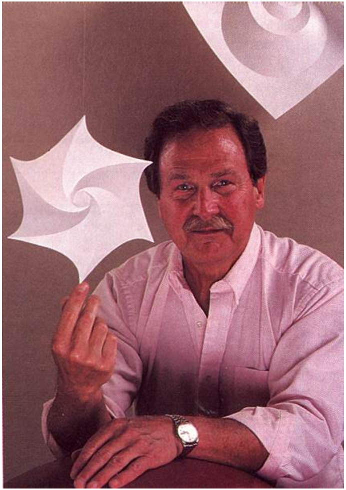
So the Shannon-Fano code is not the optimal symbol code. This is where Professor Fano and a student called David Huffman enter:

Huffman Code

So the Shannon-Fano code is not the optimal symbol code. This is where Professor Fano and a student called David Huffman enter:

`"Design with the help of binary code (0 and 1) the most efficient method to represent characters, figures and symbols."`

David Huffman (1925–1999)



Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.
2. Join the two least probable symbols, i and j, and remove them from the list. Add a new *pseudosymbol* whose probability is $p_i + p_j$.

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.
2. Join the two least probable symbols, i and j, and remove them from the list. Add a new *pseudosymbol* whose probability is $p_i + p_j$.
3. If there is more than one symbol left, go to Step 1.

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.
2. Join the two least probable symbols, i and j, and remove them from the list. Add a new *pseudosymbol* whose probability is $p_i + p_j$.
3. If there is more than one symbol left, go to Step 1.
4. Use the resulting binary tree to define the codewords.

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.
2. Join the two least probable symbols, i and j, and remove them from the list. Add a new *pseudosymbol* whose probability is $p_i + p_j$.
3. If there is more than one symbol left, go to Step 1.
4. Use the resulting binary tree to define the codewords.

See the demo at
`www.cs.auckland.ac.nz/software/AlgAnim/huffman.html`

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.
2. The longest two codewords have the same length.

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.
2. The longest two codewords have the same length.
3. The longest two codewords differ only at the last bit and correspond to the two least probable symbols.

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.
2. The longest two codewords have the same length.
3. The longest two codewords differ only at the last bit and correspond to the two least probable symbols.

Points 2 & 3 suggest the first step of Huffman's algorithm. Any subtree must satisfy the same conditions $\Rightarrow$ Induction.

Huffman Code: Optimality

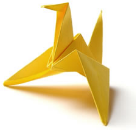
The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.
2. The longest two codewords have the same length.
3. The longest two codewords differ only at the last bit and correspond to the two least probable symbols.

Points 2 & 3 suggest the first step of Huffman's algorithm. Any subtree must satisfy the same conditions $\Rightarrow$ Induction.

Note that since Shannon-Fano gives $E[\ell(X)] \leq H(X) + 1$, and Huffman is optimal, Huffman must satisfy the same bound.



Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.
2. Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.
2. Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.
   - We can of course first estimate the distribution from the data to be compressed, but how about the decoder?

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.
2. Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.
   - We can of course first estimate the distribution from the data to be compressed, but how about the decoder?
3. Distribution is not i.i.d.: Dependence and changes.

Two-Part Codes

Solution to problem 2:

2. The Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.
   - We can of course first estimate the distribution from the data to be compressed, but how about the decoder?

Two-Part Codes

Write the distribution (or code) in the beginning of the file.

Two-Part Codes

Solution to problem 2:

2. The Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.
   - We can of course first estimate the distribution from the data to be compressed, but how about the decoder?

Two-Part Codes

Write the distribution (or code) in the beginning of the file.

Usually the overhead is minor compared to the total file size.

Block Codes

Solution to problems 1 & 3:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.
3. Distribution is not i.i.d.: Dependence and changes.

Block Codes

Combine successive symbols into blocks and treat blocks as symbols. $\Rightarrow$ One extra bit per block.

Block Codes

Solution to problems 1 & 3:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.

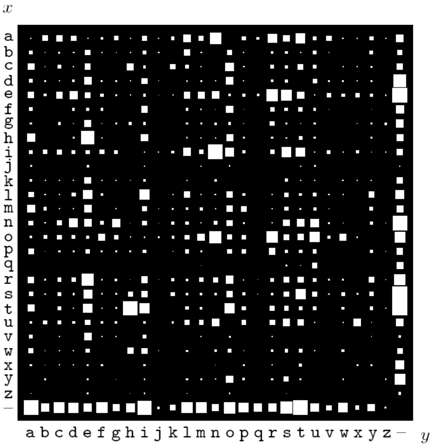
3. Distribution is not i.i.d.: Dependence and changes.

Block Codes

Combine successive symbols into blocks and treat blocks as symbols. $\Rightarrow$ One extra bit per block.

Allows modeling of dependence.

Block Codes


x
a b c d e f g h i j k l m n o p q r s t u v w x y z –
a b c d e f g h i j k l m n o p q r s t u v w x y z –
y

Block Codes

Combining solutions to problems 1–3, we get **two-part block codes**: Write first the joint distribution of blocks of N symbols, and then encode using blocks of length N.

Block Codes

Combining solutions to problems 1–3, we get **two-part block codes**: Write first the joint distribution of blocks of N symbols, and then encode using blocks of length N.

The size of the first part (distribution/code) grows with N, but the performance of the block code get better.

Block Codes

Combining solutions to problems 1–3, we get **two-part block codes**: Write first the joint distribution of blocks of N symbols, and then encode using blocks of length N.

The size of the first part (distribution/code) grows with N, but the performance of the block code get better.

Complexity Tradeoff

Find suitable balance between complexity of the model (increases with N) and codelength of data given model (decreases with N).
$\Rightarrow$ **Minimum Description Length (MDL) Principle**

Adaptive Codes

Alternative Solution to Problems 2 & 3:

Adaptive Codes

For each symbol (or a block of symbols), we can construct a code based on the probability $p(x_{\text{new}} \mid x_1, \ldots, x_n)$.

Adaptive Codes

Alternative Solution to Problems 2 & 3:

Adaptive Codes

For each symbol (or a block of symbols), we can construct a code based on the probability $p(x_{\mathrm{new}} \mid x_1, \ldots, x_n)$.

This may lead to computational problems since the code tree has to be constantly updated.

Adaptive Codes

Alternative Solution to Problems 2 & 3:

Adaptive Codes

For each symbol (or a block of symbols), we can construct a code based on the probability $p(x_{\text{new}} \mid x_1, \ldots, x_n)$.

This may lead to computational problems since the code tree has to be constantly updated.

Block coding with long blocks is another solution, but it introduces delay in decoding: the first symbol can be read only after the whole block is decoded.

Adaptive Codes

Alternative Solution to Problems 2 & 3:

Adaptive Codes

For each symbol (or a block of symbols), we can construct a code based on the probability $p(x_{\text{new}} \mid x_1, \ldots, x_n)$.

This may lead to computational problems since the code tree has to be constantly updated.

Block coding with long blocks is another solution, but it introduces delay in decoding: the first symbol can be read only after the whole block is decoded.

Arithmetic coding avoids "all problems": adaptive, spreads the one additional bit over the whole sequence, and can be decoded instantaneously. $\Rightarrow$ Read the material.