MATH3411 Information, Codes And Ciphers

2019 T3

Course Notes

Contents

School of Mathematics and Statistics, UNSW Sydney

These notes are based on lectures given by Dennis Trenerry, with some additional contributions and editing by other members of the School of Mathematics and Statistics: Catherine Greenhill, Jonathan Kress, John Steele, David Angell and Thomas Britz.

Chapter 1

Introduction

MATH3411 Information, Codes and Ciphers is a subject which deals with various mathematical aspects of discrete communication theory: in particular, digital or binary communication. It does not look at any Engineering or implementation aspects, and is also unconcerned with the meaning of the communicated message, just its transmission.

The basic idea is that a **sender** wishes to send some **information** along some sort of communications link or **channel** to a **destination**. Usually this is from one place to another, but it could equally well be between one time and another by recording, then later playing back the information.

Unfortunately there is almost certainly some **noise** or interference on the channel and this means that the signal does not get through correctly. For example, a small amount of noise on a phone line is usually not a disaster as the human ear can make out what was said even if some parts of it are distorted. This is because natural language is highly redundant.

On the other hand, the same noise on a modem or hard disk, where every single bit matters, can completely destroy any information being sent.

Our first problem is how to **encode** the signal so that, despite the noise, enough gets through so that the message is intelligible. Usually this encoding adds some extra information to the message, expanding its size, so that even after it has been corrupted by the noise, enough remains to be decoded into a meaningful message at the destination. This aspect is covered in Chapter 2 and later revisited in Chapter 6.

We use the following model of a channel subject to noise:

$$\begin{tikzcd}
\text{source} \arrow[r] & \text{encode} \arrow[r] & \text{channel} \arrow[r] & \text{decode} \arrow[r] & \text{destination} \\
 & & \text{noise} \arrow[u] & &
\end{tikzcd}$$

On the other hand, sometimes we want to fit as much information as we can into as small a space as possible (assuming that the amount of noise is very small). This means that we want to **compress** the information in some way. In Chapter 3 we consider some of the simpler text compression methods. Then in Chapter 4 we examine how much information can be passed along a noisy channel and consider the ideas of **information theory**.

Sometimes an enemy may be listening to the channel and so you want to disguise the message to make it unintelligible to them. Even worse, the person listening in may be trying to alter the message you are sending or even trying to pretend that they are you. This is the realm of **cryptography** which we study in Chapter 7.

For cryptography we have a model of the channel

$$\begin{tikzcd}
\text{source} \arrow[r] & \text{encrypt} \arrow[r] & \text{channel} \arrow[r] \arrow[d, leftrightarrow] & \text{decrypt} \arrow[r] & \text{destination} \\
 & & \text{listener} & &
\end{tikzcd}$$

In real life all three aspects may need to operate at once, so we would encrypt, then compress, then encode.

In this subject we deal only with digital information. There are a variety of techniques for compressing and encrypting analogue information, but we will not consider them here. Analogue information can always be digitised and sent as a string of digits if we wish, for example:

> *pictures:* each pixel or small element of the picture is given a set of numbers which describe its colour and brightness. These are then transmitted as digital data.
>
> *audio:* the height of the waveform is measured at certain time intervals and the heights are sent as digital data. For example, a standard music CD samples the height of the sound wave 44,100 times per second and uses $2^{16} = 65536$ different 16-bit height numbers.

Also we will not consider audio or visual information compression, coding or transmission, but restrict ourselves to text transmission only.

1.1 Mathematical Model

In general we will assume we have some **source** S consisting of q symbols from a given **source alphabet**

$$S = \{s_1,\ s_2,\ \cdots\ ,\ s_q\}$$

with associated probabilities of occuring of

$$p_1,\ p_2,\ \cdots\ ,\ p_q.$$

Sometimes we will also use conditional probabilities such as p_{ij} to mean the probability that symbol s_i follows symbol s_j. For example standard English uses $q = 26$ letters and these have various probabilities of occuring or following other letters. For example, the probability that in English text the letter 'U' occurs is 0.0271, but the probability that 'U' occurs immediately after a 'Q' is close to 1.

We think of the source as emitting the symbols one at a time.

The q symbols are then encoded, each by a string or word (which can be viewed as a vector) from a **code alphabet** having r symbols, ($r < q$ usually).

Each of the **source** symbols is given a **codeword** made up of a number of symbols in the code alphabet. The length of a codeword is the number of symbols that occur in it.

The code alphabet will usually be $\{0, 1\}$ and we then have **binary** codewords. The individual symbols in a binary codeword are called **bits**. In general when r symbols are used for the code alphabet we say this is a **radix** r code. Usually a radix r code uses the symbols $\{0, 1, 2, \cdots, r-1\}$ or $\{1, 2, 3, \cdots, r\}$.

For example, 1001101 or $(1, 0, 0, 1, 1, 0, 1)$ might be a particular binary codeword of length 7 or having 7 bits, and 25106 or $(2, 5, 1, 0, 6)$ could be a radix 7 codeword of length 5. (But it could also be a radix 5 codeword of length 5, using the symbols $\{0, 1, 2, 5, 6\}$ only.)

1.2 Modular Arithmetic

This subject will make extensive use of modular arithmetic. Modular arithmetic is *assumed knowledge* for this subject and you should revise the basics now.

We use the following notation:
$\mathbb{Z}$ denotes the **integers** $\{0, \pm 1, \pm 2, \cdots\}$
$\mathbb{N}$ denotes the **natural numbers** $\{0, 1, 2, \cdots\}$
$\mathbb{Z}^+$ denotes the **positive integers** $\{1, 2, \cdots\}$

The **Division Algorithm** says that for any $a \in \mathbb{Z},\ b \in \mathbb{Z}^+$ there exist unique integers q, r with

$$a = bq + r, \qquad 0 \le r < b.$$

Here q is the **quotient** and r, written as $a \pmod b$ is the **remainder**.

For a positive integer m, we write

$$a \equiv b \pmod m,$$

which is read as a is **congruent** to b **modulo** m, to mean that a and b leave the same (nonnegative) remainder when each is divided by m. This is equivalent to the condition that

$$a - b \text{ is exactly divisible by } m$$

and we denote this last line by

$$m \mid (a - b)$$

which is read as m **divides** $a - b$ or m is a **factor** of $a - b$.

Do not confuse the notations $\mid$ and $/$:

- $\mid$ is a *relation*: $b \mid a$ is either true or not, whereas
- $/$ is an *operator*: b/a is a number.

For example, $3 \mid 12$ is true and $12/3 = 4$ is an integer.

Addition, subtraction and multiplication (but NOT division) modulo m are done by doing the arithmetic as usual and then taking the positive remainder when divided by m.

Division is not in general allowed, but if for a given a we can solve the equation $ax \equiv 1 \pmod m$ then we say that a has an **inverse** (**modulo** m). We write this as $x \equiv a^{-1} (\text{mod } m)$. We will look at how to find inverses in chapter 5.

For example modulo 11, we have

$$\begin{array}{ccl}
5 + 9 = 14 \equiv 3 \pmod{11} & \text{as} & 14 - 3 = 11 \text{ is divisible by } 11 \\
5 - 9 = -4 \equiv 7 \pmod{11} & \text{as} & -4 - 7 = -11 \text{ is divisible by } 11 \\
5 \times 9 = 45 \equiv 1 \pmod{11} & \text{as} & 45 - 1 = 44 \text{ is divisible by } 11 \\
5^{-1} \equiv 9 \pmod{11} & \text{as} & 5 \times 9 \equiv 1 \pmod{11}
\end{array}$$

We also use the following notation:

$$\mathbb{Z}_m = \{0, 1, 2, \cdots, m - 1\}$$

to denote the set of nonnegative remainders when an integer is divided by m.

Occasionally it is useful to use negative remainders for the larger elements, in which case we take

$$\mathbb{Z}_m = \begin{cases} \left\{-\frac{m-1}{2},\ -\frac{m-1}{2}+1,\ \cdots\ ,-1,0,1,2,\ \cdots\ ,\frac{m-1}{2}\right\} & \text{when } m \text{ is odd} \\ \left\{-\frac{m}{2}+1,\ -\frac{m}{2}+2,\ \cdots\ ,-1,0,1,2,\ \cdots\ ,\frac{m}{2}\right\} & \text{when } m \text{ is even.} \end{cases}$$

For example,

$$\begin{aligned} \mathbb{Z}_6 &= \{0,1,2,3,4,5\} \qquad \text{or} \qquad \{-2,-1,0,1,2,3\} \\ \mathbb{Z}_7 &= \{0,1,2,3,4,5,6\} \qquad \text{or} \qquad \{-3,-2,-1,0,1,2,3\} \end{aligned}$$

We perform arithmetic **in** the set $\mathbb{Z}_m$ by doing the arithmetic as usual and then taking the nonnegative remainder of the result when divided by m.

So the above examples done **in** $\mathbb{Z}_{11}$ are written as

$$5+9=3, \quad 5-9=7, \quad 5\times 9=1, \quad 5^{-1}=9 \quad \textbf{in } \mathbb{Z}_{11}.$$

Note:

If $a \equiv b \pmod m$ and $c \equiv d \pmod m$ then

$$\begin{aligned} a+c &\equiv b+d \pmod m \\ a-c &\equiv b-d \pmod m \\ ac &\equiv bd \pmod m \end{aligned}$$

This allows us to mix integers and elements of $\mathbb{Z}_m$ as in the calculation

$5 \cdot 9 \equiv 45 \equiv 9 \pmod{12}$.

We will look more closely at $\mathbb{Z}_m$ in chapter 5.

Mostly we will be working with $\mathbb{Z}_2$, that is the binary case, and then the arithmetic is particularly simple.

$$0+0=0, \quad 0+1=1+0=1, \quad 1+1=0, \quad 0\cdot 0=0\cdot 1=1\cdot 0=0, \quad 1\cdot 1=1$$

For those who have done some logic, binary arithmetic is the same as the algebra of logic using XOR for "+" and AND for "·".

Binary codewords of length n can be viewed as vectors in $\mathbb{Z}_2^n$. All the usual rules of vector addition hold for such vectors. For example

$$\begin{aligned} 1001101+1101010 &= (1,0,0,1,1,0,1)+(1,1,0,1,0,1,0) \\ &= (0,1,0,0,1,1,1) \qquad \text{arithmetic mod 2} \\ &= 0100111 \end{aligned}$$

We usually omit the steps in the middle of the above calculation.

Primes

A positive integer $p > 1$ is a **prime** if and only if p has no proper divisors, that is,

if and only if the only divisors in $\mathbb{Z}^+$ of p are 1 and p

if and only if $p \neq 1$ and for all $d \in \mathbb{Z}^+$, if $d \mid p$ then $d = 1$ or $d = p$.

Example: 2, 3, 7, 101 are primes.

A number which is not prime is called **composite**. For example, $12 = 3 \cdot 4$ and $1001 = 7 \cdot 143$ are composites.

Every positive integer n has a **unique prime factorization**, that is, there exist unique primes $p_1 < p_2 < \cdots < p_r$ and unique $\alpha_1, \alpha_2, \cdots, \alpha_r \in \mathbb{Z}^+$ such that

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_r^{\alpha_r} .$$

For example, $12 = 3 \cdot 4$, $\quad 1001 = 7 \cdot 11 \cdot 13$, $\quad 123456 = 2^8 \cdot 3 \cdot 643$.

(We will later discuss how to prove a number is prime and how to factor it if it is composite.)

Another way of saying the definition of a prime p, is the following: if $ab \equiv 0 \pmod{p}$ then at least one of $a \equiv 0 \pmod{p}$ or $b \equiv 0 \pmod{p}$.

This can also be expressed by saying that **in** $\mathbb{Z}_p$, if $ab = 0$ then either $a = 0$ or $b = 0$.

1.3 Probability

If X is a (discrete) random variable, then write $P(X = x)$ for the probability that X takes the value x. We will only be interested in discrete random variables in this course.

The most important probability distribution for us will be the **binomial distribution**: if X is a random variable with probability distribution given by the binomial distribution with parameters $n \in \mathbb{N}$ and $p \in [0, 1]$, then X has positive probability on the set $\{0, 1, 2, \ldots, n\}$, namely

$$P(X = x) = \binom{n}{k} p^x (1-p)^{n-k}$$

For two discrete random variables X and Y, $P(Y = y, X = x)$ is the probability that $Y = y$ *and* $X = x$. Also, $P(Y = y \mid X = x)$ is the probability that $Y = y$ *given that* $X = x$ (i.e. *conditional upon* $X = x$). Now

$$\begin{aligned} P(Y = y, X = x) &= P(Y = y \mid X = x) \times P(X = x) \\ &= P(X = x \mid Y = y) \times P(Y = y) \end{aligned}$$

If $P(X = x|Y = y) = P(X = x)$ for all x, y, then X and Y are *independent*.

Bayes' Rule is very useful. It states that if we know $P(X = x)$ and $P(Y = y \mid X = x)$ for all x, y, then

$$\begin{aligned} P(X = x \mid Y = y) &= \frac{P(X = x, Y = y)}{P(Y = y)} \\ &= \frac{P(Y = y \mid X = x)\ P(X = x)}{\sum\limits_{x_i} P(Y = y \mid X = x_i)\ P(X = x_i)} \end{aligned}$$

Bayes' Rule tells us how to reverse a conditional probability.

1.4 Morse Code

Morse code is a code that was designed for early wire telegraph systems and then was used on radio. It consists of two symbols **dot** (or **di** or **dit**) shown here as $\cdot$ and **dash** (or **dah**) shown here as $-$ and letters are represented by a string of these symbols. Individual letters are separated by a third symbol which is a **pause** or `p`, so this is a radix 3 code. The `p` is needed to avoid confusion about where each letter's codeword begins and ends.

In Appendix A.1 is a full listing of the International Morse Code codewords. For example `A` has codeword $\cdot - \texttt{p}$, `B` has codeword $- \cdot \texttt{p}$, etc.

For example, in Morse code the English word

`MATH3411`

gets coded as

$$- - \texttt{p} \cdot - \texttt{p} - \texttt{p} \cdot\cdot\cdot\cdot \texttt{p} \cdot\cdot\cdot - - \texttt{p} \cdot\cdot\cdot\cdot - \texttt{p} \cdot - - - - \texttt{p} \cdot - - - - \texttt{p}$$

Morse code is based roughly on the relative frequencies of letters in military English so as to keep messages short — it actually is a sort of compression code.

`E` ($\cdot$`p`), `T` ($-$`p`) and `A` ($\cdot -$ `p`) are the most frequent letters and `Q` ($- - \cdot -$ `p`) is far less frequent.

We call this a **variable length** codeword code, as codewords can have different lengths.

1.5 ASCII code

The 7-bit ASCII code is listed in full in Appendix A.2 and is a **fixed length** codeword code or **block code** with blocks of length 7 in the binary alphabet $\{0, 1\}$.

For example

$$\begin{array}{ll|ccccccc} & & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 \\ A & \text{is} & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ a & \text{is} & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ \} & \text{is} & 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{array}$$

Here the b_i label the bits and the convention for ASCII is to number them from the right.

There are $128 = 2^7$ symbols or characters that can be represented by 7-bit ASCII. (The first 32 symbols are special control characters and do not all appear on keyboards.)

Often, or rather usually, ASCII has an 8th bit in front of the other 7, but what it signifies depends on the application.

In some applications the 8th bit gives $2^8 = 256$ different characters in the **extended ASCII code**, where for example

11110100   means the top half of an integral sign.

In other applications or the 8th bit is used as a **parity bit** or **check bit**. Usually **even** parity is used, which means that the 8th bit is chosen so that every 8-bit codeword contains an even number of '1's. This code is usually just called the ASCII code.

For example in (8-bit even parity) ASCII code

$$\begin{array}{lll} & & \text{parity bit} \\ & & \downarrow \\ A & \text{is} & 01000001 \\ a & \text{is} & 11100001 \end{array}$$

Even something as simple as a parity check can make a big difference to error detection.

1.6 ISBN Numbers

ISBN numbers are used to identify printed books, in particular, the country of publication, the publisher and the particular book number allocated by the publisher.

If you look inside the front cover of any book published before 2007 you will find that there is a 10 digit ISBN number of the form

$$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}$$

where $x_1, \cdots, x_9$ are decimal digits $0, 1, \cdots, 9$ and x_{10} is a decimal digit or an X (Roman numeral for 10). (There are usually some dashes as well but they are there to make it more readable and are not formally part of the number.)

The first 9 digits $x_1 x_2 \cdots x_9$ have meaning and the 10th digit x_{10} is a generalised parity number or **check digit** chosen so that

$$\sum_{i=1}^{10} i x_i \equiv 0 \pmod{11}.$$

For example the reference book by Roman has ISBN

$$0 - 387 - 97812 - 7$$

$$\begin{array}{cccc} \nearrow & \uparrow & \uparrow & \nwarrow \\ \text{country} & \text{publisher} & \text{publishers} & \text{check} \\ & & \text{book no.} & \text{digit} \end{array}$$

Let us check that this is a valid ISBN number:

$$\begin{aligned} \sum i x_i &= 1 \times 0 + 2 \times 3 + 3 \times 8 + 4 \times 7 + 5 \times 9 + 6 \times 7 + 7 \times 8 + 8 \times 1 + 9 \times 2 + 10 \times 7 \\ &= 297 \\ &\equiv 0 \pmod{11}. \end{aligned}$$

In other words, $\sum i x_i$ is exactly divisible by 11.

The check digit x_{10} needs to be able to take a value in the range $0, 1, \cdots, 9, 10 = \text{X}$ to balance the equation mod 11.

The expression $S(\boldsymbol{x}) = \sum_{i=1}^{10} i x_i$ for $\boldsymbol{x} = x_1 x_2 \cdots x_{10}$ is called the **syndrome** of $\boldsymbol{x}$ in this code. The syndrome is a particular weighted check sum of $\boldsymbol{x}$.

From the beginning of 2007 a 13 digit ISBN number, with a different check digit calculation, has been used.

Chapter 2

Error Detection and Correction Codes

Now suppose that an (8-bit even parity) ASCII codeword is transmitted and some noise causes one of the bits to change. That is, a '0' becomes a '1' or a '1' becomes a '0'.

We say the codeword is **corrupted** to a corrupted codeword or just to a "word" and denote this by codeword $\rightsquigarrow$ word.

For example

$$\begin{array}{c} \text{noise} \\ \downarrow \\ 11100001 \rightsquigarrow 10100001 \end{array}$$

If the decoder at the destination is set up to expect 8-bit even parity code words, it will recognise that 10100001 has odd parity (odd number of '1's) and so cannot be a codeword.

It will thus have **detected** that there is an error. So this code is capable of detecting a single error in a codeword.

The only action that can be taken by the decoder is to **reject** the corrupted codeword. This results in a "parity error" message or a request to send the codeword again.

If there happen to be 2 errors, then the corrupted codeword will have an even parity and the double error will **not** be detected. It is obvious that (8-bit even parity) ASCII code detects any *odd* number of errors, but an *even* number of errors will be undetected – the parity will again be even.

So we call it a **single-error detecting** code as it can detect 1 error but not 2 errors.

2.1 Random Noise

Now suppose the noise on the channel is **white noise**, by which we mean it is purely random and each location in a codeword is just as likely as any other to be altered.

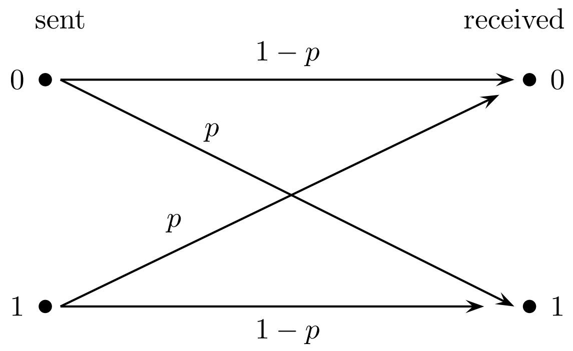
We assume that the

- probability of error in each bit position is p, and
- errors in different positions are independent.

We call p the **bit-error probability**.

We call this situation of white noise on a channel transmitting '0's and '1's a **binary symmetric channel** or **BSC** and will draw a diagram showing the various probabilities of what is sent and what arrives, where the labels on the transition arrows show the probability of a symbol taking that path.

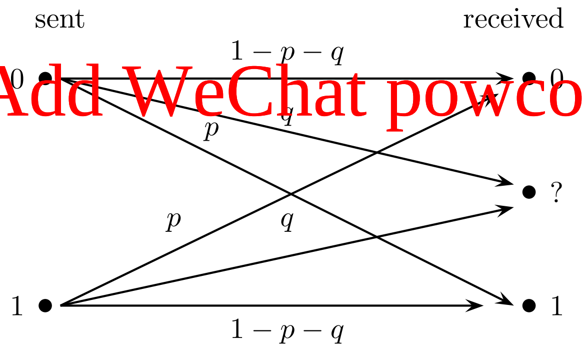
This diagram of a BSC applies only to the sending of a single bit.



We will study such channels in more detail later.

Another type of channel is where, instead of just the possiblilities of $0 \rightsquigarrow 1$ or $1 \rightsquigarrow 0$ changes, the 0 or 1 gets changed to an ambiguous symbol which is neither 0 nor 1. We call this an **erasure** and denote it by '?'. Note that here we are assuming the bit is not missing, just ambiguous. For example, if a '0' is a -1 volt signal and '1' is a 2 volt signal, then '?' might be a signal between -0.3 and $+0.3$ volts.

This is the **binary symmetric erasure channel** or **BSEC**



Notice that ASCII can actually **correct** one erasure. That is, it can fill in the '?' with a '0' or '1' as appropriate since we know that the total number of '1's must be even. For example

0101?001 corrects to 01011001

On the other hand, if one of the bits goes completely missing then we only receive 7 bits (for example, we may receive the word 0101001). We can detect that an error of this type has occured, we can work out whether it is a '0' or '1' missing (by parity), but we have no idea which one of the bits is missing.

2.2 Error Probabilities

Suppose we have a binary symmetric channel subject to white noise with bit-error probability of p, and suppose that we have codewords of length n (that is, n-bit codewords).

Suppose also that we are using an even parity check code. That is, all codewords of length n contain an even number of '1's. We call this an **n-bit even parity code**.

Let X be the number of errors in the n-bit codeword. Then the assumptions of white noise mean that X is a random variable following the **binomial** distribution, where

$$P(X=k)=\binom{n}{k}p^k(1-p)^{n-k}, \qquad \text{for } k=0,1,\cdots,n.$$

As with the (8-bit even parity) ASCII code, an odd number of errors will be detected but not an even number.

Hence the probability that some errors occur but that they are not detected is

$$P(X=2)+P(X=4)+P(X=6)+P(X=8)+\ \cdots$$

We call this the probability of **undetected error**. (Remember that if $X=0$ then there is *no error*, so we do not include this case.)

Example 2.1 ASCII. Then $n=8$ and

if	$p=0.1$	then the prob of undetected error	$=0.1534$
if	$p=0.001$	then the prob of undetected error	$=2.8\times 10^{-5}$

2.3 Error capabilities of ISBN numbers

ISBN-10 numbers are designed to **detect** 2 types of common human error.

Theorem 2.1 : ISBN-10 Error Capability

ISBN-10 numbers are capable of detecting the two types of errors:

1. *getting a digit wrong*

2. *interchanging two (unequal) digits.*

Proof: Suppose a codeword $\boldsymbol{x}=x_1x_2\cdots x_{10}$ with $S(\boldsymbol{x})=\sum_{i=1}^{10} ix_i\equiv 0 \pmod{11}$ is sent and $\boldsymbol{y}=y_1y_2\cdots y_{10}$ is received. That is, $\boldsymbol{x}\rightsquigarrow\boldsymbol{y}$ with one error of type 1 or type 2.

Type 1 error: We have $y_k=x_k+m$ for some k and $y_i=x_i$ for all $i\neq k$. (Here m is the magnitude of the error (m

For example, if $y_k=x_k+3$ then $m=3$ and if $y_k=x_k-3$ then $m=8\equiv -3 \pmod{11}$. We may assume that $m\neq 0$ otherwise we have no error.

In this case, calculating the syndrome gives

$$\begin{aligned} S(\boldsymbol{y}) &= \sum_{i=1}^{10} iy_i \\ &= \sum_{i=1}^{10} ix_i + km \\ &\equiv 0+km \\ &\equiv km \pmod{11} \end{aligned}$$

Now $m\neq 0$ and $k\neq 0$ so $km\neq 0$ since 11 is a prime, and we conclude that $S(\boldsymbol{y})\neq 0$ and hence we have detected the error.

Type 2 error: In this case for some $k>\ell$ we have

$$y_k=x_\ell, \quad y_\ell=x_k \text{ and } y_i=x_i \text{ for all } \ i\neq \ell,k.$$

We can assume that $x_\ell \neq x_k$, since there is no error if $x_\ell = x_k$.

Now, calculating the syndrome gives

$$\begin{aligned}
S(\boldsymbol{y}) &= \sum_{i=1}^{10} i y_i \\
&= \sum_{i=1}^{10} i x_i + (k-\ell)x_\ell + (\ell-k)x_k \\
&\qquad\qquad \text{since } \; k y_k = k x_\ell \quad \text{and} \quad \ell y_\ell = \ell x_k \\
&\equiv 0 + (k-\ell)(x_\ell - x_k) \\
&\not\equiv 0 \qquad\qquad \text{since } k \neq \ell,\; x_k \neq x_\ell
\end{aligned}$$

and again the error is detected.

□

Example 2.2 Given that $\mathbf{x} = 1581691750$ is a valid ISBN. Swap two digits to get $\mathbf{y} =$ 1581961750. Then

$$\begin{aligned}
\sum_{i=1}^{10} i y_i &= 1\times 1 + 2\times 5 + 3\times 8 + 4\times 1 + 5\times 9 \\
&\quad + 6\times 6 + 7\times 1 + 8\times 7 + 9\times 5 + 10\times 0 \\
&\equiv 1 + 10 + 2 + 4 + 1 + 3 + 7 + 1 + 1 + 0 \pmod{11} \\
&\equiv 8 \pmod{11} \\
&\not\equiv 0 \pmod{11}
\end{aligned}$$

Hence $\mathbf{y}$ is not a valid ISBN.

Note that the ISBN-13 standard does not detect all digit transpositions.
There are many other codes in common use that incorporate check digits, for example:

- bar codes used in supermarkets;
- the Australian Business Number (ABN) and Tax File Number (TFN);
- credit card numbers.

2.4 Burst noise

Often noise occurs in bursts that affect a number of consecutive bits.

For example, 5 bits in a row may all be swapped $0 \rightsquigarrow 1$, $1 \rightsquigarrow 0$. Or, more likely, all may become '0' (that is, $0 \rightsquigarrow 0$, $1 \rightsquigarrow 0$) or all may become '1'.

If we assume that the burst noise affects fewer consecutive bits than the codeword length and is fairly rare, then we can construct a **burst code** by using a **check codeword** or a **check sum**.

This is used as a check on a group of codewords or even on the whole message. It is commonly formed by choosing the bits of the check codeword so that the total message has an even number of '1's *in each bit position.*

Example 2.3 With the **7-bit ASCII burst code** applied to the message MATH3411

MATH3411

gives

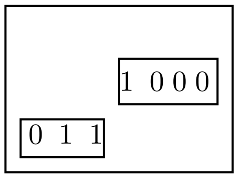
	ASCII
M	1001101
A	1000001
T	1010100
H	1001000
3	0110011
4	0110100
1	0110001
1	0110001
	0010111 $\to$ `ETB` symbol

Here there are an even number of '1's in each column (that is, in each bit position). The bits in the checksum codeword are found by adding down columns modulo 2 and using this value for the bit.

So the message sent would be MATH3411$\overline{\texttt{ETB}}$

Then any block of ≤ 7 consecutive bits changing all to '0' or all to '1' will mean that the check word or check sum is no longer correct and so the burst error is detected.

For example, suppose in the following that all the bits shown change to '1'



then F ✓ ✓ ✓ F F F is the pattern of failures in the columns

✓ indicates that the column parity is OK.
F indicates column parity fails

Note that real-life checksums are usually much more sophisticated than this.

2.5 Error-correcting codes

Suppose now we use (8-bit even parity) ASCII with a check codeword

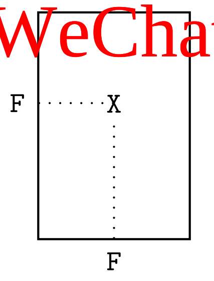
	check bit ↓
M	01001101
A	01000001
T	11010100
H	01001000
3	00110011
4	10110100
1	10110001
1	10110001
	00010111 $\to$ $\overline{\texttt{ETB}}$

Then a single error anywhere will be detected by a parity check failure in the *row* where the error occurs and in the *column* where the error occurs.

Hence the *location* of the error can be found. Since this is a binary code then the size of the error can only be $= 1$ (as $1 \equiv -1 \pmod 2$), so the error can actually be **corrected**.

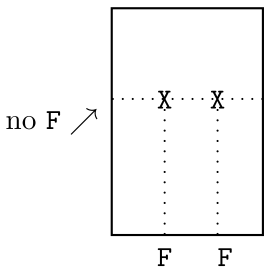
Here are some pictures showing the pattern of row and column parity failures for various errors (shown as X). In some cases this information is sufficient to locate and hence correct the errors, while in others it is insufficient or misleading.

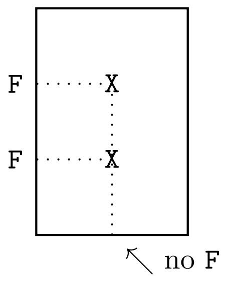
1 error picture

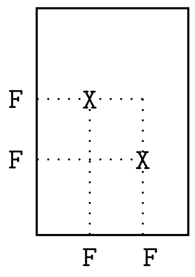


The failure pattern allows us to detect and correct any single error.

2 error pictures

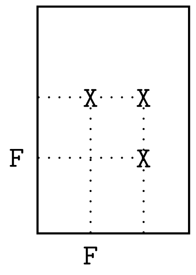






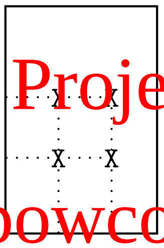
We see that all patterns of 2 errors are detected but they cannot be corrected as we do not know exactly where the errors occurred. For example, in the first diagram all we can tell is that the 2 errors are in a certain 2 columns and in the same (unknown) row, while in the third diagram the errors may be in the places shown or they may be at the other vertices of the small rectangle.

one 3 error picture



This pattern of 3 errors produces exactly the same row/column failure pattern as a single error — so we would normally take it as a single error (in the other vertex of the small rectangle) as that is more likely to occur than 3 errors. All we know is that either 1 or 3 errors (or maybe more) have occurred.

one 4 error picture



This pattern of 4 errors is not detected at all.

Hence, with 8-bit ASCII with check codeword (and similar rectangular array codes) we can adopt various *mutually exclusive* decoding strategies:

Strategy 1: correct 1 error and simultaneously detect 2 errors (and treat 3 errors as 1 error)

Strategy 2: detect 1, 2 or 3 errors (while not distinguishing between 1 and 3 errors)

If we have a long message then we divide it up into say 8 character blocks (with blanks at the end if necessary) each of which is encoded like the above using the 9th character as a check codeword.

For example MATH3411␣IS␣FUN. (␣ = space) gets encoded as

MATH3411$\overline{\text{ETB}}$	␣IS␣FUN.i
block 1	block 2

We have done the first block above, so now let's check the second block as well.

␣	10100000	
I	11001001	
S	01010011	
␣	10100000	
F	11000110	
U	01010101	
N	01001110	
.	00101110	
	01101001	$\to$ i

Let's call this the **9-character 8-bit ASCII code**.

Then there are $9 \times 8 = 72$ bits in total per block, of which $8 \times 7 = 56$ are the **information bits** (that is, they contain the information), and $8 + 8 = 16$ bits are **check bits**. (There are 8 parity checks on rows and 8 parity checks on columns, since the last row check depends on the others).

Now suppose we labelled the bits in a matrix-style notation where x_{ij} is the ith character's jth bit.

$$\begin{array}{cccclcl}
x_{11} & x_{12} & \cdots & x_{18} & & \text{for 1st character} \\
 & & \vdots & & & \vdots \\
x_{81} & x_{82} & \cdots & x_{88} & & \text{for 8th character} \\
x_{91} & x_{92} & \cdots & x_{98} & & \text{for check character}
\end{array}$$

Then there are 16 **independent check equations** among the 72 bits, namely

$$\begin{array}{llll}
x_{11} + x_{12} + \cdots + x_{18} & \equiv 0 & \pmod 2 & \text{(first row)} \\
\vdots & \vdots & & \\
x_{81} + x_{82} + \cdots + x_{88} & \equiv 0 & \pmod 2 & \text{(eighth row)} \\
x_{11} + x_{21} + \cdots + x_{81} + x_{91} & \equiv 0 & \pmod 2 & \text{(first column)} \\
\vdots & \vdots & & \\
x_{18} + x_{28} + \cdots + x_{88} + x_{98} & \equiv 0 & \pmod 2 & \text{(eighth column)}
\end{array}$$

By "independent" here we mean that the above 16 equations are linearly independent in the usual linear algebra sense.

The 17th checksum is the sum across the ninth row

$$x_{91} + x_{92} + \cdots + x_{98} \equiv 0 \pmod 2.$$

This equation is not independent of the above 16 — it is in fact the sum of all of them.

As we have seen, the 9-character 8-bit ASCII code is able to correct single errors in each block of 8 original (9 encoded) characters.

This is an example of a length 72 binary code. We take as source symbols the 8-character words and code them as 72-bit codewords $\boldsymbol{x} = x_{11} \cdots x_{18} x_{21} \cdots x_{28} \cdots x_{91} \cdots x_{98}$.

The same approach can be applied to any rectangular array of $m \times n$ bits where there are parity checks on each row and each column.

2.6 Types of codes

Codes like Morse Code where the codewords have different lengths are called **variable length** codes. Codes like 7-bit ASCII or 9-character 8-bit ASCII where every codeword is the same **length** are called **block** codes.

If t is the maximum number of errors such that *every* pattern of t or fewer errors can be corrected, then we call the code a **t-error correcting code**.

A code where there is a clear distinction between the information digits and the check digits is called a **systematic** code.

For example, in ISBN the first 9 numbers are information digits and the 10th is the check digit.

In 9-character 8-bit ASCII code there are 16 check bits or parity bits and the other 56 are information bits.

Some other codes do not have clearly defined information and check digits.

For example, consider the code C_1 below consisting of the 8 codewords, where each codeword is written as a row. It looks like C_1 is systematic, with the first 3 bits holding the information (since the binary numbers 000 to 111 each occur exactly once). But the code C_2 is clearly not systematic.

C_1	C_2
11111111	00000000
01010101	11000000
00110011	10100000
10011001	10010000
00001111	10001000
10100101	10000100
11000011	10000010
01101001	10000001

2.7 Binary Repetition codes

Binary repetition codes are the simplest error-correcting codes: simply repeat each symbol a fixed number of times.

Example 2.4

	source symbol		code word
triple repetition	0	$\to$	000
	1	$\to$	111
5-repetition	0	$\to$	00000
	1	$\to$	11111

If in the 5-repetition code, say 10010 is received then the majority of bits are ‘0’ so it is most natural to correct to 00000 and then decode as 0. Here we have corrected 2 errors. We write this complete process (of taking a corrupted codeword back to the original source symbol), which we will call **decoding** for short, as 10010 $\to$ 0.

As well as the obvious way of decoding described above, there are other ways in which we can decode.

The *mutually exclusive* **decoding strategies** we can adopt here are the following, where $\to$ indicates what the codeword is decoded to and `F` indicates that the decoder cannot decide what to decode to so simply indicates a failure.

Strategy 1: correct up to 2 errors.

For example, $\begin{array}{lcl} 00001 & \to & 0 \\ 00011 & \to & 0 \end{array}$

Strategy 2: correct 1 error and simultaneously detect 2 or 3 errors.

For example, $\begin{array}{lcl} 00001 & \to & 0 \\ 00011 & & \texttt{F} \\ 00111 & & \texttt{F} \end{array}$

Strategy 3: detect up to 4 errors.

For example, $\left.\begin{array}{l} 00001 \\ 00011 \\ 00111 \\ 01111 \end{array}\right\} \quad \texttt{F}$

Since the largest number of errors that can be corrected is 2 we call this a **2-error correcting code**.

Convince yourself that a $(2t + 1)$ – repetition code can be used to

correct t errors
XOR
correct $t - 1$ and simultaneously detect t or $t + 1$
XOR
etc.

In general, for each $0 \le i \le t$ there is a decoding strategy whereby we can correct up to $t - i$ errors and simultaneously detect up to $t + i$ errors.

Since the maximum number of errors we can correct is t, we say that this code is a t-error correcting code.

Similarly a $(2t + 2)$ – repetition code can be used to

correct t and simultaneously detect $t + 1$
XOR
correct $t - 1$ and simultaneously detect t or $t + 1$
XOR
etc.

In general, for each $0 \le i \le t$ there is a decoding strategy whereby we can correct up to $t - i$ errors and simultaneously detect up to $t + 1 + i$ errors.

We say that this code is a t-error correcting code and a $(t + 1)$-error detecting code.

2.8 Information rate and redundancy

Read the following out loud:

“We kn nrmly umdersant ritn Inclsh”

and you can probably tell that it says

“We can normally understand written English”.

Despite the errors and missing letters this is easy to understand. The same sort of thing holds for a moderately noisy phone call.

Why?

Because English has a lot of redundancy and context which can be used to reconstruct the missing and incorrect letters.

We define the mathematical **redundancy** for block codes as follows.

For systematic codes the **information rate** R is given by

$$R = \frac{\text{number of information digits}}{\text{total number of digits}}$$

$$= \frac{\#\text{ information digits}}{\text{length of code}} .$$

For any general set of radix r codewords C the information rate R is given by

$$R = \frac{\log_r |C|}{\text{length of code}}$$

We then define

$$\textbf{redundancy} = \frac{1}{R}.$$

For our earlier examples, see figure 2.1 on the next page.

We will prove the error correction and detection results for C_1 and C_2 later.

2.9 Binary Hamming error correcting codes

We will now construct a code from first principles with the following properties: the code

- is binary with code alphabet $\{0, 1\}$,
- has fixed length n codewords $\boldsymbol{x} = x_1 x_2 \cdots x_n$,
- is single error correcting.

This code will be called a **binary Hamming-type code**. (NOTE: there are alternative ways of constructing these codes, some of which we will consider later.)

Suppose this code satisfies m independent parity checks or check equations of the form

$$\sum_{j=1}^{n} a_{ij} x_j \equiv 0 \pmod 2), \quad \text{for } i = 1, \ldots, m, \quad \text{where all } a_{ij} \in \{0, 1\}.$$

Suppose that $\boldsymbol{y}$ is $\boldsymbol{x}$ with a single bit error: that is, $\boldsymbol{x} \rightsquigarrow \boldsymbol{y}$.

We want to correct the error in $\boldsymbol{y}$ to get back to $\boldsymbol{x}$. Since the code is binary, we need only to be able to *locate* the position of the error: once you know the position you simply flip the bit in that position to correct the error (that is, a ‘1’ gets corrected to a ‘0’ and vice versa).

code	**rate R**	**redundancy**	**error correcting capability**
7-bit ASCII	$\frac{7}{7}$	1	0 detect
8-bit ASCII	$\frac{7}{8}$	$\frac{8}{7} = 1.14$	1 detect
ISBN	$\frac{9}{10}$	$\frac{10}{9} = 1.11$	1 detect transpose detect
9-char 8-bit ASCII	$\frac{56}{72}$	$\frac{9}{7} = 1.29$	1 correct XOR 3 detect XOR 8 burst detect
$(2t+1)-$repet.	$\frac{1}{2t+1}$	$2t+1$	t correct etc.
$(2t+2)-$repet.	$\frac{1}{2t+2}$	$2t+2$	t corr & detect $t+1$ etc.
C_1	$\frac{\log_2 8}{8} = \frac{3}{8}$	$\frac{8}{3} = 2.67$	1 and detect 2
C_2	$\frac{\log_2 8}{8} = \frac{3}{8}$	$\frac{8}{3} = 2.67$	1 detect

Figure 2.1: Redundancy of Codes

There are n possible locations for the error.
Now apply the parity check equations to $\boldsymbol{y}$ to give m bits, $u_1, \ldots, u_m$, as follows

$$\sum_{j=1}^{n} a_{ij} y_j \equiv u_i \pmod 2, \quad i = 1, \ldots, m.$$

Then form the m-bit number

$$S(\boldsymbol{y}) = u_m u_{m-1} \ldots u_2 u_1$$

called the **syndrome** of $\boldsymbol{y}$.

Now $S(\boldsymbol{x}) = \mathbf{0}$ for all codewords $\boldsymbol{x}$ and so we do not want $S(\boldsymbol{y}) = \mathbf{0}$ if $\boldsymbol{y}$ is not a codeword.
Hence there are 2^m possible syndromes of which $2^m - 1$ can be used to locate the error.
So to be sure of locating an error, we need to have more useable syndromes than locations. That is, we need $2^m - 1 \geq n$, which rearranged gives

$$2^m \geq n + 1.$$

In fact we can associate the syndromes to locations in an obvious way by taking the syndromes as the *binary numbers for the locations.*

error position	syndrome
no error	0000
1st	0001
2nd	0010
3rd	0011
4th	0100
5th	0101
6th	0110
7th	0111
8th	1000
etc.	etc.

Suppose that $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e}_k$ where $\boldsymbol{e}_k$ is the k'th standard basis vector with a 1 in the k'th place and zeroes everywhere else. Then you can check that

$$S(\boldsymbol{y}) = S(\boldsymbol{x} + \boldsymbol{e}_k) = S(\boldsymbol{x}) + S(\boldsymbol{e}_k) = S(\boldsymbol{e}_k)$$

since $S(\boldsymbol{x}) = \mathbf{0}$, as $\boldsymbol{x}$ is a codeword. That means that we want to design the check equations to make sure that $S(\boldsymbol{e}_k)$ equals the *binary expansion of* k.

Since the syndrome is $u_m u_{m-1} \cdots u_1$, we want the right-most bit to correspond to the first check equation, and the next bit to correspond to the second check equation, and so on, with the left-most bit corresponding to the last check equation.

Hence it is natural to take

$$\begin{array}{lllll} \text{parity check equation 1} & \text{to be} & x_1 + x_3 + x_5 + x_7 & + \cdots & \equiv 0 \pmod 2 \\ \phantom{\text{parity check equation }} 2 & & x_2 + x_3 + x_6 + x_7 & + \cdots & \equiv 0 \\ \phantom{\text{parity check equation }} 3 & & x_4 + x_5 + x_6 + x_7 & + \cdots & \equiv 0 \\ \phantom{\text{parity check equation }} 4 & & x_8 + \cdots & & \equiv 0 \\ & \text{etc.} & & & \end{array}$$

Note that

$$\begin{array}{lll} x_1 & \text{occurs only in check equation} & 1 \\ x_2 & & 2 \\ x_4 & & 3 \\ x_8 & & 4 \\ \vdots & & \vdots \\ x_{2^{p-1}} & & p \end{array}$$

So we can take $x_{2^{p-1}}$ to be the check bit for the pth check equation, as it can be written as a sum of all the others. Then the pth check equation is of the form

$$\sum_{j=1}^{n} a_{pj} x_j \equiv 0 \pmod 2$$

where

$$a_{pj} = \begin{cases} 1, & \text{if the binary representation of } j \text{ has a `1' in the } 2^{p-1} \text{ digit} \\ 0, & \text{otherwise} \end{cases}$$

This determines *exactly* m **parity bits.**
The other $k = n - m$ bits in the codeword can then be used as **information bits.**

Hence we see that the k information bits are arbitrary and the m parity bits are uniquely determined by them.

Hence we have constructed a code with

- length n,
- m parity bits, where $2^m \geq n+1$,
- $k = n - m$ information bits and hence 2^k codewords.

Such codes are called **Hamming-type codes.**

The name **Hamming code** will be used only in the special case when $2^m = n+1$.

From this point on we will usually write "congruence modulo 2" as "equals".

Examples:

1. Suppose that we want a single-error correcting code with codewords of length $n = 5$. Then we need $2^m \geq 6$, so the least possible value of m is $m = 3$, and hence $k = n - m = 2$.

   The vector $\boldsymbol{x} = x_1x_2x_3x_4x_5$ is a codeword if and only if

$$\begin{cases} x_1 \quad + x_3 \quad + x_5 = 0 \\ \quad x_2 + x_3 \qquad = 0 \\ \qquad\quad x_4 + x_5 = 0 \end{cases}$$

   The information bits are naturally x_3, x_5

   and the check bits are x_1, x_2, x_4 where $\begin{cases} x_1 = x_3 + x_5 \\ x_2 = x_3 \\ x_4 = x_5 \end{cases}$

   In fact the $2^2 = 4$ codewords are exactly the following, where the check bits are underlined:

info word	→	$\underline{x_1}$	$\underline{x_2}$	x_3	$\underline{x_4}$	x_5
00	→	0	0	0	0	0
01	→	1	0	0	1	1
10	→	1	1	1	0	0
11	→	0	1	1	1	1

2. This time we will try to get the most out of the number of parity checks. So we will construct the *largest* single-error correcting code with $m = 3$. The biggest value of n which satisfies the bound

$$2^m = 2^3 = 8 \geq n + 1$$

   is $n = 7$. Hence the codewords have length 7 and there are $k = 4$ information bits. Thus we have $2^4 = 16$ codewords. The vector $\boldsymbol{x} = x_1x_2x_3x_4x_5x_6x_7$ is a codeword if and only if

$$\begin{cases} x_1 \quad +x_3 \quad +x_5 \quad +x_7 = 0 \\ \quad x_2 +x_3 \quad\quad +x_6 +x_7 = 0 \\ \qquad\quad x_4 +x_5 +x_6 +x_7 = 0 \end{cases}$$

   So, for example, to encode 0101 = (information word) we use a codeword of the form _ _ 0 _ 1 0 1 where _ signifies a check bit.

Applying the check equations fills in the missing bits as $\underline{0}\;\underline{1}\;0\;\underline{0}\;1\;0\;1 = \boldsymbol{x}$

Now suppose we have an error in location 6 so we receive

$$\boldsymbol{y} = 0\;1\;0\;0\;1\;1\;1$$

Calculations for the syndrome $S(\boldsymbol{y})$ are

$$\begin{array}{lccccccccl}
\text{check } 1 & y_1 & & +y_3 & & +y_5 & & +y_7 & = 0 = u_1 \\
\text{check } 2 & & y_2 & +y_3 & & & +y_6 & +y_7 & = 1 = u_2 \\
\text{check } 3 & & & & y_4 & +y_5 & +y_6 & +y_7 & = 1 = u_3
\end{array}$$

so the syndrome $S(\boldsymbol{y}) = u_3u_2u_1 = 110$ (reading upwards). This is the binary expansion for 6 and so the error is in 6th place.

Hence we can correct the error to give $\boldsymbol{x} = \underline{0}\;\underline{1}\;0\;\underline{0}\;1\;0\;1$ and then decode by omitting the check bits to give

$$\text{information word } = 0\;1\;0\;1$$

This code is the **Hamming (7,4) code** with length 7 and containing 4 information bits.

All the above can be done much more compactly using matrices and matrix multiplication.

The matrix H of the check equations is called the **parity check matrix**. In this example

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

which we recognise as simply the binary numbers for the columns turned on their sides and reading from bottom to top – this was the intent of our construction.

Codewords are vectors and hence columns in matrix notation. Hence $\boldsymbol{x}$ is a codeword if and only if $H\boldsymbol{x}^T = \mathbf{0}$,

and the syndrome $S(\boldsymbol{y}) = H\boldsymbol{y}^T = H \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_7 \end{pmatrix}$.

Applying this to Example 2 above, we see that $H \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$ which is the the 6th column since $6 = 110$ in binary.

3. This time we will try for the smallest code needed to encode $k = 8$ information bits.

   We require m so that $\quad 2^m \geq n + 1 = m + k + 1 = m + 9$.

   By trial and error, we find that $m = 4$ is the smallest such integer, using the table

m	1	2	3	4
$m+9$	10	11	12	13
2^m	2	4	8	16

Hence $n = 8 + 4 = 12$ and

$$H = \begin{pmatrix} 1&0&1&0&1&0&1&0&1&0&1&0 \\ 0&1&1&0&0&1&1&0&0&1&1&0 \\ 0&0&0&1&1&1&1&0&0&0&0&1 \\ 0&0&0&0&0&0&0&1&1&1&1&1 \end{pmatrix}$$

Notice the pattern of the '1's in the check matrix of a Hamming-type code constructed in this way. If we put a zero column in front of the matrix, then it would have the following property — row i consists of 2^{i-1} '0's followed by 2^{i-1} '1's periodically until the length n is reached.

2.10 Binary Hamming (n, k) codes

Binary Hamming (n, k) codes have the following parameters:

m	$k = 2^m - m - 1$	$n = k + m$	$R = k/n$
3	4	7	0.57
4	11	15	0.73
5	26	31	0.84
6	57	63	0.90
7	120	127	0.94
8	247	255	0.97
9	502	511	0.98
10	1013	1023	0.99

Clearly $R \to 1$ as $m, n \to \infty$.

However the Hamming (1023, 1013) code only corrects 1 error in a 1023-bit codeword, so it is only of use when the probability of error is extremely low.

2.11 Hamming Distance, Weights (Binary case)

Given two n-bit words $\boldsymbol{x}$ and $\boldsymbol{y}$, the **Hamming distance** between them is defined to be

$$\begin{aligned} d(\boldsymbol{x}, \boldsymbol{y}) &= \text{ number of bits in } \boldsymbol{x} \text{ and } \boldsymbol{y} \text{ that differ} \\ &= \#\{i : 1 \le i \le n,\ x_i \ne y_i\}. \end{aligned}$$

If a codeword $\boldsymbol{x} \rightsquigarrow \boldsymbol{y}$ by some random errors then

$$d(\boldsymbol{x}, \boldsymbol{y}) = \text{ number of errors in } \boldsymbol{y}.$$

(Note that $d(\boldsymbol{x}, \boldsymbol{y})$ is a true metric on $(\mathbb{Z}_2)^n$ in the usual analysis sense.)

The **weight** of an n-bit word $\boldsymbol{x}$ is defined to be

$$w(\boldsymbol{x}) = d(\boldsymbol{x}, \boldsymbol{0}) = \text{ number of ones in } \boldsymbol{x}.$$

Now, given some code with set of codewords C, we define the **(minimum) weight** of C to be

$$w = w(C) = \min\{w(\boldsymbol{x}) : \boldsymbol{x} \ne \boldsymbol{0},\ \boldsymbol{x} \in C\}$$

Similarly, the **(minimum) distance of** C is defined by

$$d = d(C) = \min\{d(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \ne \boldsymbol{y},\ \boldsymbol{x}, \boldsymbol{y} \in C\}.$$

We also define the **weight numbers** A_i for a code C by

$$A_i = \text{ number of codewords of weight } i \text{ in } C.$$

For example, in the Hamming (7,4) code, some codewords are

$$\begin{array}{ll} \boldsymbol{v} = 1111111 & \qquad \boldsymbol{y} = 1010101 \\ \boldsymbol{x} = 1110000 & \qquad \boldsymbol{z} = 0000000 \end{array}$$

for which $d(\boldsymbol{v},\boldsymbol{x}) = 4,\ d(\boldsymbol{x},\boldsymbol{y}) = 3,\ w(\boldsymbol{v}) = 7,\ w(\boldsymbol{x}) = 3$.

By listing all 16 codewords and exhaustively checking (do it as an exercise) we find that

$$w = 3,\ d = 3 \qquad \text{and}$$

$$A_0 = 1,\ A_3 = 7,\ A_4 = 7,\ A_7 = 1 \quad \text{with all other} \quad A_i = 0.$$

Note: For many codes, $d = w$. In particular, if $\mathbf{0} \in C$, then certainly $d \leq w$. (Think about why.)

2.12 Connection between d and error correction

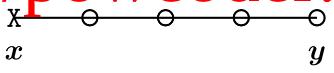
Suppose we have a code C of minimum distance d. If a codeword $\boldsymbol{x} \rightsquigarrow \boldsymbol{y}$ after noise then

$$d(\boldsymbol{x},\boldsymbol{y}) = \text{ number of errors in } \boldsymbol{y}.$$

We will draw a picture of this as follows:

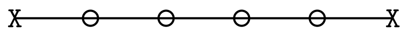
We show codewords by `X` and show other words by O

For example, if $d(\boldsymbol{x},\boldsymbol{y}) = 4$ and $\boldsymbol{x}$ is a codeword, then we have



We are using this simplified picture, since in reality, to get from $\boldsymbol{x}$ to $\boldsymbol{y}$ you have moved 4 steps of length 1 along lines parallel to the axes in n-dimensional space — this is too hard to visualize.

If the code has distance d, then somewhere there are 2 codewords at distance exactly d apart. For example, if $d = 5$ then we have a picture for the minimum distance of



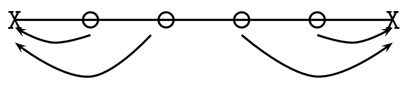
Now we develop a strategy for error correction called **minimum distance decoding** as follows:

If $\boldsymbol{y} \notin C$ then take the *closest* codeword ($\boldsymbol{x}$) to $\boldsymbol{y}$ as the codeword that $\boldsymbol{y}$ is corrected/decoded to, so we decode $\boldsymbol{y} \rightarrow \boldsymbol{x}$. In other words we are saying that this codeword $\boldsymbol{x}$ is the most likely codeword for which $\boldsymbol{x} \rightsquigarrow \boldsymbol{y}$.

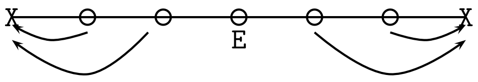
If there is more than one codeword at the same distance then flag an unspecified error (shown by `E`) of this size.

The value of the minimum distance d determines the maximum number of errors that can be *guaranteed* to be corrected.

In our example with $d = 5$ it is clear that 2 errors can be corrected. We will picture this as follows, with the arrows showing to which codeword a non-codeword is decoded using minimum distance decoding.



Similarly, when $d = 6$



and we see that 2 errors can be corrected but a triple error can only be detected, as it is an equal distance from 2 codewords.
In general it is clear that for a code C:

If $d = 2t + 1$ then C is a t-error correcting code, and
if $d = 2t + 2$ then C is a t-error correcting / $(t + 1)$-error detecting code.

Looking back at our earlier examples, we see that

Hamming codes	have	$d = 3$	(proof later)
8-bit ASCII	has	$d = 2$	(check)
code C_1	has	$d = 4$	(tedious checking)
code C_2	has	$d = 2$	(obvious)

We also notice that the relationship between d and t is given by

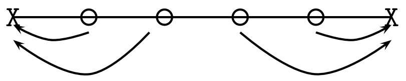
$$t = \left\lfloor \frac{d-1}{2} \right\rfloor \quad \text{(where } \lfloor\ \rfloor \text{ is the floor function).}$$

Note that minimum distance decoding may at times be able to correct more than t errors – there may be places where codewords are further than d apart, for example when the codewords are not evenly spread around in $\mathbb{Z}_2^n$ space.

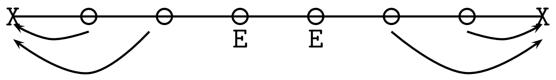
When we consider probabilities we will adopt a slightly different strategy which is easier to analyse. We will call it our **standard strategy.**

If $d = 2t + 1$ or $2t + 2$ then the standard strategy is the following:
"if $\boldsymbol{y}$ is distance $\leq t$ from a codeword $\boldsymbol{x}$ then correct $\boldsymbol{y}$ to $\boldsymbol{x}$,
otherwise simply flag an error."

For instance, in a code with $d = 5$ we can apply the standard strategy as follows. If two codewords are at distance $d = 5$ apart then the standard strategy looks just the same as the minimum distance decoding strategy:



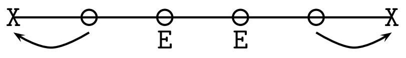
However, in a code with $d = 5$ we can usually find pairs of codewords which have a greater Hamming distance than d. For example, here is how the standard strategy applies to two codewords which are at distance 7 apart, in a code with minimum distance $d = 5$.



We see that two unspecified errors are flagged, whereas in the minimum distance decoding strategy there would have been no errors flagged.

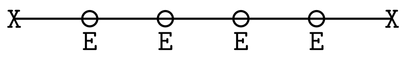
Other strategies than our standard strategy may be adopted just as with the repetition codes. For example, when $d = 5$ we have

Strategy 2:



Correct 1 error and (simultaneously) detect 2 or 3 errors (sometimes more)

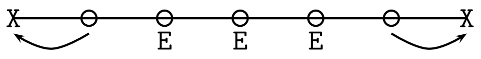
Strategy 3:



Detect up to 4 errors (sometimes more) and do not try to correct

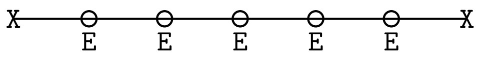
When $d = 6$ we have

Strategy 2:



Correct 1 error and simultaneously detect up to 4 errors (sometimes more)

Strategy 3:



Detect up to 5 errors (sometimes more) and do not try to correct

In general it can be shown that if

$$e + f = d - 1 \quad \text{and} \quad f \geq e$$

then there exists a strategy which simultaneously corrects up to e errors and detects up to f errors. Sometimes this strategy will detect more errors, when the closest codewords are further than d apart.

Finally, the strategy "if y is not a codeword then flag an error" will be called the **pure error detection strategy.**

2.13 Sphere packing.

Let $\rho \geq 0$ be given. In normal 3-dimensional geometry we call the set of points of distance at most ρ from a given point the sphere of radius ρ around the point. (Perhaps more correctly we should call it the sphere and its interior.)

Similarly, we define for a fixed codeword $\boldsymbol{c}$, the **sphere of radius ρ around $\boldsymbol{c}$** as the set of points $\{\boldsymbol{x} \in \mathbb{Z}_2^n : d(\boldsymbol{x}, \boldsymbol{c}) \leq \rho\}$. Here we usually take ρ to be a nonnegative integer.

By the **volume of a sphere** in this context we mean the **number of points** in $\mathbb{Z}_2^n$ that lie in the sphere.

Remember, a point in $\mathbb{Z}_2^n$ is at distance ρ from $\boldsymbol{c}$ if and only if it differs from $\boldsymbol{c}$ in ρ of the n places. But since the code is binary, there is only one way to change the value at a given position. Therefore the number of points at distance ρ from c is the number of choices of ρ positions out of n, which is $\binom{n}{\rho}$. In particular,

$$\begin{array}{lclcl}
\text{number of points at distance} & 0 & \text{from } \boldsymbol{c} & = & 1 \\
\text{number of points at distance} & 1 & \text{from } \boldsymbol{c} & = & \binom{n}{1} \\
\text{number of points at distance} & 2 & \text{from } \boldsymbol{c} & = & \binom{n}{2} \\
\text{number of points at distance} & 3 & \text{from } \boldsymbol{c} & = & \binom{n}{3} \\
 & & \text{and so on.} & &
\end{array}$$

Hence the volume of the sphere of radius ρ is the number of points at distance at most ρ from the centre, which equals

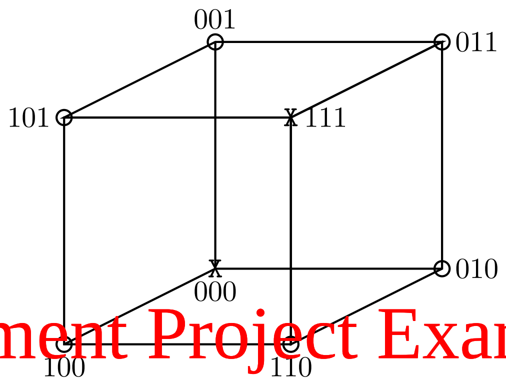
$$1 + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{\rho}.$$

Now we wish to draw a sphere of radius ρ around **every** codeword. How can we picture these spheres?

1. $\mathbb{Z}_2^3$ is the only case where we can view it in true perspective using the usual 3-dimensional axes.

   We show the triple repetition code $C = \{000,\ 111\}$ with codewords as usual shown by X and other words by o

   $\mathbb{Z}_2^3$ is the cube



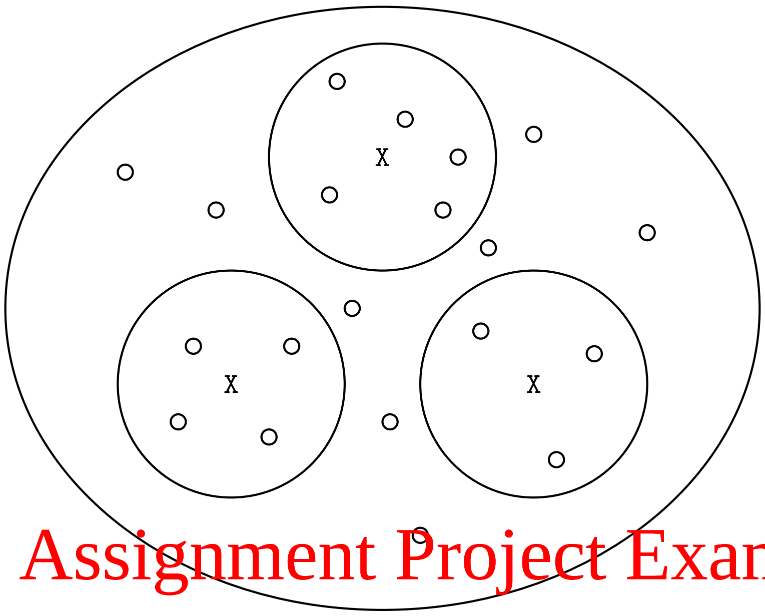
The sphere	of radius	1	around	000	=	$\{000,\ 100,\ 010,\ 001\}$
”	”	1	”	111	=	$\{111,\ 011,\ 101,\ 110\}$
”	”	2	”	000	=	$\{000,\ 100,\ 010,\ 001,\ 011,\ 101,\ 110\}$
”	”	2	”	111	=	$\{111,\ 011,\ 101,\ 110,\ 100,\ 010,\ 001\}$
”	”	3	”	000	=	all the points
”	”	3	”	111	=	”

Now notice that the spheres of radius 2 at the codewords overlap but the spheres of radius 1 do not overlap.

If *none* of the spheres of radius ρ placed around *all* of the codewords overlap then this is called a **sphere packing.**

In this example, we have a sphere packing with radius 1 but not with any larger radius.

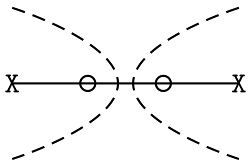
2. More generally in $\mathbb{Z}_2^n$ we might picture a sphere packing as something like the following, where again the codeword points are shown X
   and non-codeword points shown ○



Here we have not tried to show which points are at distance 1 from each other, just the points inside the spheres.

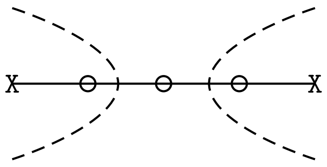
A natural question is to ask how *large* a radius we can have and still get a sphere packing. The answer is to look at the closest codewords to one another, that is, the codewords at the minimum distance d. For example,

if $d = 3$



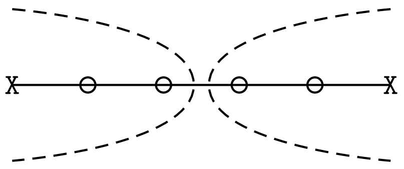
then largest radius of a sphere packing is 1, and

if $d = 4$



then again the largest radius is 1 since radius 2 overlaps at the middle.

When $d = 5$



the largest radius is 2.

It is clear that the largest sphere packing radius is the same as the (maximum) error correcting capability.

Now suppose that we have a t-error correcting code C of length n. We will count all the points in all the spheres of radius t around all the codewords.

There are $|C|$ codewords, each surrounded by a sphere of radius t containing

$$1 + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t} \text{ points}$$

and none of them overlap as it is a sphere packing.

This accounts for $|C| \times \sum_{i=0}^{t} \binom{n}{i}$ points.

But there are 2^n points in total in $\mathbb{Z}_2^n$, so we conclude that

$$2^n \geq |C| \times \sum_{i=0}^{t} \binom{n}{i}.$$

We have proved the following.

Theorem 2.2 : Sphere-Packing Condition Theorem (binary case)

A t-error correcting binary code C of length n has minimum distance $d = 2t + 1$ or $2t + 2$ and the number of codewords satisfies

$$|C| \leq \frac{2^n}{\sum_{i=0}^{t} \binom{n}{i}}.$$

Note that with fixed n, as t increases the denominator in this inequality increases, so the more errors we want to correct the fewer codewords we have, and conversely.

Examples

1. Any 1-error correcting code must have

$$|C| \leq \frac{2^n}{1+n}.$$

So, for example, consider Hamming-type codes with k information bits and 2^k codewords. We have

$$2^k \leq \frac{2^n}{1+n}.$$

But $n = k + m$, so

$$2^k \leq \frac{2^{k+m}}{1+n}$$

or, after rearranging,

$$2^m \geq 1 + n$$

as we had before in the earlier section.

2. Any 2-error correcting code must have

$$|C| \leq \frac{2^n}{1 + n + \frac{1}{2}n(n-1)}.$$

So for any systematic 2-error correcting code with $n = k + m$ and with 2^k codewords we obtain

$$2^m \geq 1 + n + \frac{1}{2}n(n-1).$$

2.14 Perfect codes

Notice with the Hamming (n, k) codes that

$$2^m = 1 + n \quad \text{rather than just } 2^m \geq 1 + n.$$

This means that there are *no* points *outside* the spheres centred at codewords.

In general if we have equality in the sphere-packing bound; that is, if

$$|C| = \frac{2^n}{\sum_{i=0}^{t} \binom{n}{i}},$$

then we say that the code is **perfect.**

In some sense this means that the codewords are evenly spread around in $\mathbb{Z}_2^n$ space.

Note the following facts:

Perfect codes are quite rare. Clearly codes with one codeword or all possible words are perfect, but these are trivial and we will ignore them.

For binary codes the following can be shown:

For $t = 1$ the Hamming codes (and some others with the same parameters) are the only perfect codes.

For $t = 3$ the Golay code is perfect. (See Chapter 6 for this code.)

For any t the $(2t + 1)$-repetition code is perfect.

There are *no other* perfect binary codes.

2.15 Binary Linear codes

We can generalize some of the ideas of Hamming codes to get the theory of binary linear codes.

But firstly, let us note that all the algebraic properties that we know about matrices, vectors and linear algebra remain true when the matrices have entries in $\mathbb{F}$ where $\mathbb{F}$ is any field, in particular in $\mathbb{Z}_2$. So we can perform row reduction to find the row echelon form of a matrix over $\mathbb{Z}_2$, and we can find null spaces of matrices over $\mathbb{Z}_2$. (We will not consider any geometric properties such as orthogonality and projection over finite fields.)

Let H be any $m \times n$ binary matrix of rank m over $\mathbb{Z}_2$. Then H is a **parity check matrix** for the code given by

$$C = \{\boldsymbol{x} \in \mathbb{Z}_2^n : H\boldsymbol{x}^T = \mathbf{0}\}$$

and such a code C is called a **(binary) linear code**, since the sum of any two codewords is a codeword. Since H has rank m it follows that $n \geq m$.

The m rows of H correspond to m parity check equations.

Note: C is a vector space over $\mathbb{Z}_2$ of dimension $k = n - m$. In fact C is the null space of H.

Recall that H can be further reduced to **reduced row echelon form (RREF)**, where all leading entries are 1 and all other entries in leading columns are zero. This echelon form has the same null-space as H, but contains an identity matrix formed by the leading (pivot) columns. Hence we can make the leading columns correspond to check bits, just as we did with Hamming-type codes. Therefore linear codes are also systematic.

Theorem 2.3 : Minimum Distance for Linear Codes

The minimum distance d of a linear code with parity check matrix H is the smallest integer r for which there are r linearly dependent columns of H.

For linear codes $w = d$.

Proof: Omitted (see Problem 19).

Hence we can calculate the minimum distance d for linear codes without ploughing through long lists of codewords and their distances. This is particularly easy if $d = 3$.

Example: The matrix

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

is rank 3, so is a parity check matrix for some code C.

All pairs of columns are linearly independent, however column 2 plus column 3 = column 4 so there exists a triple of dependent columns.

Hence $d = 3$ and this is a 1-error correcting code.

Now reducing to reduced row echelon form (using row operations over $\mathbb{Z}_2$) we get

$$H \xrightarrow{\text{row ops}} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix},$$

so the first 3 bits are check bits and the last 2 bits are information bits. The code is

$$\begin{aligned} C &= \{00000, 11101, 01110, 10011\} \\ &= \text{span}\{11101, 01110\} \end{aligned}$$

and we can easily confirm that $d = 3$.

Example: Consider the Hamming-type codes as described in Section 2.9. Assume that $n \geq 3$ and $m \geq 2$. The check matrix H has no zero column and no column is repeated, so all pairs of columns are linearly independent. However, the sum of the first three columns of H is zero. So there exists a triple of dependent columns.

Hence $d = 3$ and all Hamming-type codes are 1-error correcting.

Now suppose that C is a linear code with parity check matrix H and distance $d = 2t + 1$ or $2t + 2$. Let $\boldsymbol{x}$ be a codeword of C suppose that $\boldsymbol{x} \rightsquigarrow \boldsymbol{y}$ with at most t errors.

Take the **syndrome** of $\boldsymbol{y}$ to be $S(\boldsymbol{y}) = H\boldsymbol{y}^T$. This can then be used to correct the errors as follows.

We must have

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e}_{i_1}^T + \cdots + \boldsymbol{e}_{i_u}^T \quad \text{where} \quad u \leq t$$

and each $\boldsymbol{e}_i$ is a unit (column) vector with a '1' in the ith place (like $00\cdots010\cdots0$) corresponding to a single error.

Then

$$\begin{aligned} H\boldsymbol{y}^T &= H(\boldsymbol{x}^T + \boldsymbol{e}_{i_1} + \cdots + \boldsymbol{e}_{i_u}) \\ &= H\boldsymbol{x}^T + H\boldsymbol{e}_{i_1} + \cdots + H\boldsymbol{e}_{i_u} \\ &= \boldsymbol{0} + H\boldsymbol{e}_{i_1} + \cdots + H\boldsymbol{e}_{i_u} \end{aligned}$$

Now recall that $H\boldsymbol{e}_i = i$th column of H, so this means that $H\mathbf{y}^T$ is the sum of u distinct columns of H.

Since $u \leq t$ it follows that $2u \leq 2t \leq d - 1$. We know that any $d - 1$ columns of H are independent. Therefore, no sum of $\leq t$ columns can equal any other sum of $\leq t$ columns.

Hence from $H\boldsymbol{y}^T$ it is possible (at least in theory) to determine both the number u and the particular u columns that sum to $H\boldsymbol{y}^T$. This means we can locate the u errors and hence correct them.

This is particularly nice with *single* errors as you need only to match a single column (as was the case with Hamming codes). For $t \geq 2$ it is more difficult to find the right linear combination of columns.

Examples

1. For the previous code with

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix},$$

if the codeword $\boldsymbol{x} = 11101 \rightsquigarrow \boldsymbol{y} = 10101$ then

$$H\boldsymbol{y}^T = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

which is the second column so the error is in the second place.

2.

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Here by extensive checking we find that no 4 columns are dependent but some 5 columns are dependent (in fact the last 5 columns sum to $\mathbf{0}$).

Hence $d = 5$.

So this is the parity check matrix of a 2-error correcting code with $m = 6,\ n = 8,\ k = 2$.

(Exercise: check that this code satisfies the Sphere Packing Bound)

There are only $2^k = 2^2 = 4$ codewords and they form the null-space of H, namely

$$\{00000000,\ 11110001,\ 11101110,\ 00011111\}$$

and again we can easily confirm that $d = 5$.

(Note that this is about the smallest possible non-trivial 2-error correcting code.)

Now, suppose that the codeword $\boldsymbol{x} = 11110001 \rightsquigarrow \boldsymbol{y} = 01110011$ (with 2 errors in the 1st and 7th places).

Let us find the errors in $\boldsymbol{y}$.

Calculating $H\boldsymbol{y}^T$ and expressing it as a sum of columns of H, we get

$$H\boldsymbol{y}^T = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \text{column } 1 \ + \ \text{column } 7$$

and it cannot be the sum of any other ≤ 2 columns of H, so the errors are located.

2.15.1 Coset decoding

(This section is extra material for background interest).

For general linear codes, one of the major problems we face is an efficient method of error detection and decoding, assuming a nearest member strategy. We have already seen how this is done for Hamming-type codes. But how in practice do we implement the nearest member strategy for other linear codes?

The method usually used is **coset decoding**.

Let V be a vector space over field $\mathbb{F}$, W a subspace of V and $\mathbf{v}$ a vector not in W. Then the **coset** $\mathbf{v} + W$ is defined to be the subset

$$\mathbf{v} + W = \{\mathbf{v} + \mathbf{x} \mid \mathbf{x} \in W\}$$

In $\mathbb{R}^3$, for example, if W is a 1-dimensional subspace it is a line through the origin, and the coset $\mathbf{v} + W$ is then the line parallel to W through $\mathbf{v}$.

Clearly, vector $\mathbf{u}$ and $\mathbf{v}$ are in the same coset if and only if $\mathbf{u} - \mathbf{v} \in W$. If W is a linear code, $\mathbf{u}$ and $\mathbf{v}$ in the same coset is equivalent to saying $\mathbf{u} - \mathbf{v}$ is a codeword.

It is a straightforward exercise in linear algebra to prove that the set of cosets of any (non-trivial) subspace W will partition the vector space V: every vector is in one and only one coset and two cosets are either disjoint or coincide.

If V is a vector space over a finite field $\mathbb{F}$, and $\mathbf{v} + W$ a coset of W then we define the **coset leader** to be any member of $\mathbf{v} + W$ of least Hamming weight. (If there is a choice the coset leader is chosen at random.) For example, the coset leader of W is the zero vector.

Now suppose $\mathcal{C}$ is a linear (n, k)-code over $\mathbb{F} = \mathbb{Z}_p$ for prime p, and so is a subspace of $\mathbb{F}^n$. If a codeword $\mathbf{x} \in \mathcal{C}$ is transmitted and word $\mathbf{y}$ is received, then $\mathbf{y}$ will lie in some coset of $\mathcal{C}$.

The nearest member strategy then involves assuming that the error in $\mathbf{y}$ is the coset leader of the coset that contains $\mathbf{y}$, and subtracting off this error gives the corrected transmitted word $\mathbf{x}$.

In practice (at least for small codes), the whole vector space $\mathbb{F}^n$ can be listed out in a table whose rows are cosets and the received word $\mathbf{y}$ looked up in the table.

For larger codes, we instead create a list of syndromes and match the syndromes up with the coset leaders. This works because two vector are in the same coset if and only if they have the same syndrome, since their difference is a codeword. This approach is of course what we use for Hamming codes too, but there the syndrome we calculate actually gives us the error position directly.

Coset decoding becomes less and less practical the larger the codes get. If we want to use a large linear code we need to create one with a simple decoding strategy: we will see one such in chapter 6.

2.16 Generator Matrices

So far, we have only considered *binary* linear codes C, and defined them as solutions to the equation $H\boldsymbol{x}^T = \mathbf{0}$ for some binary matrix H, a parity check matrix for C. In particular, a binary linear code is a *vector space* over the binary field $\mathbb{Z}^2$.

More generally, a *linear code* C is a vector space over some (usually finite) field $\mathbb{F}$. As such, it can be represented in two main ways. The first way is as we have already seen: C is the set of solutions to the equation $H\boldsymbol{x}^T = \mathbf{0}$ for some matrix H over $\mathbb{F}$, a parity check matrix for C. The second way is to find a basis B for C and to let G be a matrix whose rows are the vectors of B. Then G is a **generator matrix** for C and generates C in the sense that C is the set of linear combinations of G:

$$C = \{\mathbf{m}G \,:\, \mathbf{m} \in \mathbb{F}^k\}$$

where k is the dimension of C.

Example: Consider the binary linear code

$$C = \{00000, 11101, 01110, 10011\}.$$

One of its parity check matrices is

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

and one of its generator matrices is

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Note that if C has length n and dimension k, then the number of columns of G and H is each n, the number of rows of G is k, and the the number of rows of H is k.

2.17 Standard Form Matrices

Now consider any parity check matrix H with m independent rows and n columns.

By row reduction it can be further reduced to reduced row echelon form (RREF), say H'.

Now we know that

$$H\boldsymbol{x}^T = \mathbf{0} \quad \text{if and only if} \quad H'\boldsymbol{x}^T = \mathbf{0}$$

so H and H' are parity check matrices of the same code C.

Now permute the columns of H' so that the identity matrix is at the beginning,

$$\text{say} \quad H'' = \left(I_m \middle| B\right)$$

where B is $m \times k$ (where $k = n - m$) and I_m is the $m \times m$ identity matrix. Then H'' is the matrix of a different code obtained from the original code by permuting the entries in the same way as the columns were permuted.

We say that such a parity check matrix is one in **standard form** and its code is an **equivalent** code to the original.

Note: Most books put the I_m on the right as $\left(B \middle| I_m\right)$, but for us it is more natural to put it on the left. Some other books also have the transpose of all this.

Now suppose that

$$H = \left(I_m \middle| B\right)$$

is a standard form check matrix. Then it is very easy to find a basis for the set of codewords C, which is of course the set of solutions to $H\boldsymbol{x}^T = \mathbf{0}$ (that is, the null-space of H).

Example: The matrix of our Hamming (7,4) was not in RREF, but swapping positions 3 and 4 gives check matrix

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} = \left(I_3 \middle| B\right).$$

To solve $H\boldsymbol{x}^T = \mathbf{0}$ we need only to use back substitution.

Let

$$\begin{array}{rrcrrrr} & x_7 & = & \lambda_4 & & & \\ & x_6 & = & \lambda_3 & & & \\ & x_5 & = & \lambda_2 & & & \\ & x_4 & = & \lambda_1 & & & \\ \text{then} & x_3 & = & & -\lambda_2 & -\lambda_3 & -\lambda_4 \\ & x_2 & = & -\lambda_1 & & -\lambda_3 & -\lambda_4 \\ & x_1 & = & -\lambda_1 & -\lambda_2 & & -\lambda_4 \end{array}$$

so

$$\boldsymbol{x} = \lambda_1 \begin{pmatrix} -1 \\ -1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \lambda_2 \begin{pmatrix} -1 \\ 0 \\ -1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \lambda_3 \begin{pmatrix} 0 \\ -1 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \lambda_4 \begin{pmatrix} -1 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$= \left(-\frac{B}{I_4} \right) \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{pmatrix} = \left(-\frac{B}{I_4} \right) \boldsymbol{\lambda}^T$$

where

$$\boldsymbol{\lambda} = \lambda_1 \lambda_2 \lambda_3 \lambda_4 = (\lambda_1\ \lambda_2\ \lambda_3\ \lambda_4)$$

can be taken as the information bits and this gives a quick way to find the corresponding codeword $\boldsymbol{x}$ — the information bits are at the end and the check bits at the front.

So both encoding and decoding are easy.

Taking the transpose of this we have

$$\boldsymbol{x}^T = \boldsymbol{\lambda} \left(-B^T \middle| I_4 \right) = \boldsymbol{\lambda} G$$

Now, $\boldsymbol{\lambda} G$ forms a linear combination of the rows of G, and as $\boldsymbol{\lambda}$ varies we get all such combinations. Thus the code C consists of all codewords (as rows) which are linear combinations of the rows of G.

That is, **the rows of G are a basis for the code**, so G is a **generator matrix** of the code, and in this shape is also said to be in **standard form**.

In our example

$$\boldsymbol{x}^T = (x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7) = (\lambda_1\ \lambda_2\ \lambda_3\ \lambda_4)\, G$$

$$\text{where} \quad G = \left(\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array} \right)$$

We have thus proved:

Theorem 2.4 : Standard Form Matrices

A binary linear code C with standard form check matrix

$$H = \left(I_m \middle| B \right)$$

has standard form generator matrix

$$G = \left(-B^T \middle| I_k \right)$$

and the rows of G form a basis for C; that is, the codewords of C are exactly the linear combinations of the rows of G.

In particular, a binary *linear code C with standard form check matrix*

$$H = \left(I_m \middle| B\right)$$

has standard form generator matrix

$$G = \left(B^T \middle| I_k\right)$$

Note: Again, some books have a slightly different format for these matrices.

2.18 Extending linear codes

Let C be a binary code with $m \times n$ parity check matrix H and with minimum distance d. We can sometimes improve the distance to $d+1$ by **extending** H to obtain an **extended code**.

Let

$$\widehat{H} = \left(\begin{array}{c|ccc} 1 & 1 & \cdots & 1 \\ \hline 0 & & & \\ \vdots & & H & \\ 0 & & & \end{array}\right)$$

Here we have put a column of zeros in front of H, then a row of ones on top of the resulting matrix.

All we have in fact done is to add an **overall parity check** bit which is in front of x_1 and so is denoted by x_0.

In other words, if $\boldsymbol{x} = x_1 x_2 \cdots x_n \in C$ then we will write the extended codewords as

$$\hat{\boldsymbol{x}} = x_0 x_1 x_2 \cdots x_n \qquad \text{where } x_0 = \sum_{i=1}^{n} x_i \pmod 2).$$

We will also call the code $\widehat{C}$.

Then

$$\widehat{H}\hat{\boldsymbol{x}}^T = \left(\frac{\sum_{i=0}^{n} x_i}{H\boldsymbol{x}^T}\right) = \left(\frac{0}{\mathbf{0}}\right) \begin{array}{l} - \text{ length } 1 \\ \\ - \text{ length } m \end{array}$$

Claim: the minimum distance $\hat{d}$ of $\widehat{C}$ is either d or $d+1$.

Proof: Clearly $d \le \hat{d}$, since a set of linearly dependent columns of $\widehat{H}$ gives a set of linearly dependent columns of H after the top '1' is deleted. If $d < \hat{d}$ then any set of d linearly dependent columns in H is no longer linearly dependent when an extra 1 is added to the top of each column. So the sum of these d columns gives the vector $(1, 0, 0, \cdots, 0)^T$. Hence adding the new column to these d columns produces the zero vector, so there exists a set of $d+1$ linearly dependent columns in $\widehat{H}$. $\square$

Example: The Hamming (n, k) codes all have $d = 3$ and the extended codes all have $d = 4$, and have size $(n+1, k)$.

Suppose that $\hat{\boldsymbol{x}} = x_0\boldsymbol{x}$ is an extended Hamming code codeword,

$$\text{so} \quad \widehat{H}\hat{\boldsymbol{x}}^T = \left(\begin{array}{c|ccc} 1 & 1 & \cdots & 1 \\ \hline 0 & & & \\ \vdots & & H & \\ 0 & & & \end{array}\right) \left(\frac{x_0}{\boldsymbol{x}}\right) = \left(\frac{0}{\mathbf{0}}\right)$$

where H is the original Hamming parity check matrix.

If $\hat{\boldsymbol{x}} \rightsquigarrow \hat{\boldsymbol{y}}$ with 1 or 2 errors then the syndrome

$$\widehat{S}(\hat{\boldsymbol{y}}) = \widehat{H}\hat{\boldsymbol{y}}^T = \begin{pmatrix} \sum_{i=0}^n y_i \\ H\boldsymbol{y}^T \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^n y_i \\ S(\boldsymbol{y}) \end{pmatrix}$$

allows the following decoding:

$$\begin{array}{llll} \text{if } \sum_{i=0}^n y_i = 0 & \text{and } S(\boldsymbol{y}) = \mathbf{0} & \text{then} & \text{no error} \\ \text{if } \sum_{i=0}^n y_i = 1 & \text{and } S(\boldsymbol{y}) \neq \mathbf{0} & \text{then} & \text{error in location given by } S(\boldsymbol{y}) \\ \text{if } \sum_{i=0}^n y_i = 1 & \text{and } S(\boldsymbol{y}) = \mathbf{0} & \text{then} & \text{error in 0th location} \\ \text{if } \sum_{i=0}^n y_i = 0 & \text{and } S(\boldsymbol{y}) \neq \mathbf{0} & \text{then} & \text{double error but do not know where.} \end{array}$$

Hence $d = 4$ for extended Hamming codes.

2.19 Radix greater than two

We now consider how the theory developed in this chapter so far extends to codes of radix greater than 2.

2.19.1 Hamming codes for radix greater than 2

Let r be a prime number. The radix r Hamming codes are constructed by forming a matrix from the base r representations of the positive integers, then crossing out every column whose first nonzero entry is not equal to 1. This ensures that no remaining column is a scalar multiple of any other column.

For example, consider the matrix

$$\begin{pmatrix} 1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2 \\ 0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2 \\ 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&2&2&2&2&2&2&2&2&2 \end{pmatrix}$$

formed from the base 3 representations of the integers $1, \dots, 26$. (Do you recognise the pattern of entries along each row?)

Now delete any column which does not have a 1 as its first nonzero entry. This gives the matrix

$$H = \begin{pmatrix} 1&0&1&1&0&1&0&1&1&1&0&1&1 \\ 0&1&1&2&0&0&1&1&2&0&1&1&2 \\ 0&0&0&0&1&1&1&1&1&2&2&2&2 \end{pmatrix}$$

with 13 columns. The kernel of H is the ternary Hamming code with $n = 13$, $m = 3$, $k = 10$. The check digits are in positions 1, 2 and 5, corresponding to the leading entries of this matrix.

In general, if you start with the integers $1, \dots, r^m - 1$ and work modulo r then the parity check matrix you get will have m rows and $n = (r^m - 1)/(r - 1)$ columns.

By construction, no two columns of the parity check matrix are linearly dependent but some sets of three columns are. So the radix r Hamming codes have $d = 3$ and are 1-error correcting. We now show how errors can be corrected.

Let $\mathbf{x}$ be a codeword and suppose that $\mathbf{x} \rightsquigarrow \mathbf{y}$ with one error. That is, $\mathbf{y} = \mathbf{x} + \alpha\,\mathbf{e}_j^T$ where $\alpha \in \mathbb{Z}_r$ is the magnitude of the error and $\mathbf{e}_j$ is the jth unit (column) vector. The syndrome is

$$S(\mathbf{y}) = H\mathbf{y}^T = \alpha\,H\mathbf{e}_j = \alpha\,\mathbf{h}_j$$

where $\mathbf{h}_j$ is the jth column of H. Moreover, since the first nonzero entry of $\mathbf{h}_j$ is 1, we can easily find α and correct the error.

Example: Using the ternary Hamming code above, suppose that the received word is $\mathbf{y} = 1111110000222$. Then

$$S(\mathbf{y}) = H\mathbf{y}^T = \begin{pmatrix} 2 \\ 0 \\ 2 \end{pmatrix} = 2\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

So the syndrome equals the 6th column of H multiplied by 2, which implies that $\mathbf{y} = \mathbf{x} + 2\mathbf{e}_6^T$.

We correct the error by subtracting 2 from the 6th position of $\mathbf{y}$ modulo 3 (that is, by adding 1 to the 6th position of $\mathbf{y}$). This gives $\mathbf{x} = 1111120000222$.

Finally we decode by removing the check digits from the 1st, 2nd and 5th positions to give 1120000222.

2.19.2 Linear codes for radix greater than 2

Let $r = p$ where $p \geq 3$ is prime. Then $\mathbb{Z}_p$ is a field (see Section 5.1). Let $\mathbb{Z}_p^n$ denote the set of n-tuples with entries in $\mathbb{Z}_p$, that is, vectors of length n over $\mathbb{Z}_p$.

If H is an $m \times n$ matrix with entries in $\mathbb{Z}_p$ and rank m then H is the check matrix for the radix p code

$$\mathcal{C} = \{\mathbf{x} \in \mathbb{Z}_p^n \mid H\mathbf{x}^T = \mathbf{0}\}.$$

This is a linear code over $\mathbb{Z}_p$.

If $k = n - m$ then $\mathcal{C}$ is a vector space of dimension k over $\mathbb{Z}_p$, and so it has p^k codewords.

Using row-reduction modulo p, we can reduce H to *standard form* $(I_m \mid B)$, where B is an $m \times k$ matrix with entries in $\mathbb{Z}_p$. Then $G = (-B^T \mid I_k)$ is the **standard form generator matrix** for $\mathcal{C}$ (note the minus sign).

As for the binary case, we define the *weight* of a vector $\mathbf{x} \in \mathbb{Z}_p^n$ to be the number of nonzero entries in $\mathbf{x}$, and define $d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} - \mathbf{y})$. We also define the minimum distance $d = d(\mathcal{C})$ and the minimum weight $w = w(\mathcal{C})$ of the code, as before. Then the minimum distance of the code equals the minimum weight of the code, which equals the smallest positive integer ℓ such that there exists ℓ linearly dependent columns of H.

As in the binary case, if $d = 2t + 1$ then the code is t-error correcting and if $d = 2t + 2$ then the code is t-error correcting and $(t+1)$-error detecting.

To decode: suppose that $\mathbf{x} \in \mathcal{C}$ and $\mathbf{y} = \mathbf{x} + \mathbf{e}^T$, where $\mathbf{e}$ is an error vector. Then the syndrome is

$$S(\mathbf{y}) = H\mathbf{y}^T = H\mathbf{e}.$$

If there were s errors in $\mathbf{y}$ then

$$\mathbf{e} = \lambda_{i_1}\mathbf{e}_{i_1} + \cdots + \lambda_{i_s}\mathbf{e}_{i_s}$$

where $\mathbf{e}_j$ is the jth unit (column) vector, and λ_j is the magnitude of the error in position j. If $\mathbf{h}_1, \ldots, \mathbf{h}_n$ are the columns of H then

$$S(\mathbf{y}) = H(\lambda_{i_1}\mathbf{e}_{i_1} + \cdots + \lambda_{i_s}\mathbf{e}_{i_s}) = \lambda_{i_1}\mathbf{h}_{i_1} + \cdots + \lambda_{i_s}\mathbf{h}_{i_s}$$

which is a linear combination of columns of H. If $s \leq t$ where $d = 2t + 1$ or $2t + 2$ then this is the only linear combination of columns of H which can equal $S(\mathbf{y})$. So we can correct $\mathbf{y}$ to

$$\mathbf{x} = \mathbf{y} - (\lambda_{i_1}\mathbf{e}_{i_1}^T + \cdots + \lambda_{i_s}\mathbf{e}_{i_s}^T).$$

2.19.3 Perfect codes with radix greater than two

The Sphere Packing Bound can be suitably modified (see Problem 22), allowing us to define perfect codes for other radices.

All the radix r Hamming codes are perfect (see Problem 22). There is one other known radix 3 perfect code, the ternary Golay code (see Chapter 6).

There are no other known perfect codes with $r \geq 3$. J.H. van Lindt (1971) and A. Tietäväinen and A. Perko (1973) between them proved that there are no others if r is a prime power, but it is not known for sure that there are none in other cases.

Chapter 3

Compression Coding

3.1 Instantaneous and UD codes

Hamming and repetition codes are block codes and were designed to correct and detect errors. That is, they were designed to code for the channel: this is called **channel coding**.

We now look at **variable length codes**. In particular we will study codes where the shorter codewords are used to encode the more likely symbols and longer codewords are used for the less likely symbols. This should allow the *average length* of the code to be *decreased* and lead to an efficient coding or a **compression code**.

Here we are coding the source as efficiently as possible and ignoring the possibility of errors: this is called **source coding**.

As usual, we assume that we have

$$\begin{array}{rrl} \text{a source } S & \text{with } q \text{ source symbols} & s_1, s_2, \ldots, s_q, \\ & \text{with probabilities} & p_1, p_2, \cdots, p_q, \\ \text{encoded by a code } C & \text{with } q \text{ codewords} & \boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_q \\ & \text{of lengths} & \ell_1, \ell_2, \cdots, \ell_q. \end{array}$$

The codewords are radix r (usually binary).

With block codes, where all codewords are the same length, it is easy to know where a codeword starts and ends. However, if the codewords are of variable length then this is a problem.

Morse code solves this by having a pause symbol (as well as dots and dashes) which is used to denote the end of the codeword. Morse code is radix 3, with alphabet $\{\cdot\,,\,-\,,\,\mathtt{p}\}$.

The same trick can be used with binary codes. A **comma code** of length n is a code in which every codeword has length $\leq n$, every codeword contains at most one 0, and if a codeword contains 0 then 0 must be the final symbol in the codeword.
Now it is easy to detect the end of a codeword: stop after n bits or the first '0', whichever comes sooner.

However other codes cause problems as we will see.

Example 3.1 The **standard comma code** of length 5 is

$$\begin{array}{lrcl} s_1 & \boldsymbol{c}_1 & = & 0 \\ s_2 & \boldsymbol{c}_2 & = & 10 \\ s_3 & \boldsymbol{c}_3 & = & 110 \\ s_4 & \boldsymbol{c}_4 & = & 1110 \\ s_5 & \boldsymbol{c}_5 & = & 11110 \\ s_6 & \boldsymbol{c}_6 & = & 11111 \end{array}$$

For example, we can easily decode

$$1100111101101011111110$$

via

$$\underbrace{110}_{s_3}\underbrace{0}_{s_1}\underbrace{11110}_{s_5}\underbrace{110}_{s_3}\underbrace{10}_{s_2}\underbrace{11111}_{s_6}\underbrace{110}_{s_3}$$

as

$$s_3s_1s_5s_3s_2s_6s_3$$

Example 3.2 The code

$$\begin{array}{lrcl} s_1 & \boldsymbol{c}_1 & = & 0 \\ s_2 & \boldsymbol{c}_2 & = & 01 \\ s_3 & \boldsymbol{c}_3 & = & 11 \\ s_4 & \boldsymbol{c}_4 & = & 00 \end{array}$$

is not a sensible code since, for example

$$0011$$

is ambiguous and could mean

$$\text{either} \qquad s_1s_1s_3 \qquad \text{or} \qquad s_4s_3\ .$$

Here we cannot uniquely decode a string of bits.

We say that a code is **uniquely decodable** (written UD) if and only if no two concatenations of codewords are the same. That is, every sequence of source symbols has a different encoded sequence of code symbols.

So our last example is not UD.

Example 3.3 Consider the code

$$\begin{array}{lrcl} s_1 & \boldsymbol{c}_1 & = & 0 \\ s_2 & \boldsymbol{c}_2 & = & 01 \\ s_3 & \boldsymbol{c}_3 & = & 011 \\ s_4 & \boldsymbol{c}_4 & = & 0111 \\ s_5 & \boldsymbol{c}_5 & = & 1111 \end{array}$$

This is UD but it is difficult to decode as it is the *reverse* of a comma code. Decoding cannot begin until the complete message has been received, and then decoding proceeds from the right-hand end working back to left-hand end.

For example

$$01111111011010$$

decodes from the right-hand end as

$$\underbrace{0111}_{s_4}\underbrace{1111}_{s_5}\underbrace{011}_{s_3}\underbrace{01}_{s_2}\underbrace{0}_{s_1}$$

We would rather not have to wait until we have received the entire message before we can start decoding: we would rather be able to decode as we go along, from left to right.

We can only do this if *no* codeword is the *initial part* of another codeword. An initial part of a codeword is also called a **prefix**.

An **instantaneous code** (or **prefix code**) is one in which no codeword is the initial part of any other codeword. We write **I-code** for short.

Comma codes are clearly I-codes, but there are plenty of others as well.

Example 3.4 Here is an I-code

s_1	0
s_2	100
s_3	1011
s_4	110
s_5	111

Note that no codeword is a prefix of any other and for example

$$0011001011111$$

decodes easily as

$$\underbrace{0}_{s_1}\underbrace{0}_{s_1}\underbrace{110}_{s_4}\underbrace{0}_{s_1}\underbrace{1011}_{s_3}\underbrace{111}_{s_5}$$

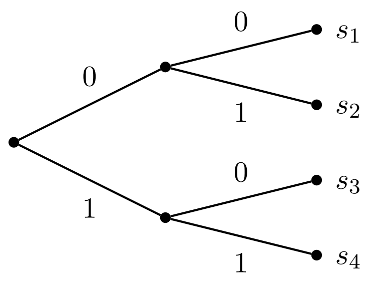
Example 3.5 The code

s_1	00
s_2	01
s_3	10
s_4	11

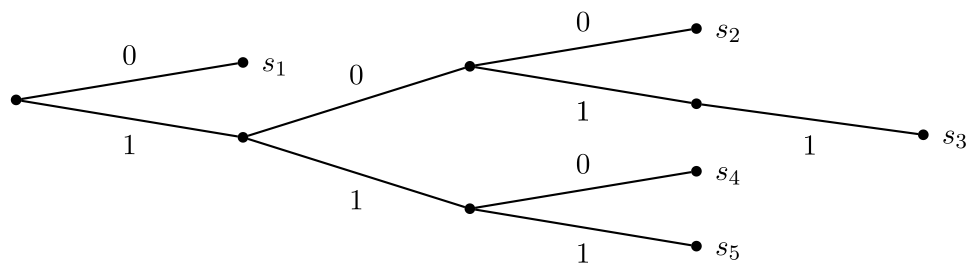
is a block code and is thus an I-code since no codeword can be a prefix of any other codeword.

I-codes can be associated with **decision trees** in which you move left to right taking the branch labelled with the code symbol shown, until you reach one of the leaves which is a codeword. Then restart with the next string of symbols.

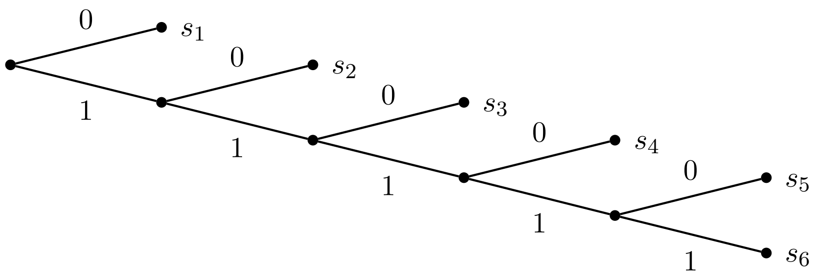
So example 3.5 is



Example 3.4 is

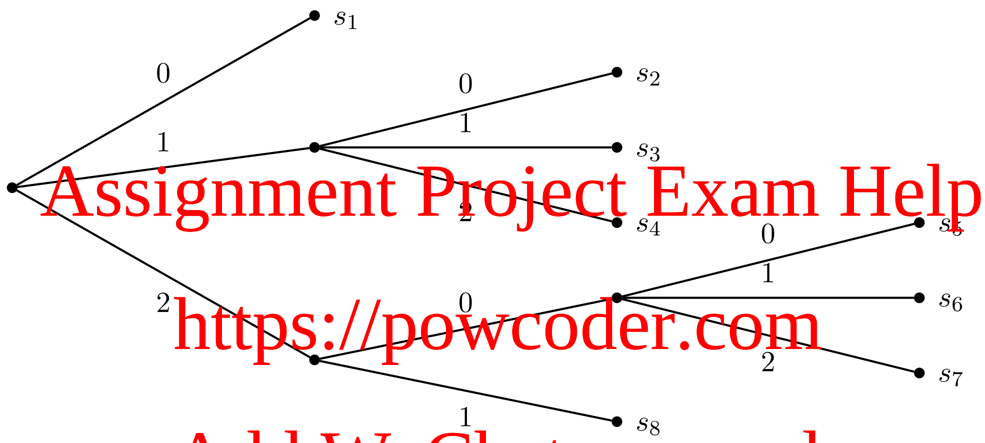


Example 3.1 is



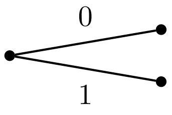
This method works for radix $r > 2$ also.

Example 3.6

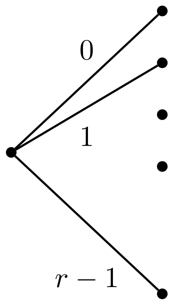


describes a radix 3 code.

Note: our **convention** is for branches to always be numbered down from the top, as shown:



for binary, and



for radix r

Also **note** that example 3.4 can be shortened by taking s_3 as 101, rather than 1011, since no decision was made at the 101 node.

We say two codes are **equivalent** if one can be obtained from the other by permuting the symbols at a given location and then, if need be, permuting symbols at another location, and so on.

For I-codes this means the decision trees are isomorphic as rooted trees.

For example in example 3.6

s_1	0		2
s_2	10		10
s_3	11		11
s_4	12	is equivalent	12
s_5	200	to	001
s_6	201		000
s_7	202		002
s_8	21		01
			$\uparrow\ \uparrow$
			permuted permuted

Also, by rearranging the source symbols if necessary, we can arrange so that the codewords are in non-decreasing length order.

So we also

$$\textbf{assume that} \quad \ell_1 \le \ell_2 \le \cdots \le \ell_q.$$

We have seen that any I-code is a UD-code, and from example 3.3 the converse is false. (That is, there are UD-codes which are not I-codes.)

The next theorem shows that whenever a UD-code exists with a given set of parameters, then there also exists an I-code with the same set of parameters, (i.e. radix, number and lengths of codewords). This means that we can focus on I-codes.

Theorem 3.1 : Kraft-McMillan

A UD-code of radix r with q codewords $\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_q$ of lengths $\ell_1 \le \ell_2 \le \cdots \le \ell_q$ exists
if and only if an I-code with the same parameters exists
if and only if

$$K = \sum_{i=1}^{q} \frac{1}{r^{\ell_i}} \le 1.$$

Proof: This proof is rather long and involved, so can be skipped on the first reading.
The next section gives examples of the constructions used in the proof.
The proof proceeds in three parts — we show

$$\exists \text{ I-code} \quad \Rightarrow \quad \exists \text{ UD-code} \quad \Rightarrow \quad K \le 1 \quad \Rightarrow \quad \exists \text{ I-code}.$$

(1) ($\exists$ *I-code* $\Rightarrow$ $\exists$ *UD-code*)
The first implication follows since any I-code is UD.

(2) ($\exists$ *UD-code* $\Rightarrow$ $K \le 1$)
Suppose we have a UD-code with the given parameters.

$$\text{Let} \qquad K = \sum_{i=1}^{q} \frac{1}{r^{\ell_i}}.$$

$$\text{Consider} \qquad K^n = \left(\sum_{i=1}^{q} \frac{1}{r^{\ell_i}} \right)^n.$$

Expanding the right hand side and collecting terms gives a generalisation of the multinomial theorem

$$K^n = \sum_{j=n\ell_1}^{n\ell_q} \frac{N_j}{r^j}$$

where N_j is the number of ways of writing j as a sum of n numbers selected from the set $\{\ell_1, \ell_2, \cdots, \ell_q\}$, where order matters.

Now N_j is the number of $(e_1, \ldots, e_n)$ such that

$$j = e_1 + \cdots + e_n$$

and $e_i \in \{\ell_1, \ell_2, \ldots, \ell_q\}$ for all i. That is, N_j is the number of ways to concatenate n codewords to give a word of length j.

Since the code is UD, no two different choices of n concatenated codewords gives the same string of length j. Hence

$$N_j \le r^j.$$

Thus

$$K^n \le \sum_{j=n\ell_1}^{n\ell_q} 1 = n(\ell_q - \ell_1) + 1.$$

Now this is true for all n.
However, the left hand side is exponential in n but the right hand side is linear in n.
This means that $K \le 1$, for otherwise, as $n \to \infty$, left hand side exceeds right hand side.
Hence $K \le 1$.

(3) $(K \le 1 \;\Rightarrow\; \exists$ *I-code*$)$

We prove this for the $r = 2$ case only. (The proof of the general case is similar.)

Given the lengths $\ell_1 \le \ell_2 \le \cdots \le \ell_q$, assume that $K = \sum_{i=1}^{q} \frac{1}{2^{\ell_i}} \le 1$.

We show that an I-code exists by constructing it.

Let $\boldsymbol{c}_1 = 00 \cdots 00$ of length ℓ_1,

let $\boldsymbol{c}_2 = 00 \cdots 010 \cdots 0$ of length ℓ_2 with a 1 in position ℓ_1,

and more generally let

let $\boldsymbol{c}_i = c_{i1} c_{i2} \cdots c_{i\ell_i}$ of length ℓ_i,

where $c_{i1}, c_{i2}, \cdots, c_{i\ell_i}$ are such that

$$\sum_{j=1}^{i-1} \frac{1}{2^{\ell_j}} = \frac{c_{i1}}{2} + \frac{c_{i2}}{2^2} + \cdots + \frac{c_{i\ell_i}}{2^{\ell_i}} = \sum_{k=1}^{\ell_i} \frac{c_{ik}}{2^k}$$

i.e. the c_{ik} are the binary digits of the sum of the first $i-1$ terms of $\sum \frac{1}{2^{\ell_j}}$.
Note that such $c_{i1}, \cdots, c_{i\ell_i}$ exist since $\sum \frac{1}{2^{\ell_j}} \le 1$.

We have finished if we can show NO $\boldsymbol{c}_u$ is a prefix of any $\boldsymbol{c}_v$.
We prove this by contradiction.

Assume that $\boldsymbol{c}_u$ is a prefix to $\boldsymbol{c}_v$ where $u < v$. Then $\ell_u < \ell_v$, so

$$\sum_{j=1}^{v-1} \frac{1}{2^{\ell_j}} - \sum_{j=1}^{u-1} \frac{1}{2^{\ell_j}} = \sum_{j=u}^{v-1} \frac{1}{2^{\ell_j}}$$

$$\ge \frac{1}{2^{\ell_u}} \quad \text{(since terms are all positive).}$$

However the left hand side also equals

$$
\begin{aligned}
&\left(\frac{c_{v1}}{2}+\cdots+\frac{c_{v\ell_v}}{2^{\ell_v}}\right)-\left(\frac{c_{u1}}{2}+\cdots+\frac{c_{u\ell_u}}{2^{\ell_u}}\right)\\
&= \frac{c_{v,(\ell_u+1)}}{2^{\ell_u+1}}+\cdots+\frac{c_{v\ell_v}}{2^{\ell_v}}\\
&\leq \frac{1}{2^{\ell_u+1}}+\cdots+\frac{1}{2^{\ell_v}}\\
&< \frac{1}{2^{\ell_u+1}}+\frac{1}{2^{\ell_u+2}}+\cdots\\
&= \frac{1}{2^{\ell_u}} \quad \text{(sum of a geometric progression).}
\end{aligned}
$$

This is a contradiction and the proof is finished. □

3.2 Illustration of parts of the proof

Example for (2) ($\exists$ *UD-code* $\Rightarrow$ $K \leq 1$):

Suppose $n = 5$, $\ell_1 = 1$, $\ell_2 = 3$, $\ell_3 = 4$, then

$$
\begin{aligned}
K^5 &= \left(\frac{1}{r}+\frac{1}{r^3}+\frac{1}{r^4}\right)^5 \quad \text{then expand out giving}\\
&= \frac{1}{r^5}+\frac{5}{r^7}+\frac{5}{r^8}+\frac{10}{r^9}+\frac{20}{r^{10}}+\frac{20}{r^{11}}+\frac{30}{r^{12}}+\frac{35}{r^{13}}\\
&\quad+\frac{30}{r^{14}}+\frac{31}{r^{15}}+\frac{25}{r^{16}}+\frac{15}{r^{17}}+\frac{10}{r^{18}}+\frac{5}{r^{19}}+\frac{1}{r^{20}}
\end{aligned}
$$

where the coefficient N_j of $1/r^j$ is the number of ways of writing j as a sum of 5 numbers selected from $\{1,3,4\} = \{\ell_1,\ell_2,\ell_3\}$.

Some of these coefficients are:

$\frac{1}{r^5}: j = 5,\ N_j = 1$ as $5 = 1+1+1+1+1$ in 1 way

$\frac{1}{r^6}: j = 6,\ N_j = 0$ as 6 is not a sum of 5 terms from $\{1,3,4\}$

$\frac{1}{r^{10}}: j = 10,\ N_j = 20$ as $10 = 1+1+1+3+4$ in $\frac{5!}{3!1!1!} = 20$ ways

$\frac{1}{r^{15}}: j = 15,\ N_j = 31$ as $15 = 3+3+3+3+3$ in 1 way

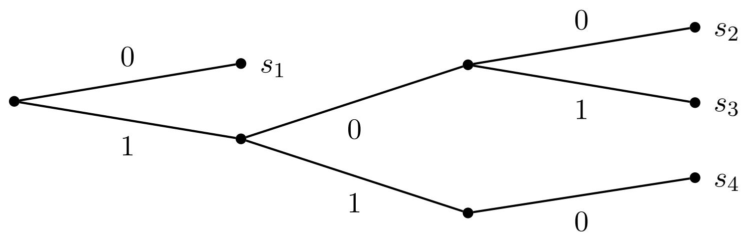
and $15 = 1+3+3+4+4$ in $\frac{5!}{1!2!2!} = 30$ ways.

Example for (3) ($K \leq 1 \Rightarrow \exists$ *I-code*): Consider $\ell_1 = 1$, $\ell_2 = 3$, $\ell_3 = 3$, $\ell_4 = 3$. Then $K = \frac{7}{8} < 1$ and we show that the code exists by constructing it:

$$
\begin{aligned}
\boldsymbol{c}_1 &= 0\\
\boldsymbol{c}_2 &= 100\\
\sum_{j=1}^{2}\frac{1}{2^{\ell_j}} &= \frac{1}{2}+\frac{1}{2^3} = \frac{1}{2}+\frac{0}{2^2}+\frac{1}{2^3} \qquad \text{so} \quad \boldsymbol{c}_3 = 101\\
\sum_{j=1}^{3}\frac{1}{2^{\ell_j}} &= \frac{1}{2}+\frac{1}{2^3}+\frac{1}{2^3} = \frac{1}{2}+\frac{1}{2^2}+\frac{0}{2^3} \qquad \text{so} \quad \boldsymbol{c}_4 = 110
\end{aligned}
$$

But we do not have to follow the method of construction used in the proof. Instead, we can simply work systematically from the shortest to longest length, constructing a decision tree from the top to bottom using the highest-most possible branches as we go. We call this our **standard** I-code for the given lengths.

Example 3.7 For lengths 1, 3, 3, 3 we construct the decision tree



Hence the standard I-code for the lengths 1, 3, 3, 3 is

$$\boldsymbol{c}_1 = 0, \quad \boldsymbol{c}_2 = 100, \quad \boldsymbol{c}_3 = 101, \quad \boldsymbol{c}_4 = 110.$$

3.3 Examples for radix 3

Example 3.8 Construct, if possible, a radix 3 I-code having the following lengths.

(a) 1, 2, 2, 2, 2, 2, 3, 3, 3, 3

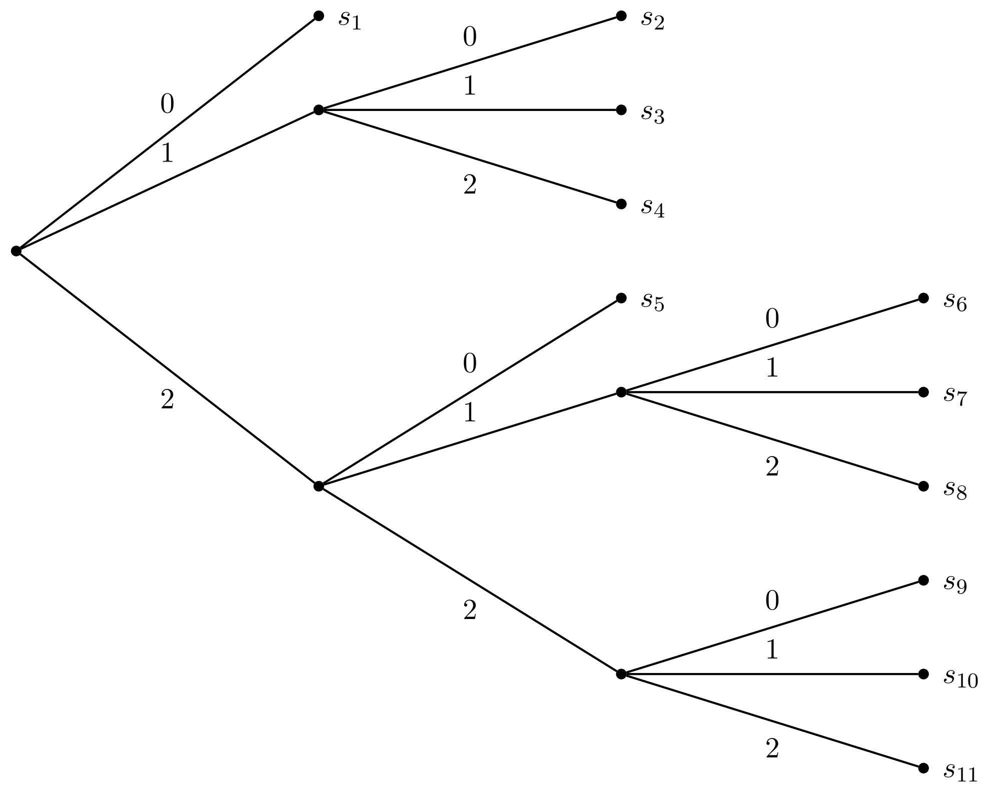
$$\text{Here} \quad K = \frac{1}{3} + 5 \times \frac{1}{9} + 4 \times \frac{1}{27} = \frac{28}{27} > 1$$

so there is no radix 3 I-code with these lengths.

(b) 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3

$$\text{Here} \quad K = \frac{1}{3} + 4 \times \frac{1}{9} + 6 \times \frac{1}{27} = 1$$

so such a code exists. We construct the decision tree



to produce the **standard** I-code

$$\begin{aligned}&\boldsymbol{c}_1 = 0, \quad \boldsymbol{c}_2 = 10, \quad \boldsymbol{c}_3 = 11, \quad \boldsymbol{c}_4 = 12, \quad \boldsymbol{c}_5 = 20, \quad c_6 = 210,\\ &\boldsymbol{c}_7 = 211, \quad \boldsymbol{c}_8 = 212, \quad \boldsymbol{c}_9 = 220, \quad \boldsymbol{c}_{10} = 221, \quad \boldsymbol{c}_{11} = 222.\end{aligned}$$

NOTE: For binary codes, $K < 1$ if and only if some branch can be shortened as a decision is not necessary at some point. (This is NOT true for non-binary codes, e.g. with $r = 3$, lengths 1, 2, 2, 2.)

So shortest binary I-codes codes always have $K = 1$.

3.4 Minimal UD-codes (binary case)

Now introduce probabilities and assume that the source symbols are labelled in non-decreasing probability order: that is, we have

$$\begin{array}{rrl} \text{a source } S & \text{with } q \text{ source symbols} & s_1, s_2, \ldots, s_q, \\ & \text{with probabilities} & p_1 \ge p_2 \ge \cdots \ge p_q, \\ \text{encoded by a code } C & \text{with } q \text{ codewords} & \boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_q \\ & \text{of lengths} & \ell_1, \ell_2, \cdots, \ell_q. \end{array}$$

(We no longer assume that these lengths $\ell_1, \ldots, \ell_q$ are in any particular order.)

The codewords are radix r (usually binary).

The expected or **average length** of codewords is given by

$$L = \sum_{i=1}^{q} p_i \ell_i$$

and the **variance** is given by

$$V = \sum_{i=1}^{q} p_i \ell_i^2 - L^2.$$

Our aim is to minimise L for a given S and, if more than one code C gives this value, to minimise V.

Theorem 3.2 : Minimal binary UD-code conditions

If a binary UD-code has minimal expected length L for the given source S then, after permuting codewords of equally likely symbols if necessary,

1. $\ell_1 \le \ell_2 \le \cdots \le \ell_q$,
2. *we can take the code to be instantaneous,*
3. $K = \sum_{i=1}^{q} 2^{-\ell_i} = 1$,
4. $\ell_{q-1} = \ell_q$,
5. $\boldsymbol{c}_{q-1}$ *and* $\boldsymbol{c}_q$ *differ only in their last place.*

Sketch proof:

1. If $p_m > p_n$ while $\ell_m > \ell_n$ then swapping $\boldsymbol{c}_m$ and $\boldsymbol{c}_n$ gives a new code with smaller L. (Simply substitute into formulae).

2. Kraft-McMillan.

3. If $\displaystyle\sum_{i=1}^{q} \frac{1}{2^{\ell_i}} = K < 1$ then decreasing ℓ_q by 1 gives a code with a smaller value of L which still satisfies $K_{\text{new}} \le 1$. (This follows since the lengths ℓ_i are non-increasing, so $K \le 1 - 2^{-\ell_q}$.)

4. If $\ell_{q-1} < \ell_q$ then, as none of $\boldsymbol{c}_1, \cdots, \boldsymbol{c}_{q-1}$ are a prefix to $\boldsymbol{c}_q$, we can omit the last digit of $\boldsymbol{c}_q$ and get a better L.

5. Let $\widehat{\boldsymbol{c}}_q$ be obtained from $\boldsymbol{c}_q$ by deleting the last digit of $\boldsymbol{c}_q$. This gives a new code with $K_{\text{new}} > 1$, so the new code is not instantaneous. Therefore $\widehat{\boldsymbol{c}}_q$ must be a prefix to some other codeword, say $\boldsymbol{c}_i$. This implies that $\ell_q - 1 \leq \ell_i$. But we also have $i \leq q-1$ and so $\ell_i \leq \ell_{q-1} = \ell_q$. Therefore $\ell_q - 1 \leq \ell_i \leq \ell_q$. However, if $\ell_i = \ell_q - 1$ then $\boldsymbol{c}_i$ is a prefix of $\boldsymbol{c}_q$, which is a contradiction. Therefore $\ell_i = \ell_q$ and $\boldsymbol{c}_i$ and $\boldsymbol{c}_q$ differ only in the last place. Swap $\boldsymbol{c}_i$ and $\boldsymbol{c}_{q-1}$ to complete the proof. □

3.5 Huffman's algorithm (binary case)

In 1951 David Huffman, a student of Robert Fano, saw that you get the minimum length UD-code by creating the decision tree in the other direction.

1. **Combining phase.**

   We are assuming the symbols are in non-increasing order of probabilities.

   Combine the two symbols of *least* probability into a single *new* symbol of combination probability $p_{q-1} + p_q$ called the **child** of the two **parents**. The new source has one less symbol. Rearrange the symbols into non-increasing order of probabilities, keeping track of all movement in a tree with child and parent nodes for the symbols. **Repeat** until only one symbol (the **root**) is left.

2. **Splitting phase.**

   Starting with the child symbol at the far right, move leftwards and label *each edge from a child node back to its parents*, using a 0 for the top edge 1 for the other. Do not label any other edges. In this way we get a backwards decision tree. Each original symbol then has as codeword the *reverse* of the string of bits you get following the edges to the single node.

We usually show both phases on the same diagram (see below) and only show the probabilities and codewords.

Work from left to right on the combining phase, then back from right to left on the splitting phase.

The arrows track the rearranging and I have used boxes for parent nodes and circles for child (new) nodes. Our **standard method** will be to always append '0' to the upper symbol and '1' to the lower symbol. (We could assign 0 and 1 arbitrarily at each child node if we wanted, it would make no practical difference.)

Various choices can be made at different places – some lead to equivalent codes, others to quite different codes. We will discuss two strategies.

Examples

(a) **place low strategy** — here when the child symbol has the same probability as others we place it in the *lowest* possible position in the list. See figure 3.1.
For this code, the average length $L = \sum_i \ell_i p_i$ is

$$\begin{aligned} L &= 2 \times 0.3 + 2 \times 0.2 + 2 \times 0.2 + 3 \times 0.1 + 4 \times 0.1 + 4 \times 0.1 \\ &= 2.5 \end{aligned}$$

and the variance $V = \sum_i {\ell_i}^2 p_i - L^2$ is

$$\begin{aligned} V &= 2^2 \times 0.3 + \cdots + 4^2 \times 0.1 - (2.5)^2 \\ &= 0.65 \end{aligned}$$

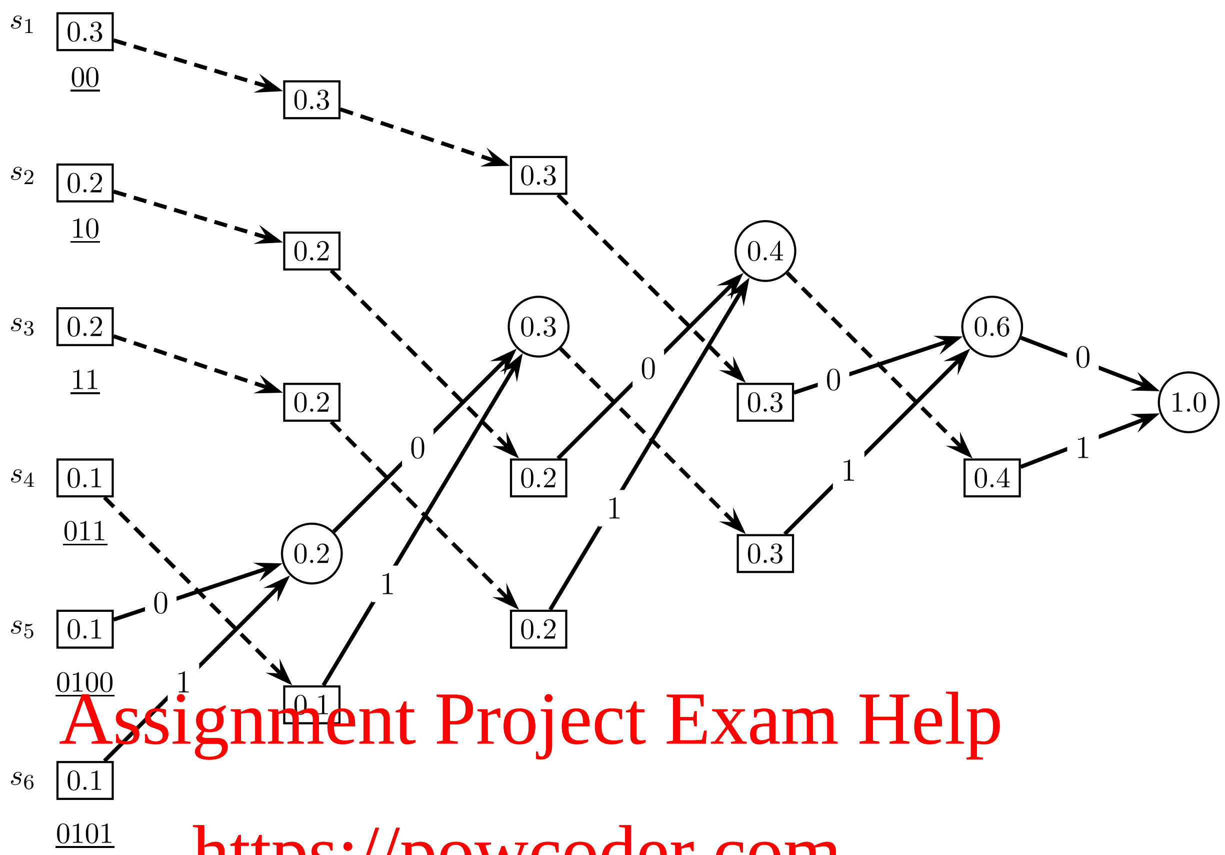


Figure 3.1: Place low strategy

(b) **place high strategy** — here we place the child symbols as high as possible in the list. See figure 3.2.

Now for this code the average length is

$$L = 2 \times 0.3 + \cdots + 3 \times 0.1 = 2.5$$

and the variance is

$$V = 2^2 \times 0.3 + \cdots + 3^2 \times 0.1 - (2.5)^2 = 0.25$$

This has the same average length L as the place low strategy, but a much lower variance V. Therefore we prefer code (b) to code (a), and we will *always use the place high strategy.*

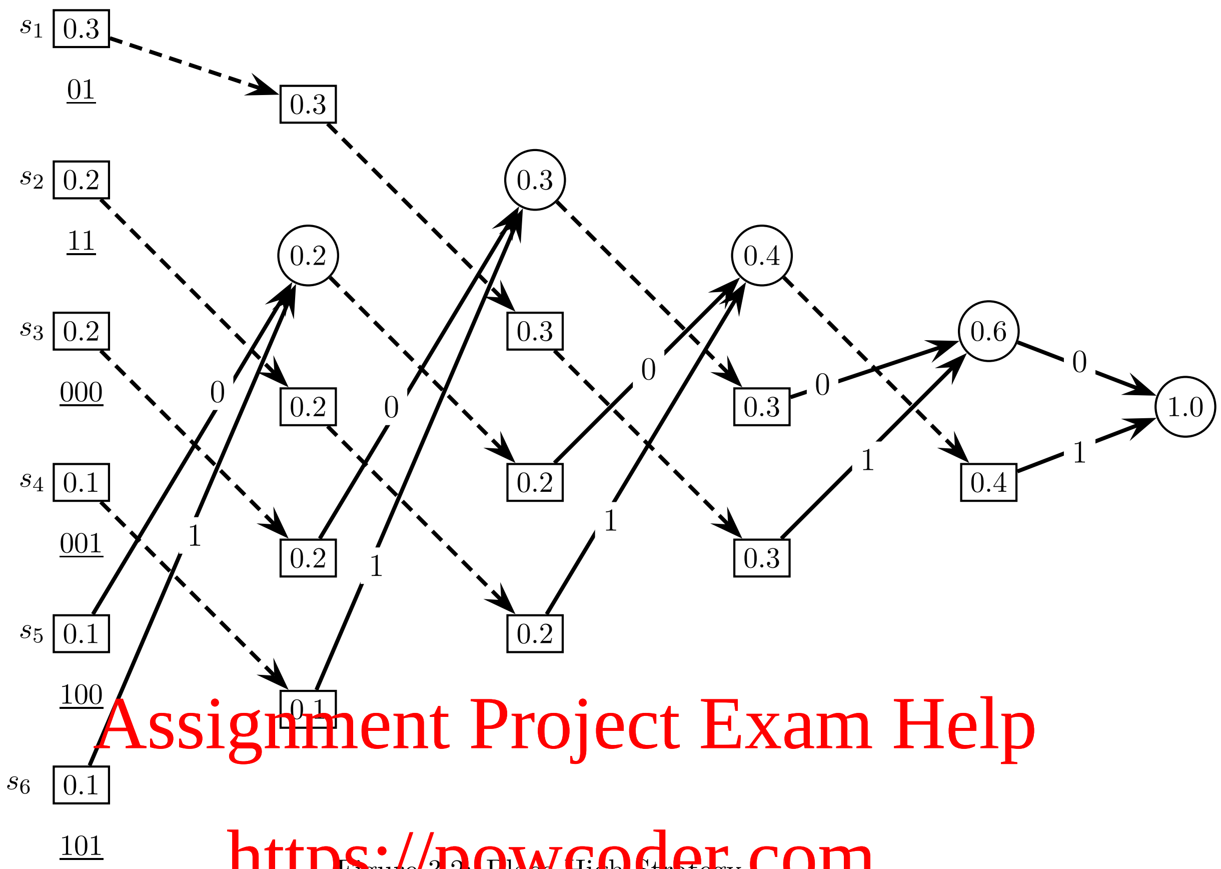


Figure 3.2: Place High Strategy

Theorem 3.3 : Huffman Code Theorem

For the given source S, the Huffman algorithm produces a minimum average length UD-code which is an instantaneous code.

Proof: By induction on $q = |S|$. The base case is $q = 2$. Here the Huffman code has minimum average length $L = 1$.

For the inductive step, assume that the Huffman code gives the minimum average length UD-code for all sources with $q - 1$ symbols.

Let $\mathcal{C}$ be a Huffman code on q symbols with average length L and let $\mathcal{C}^*$ be any *minimum* average length UD code with codeword lengths $\ell_1^*, \cdots, \ell_q^*$ and average length L^*.

In the Huffman code $\mathcal{C}$, by construction, $\boldsymbol{c}_q$ and $\boldsymbol{c}_{q-1}$ differ only in last place.
By Theorem 2, the UD code $\mathcal{C}^*$ has codewords $\boldsymbol{c}_q^*$ and $\boldsymbol{c}_{q-1}^*$ which differ only in the last place.

So (one step) combining on both gives the following:

The code obtained from the Huffman code $\mathcal{C}$ is still a Huffman code. Denote its average length by M.

The code obtained from the UD-code $\mathcal{C}^*$ is UD. Denote its average length by M^*. (Note that it might not be a *minimum* average length UD).

By the induction hypothesis $M \leq M^*$. Hence

$$\begin{aligned} L - L^* &= \sum_{i=1}^{q} \ell_i p_i - \sum_{i=1}^{q} \ell_i^* p_i \\ &= \left[\sum_{i=1}^{q-2} \ell_i p_i + (\ell_q - 1)(p_{q-1} + p_q)\right] \\ &\qquad - \left[\sum_{i=1}^{q-2} \ell_i^* p_i + (\ell_q^* - 1)(p_{q-1} + p_q)\right] \\ &= M - M^* \leq 0. \end{aligned}$$

Thus $L \leq L^*$ and we conclude that the Huffman code is the best for the case q also.

Since the code is created using a decision tree it is instantaneous. □

With the Huffman algorithm we have a quick way of calculating the average length:

Proposition 3.4 (Knuth) *For a Huffman code created by the given algorithm, the average code word length is sum of all the probabilities at child nodes.*

So, in the example in figures 3.1 and 3.2, the probabilities at child nodes were 0.2, 0.3, 0.4, 0.6 and 1, which add up to 2.5, the average codeword length that we calculated.

Proof:

The path through the tree that gives the codeword for symbol s_i will pass through exactly ℓ_i child nodes. But p_i will occur as part of the sum in each of the these child nodes, so adding up all the probabilities at all child nodes will add up ℓ_i copies of p_i for each symbol s_i, but this is L. □

Notes: The following points can be shown:

1. The place high strategy always produces a **minimum variance** Huffman code (L.T. Kau, *SIAM J. Comput.* **11** (1980) p. 138). So we will *always* use this strategy.

2. If there are 2^n equally likely source symbols then the Huffman code is a block code of length n (problem 31).

3. If for all j, $3p_j \geq 2\sum_{k=j+1}^{q} p_k$ then the Huffman code is a comma code.

4. Small changes in the p_i can change the Huffman code substantially, but have little effect on the average length L. This effect is smaller with smaller variance.

3.6 Radix r Huffman codes

Instead of combining two symbols at a time, we now need to combine r symbols at a time. To do this we can use two strategies.

(1) At each stage, combine the last r symbols until no more than r symbols are left. Then assign codewords and perform the splitting phase.

(2) Introduce new dummy variables of zero probability at the end, so that the combining stage ends with exactly r symbols. To do this there must be

$$k(r-1) + r = (k+1)(r-1) + 1$$

symbols to start with. That is, we need $q \equiv 1 \pmod{r-1}$ symbols before we start combining. So if the number of symbols q is not 1 plus some multiple of $(r-1)$, then introduce enough extra symbols to make it so.

Strategy (2) is better than strategy (1), hence we will always use strategy (2).

Example: We code the same example as before using radix 3, see figure 3.3
Now $q = 6 \not\equiv 1 \pmod 2$ so we introduce a dummy symbol.

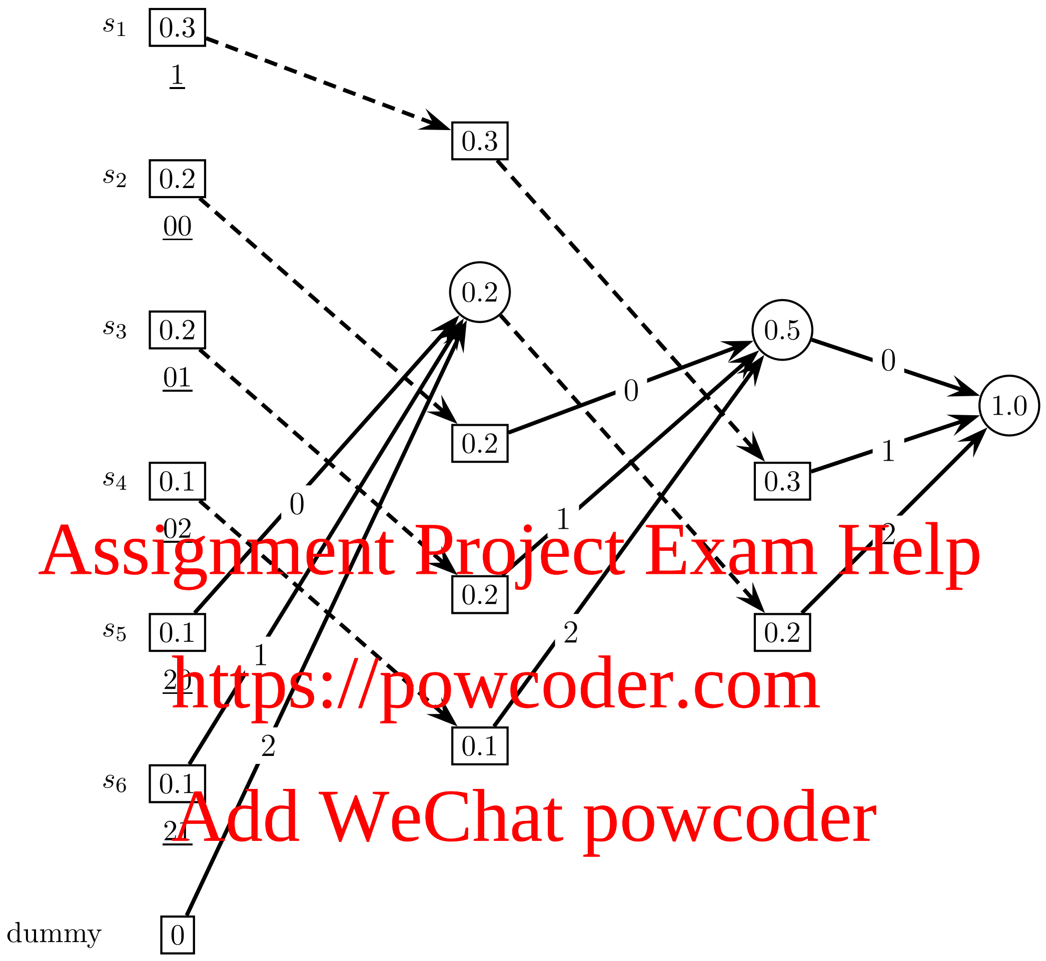


Figure 3.3: Radix 3 Example

Here the average codeword length and variance (denoted by L_3 and V_3 to show radix 3) are

$$\begin{aligned} L_3 &= 1 \times .3 + 2 \times .2 + \cdots = 1.7 \\ V_3 &= 1^2 \times .3 + 2^2 \times .2 + \cdots - (1.7)^2 = 0.21 \end{aligned}$$

Naturally L_3 is smaller than the average length $L = L_2$ of the binary code, since $r = 3$ means you are able to use shorter codewords. We also see that proposition 3.4 applies here too with identical proof.

3.7 Extensions

So far we have been coding for single letters or symbols. Maybe we can gain something by coding whole words.

By the **nth extension** of a source we mean the set of all words of length n formed from the source symbols, together with their corresponding probabilities. We denote this by S^n.

Lacking any other information, we assume **independence** for the probabilities of successive symbols.

So a symbol $\sigma_i \in S^n$ has the form

$$\sigma_i = s_{i_1} s_{i_2} \cdots s_{i_n} \quad \text{where } s_{i_j} \in S$$

and

$$P(\sigma_i) = P(s_{i_1} \cdots s_{i_n}) = p_{i_1} p_{i_2} \cdots p_{i_n}.$$

Here $1 \le i_j \le q$ and σ_i has $1 \le i \le q^n$.

It does not matter how we number the extension symbols, so normally it would be done so that their probabilities are in **non-increasing** order.

Example 3.9 The first few extensions of $S = \{s_1, s_2\}$ of relative probabilities $\frac{3}{4}, \frac{1}{4}$, coded using Huffman coding are:

$n = 1$	$S^1 = S$	p_i	$\boldsymbol{c}_i$
	s_1	$\frac{3}{4}$	0
	s_2	$\frac{1}{4}$	1

Average length of code for S^1 is $L^{(1)} = L = 1$.

$n = 2$	S^2	$16p_i$	$\boldsymbol{c}_i$
	$\sigma_1 = s_1s_1$	9	0
	$\sigma_2 = s_1s_2$	3	11
	$\sigma_3 = s_2s_1$	3	100
	$\sigma_4 = s_2s_2$	1	101

Average length of code for S^2 is $L^{(2)} = \dfrac{27}{16}$.

Average length per (original source) symbol $= \dfrac{L^{(2)}}{2} = \dfrac{27}{32} = 0.84$.

$n = 3$	S^3	$64 \times$ prob	$\boldsymbol{c}_i$
	$\sigma_1 = s_1s_1s_1$	27	1
	$\sigma_2 = s_1s_1s_2$	9	001
	$\sigma_3 = s_1s_2s_1$	9	010
	$\sigma_4 = s_2s_1s_1$	9	011
	$\sigma_5 = s_1s_2s_2$	3	00000
	$\sigma_6 = s_2s_1s_2$	3	00001
	$\sigma_7 = s_2s_2s_1$	3	00010
	$\sigma_8 = s_2s_2s_2$	1	00011

Average length of code $L^{(3)} = \dfrac{158}{64}$

Average length per (original source) symbol $= \dfrac{L^{(3)}}{3} = \dfrac{158}{192} = 0.82$.

Notes:

1. We use the following notation:
   $L^{(n)}$ is average length of code for nth extension,
   $\dfrac{L^{(n)}}{n}$ is average length per original source symbol.

2. We notice that $\dfrac{L^{(n)}}{n}$ seems to decrease as n increases, which is to be expected.
   Does $\dfrac{L^{(n)}}{n}$ tend to zero, or does $\dfrac{L^{(n)}}{n}$ tend to a positive limit?
   We answer these questions in Chapter 4.

3.8 Markov sources

We previously assumed that consecutive source symbols were independent: that is, we assumed that the probability of a particular symbol being the next source symbol did not depend on the previous symbol. This is clearly not appropriate in all circumstances. For instance, in English the letter ‘Q’ is usually followed by ‘U’.

We say that a source is a **memoryless source** if successive source symbols are independent.

A ***k*-memory source** is one in which the probability of any symbol depends on the previous k symbols in the symbol string.

Here $P(s_i | s_{i_1} s_{i_2} \cdots s_{i_k})$ denotes the probability that s_i occurs given that $s_{i_1} s_{i_2} \cdots s_{i_k}$ has just occurred. This is conditional probability.

These are difficult to deal with in general and we restrict ourselves to the **1-memory** case which is called a **Markov source**.

Denote $P(s_i | s_j)$, the probability that s_i occurs immediately following a given s_j, by p_{ij}.

The matrix $M = (p_{ij})$ is called the **transition matrix** of the Markov process.
Note the pattern in the matrix:

$$\begin{array}{cc} & \text{from } s_j \\ & \downarrow \\ \text{to } s_i & \left(\leftarrow \; p_{ij} \qquad\qquad \right) \end{array}$$

That is, the transition from s_j to s_i occurs with probability p_{ij} and is stored in row i and column j.

This notation is not standard and many books use the transpose of this. *Take care.*

Now one of the properties of conditional probabilities is that

$$P(s_1|s_j) + P(s_2|s_j) + \cdots + P(s_q|s_j) = 1 \ .$$

That is,

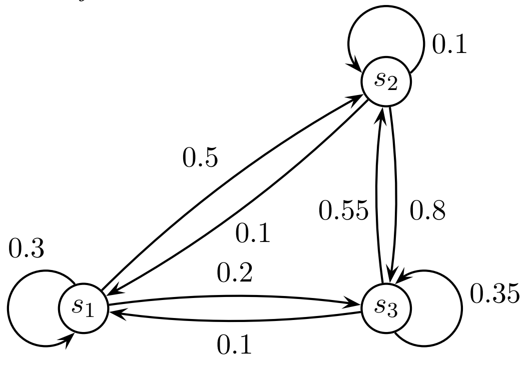
$$p_{1j} + p_{2j} + \cdots + p_{qj} = 1, \qquad \text{for } j = 1, \cdots, q$$

so that each *column sums to 1.* All the entries are also greater than or equal to zero.

We can also view a Markov source as a finite state process in which the transitions are between states $s_1, \cdots, s_q$ and occur with probabilities p_{ij}.

Example 3.10

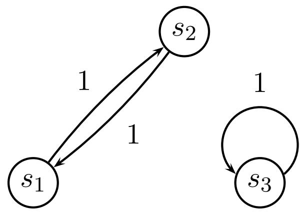
$$M = \begin{pmatrix} 0.3 & 0.1 & 0.10 \\ 0.5 & 0.1 & 0.55 \\ 0.2 & 0.8 & 0.35 \end{pmatrix}$$



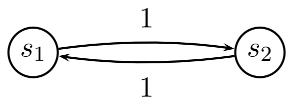
Since each column of M sums to 1 then the probabilities on arrows leading **out** of a state sum to 1.

We will **assume** that all our Markov processes are **stationary** (or **ergodic**), which means that it is possible to get from any state to any other state by a sequence of arrows with **positive** probability and also **aperiodic**, which means the greatest common divisor of all cycle lengths is 1 (a cycle is a walk that returns to its start).

Example 3.11 Consider



not irreducible



not aperiodic

Under these assumptions it can be proved that, after some time, the probabilities that the process is in state s_i settle down to unique **stationary** or **equilibrium** values whatever the starting conditions.

We denote the equilibrium probabilities by $p_1, \cdots, p_q$.

This means that no matter where the process starts then after a large number of iterations the probability that the current state is s_i is equal to p_i. (We still have that $P(s_i|s_j) = p_{ij}$, of course.)

Suppose that the Markov chain is in equilibrium at the current state X_m, and that the next state is X_{m+1}. (Both X_m, X_{m+1} are elements of $\{s_1, \ldots, s_q\}$.) Then $P(X_{m+1} = s_i) = p_i$ since the Markov chain is in equilibrium. So

$$\begin{aligned}
p_i &= P(X_{m+1} = s_i) \\
&= P(X_m = s_1 \text{ and } X_{m+1} = s_i) + \cdots + P(X_m = s_q \text{ and } X_{m+1} = s_i) \\
&= P(X_{m+1} = s_i | X_m = s_1) P(X_m = s_1) + \cdots + P(X_{m+1} = s_i | X_m = s_q) P(X_m = s_q)
\end{aligned}$$

We can rewrite this as

$$\begin{aligned}
p_i &= P(s_i|s_1)P(s_1) + \cdots + P(s_i|s_q)P(s_q) \\
&= p_{i1}p_1 + p_{i2}p_2 + \cdots + p_{iq}p_q
\end{aligned}$$

and this equation holds for $1 \le i \le q$.

Writing the vector of probabilities as a column vector $\quad \boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_q \end{pmatrix}, \quad$ our equation simply says that

$$\boldsymbol{p} = M\boldsymbol{p}.$$

Hence the equilibrium probability vector $\boldsymbol{p}$ is an *eigenvector* of the transition matrix M corresponding to the eigenvalue $\lambda = 1$. To find $\boldsymbol{p}$, find any eigenvector of M with eigenvalue 1, and then *scale* this vector by dividing by the sum of its entries. This gives the equilibrium probability vector $\boldsymbol{p}$ which satisfies $M\boldsymbol{p} = \boldsymbol{p}$ and $p_1 + \cdots + p_q = 1$.

Example 3.12 We can find the equilibrium probabilities for the given matrix M by finding a vector in the null space (kernel) of $M - I$. Using row reduction we obtain

$$\begin{pmatrix} -0.7 & 0.1 & 0.1 \\ 0.5 & -0.9 & 0.55 \\ 0.2 & 0.8 & -0.65 \end{pmatrix} \to \begin{pmatrix} -7 & 1 & 1 \\ 10 & -18 & 11 \\ 4 & 16 & -13 \end{pmatrix} \to \begin{pmatrix} -7 & 1 & 1 \\ 3 & -17 & 12 \\ 4 & 16 & -13 \end{pmatrix}$$

$$\to \begin{pmatrix} 3 & -17 & 12 \\ 1 & 33 & -25 \\ 0 & 0 & 0 \end{pmatrix} \to \begin{pmatrix} 1 & 33 & -25 \\ 0 & -116 & 87 \\ 0 & 0 & 0 \end{pmatrix} \to \begin{pmatrix} 1 & 33 & -25 \\ 0 & -4 & 3 \\ 0 & 0 & 0 \end{pmatrix}.$$

Hence by back-substitution we find an eigenvector $\mathbf{v} = (1, 3, 4)$. Since $1+3+4 = 8$ we divide by 8 to give the vector of equilibrium probabilities $\boldsymbol{p} = \frac{1}{8}\begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix}$.

NOTE: In the exam we will give you the vector $\boldsymbol{p}$. You then need only to check that $\boldsymbol{p}$ is a probability vector and $M\boldsymbol{p} = \boldsymbol{p}$.

3.9 Huffman coding for stationary Markov sources

We construct q possibly different Huffman codes, one for each *previous* symbol s_j, based on the probabilities that $s_1, \cdots, s_q$ follow s_j. Recall that

$$\left.\begin{array}{llll} & s_1 & & p_{1j} \\ s_j \text{ is followed by} & \vdots & \text{with probability} & \vdots \\ & s_q & & p_{qj} \end{array}\right\} \text{ this is column } j \text{ of } M.$$

Remembering to re-order so probabilities are *non-increasing*, construct a code $\text{Huff}_{(j)}$ for this situation having average length $L_{(j)}$.

We assume that the Markov source is in equilibrium, so s_j occurs with probability p_j. Therefore the code $\text{Huff}_{(j)}$ will be used with probability p_j, and the overall average length of codeword will be

$$L_M = p_1 L_{(1)} + p_2 L_{(2)} + \cdots + p_q L_{(q)}.$$

This is the **average length for the Markov Huffman code**. (We write Huff_M to denote this code.)

This ignores the code for the first symbol since it is not preceded by any other symbol. The first symbol is normally encoded using the Huffman code for the equilibrium probabilities, denoted Huff_E, of average length L_E.

The above formula for L_M is then not strictly correct, but if a large number of source symbols are encoded the effect of the slightly different situation for the first will be negligible and so we ignore it in L_M calculations.

Example 3.13 Using the above $M = \begin{pmatrix} 0.3 & 0.1 & 0.10 \\ 0.5 & 0.1 & 0.55 \\ 0.2 & 0.8 & 0.35 \end{pmatrix}$ with equilibrium $\boldsymbol{p} = \frac{1}{8}(1, 3, 4)$, we have equilibrium Huffman code

Huff$_E$	prob $\times$ 8	code
s_1	1	01
s_2	3	00
s_3	4	1

with $L_E = \frac{1}{8}(2 \times 1 + 2 \times 3 + 1 \times 4) = 1.5$.

The Huffman code for symbols following the symbol s_1 is

Huff$_{(1)}$

s_1	0.3	00
s_2	0.5	1
s_3	0.2	01

with $L_{(1)} = 2 \times 0.3 + 1 \times 0.5 + 2 \times 0.2 = 1.5$

The Huffman code for symbols following the symbol s_2 is

Huff$_{(2)}$

s_1	0.1	10
s_2	0.1	11
s_3	0.8	0

with $L_{(2)} = 1.2$.

Finally, the Huffman code for symbols following the symbol s_3 is

Huff$_{(3)}$ (i.e. probabilities of following s_3)

s_1	0.1	11
s_2	0.55	0
s_3	0.35	10

with $L_{(3)} = 1.45$.

Then the average length of the Markov Huffman code (ignoring the "first symbol effect") is

$$\begin{aligned} L_M &= \frac{1}{8} \times 1.5 + \frac{3}{8} \times 1.2 + \frac{1}{2} \times 1.45 \\ &\approx 1.36 \end{aligned}$$

So $L_M < L_E$ as expected, since L_E is the average length if you ignore the Markov dependencies among the source symbols.

Now, if we simply used a binary block code to encode this example, ignoring the source probabilities completely, we would need 2 bits per symbol and so

$$L_{\text{block}} = 2$$

which is considerably worse than either L_E or L_M.

Hence using the variable length code Huff$_E$ *compresses* the message length by a factor of

$$\frac{L_E}{L_{\text{block}}} = \frac{1.5}{2} = 0.75, \qquad \text{that is, 25\% saved.}$$

Using Huff_M compresses by

$$\frac{1.36}{2} = 0.68, \qquad \text{that is, 32\% saved.}$$

Even if we ignore the source probabilities completely and assume that each source symbol is equally likely, then we obtain

s_1	$\frac{1}{3}$	1
s_2	$\frac{1}{3}$	00
s_3	$\frac{1}{3}$	01

which gives

$$L = 1.67.$$

Here the compression factor is $\frac{1.67}{2} = 0.83$, that is, 17% saved.

Example 3.14 Finally, here is an example of using Huff_M to encode with the above M.

The message $s_1 s_2 s_3 s_3 s_2 s_1 s_2$ gets coded via

s_1	first, so use	Huff_E	giving	01
s_2	after s_1 so use	$\text{Huff}_{(1)}$		1
s_3	s_2	$\text{Huff}_{(2)}$		0
s_3	s_3	$\text{Huff}_{(3)}$		10
s_2	s_3	$\text{Huff}_{(3)}$		0
s_1	s_2	$\text{Huff}_{(2)}$		10
s_2	s_1	$\text{Huff}_{(1)}$		1

so this is encoded as 0110100101 .

Here $\frac{\text{length}}{7} = \frac{10}{7} = 1.43$ is the code length per symbol.

Since each of the Huffman codes used is an I-code, any valid encoding will decode uniquely.

Example 3.15 We decode by running the above in reverse, i.e. 1st letter using Huff_E as s_{i_1} then 2nd using $\text{Huff}_{(i_1)}$ as s_{i_2} 3rd using $\text{Huff}_{(i_2)}$ etc.

first using	Huff_E	01	is first codeword and decodes as	s_1
next using	$\text{Huff}_{(1)}$	1	is next codeword and decodes as	s_2
	$\text{Huff}_{(2)}$	0		s_3
	$\text{Huff}_{(3)}$	10		s_3
	$\text{Huff}_{(3)}$	0		s_2
	$\text{Huff}_{(2)}$	10		s_1
	$\text{Huff}_{(1)}$	1		s_2

3.10 Other Text Compression Methods

There are many ways of approaching text compression and many considerations: for example, compression efficiency, speed, memory use, uncompression speed, memory use.

The model of the source symbols is also important. Some methods use a *static* model where the probabilities of each source symbol are fixed. Others are *adaptive* where the probabilities are updated after each symbol is received, to reflect their relative frequency.

We discuss two methods.

3.10.1 Arithmetic Coding

Arithmetic coding is a form of compression coding which encodes the message into a rational number between 0 and 1. It is very efficient and approaches the entropy limit faster than Huffman. This is not a contradiction because arithmetic coding is not a UD-code. The reason it is so efficient is that it does not need to use an integral number of bits for each symbol of the message. Huffman is more commonly used as arithmetic coding is more computationally expensive to implement. Arithmetic coding is also subject to patents.

The idea is to assign a subinterval of $[0, 1) \subseteq \mathbb{R}$ to the message and successively narrow this subinterval down as each symbol is encoded.

The message must end with a stop symbol $\bullet$, and after the subinterval corresponding to the message plus $\bullet$ is found, then any suitable *single number* in the subinterval is transmitted — this is the actual code number or codeword.

Encoding:

Firstly, to each symbol is associated a subinterval of $[0, 1)$ whose length is proportional to the relative frequency of the symbol.

These subintervals are chosen so they do not overlap, but fill $[0, 1)$.

If the "message so far" subinterval is $[s, s + w)$ where s is the start, w is the width

and the next symbol has associated subinterval $[s', s' + w')$

then the new message subinterval is the $[s', s' + w')$ part of $[s, s + w)$.

That is, the new message subinterval becomes

$$[\, s + s'w, (s + s'w) + w'w\,)$$

Decoding:

We reverse the above, by finding which of the symbol subintervals contains the code number.

This then gives the first symbol of the message.

The code number is then re-scaled to that interval as follows. If x is the code number and it lies in the subinterval $[s, s + w)$, then the rescaled code number is $(x - s)/w$.

The process is repeated until the stop symbol $\bullet$ is encountered.

Example 3.16 Suppose we have a fixed probability model

	probability	subinterval	start	width
s_1	0.4	$[0, .4)$	0	.4
s_2	0.2	$[.4, .6)$	.4	.2
s_3	0.2	$[.6, .8)$	.6	.2
s_4	0.1	$[.8, .9)$	.8	.1
$\bullet$	0.1	$[.9, 1)$	.9	.1

We wish to encode the message $s_1 s_2 s_1 s_4 s_2 s_1 s_3 s_3 s_1 \bullet$

Encoding proceeds as follows:

	subinterval start	**width**
begin	0	1
s_1	$0+0=0$	$.4\times 1=.4$
s_2	$0+.4\times .4=.16$	$.2\times .4=.08$
s_1	$.16+0\times .08=.16$	$.4\times .08=.032$
s_4	$.16+.8\times .032=.1856$	$.1\times .032=.0032$
s_2	$.1856+.4\times .0032=.18688$	$.2\times .0032=.00064$
s_1	$.18688+0\times .00064=.18688$	$.4\times .00064=2.56\times 10^{-4}$
s_3	$''+.6\times 2.56\times 10^{-4}=.1870336$	$.2\times ''=5.12\times 10^{-5}$
s_3	$''+.6\times 5.12\times 10^{-5}=.18706432$	$.2\times ''1.024\times 10^{-5}$
s_1	$''+0\times 1.024\times 10^{-5}=.18706432$	$.4\times ''=4.096\times 10^{-6}$
$\bullet$	$''+.9\times 4.096\times 10^{-6}=.187068006$	$.1\times ''=4.096\times 10^{-7}$

The message subinterval is [.187068006, .187068415) and we transmit say .1870681, which is the shortest decimal in this subinterval, as the code number or codeword. Using a Huffman coding for the same source probabilities would give a 22 bit message.
Decoding proceeds as follows:

Example 3.17

code number rescaled	**in interval**	**so symbol**
$.1870681$	$[0,.4)$	s_1
$(.1870681-0)/.4=.46767025$	$[.4,.6)$	s_2
$(.46767025-.4)/.2=.33835125$	$[0,.4)$	s_1
$(.33835125-0)/.4=.845878125$	$[.8,.9)$	s_4
$(''-.8)/.1=.45878125$	$[.4,.6)$	s_2
$(''-.4)/.2=.29390625$	$[0,.4)$	s_1
$(''-0)/.4=.734765625$	$[.6,.8)$	s_3
$(''-.6)/.2=.673828125$	$[.6,.8)$	s_3
$(''-.6)/.2=.369140626$	$[0,.4)$	s_1
$(''-0)/.4=.922851562$	$[.9,1)$	$\bullet$
so stop		

Note:

1. Without the stop symbol, decoding would usually go on forever.

2. It is possible to implement arithmetic coding and decoding using integer arithmetic, and if we use binary representation for the subintervals then it can actually be done using binary arithmetic. Successive digits can be transmitted as soon as they are finalized; that is, when both start and end points of the interval agree to sufficiently many significant places. Decoding can also be done as the digits arrive. Hence arithmetic coding does not need much memory.

3. We have described a static version of arithmetic coding. But the process can also be made adaptive by continually adjusting the start/width values for symbols while decoding takes place.

3.10.2 Dictionary methods

In these methods, a dictionary is maintained and parts of the text are replaced by pointers to the appropriate dictionary entry. Examples are Ziv and Lempel's algorithms `LZ77` and `LZ78` and Welch's improvement, called the `LZW` algorithm. These compression algorithms are used in gzip, gif and postscript. It is known that `LZ78` and `LZW` approach the entropy limit for very long messages. But it approaches the limit so slowly that in practice it never gets close.

The `LZ77` algorithm uses a system of pointers looking back in the data stream: the `LZ78` algorithm creates a system of pointers (a dictionary) looking forward in the data stream. One drawback with `LZ77` is that it takes a lot of time to encode, as it involves a lot of comparisons: decoding is quick though.

We describe the `LZ78` algorithm for encoding a message m.

Encoding: start with an empty dictionary and set $r = m$. (Here r is the part of the message which we have not yet encoded.) Let "+" denote concatenation of symbols. Repeat the following steps until r is empty:

(i) Let s be the longest prefix of r which corresponds to a dictionary entry. If there is no such prefix then set $s = \emptyset$. (By *prefix* we mean a subsequence of r which begins at the first symbol.)

(ii) Suppose that s is in line ℓ of the dictionary, and put $\ell = 0$ if $s = \emptyset$. Add a new dictionary entry for $s + c$, where c is the next symbol after s in r. Output (ℓ, c) to the encoding and delete $s + c$ from r.

Example 3.18 Suppose we wish to encode the message $m = abbcbcababcaa$.

r	s	ℓ	new dictionary entry	output
$abbcbcababcaa$	$\emptyset$	0	1. a	$(0, a)$
$bbcbcababcaa$	$\emptyset$	0	2. b	$(0, b)$
$bcbcababcaa$	b	2	3. bc	$(2, c)$
$bcababcaa$	bc	3	4. bca	$(3, a)$
$babcaa$	b	2	5. ba	$(2, a)$
$bcaa$	bca	4	6. $bcaa$	$(4, a)$

The output is $(0, a)(0, b)(2, c)(3, a)(2, a)(4, a)$. Ignoring brackets this has 12 symbols consisting of 6 numbers and 6 letters whereas the original message has 13 symbols, all letters. Much better compression rates are achieved when encoding long texts, or data streams with a lot of repetition, as the dictionary entries themselves become longer.

Decoding: start with an empty dictionary. If the next code pair is (ℓ, c) then output $D(\ell) + c$ to the message, where $D(\ell)$ denotes entry ℓ of the dictionary and $D(0) = \emptyset$. Also add $D(\ell) + c$ to the dictionary. Repeat until all code pairs have been processed. The dictionary after the decoding process is exactly the same as the dictionary after the encoding process.

Example 3.19 To decode the codeword $(0, c)(0, a)(2, a)(3, b)(4, c)(4, b)$, start with the empty dictionary.

output	new dictionary entry
c	1. c
a	2. a
aa	3. aa
aab	4. aab
$aabc$	5. $aabc$
$aabb$	6. $aabb$

The decoded text is $caaaaabaabcaabb$.

3.11 Other types of Compression

We have only considered compression methods that are **lossless**: there is no information lost. There are **lossy** methods of compression that can typically achieve better compression at the expense of losing (usually superfluous) information. For example, JPEG and MPEG use lossy compression. The difficulty with lossy compression is that you have to choose which information to lose. With images and music this is usually done so that the human eye or ear cannot detect the loss: you drop subtle colour differences or extreme frequency values for example. Once these extremes are removed, a Huffman, arithmetic or LZ78 coding of what remains will be much more efficient.

See `www.whydomath.org` under **wavelets** for more information.

Chapter 4

Information Theory

As usual we have a source

$$S = \{s_1, s_2, \cdots, s_q\} \quad \text{with probabilities} \quad p_1, p_2, \cdots, p_q.$$

If we receive a symbol from this source, how much "information" do we receive?

If

$$p_1 = 1 \qquad \text{and} \qquad p_2 = \cdots = p_q = 0$$

and s_1 is received then we are not at all *surprised* and we receive no "information".

But if

$$p_1 = \epsilon > 0 \qquad \text{(very small)}$$

and s_1 arrives we are *very surprised* and receive lots of "information".

"Information" will measure the amount of *surprise* or *uncertainty* about receiving a given symbol: it is somehow *inversely* related to probability.

Let $I(p)$ be the **information** contained in an event of probability p.

We make some natural assumptions about $I(p)$ and see where these assumptions lead us.

Assume that

(a) $I(p) \geq 0$ for $0 \leq p \leq 1$,

(b) $I(p_1 p_2) = I(p_1) + I(p_2)$ for **independent** events of probability p_1 and p_2,

(c) $I(p)$ is a continuous function of p.

Now if we independently repeat the same event we get

$$I(p^2) = I(p.p) = I(p) + I(p) = 2I(p)$$

and by induction, for each positive integer n we have

$$I(p^n) = nI(p).$$

Next, let m, n be positive integers and let

$$p^n = x, \qquad \text{so} \quad p = x^{\frac{1}{n}}.$$

Then

$$I(x) = nI(x^{\frac{1}{n}})$$

and multiplying by m gives

$$mI(x) = nI(x^{\frac{m}{n}}).$$

Hence
$$I(x^{\frac{m}{n}}) = \tfrac{m}{n} I(x)$$
and we have
$$I(x^\alpha) = \alpha I(x) \quad \text{for all positive rational numbers} \quad \alpha.$$
Then by continuity we have
$$I(x^\alpha) = \alpha I(x) \qquad \text{for all real } \alpha > 0.$$
Hence $I(x)$ behaves like a logarithm.

In fact it can be shown that the *only* function which satisfies (a), (b), (c) is
$$I(x) = k \ln x \quad \text{where} \quad k < 0 \quad \text{is a constant.}$$

For the **binary** case (radix 2), define for $0 < p \le 1$,
$$\begin{aligned} I(p) &= I_2(p) = -\log_2 p & \left(= -\frac{1}{\ln 2}\ln p\right) \\ &= \log_2\left(\frac{1}{p}\right) \end{aligned}$$

Here we have chosen the scale factor

$$k = -\frac{1}{\ln 2}.$$
With this scale factor we say that information is measured in "**binary units**" of information. Some books shorten this to **binits**, but as with many books we will call it **bits**. However, this is a different use of the word to **binary digits** = **bits** used earlier.

For general radix r, define

$$\begin{aligned} I_r(p) &= -\log_r p & \left(= -\frac{1}{\ln r}\ln p\right) \\ &= \log_r\left(\frac{1}{p}\right) \end{aligned}$$

measured in **radix** r **units**. Here the scale factor used is
$$k = -\frac{1}{\ln r}.$$

The reason for the differing scale factors will be clearer later on.

We will *always* use radix 2, and hence $\log_2$, unless specifically stated.

Now for our source $S = \{s_1, \cdots, s_q\}$ we write
$$I(s_j) = I(p_j) = -\log_2 p_j$$
for the information contained in observing or receiving the symbol s_j of probability p_j.

Define the **entropy** of the source S to be the average or expected **information per symbol**,
$$H(S) = H_2(S) = \sum_{i=1}^{q} p_i I(p_i) = -\sum_{i=1}^{q} p_i \log_2 p_i \quad \text{(binary)}$$
and for radix r
$$H_r(S) = -\sum_{i=1}^{q} p_i \log_r p_i\ .$$
(We often omit the subscript r when it is clear what the radix/log base should be.)

Example 4.1 Toss a coin. This has 2 outcomes h, t of probability $\frac{1}{2}, \frac{1}{2}$, so the entropy is

$$\begin{aligned} H &= H(S) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) \\ &= \log_2 2 \\ &= 1 \quad \text{bits/symbol} \end{aligned}$$

Example 4.2 Toss a biased coin with $P(h) = 0.05, \quad P(t) = 0.95$. Equivalently, observe a binary source in which $P(0) = 0.05, \quad P(1) = 0.95$. This gives an entropy of

$$\begin{aligned} H &= -0.05\log_2(0.05) - 0.95\log_2(0.95) \\ &= 0.286 \quad \text{bits/symbol} \end{aligned}$$

Example 4.3 Our initial example on Huffman coding in Chapter 3 had probabilities 0.3, 0.2, 0.2, 0.1, 0.1, 0.1 and so has entropy

$$\begin{aligned} H &= -0.3\log_2(0.3) - 2\times 0.2\log_2(0.2) - 3\times 0.1\log_2(0.1) \\ &= 2.446 \quad \text{bits/symbol (in binary units)} \end{aligned}$$

The Huffman code for this had $L = 2.5$ bits/symbol (binary digits), which is close to the $H = 2.446$ bits/symbol (binary units of information).

As we will see later, this relationship $H < L$ is no accident, and is related to the dual use of bits/symbol (being the units for code length on the one hand, and units of information on the other).

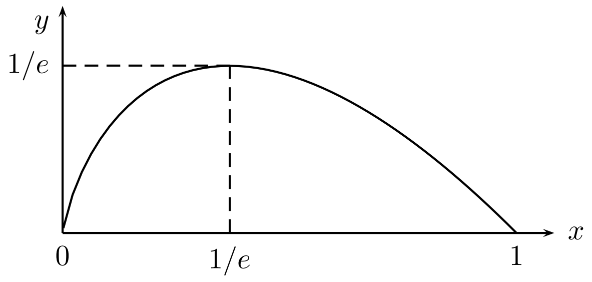
Example 4.4 Our initial example on arithmetic coding in section 3.10.1 had probabilities 0.4, 0.2, 0.2, 0.1, 0.1 (including the stop symbol) and we encoded a 10-symbol message into a 7-digit decimal number. The *decimal* entropy of this source is 0.639 digits per symbol, so we see the arithmetic coding has got as close as possible to the entropy.

4.1 Some mathematical preliminaries

1. Consider the function $f(x) = -x \log_r x$ for $0 < x \leq 1$. What is its graph?

   Ignoring scale factors consider

$$y = f(x) = -x \ln x$$



   Here $\dfrac{dy}{dx} = -1 - \ln x$ is infinite at $x = 0$ and the graph has a maximum when $-1 = \ln x$; that is, when $x = e^{-1}$.

   Also

$$\begin{aligned}
\lim_{x\to 0^+} f(x) &= \lim_{x\to 0^+} \left(\frac{-\ln x}{1/x} \right) \\
&= \lim_{x\to 0^+} \frac{-1/x}{-1/x^2} \quad \text{by L'Hôpital} \\
&= \lim_{x\to 0^+} x \\
&= 0
\end{aligned}$$

   So we **define** $f(x) = -x \log x$ to equal 0 when $x = 0$.

   Note: If $p = 0$ then $I(p)$ is infinite but this event **never** occurs so there is no problem.

   However if in our source there is a $p_i = 0$ we want it to contribute nothing to the entropy. Hence we say that $-p \log p = 0$ when $p = 0$.

   This means that we can use the formula for H without worrying about whether $p = 0$ is possible or not.

2. Suppose

$$p_1, p_2, \cdots, p_q \text{ and } p_1', p_2', \cdots, p_q'$$

   are two probability distributions. Then $0 \leq p_i,\ p_i' \leq 1$ for all i and

$$\sum_{i=1}^{q} p_i = 1 \quad \text{and} \quad \sum_{i=1}^{q} p_i' = 1.$$

   Since $\ln x \leq x - 1$ for all $x > 0$ (using first year calculus), we have

$$\begin{aligned}
\sum_{i=1}^{q} p_i \ln\left(\frac{p_i'}{p_i}\right) &\leq \sum_{i=1}^{q} p_i \left(\frac{p_i'}{p_i} - 1\right) \\
&= \sum_{i=1}^{q} (p_i' - p_i) \\
&= \sum_{i=1}^{q} p_i' - \sum_{i=1}^{q} p_i = 1 - 1 = 0.
\end{aligned}$$

Hence on scaling,

$$\begin{aligned} \sum_{i=1}^{q} p_i \log_r \left(\frac{p_i'}{p_i}\right) &\leq 0, \\ \text{or} \qquad \sum_{i=1}^{q} p_i \log_r \left(\frac{p_i}{p_i'}\right) &\geq 0. \end{aligned}$$

This inequality (in either form) is called **Gibbs' inequality** and will be used in some proofs in this chapter. Moreover *equality* occurs in Gibbs' inequality if and only if $p_i'/p_i = 1$ for all i; that is, if and only if $p_i' = p_i$ for $1 \leq i \leq q$.

4.2 Maximum Entropy Theorem

Theorem 4.1 : Maximum Entropy Theorem

For any source S with q symbols, the base r entropy satisfies

$$H_r(S) \leq \log_r q$$

with equality if and only if all symbols are equally likely.

Proof: Throughout this proof we write log for $\log_r$. Now

$$\begin{aligned} H - \log q &= -\sum_{i=1}^{q} p_i \log p_i - \log q \sum_{i=1}^{q} p_i \qquad (\text{as } \sum_{i=1}^{q} p_i = 1) \\ &= -\sum_{i=1}^{q} p_i(\log p_i + \log q) \\ &= -\sum_{i=1}^{q} p_i \log(q p_i) \\ &= -\sum_{i=1}^{q} p_i \log\left(\frac{p_i}{p_i'}\right) \qquad \text{where } p_i' = \frac{1}{q} \text{ for } 1 \leq i \leq q \\ &\leq 0 \end{aligned}$$

Here $p_i' = \frac{1}{q}$ is an equally likely source and we have used Gibbs' inequality.

The result follows. □

Consider the upper bound in this theorem, which occurs when S is source with q equally-likely symbols. where every symbol has equal probability. We can check the value of the radix r entropy of S from the definition (without using Gibbs' inequality as in the above proof). Since $p_i = 1/q$ for $1 \leq i \leq q$ we have

$$H_r(S) = q \times (-1/q \log_r(1/q)) = -\log_r(1/q) = \log_r(q) = \log_r(|S|).$$

So a source with equally-likely symbols has the *maximum* entropy.

At the other extreme, the source where

$$p_1 = 1, \ p_2 = \cdots = p_q = 0$$

has

$$H_r = -\sum_{i=1}^{q} p_i \log_r p_i = -1 \log 1 - 0 \log 0 \cdots = 0$$

giving the *minimum* entropy.

Notes:

1. The more symbols there are, the larger the maximum entropy, that is, the larger the average amount of information/surprise.
2. Which book contains the most information (in our sense)?

   The surprising answer is that it is a book of random letters, each one coming as a complete surprise.
3. An increase in entropy can be viewed as an increase in randomness, just as with entropy in physics (in fact that's why it is called entropy).

4.3 Entropy and coding

Suppose we have a binary instantaneous or UD-code for the source S, with codeword lengths $\ell_1, \ldots, \ell_q$. Then by the Kraft-McMillan inequality,

$$K = \sum_{i=1}^{q} \frac{1}{2^{\ell_i}} \le 1$$

Now, define $p_i' = \dfrac{2^{-\ell_i}}{K}$. Then $0 \le p_i' \le 1$ for $1 \le i \le q$, and $\sum_{i=1}^{q} p_i' = 1$.

Hence by Gibbs' inequality,

$$\sum_{i=1}^{q} p_i \log_2 \frac{p_i}{p_i'} \ge 0\,.$$

$$\text{That is,} \quad -\sum_{i=1}^{q} p_i \log_2 p_i \le -\sum_{i=1}^{q} p_i \log_2 p_i'$$

$$\begin{aligned} \text{so} \quad H(S) &\le -\sum_{i=1}^{q} p_i \log_2 \left(\frac{2^{-\ell_i}}{K}\right) \\ &= \sum_{i=1}^{q} p_i \log_2 K + \sum_{i=1}^{q} p_i \log_2 2^{\ell_i} \\ &= \log_2 K + \sum_{i=1}^{q} p_i \ell_i \\ &= \log_2 K + L \end{aligned}$$

Now $K \le 1$ so $\log_2 K \le 0$ and hence $\quad H(S) \le L$.

Hence we have proved:

Theorem 4.2 : First Source-coding Theorem

Any binary UD-code for the source S satisfies

$$H(S) \le L$$

with equality if and only if $K = 1$ and $p_i = 2^{-\ell_i}$ for all i (that is, the code is efficient and S has probabilities which are all powers of 2).

Notes:

1. L is measured in bits = binary digits/symbol

   H is measured in bits = binary units/symbol

   so we now see the connection between bits and bits.

2. This result is the reason that $\log_2$ is used for entropy.

3. For radix r, the corresponding result is:

   if $H_r(S) = -\sum_{i=1}^{q} p_i \log_r p_i$ and L_r = average length of radix r UD-code then

$$H_r(S) \le L_r .$$

   This result explains the use of the $\log_r$ factor for radix r entropy.

4.4 Shannon-Fano Coding

Huffman coding is the best UD-coding, but has the disadvantage that all the probabilities have to be known before *any* codeword can be found and also that there is no *formula* for the length of the codewords.

Shannon-Fano coding pre-dates Huffman and is more useful in the above respects. It is constructed as follows:

Given p_i, define the integer $\ell_i = \lceil -\log_2 p_i \rceil$. Then

$$-\log_2 p_i \le \ell_i < -\log_2 p_i + 1$$

We will use this ℓ_i as the length for *any* symbol having probability p_i and construct a corresponding I-code (using a decision tree) which is called a **Shannon-Fano code** or SF-code. First we must check that an I-code with these parameters exists, by verifying that the Kraft-McMillan inequality holds.

We have

$$-\log_2 p_i \le \ell_i < -\log_2 p_i + 1$$

Now, raise 2 to this power, giving

$$\frac{1}{p_i} \le 2^{\ell_i} < \frac{2}{p_i} \; .$$

Next, invert these inequalities to give

$$\frac{p_i}{2} < \frac{1}{2^{\ell_i}} \le p_i$$

and finally sum the terms to give

$$\frac{1}{2}\sum_{i=1}^{q} p_i < \sum_{i=1}^{q} \frac{1}{2^{\ell_i}} \le \sum_{i=1}^{q} p_i \; ,$$

$$\text{that is,} \qquad \frac{1}{2} < K \le 1 \; .$$

Hence these are the lengths of a legitimate UD-code, and we can construct the SF-code using a decision tree.

To find the length ℓ_i without the help of a calculator, use the following:

$$2^{\ell_i - 1} < \frac{1}{p_i} \le 2^{\ell_i} .$$

Example 4.5

Huff code	p_i	$1/p_i$	SF ℓ_i	standard SF code
01	0.3	10/3	2	00
11	0.2	5	3	010
000	0.2	5	3	011
001	0.1	10	4	1000
100	0.1	10	4	1001
101	0.1	10	4	1010

So the average length of the SF-code is

$$L_{\text{SF}} = 2 \times 0.3 + \cdots + 4 \times 0.1 = 3.0$$

compared with $L_{\text{Huff}} = 2.5$ and $H(S) = 2.446$.

The SF-codes are normally *quite inefficient*,
for example, the above has $K = \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16} = \frac{11}{16}$ and decision tree in figure 4.1.

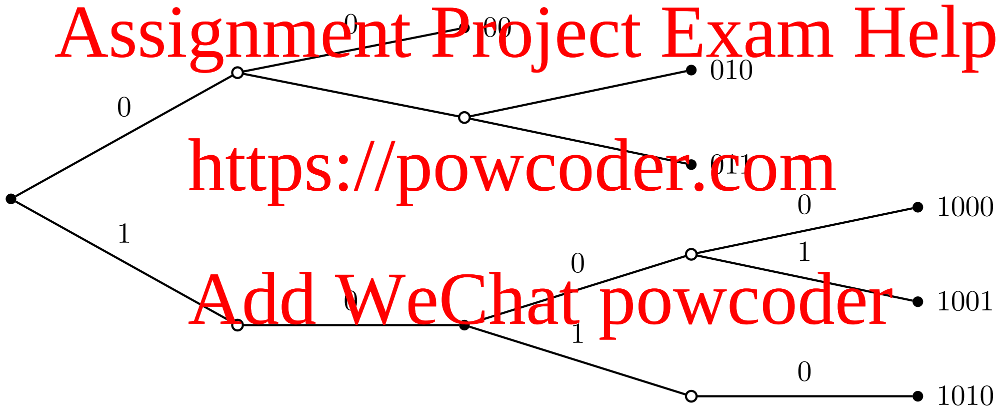


Figure 4.1: Descision tree for Shannon-Fano example

Notes

1. Since
$$-\log_2 p_i \le \ell_i < -\log_2 p_i + 1 \ ,$$
multiplying by p_i and summing gives
$$-\sum_{i=1}^{q} p_i \log_2 p_i \le \sum_{i=1}^{q} p_i \ell_i < -\sum_{i=1}^{q} p_i \log_2 p_i + \sum_{i=1}^{q} p_i$$
$$\text{i.e.} \qquad H \le L_{\text{SF}} < H + 1.$$
Also, we know that Huffman is better than SF and that any binary UD-code has expected length at least H. Therefore
$$H \le L_{\text{Huff}} \le L_{\text{SF}} < H + 1. \qquad\qquad (*)$$

2. For radix r, we similarly have
$$H_r(S) \le L_{r,\text{Huff}} \le L_{r,\text{SF}} < H_r(S) + 1.$$
The radix r Shannon-Fano lengths satisfy

$$-\log_r p_i \le \ell_i < -\log_r p_i + 1$$
or equivalently,

$$r^{\ell_i - 1} < \frac{1}{p_i} \le r^{\ell_i}$$
to find ℓ_i without the use of a calculator.

4.5 Entropy of extensions for memoryless sources

For the rest of this chapter, we write log to mean $\log_2$.

Let n be a positive integer. Consider S^n, the nth extension of S, having symbols

$$\sigma_i = s_{i_1}s_{i_2}\cdots s_{i_n} \qquad \text{for } \; i = 1, 2, \cdots, q^n$$

of probability

$$P(\sigma_i) = p_{i_1}p_{i_2}\cdots p_{i_n}.$$

The entropy of S^n is

$$\begin{aligned}
H(S^n) &= -\sum_{i=1}^{q^n} P(\sigma_i)\log P(\sigma_i) \\
&= -\sum_{i_1=1}^{q}\sum_{i_2=1}^{q}\cdots\sum_{i_n=1}^{q}(p_{i_1}\cdots p_{i_n})\log(p_{i_1}\cdots p_{i_n}) \\
&= -\sum_{i_1=1}^{q}\cdots\sum_{i_n=1}^{q} p_{i_1}\cdots p_{i_n}\left(\sum_{k=1}^{n}\log p_{i_k}\right) \\
&= -\sum_{k=1}^{n}\left[\left(\sum_{i_1=1}^{q} p_{i_1}\right)\cdots\underbrace{\left(\sum_{i_k=1}^{q} p_{i_k}\log p_{i_k}\right)}_{k^{\text{th}}\text{ term only}}\cdots\left(\sum_{i_n=1}^{q} p_{i_n}\right)\right] \\
&= -\sum_{k=1}^{n}\sum_{i_k=1}^{q} p_{i_k}\log p_{i_k} \qquad \text{as} \quad \sum_{i=1}^{q} p_i = 1 \\
&= \sum_{k=1}^{n} H(S) = nH(S).
\end{aligned}$$

Hence

Theorem 4.3 : Entropy of Extensions

$$H(S^n) = nH(S).$$

Now encode S^n by an SF-code (or a Huffman code) having average length $L^{(n)}$.

Then by the previous note 1 $(*)$ we have

$$\begin{aligned}
& H(S^n) \le L^{(n)} < H(S^n) + 1 \\
\text{that is,} \quad & nH(S) \le L^{(n)} < nH(S) + 1 \\
\text{which implies} \quad & H(S) \le \underset{\substack{\uparrow \\ \text{average length per source symbol.}}}{\frac{L^{(n)}}{n}} < H(S) + \frac{1}{n}
\end{aligned}$$

Hence,

$$\text{as } n \to \infty, \qquad \frac{L^{(n)}}{n} \to H(S)$$

for both SF and Huffman codes, with the Huffman approaching the limit faster.

Thus we have proved:

Theorem 4.4 : Shannon's Source Coding Theorem (1948)

(also called Shannon's Noiseless Channel Coding Theorem, or the S-F theorem)

The average codeword length per source symbol can be made arbitrarily close to the entropy of the source by coding sufficiently large extensions.

[Again, all this similarly applies to radix r].

Example 4.6 The source $S = \{p_1 = \frac{3}{4},\ p_2 = \frac{1}{4}\}$ has $H = -\frac{3}{4}\log\frac{3}{4} - \frac{1}{4}\log\frac{1}{4} = 0.811$.

As we saw earlier, binary Huffman coding for extensions gave

$$\frac{L^{(1)}_{\text{Huff}}}{1} = 1,\quad \frac{L^{(2)}_{\text{Huff}}}{2} = 0.84,\quad \frac{L^{(3)}_{\text{Huff}}}{3} = 0.82$$

while the binary SF-code is not as good: we calculate that

	p_i	$1/p_i$	SF ℓ_i	
S^1 :	$\frac{3}{4}$	$\frac{4}{3}$	1	giving $L^{(1)}_{\text{SF}} = 1.25$
	$\frac{1}{4}$	4	2	

		p_i	$1/p_i$	SF ℓ_i	
		$\frac{9}{16}$	$\frac{16}{9}$	1	
S^2 :	2 at	$\frac{3}{16}$	$\frac{16}{3}$	3	giving $\frac{L^{(2)}_{\text{SF}}}{2} = \frac{31}{32}$
		$\frac{1}{16}$	16	4	

		p_i	$1/p_i$	SF ℓ_i	
		$\frac{27}{64}$	$\frac{64}{27}$	2	
S^3 :	3 at	$\frac{9}{64}$	$\frac{64}{9}$	3	giving $\frac{L^{(3)}_{\text{SF}}}{3} = \frac{31}{32}$
	3 at	$\frac{3}{64}$	$\frac{64}{3}$	5	
		$\frac{1}{64}$	64	6	

(The fact that $\frac{L^{(2)}_{\text{SF}}}{2} = \frac{L^{(3)}_{\text{SF}}}{3}$ in this example is a coincidence.)

4.6 Entropy for Markov sources

Recall the definitions from Section 3.8 for Markov sources.

Suppose that source symbol s_j is immediately followed by an s_i. How much extra information have we received from the symbol s_i, given that we knew the last symbol was s_j? This is called **conditional information** and is written

$$I(s_i|s_j) = -\log P(s_i \,|\, s_j) = -\log p_{ij}.$$

The **conditional entropy given** s_j is defined as

$$\begin{aligned} H(S \,|\, s_j) &= -\sum_{i=1}^{q} P(s_i \,|\, s_j) \log P(s_i \,|\, s_j) \\ &= -\sum_{i=1}^{q} p_{ij} \log p_{ij} \\ &= \text{entropy of } j^{\text{th}} \text{ column of } M. \end{aligned}$$

Now, the symbol s_j occurs with equilibrium probability p_j and so the natural definition of the **Markov entropy** or **entropy of** S **as a Markov source** is the expected value of conditional entropies:

$$\begin{aligned} H_M(S) &= \sum_{j=1}^{q} p_j H(S \,|\, s_j) \\ &= -\sum_{i=1}^{q}\sum_{j=1}^{q} p_j p_{ij} \log p_{ij} \\ &= -\sum_{i=1}^{q}\sum_{j=1}^{q} P(s_j s_i) \log P(s_i \,|\, s_j). \end{aligned}$$

If we view S without the Markov probabilities as an equilibrium source we then have the **equilibrium entropy** or **entropy of** S **as a memoryless source**

$$\begin{aligned} H_E(S) &= -\sum_{j=1}^{q} p_j \log p_j \\ &= H(S_E), \end{aligned}$$

where S_E is the equilibrium source with probabilities $p_1, \cdots, p_q$. (Here we just ignore the Markov structure.)

Example 4.7 Consider

$$M = \begin{pmatrix} 0.3 & 0.1 & 0.10 \\ 0.5 & 0.1 & 0.55 \\ 0.2 & 0.8 & 0.35 \end{pmatrix} \qquad \text{which has} \qquad \boldsymbol{p} = \frac{1}{8}\begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix}$$

so the equilibrium entropy is

$$H_E = -\frac{1}{8}\left(\log\frac{1}{8} + 3\log\frac{3}{8} + 4\log\frac{1}{2}\right) \approx 1.41$$

(compared with $L_E = 1.50$ for Huff$_E$). The Markov entropy of S is

$$\begin{aligned} H_M &= \frac{1}{8}H(S \mid s_1) + \frac{3}{8}H(S \mid s_2) + \frac{4}{8}H(S \mid s_3) \\ &= -\frac{1}{8}(0.3\log 0.3 + 0.5\log 0.5 + 0.2\log 0.2) \\ &\quad -\frac{3}{8}(0.1\log 0.1 + 0.1\log 0.1 + 0.8\log 0.8) \\ &\quad -\frac{4}{8}(0.1\log 0.1 + 0.55\log 0.55 + 0.35\log 0.35) \\ &\approx 1.20 \end{aligned}$$

(compared with $L_M = 1.36$ for Huff$_M$).

Theorem 4.5 : Theorem on Markov Entropy

For a Markov source S

$$H_M(S) \le H_E(S)$$

with equality if and only if the symbols are pairwise independent (that is, there is no Markov structure).

So if any Markov structure is present then the entropy is lowered, i.e. there is less information transmitted.

Proof. Recall that

$$P(s_j s_i) = P(s_i \mid s_j)P(s_j) = p_j p_{ij}$$

and $$\sum_{i=1}^{q}\sum_{j=1}^{q} P(s_j s_i) = 1, \quad \sum_{i=1}^{q}\sum_{j=1}^{q} p_i p_j = 1, \quad \sum_{j=1}^{q} p_j p_{ij} = p_i$$

So by Gibbs' inequality,

$$\sum_{i=1}^{q}\sum_{j=1}^{q} P(s_j s_i)\log\left(\frac{P(s_j s_i)}{p_i p_j}\right) \ge 0\ ,$$

that is, $$\sum_{i=1}^{q}\sum_{j=1}^{q} p_j p_{ij}\log\left(\frac{p_j p_{ij}}{p_i p_j}\right) \ge 0\ .$$

Therefore

$$\begin{aligned} H_M(S) = -\sum_{i=1}^{q}\sum_{j=1}^{q} p_j p_{ij}\log p_{ij} &\le -\sum_{i=1}^{q}\sum_{j=1}^{q} p_j p_{ij}\log p_i \\ &= -\sum_{i=1}^{q} p_i \log p_i \ = H_E(S). \end{aligned}$$

This says that $H_M(S) \le H_E(S)$ and equality holds if and only if $P(s_j s_i) = p_i p_j$ for all i, j (that is, if successive symbols are independent). $\square$

4.7 Extensions of Markov sources

The nth extension of a source is

$$S^n = \{\sigma_i\} = \{s_{i_1} s_{i_2} \cdots s_{i_n}\} \quad \text{having } q^n \text{ elements.}$$

This can be viewed as a Markov source and we define as before

$$\begin{aligned} H_M(S^n) &= -\sum_{i=1}^{q^n} \sum_{j=1}^{q^n} P(\sigma_j \sigma_i) \log P(\sigma_i \,|\, \sigma_j) \\ &= -\sum_{i=1}^{q^n} \sum_{j=1}^{q^n} P(\sigma_j \sigma_i) \log P(\sigma_i \,|\, s_{j_n}) \\ &\quad \text{where } s_{j_n} \text{ is the last symbol of } \sigma_j = s_{j_1} \cdots s_{j_n} \end{aligned}$$

since in a Markov source the dependence is only on the previous symbol.

Similarly we can view S^n as an equilibrium source and obtain

$$H_E(S^n) = -\sum_{i=1}^{q^n} P(\sigma_i) \log P(\sigma_i) \ .$$

Note that this is not the same as $H((S_E)^n)$.

It can be shown that

$$\begin{aligned} H_M(S^n) &= nH_M(S), \\ H_E(S^n) &= H_E(S) + (n-1)H_M(S) \end{aligned}$$

and hence

$$\begin{aligned} \frac{H_M(S^n)}{n} &= H_M(S) \\ \text{and} \qquad \lim_{n\to\infty} \frac{H_E(S^n)}{n} &= H_M(S). \end{aligned}$$

4.8 Noisy channels

We now turn our attention from the source/noiseless channels to channels with noise, and we ask how much information can be transmitted in the presence of noise. Let A be the source or **input** or sent symbols

$$A = \{a_1, a_2, \cdots, a_u\} \quad \text{of probabilities} \quad P(a_1), \cdots, P(a_u).$$

Let B be the sink or **output** or received symbols

$$B = \{b_1, b_2, \cdots, b_v\} \quad \text{of probabilities} \quad P(b_1), \cdots, P(b_v).$$

The obvious conditional probabilities are $P(b_i \,|\, a_j)$, the probability that the output is b_i when the input is a_j (that is, the probability of receiving b_i when a_j is sent).

We are thinking of these symbols as the single symbols of a codeword, so for example, a binary channel has $A = \{0, 1\}, B = \{0, 1\}$ or possibly $B = \{0, 1, ?\}$. However, the theory copes with more generality.

It is possible to form a channel matrix holding the probabilities $P(b_i \,|\, a_j)$ and use matrix notation. We gain little by this and will not bother with it. Also, there is no standard notation in the books, so be careful when reading about this topic.

Our model is thus

$$\begin{tikzcd} & \text{noise} \arrow[d] & \\ \boxed{\text{source}} \arrow[r] & \boxed{\text{channel}} \arrow[r] & \boxed{\text{output}} \end{tikzcd}$$

$$\begin{tikzcd} a_j \bullet \arrow[rr, "P(b_i \,|\, a_j)"'] & & \bullet\, b_i \end{tikzcd}$$

It is assumed that for each symbol which enters the channel, exactly one symbol comes out. That is, no symbols are lost and the same number of symbols come out as go in. However, if errors have occurred then the sequence of symbols which come out may be different to the sequence of symbols that went in.

Recall that

$$\sum_{j=1}^{u} P(a_j) = 1, \quad \sum_{i=1}^{v} P(b_i) = 1,$$

$$\sum_{i=1}^{v} P(b_i \,|\, a_j) = 1 \quad \text{for } 1 \le j \le u.$$

Now the probability that a_j went into the channel and b_i came out is written $P(a_j \text{ and } b_i)$. We have

$$P(a_j \text{ and } b_i) = P(a_j)P(b_i \,|\, a_j) = P(b_i)P(a_j \,|\, b_i).$$

Here we have the reverse probability $P(a_j \,|\, b_i)$: that is, the probability that a_j was input given that b_i was received. These are of course the probabilities of main interest when decoding.

Also

$$\begin{aligned} \sum_{j=1}^{u} P(a_j \,|\, b_i) &= 1 \quad \text{for } 1 \le i \le v, \\ \sum_{i=1}^{v}\sum_{j=1}^{u} P(a_j \text{ and } b_i) &= 1 \\ \sum_{j=1}^{u} P(a_j)P(b_i \,|\, a_j) &= P(b_i) \quad \text{for } 1 \le i \le v, \qquad (*) \\ \text{and} \quad \sum_{i=1}^{v} P(b_i)P(a_j \,|\, b_i) &= P(a_j) \quad \text{for } 1 \le j \le u \end{aligned}$$

which can be combined to give

$$P(a_j \,|\, b_i) = \frac{P(a_j)P(b_i \,|\, a_j)}{\sum_j P(a_j)P(b_i \,|\, a_j)} \qquad (*)$$

which is called **Bayes' Rule**.

The identities $(*)$ enable us to find $P(b_i)$ and $P(a_j \,|\, b_i)$ (the results of transmission) from $P(a_j)$ and $P(b_i \,|\, a_j)$ (the input and channel properties).

From all these we have some natural definitions of entropy (units = binary units/symbol).

$$\textbf{source entropy} \quad H(A) = -\sum_{j=1}^{u} P(a_j) \log P(a_j)$$

$$\textbf{output entropy} \quad H(B) = -\sum_{i=1}^{v} P(b_i) \log P(b_i)$$

Conditional entropies

$$\textbf{of } B \textbf{ given } a_j \quad H(B \,|\, a_j) = -\sum_{i=1}^{v} P(b_i \,|\, a_j) \log P(b_i \,|\, a_j)$$

$$\textbf{of } A \textbf{ given } b_i \quad H(A \,|\, b_i) = -\sum_{j=1}^{u} P(a_j \,|\, b_i) \log P(a_j \,|\, b_i)$$

which are combined as weighted averages in

$$\begin{aligned} H(B \,|\, A) &= \sum_{j=1}^{u} P(a_j) H(B \,|\, a_j) \\ &= -\sum_{i=1}^{v} \sum_{j=1}^{u} P(a_j) P(b_i \,|\, a_j) \log P(b_i \,|\, a_j) \\ &= -\sum_{i=1}^{v} \sum_{j=1}^{u} P(a_j \text{ and } b_i) \log P(b_i \,|\, a_j) \\ H(A \,|\, B) &= \sum_{i=1}^{v} P(b_i) H(A \,|\, b_i) \\ &= -\sum_{i=1}^{v} \sum_{j=1}^{u} P(b_i) P(a_j \,|\, b_i) \log P(a_j \,|\, b_i) \\ &= -\sum_{i=1}^{v} \sum_{j=1}^{u} P(a_j \text{ and } b_i) \log P(a_j \,|\, b_i) \end{aligned}$$

We can view $H(B \,|\, A)$ as a measure of the uncertainty (average information) in the output given the input, and similarly we can view $H(A \,|\, B)$ as a measure of the uncertainty in the input given the output. $H(A \,|\, B)$ is called the **equivocation** of the channel.

Finally there is the **joint entropy**

$$\begin{aligned} H(A, B) &= H(B, A) \quad \text{(also denoted } \; H(A \text{ and } B)) \\ &= -\sum_{i=1}^{v} \sum_{j=1}^{u} P(a_j \text{ and } b_i) \log P(a_j \text{ and } b_i) \end{aligned}$$

Now by manipulating sums.

$$\begin{aligned} H(A, B) &= -\sum_{i=1}^{v} \sum_{j=1}^{u} P(a_j) P(b_i \,|\, a_j) \log[P(a_j) P(b_i \,|\, a_j)] \\ &= -\sum_{j} \underbrace{\sum_{i=1}^{v} P(b_i \,|\, a_j)}_{=1} P(a_j) \log P(a_j) - \sum_{i=1}^{v} \sum_{j} P(a_j) P(b_i \,|\, a_j) \log P(b_i \,|\, a_j) \\ &= H(A) + H(B \,|\, A) \end{aligned}$$

and similarly $H(A,B) = H(B) + H(A \mid B)$.

Example. If A and B are *independent* then for $1 \le i \le v$, $1 \le j \le u$,

$$P(a_j \mid b_i) = P(a_j) \quad \text{and} \quad P(b_i \mid a_j) = P(b_i)$$

and so

$$H(A \mid B) = H(A), \quad H(B \mid A) = H(B)$$

and

$$H(A,B) = H(A) + H(B).$$

This case is of course *useless* as a channel since we do not want to use a channel whose output is independent of its input. □

More generally

$$H(B \mid A) = H(A,B) - H(A)$$

is the *gain* in uncertainty going from the source to the joint situation, and is called the **noise entropy.**

Similarly the equivocation satisfies $H(A \mid B) = H(A,B) - H(B)$.

We also have

$$H(A) - H(A \mid B) = H(B) - H(B|A)$$

and we call this common value $I(A,B) = I(B,A)$, the **mutual information** between A and B.

$I(A,B)$ equals the average information in B about A and also equals the average information in A about B.

Now

$$\begin{aligned}
& I(A,B) \\
= \; & H(A) + H(B) - H(A,B) \\
= \; & -\sum_{j=1}^{u} P(a_j) \log P(a_j) - \sum_{i=1}^{v} P(b_j) \log P(b_j) + \sum_{i=1}^{v}\sum_{j=1}^{u} P(a_j \text{ and } b_i) \log P(a_j \text{ and } b_i) \\
= \; & -\sum_{j=1}^{u} P(a_j) \underbrace{\sum_{i=1}^{v} P(b_i \mid a_j)}_{=1} \log P(a_j) - \sum_{i=1}^{v} P(b_i) \underbrace{\sum_{j=1}^{u} P(a_j \mid b_i)}_{=1} \log P(b_i) \\
& + \sum_{i=1}^{v}\sum_{j=1}^{u} P(a_j \text{ and } b_i) \log P(a_j \text{ and } b_i)
\end{aligned}$$

However, $P(a_j \text{ and } b_i) = P(a_j)P(b_i \mid a_j) = P(b_i)P(a_j \mid b_i)$
and so (by Gibbs' inequality),

$$I(A,B) = \sum_{i=1}^{v}\sum_{j=1}^{v} P(a_j \text{ and } b_i) \log\left(\frac{P(a_j \text{ and } b_i)}{P(a_j)P(b_i)}\right) \ge 0.$$

Hence $I(A,B) \ge 0$

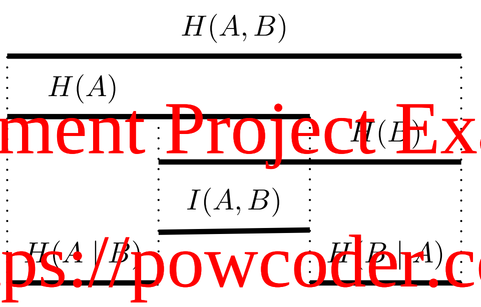
and so $H(A \mid B) \le H(A)$ (that is, equivocation $\le$ source entropy),
$H(B \mid A) \le H(B)$ (that is, noise entropy $\le$ output entropy).

This means that uncertainty in A decreases once you have seen B and vice versa, and $I(A,B) =$ loss in entropy in A when B is seen and vice versa.

$$\begin{aligned}
\text{Also} \qquad I(A,B) &= -\sum_{i=1}^{v}\sum_{j=1}^{u} P(a_j \text{ and } b_i) \log \frac{P(a_j \mid b_i)}{P(a_j)} \\
&= -\sum_{i=1}^{v}\sum_{j=1}^{u} P(a_j \text{ and } b_i) I(a_j, b_i) \\
\text{where} \qquad I(a_j, b_i) &= I(a_j) - I(a_j \mid b_i) \\
&= (-\log P(a_j)) - (-\log P(a_j \mid b_i)) \\
&= I(b_i) - I(b_i \mid a_j)
\end{aligned}$$

hence the name **mutual information**.

We can picture the relationships between the entropies using the following diagram:



Here, the length of each lines show how big the entropies are, relative to each other. For instance, the sum of line lengths representing H(A) and H(B—A) are just as the length of the line representing H(A,B), which illustrates the identity H(A,B) = H(A) + H(B—A).

Notes:

1. Most books calculate all these entropies in units of bits/symbol and *so will we.* (It would perhaps be more natural to use a radix equal to the number of input symbols since these are often coming straight from encoding the source, but we will not do this as often the numbers of input and output symbols will differ.)

2. It can be shown that the equivocation satisfies

$$H(A \mid B) \leq H(p_e) + p_e \log(u-1)$$

where u is the number of input symbols and p_e is the average probability of error per symbol and hence p_e can be estimated from $H(A \mid B)$.

This is called the **Fano bound.**

3. For binary channels it will be convenient to define the function

$$H(x) = -x \log x - (1-x)\log(1-x)$$

(where as usual, the logarithms are to base 2). This expression occurs often, so we give it this "name" $H(x)$. See the problems, where this function is investigated.

Before an example, let us consider what these entropies actually mean:

Given a source A we need to provide (at least) an average $H(A)$ bits/symbol to represent the symbols.

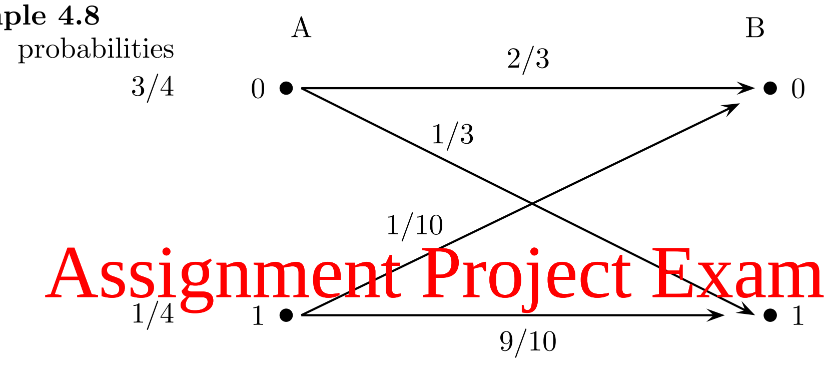
If these symbols are transmitted through a channel and the output B is observed then we need only $H(A \,|\, B)$ bits/symbol to represent the input A.

So the reception of the output has provided $I(A,B) = H(A) - H(A \,|\, B)$ bits/symbol; that is, we have transmitted $I(A,B)$ bits/symbol.

If thechannel is noiseless then $H(A \,|\, B) = 0$ and each output symbol contains $H(A)$ bits, so we receive the $H(A)$ bits error-free.

On the other hand a noisy channel allows only $H(A) - H(A \,|\, B) < H(A)$ bits/symbol through and so, since we need $H(A)$ bits to reconstruct the source then we are $H(A \,|\, B)$ bits short. So although some information gets through, it will not be error-free.

Example 4.8



Here

$$
\begin{aligned}
P(a_1) &= 3/4, & P(a_2) &= 1/4,\\
P(b_1 \mid a_1) &= 2/3, & P(b_2 \mid a_1) &= 1/3,\\
P(b_1 \mid a_2) &= 1/10, & P(b_2 \mid a_2) &= 9/10
\end{aligned}
$$

Hence, using the function $H(x)$ defined above,

$$
\begin{aligned}
H(A) &= H(\tfrac{3}{4}) = -\frac{3}{4}\log\frac{3}{4} - \frac{1}{4}\log\frac{1}{4} = 0.8113\\
H(B \,|\, a_1) &= H(\tfrac{2}{3}) = -\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3} = 0.9183\\
H(B|a_2) &= H(\tfrac{1}{10}) = -\frac{1}{10}\log\frac{1}{10} - \frac{9}{10}\log\frac{9}{10} = 0.4690\\
\text{so} \qquad H(B \,|\, A) &= \frac{3}{4}H(\tfrac{2}{3}) + \frac{1}{4}H(\tfrac{1}{10})\\
&= \frac{3}{4}(0.9183) + \frac{1}{4}(0.4690) = 0.8060
\end{aligned}
$$

$$
\begin{aligned}
\text{Therefore} \qquad H(A,B) &= H(A) + H(B \,|\, A) = 0.8113 + 0.8060 = 1.6173.\\
\text{Also} \qquad P(b_1) &= \frac{3}{4}\cdot\frac{2}{3} + \frac{1}{4}\cdot\frac{1}{10} = \frac{21}{40},\\
\text{so} \qquad P(b_2) &= \frac{19}{40}\\
\text{and} \qquad H(B) &= H(\tfrac{19}{40}) = -\frac{21}{40}\log\frac{21}{40} - \frac{19}{40}\log\frac{19}{40} = 0.9982.\\
\text{Hence} \qquad I(A,B) &= H(B) - H(B \,|\, A) = 0.9982 - 0.8060 = 0.1922.\\
\text{Hence} \qquad H(A \,|\, B) &= H(A) - I(A,B) = 0.8113 - 0.1922 = 0.6191.
\end{aligned}
$$

Alternatively, we can calculate directly using Bayes' Rule:

$$P(a_1 \,|\, b_1) = \frac{\frac{3}{4}\cdot\frac{2}{3}}{\frac{21}{40}} = \frac{20}{21} \qquad \text{so } P(a_2 \,|\, b_1) = \frac{1}{21}$$

$$P(a_1 \,|\, b_2) = \frac{\frac{3}{4}\cdot\frac{1}{3}}{\frac{19}{40}} = \frac{10}{19} \qquad \text{so } P(a_2 \,|\, b_2) = \frac{9}{19}.$$

Therefore

$$H(A \,|\, b_1) = H(\tfrac{20}{21}) = -\frac{20}{21}\log\frac{20}{21} - \frac{1}{21}\log\frac{1}{21} = 0.2762$$

$$H(A \,|\, b_2) = H(\tfrac{9}{19}) = -\frac{9}{19}\log\frac{9}{19} - \frac{10}{19}\log\frac{10}{19} = 0.9980$$

$$\begin{aligned} \text{so} \qquad H(A \,|\, B) &= \frac{21}{40}H(\tfrac{20}{21}) + \frac{19}{40}H(\tfrac{9}{19}) \\ &= \frac{21}{40}(0.2762) + \frac{19}{40}(0.9980) = 0.6190 \end{aligned}$$

which agrees up to rounding errors.

4.9 Channel Capacity

Suppose the the channel probabilities, $P(b_j \,|\, a_i)$, are given. The **channel capacity** is defined to be

$$C = C(A, B) = \max\ I(A, B)$$

where the maximum is taken over all possible probabilities of the source symbols A.

This is tricky to calculate in general, but for a binary source it is only a single variable problem and so is reasonable.

The BSC (Binary Symmetric Channel) with crossover probability p (fixed) and variable probablity of a 0-symbol of x is shown in figure 4.2

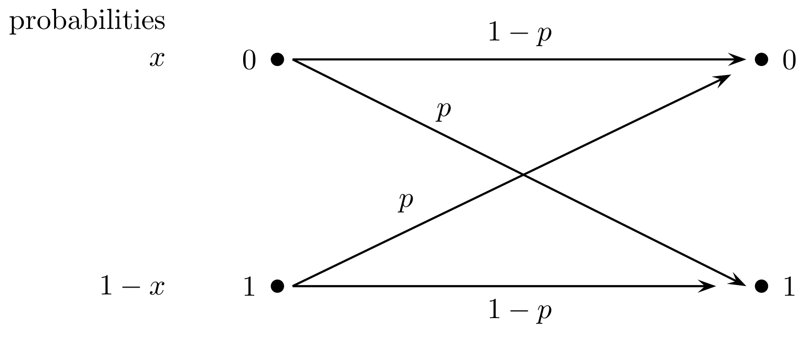


Figure 4.2: binary symmetric channel

We will continue to use the function

$$H(x) = -x\log x - (1-x)\log(1-x)$$

to write down the various entropies. In particular $H(A) = H(x)$, since $P(a_1) = x$ and $P(a_2) = 1-x$.

We calculate

$$\begin{aligned}
H(B \,|\, a_1) &= -(1-p)\log(1-p) - p\log p = H(p)\\
H(B \,|\, a_2) &= -p\log p - (1-p)\log(1-p) = H(p)\\
\text{so}\quad H(B \,|\, A) &= xH(p) + (1-x)H(p) = H(p)\\
\text{so}\quad H(A,B) &= H(x) + H(p).\\
\text{Also}\quad P(b_1) &= x(1-p) + (1-x)p\\
&= x(1-2p) + p = y \quad \text{say}\\
\text{so}\quad P(b_2) &= 1-y.\\
\text{Hence}\quad H(B) &= -y\log y - (1-y)\log(1-y) = H(y)\\
\text{and finally}\quad I(A,B) &= H(y) - H(p)\\
&= H(x(1-2p)+p) - H(p)
\end{aligned}$$

Now we need to maximise this over all possible choices of x which we do by calculus. From Problem Sheet 4, we know that $H(y)$ has maximum of 1 when $y = \dfrac{1}{2}$.

Hence $I(A,B)$ has maximum of $1 - H(p)$.

That is, the **channel capacity for BSC is** $1 - H(p)$.

Also when $y = \frac{1}{2}$ then $x(1-2p) + p = \frac{1}{2}$ so $x = \frac{\frac{1}{2}-p}{1-2p} = \frac{1}{2}$ (provided $p \neq \frac{1}{2}$) and the capacity is reached for equally likely source symbols, whatever p is.

From the graph of $H(p)$ we get the graph of $C = 1 - H(p)$ in figure 4.3.

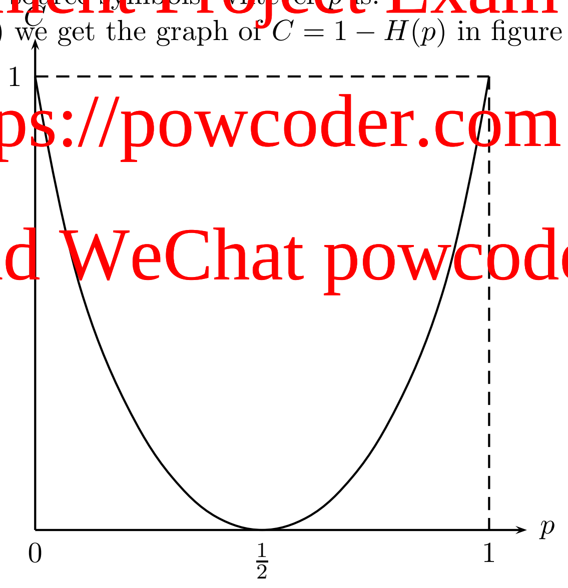
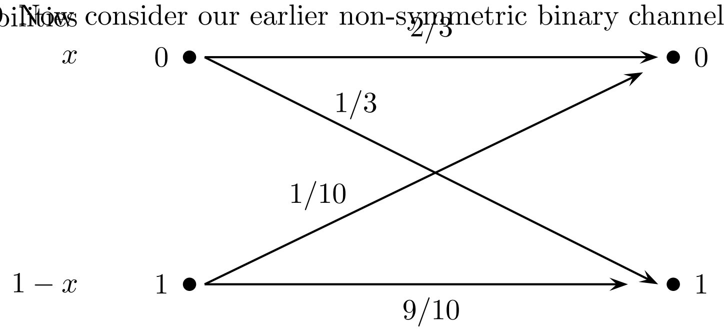


Figure 4.3: Channel Capacity for BSC

When $p = \frac{1}{2}$ then the capacity is 0. In this case $I(A,B) = 0$ no matter what value x takes, so nothing gets through the channel — the output is *independent* of the input.

The channel capacity achieves its maximum of 1 when $p = 0$ (that is, no errors), or $p = 1$ (that is, 100% error, so *every* bit is flipped).

Example 4.9 Now consider our earlier non-symmetric binary channel with probabilities



Now, calculating (some done before) gives

$$\begin{aligned}
H(A) &= -x\log x - (1-x)\log(1-x) = H(x),\\
H(B\,|\,a_1) &= 0.9183,\\
H(B\,|\,a_2) &= 0.4690\\
H(B\,|\,A) &= x(0.9183) + (1-x)(0.4690)\\
&= 0.4493x + 0.4690\\
P(b_1) &= \frac{2}{3}x + \frac{1}{10}(1-x) = \frac{3+17x}{30} = y \quad \text{say}\\
\text{and } P(b_2) &= \frac{27-17x}{30} = 1-y.
\end{aligned}$$

$$\begin{aligned}
\text{So}\quad H(B) &= -y\log y - (1-y)\log(1-y) = H(y)\\
\text{and}\quad I(A,B) &= H(y) - 0.4493x - 0.4690\\
\text{where}\quad y &= \frac{3+17x}{30}\\
\text{so}\quad \frac{dy}{dx} &= \frac{17}{30}
\end{aligned}$$

Also recall

$$\begin{aligned}
\frac{d}{dx}\log x &= \frac{1}{\ln 2}\times\frac{1}{x}\\
\text{so}\quad \frac{d}{dx}I(A,B) &= \frac{d}{dx}(-y\log y - (1-y)\log(1-y) - 0.4493x - 0.4690)\\
&= -\frac{17}{30}\log y - y\times\frac{1}{\ln 2}\times\frac{1}{y}\times\frac{17}{30}\\
&\quad + \frac{17}{30}\log(1-y) + (1-y)\times\frac{1}{\ln 2}\times\frac{1}{1-y}\times\frac{17}{30} - 0.4493\\
&= \frac{17}{30}\log\left(\frac{1-y}{y}\right) - 0.4493
\end{aligned}$$

For a maximum we set this equal to 0

$$\begin{aligned}
\text{so}\qquad \frac{17}{30}\log\left(\frac{1-y}{y}\right) &= 0.4493\\
\log\left(\frac{1-y}{y}\right) &= 0.7929\\
\frac{1-y}{y} &= 2^{0.7929} = 1.7326\\
y &= \frac{1}{2.7326} = 0.3660\\
\text{and}\qquad x &= \frac{1}{17}(30y-3) = 0.4693
\end{aligned}$$

Hence the channel capacity C, which is the maximum value of $I(A,B)$, is given by

$$\begin{aligned}
C &= -y\log y - (1-y)\log(1-y) - 0.4493x - 0.4690\\
&= 0.5307 + 0.4168 - 0.2109 - 0.4690\\
&= 0.2676
\end{aligned}$$

This is well above the value $I(A,B) = 0.1922$ obtained with $x = \frac{3}{4}$.

Notes:

1. It can be shown that $I(A, B)$ is always a convex function of the input/source probabilities (see e.g. Ash). Hence, if there is an isolated critical point it must be a maximum. This applies no matter how many source symbols there are. Hence, you may *assume* that the critical point which you find is a maximum, without checking.

2. There are several interesting results that can be proved for our definitions. For example, one can show that if the output of a memoryless channel of capacity C_1 is fed into another channel of capacity C_2, the capacity of the combination is bounded by $\min(C_1, C_2)$. In other words, *information cannot be increased by data processing*, see Ash.

4.10 Shannon's Noisy Channel Coding Theorem

Theorem 4.6 : Shannon's Noisy Channel Coding Theorem

Let $\varepsilon > 0$ be a small constant and consider a channel with channel capacity $C = \max I(A, B)$. Given any *positive number $R < C$, there exists a code with information rate R such that the probability of error after decoding the output is less than ε.*

This means that there exist codes that are arbitrarily reliable and which can be used to transmit information at a rate arbitrarily close to the channel capacity.

Proof: skipped – see Hamming's book for a 12 page proof for the special case of the BSC, or Ash's book for two different proofs.

Shannon's original proof (which Ash gives) is an existence proof that works by averaging over random codes of very long length with high error correcting capability.

Ash's other proof consists of an (unfortuantely impractical) algorithm for constructing the code. Ash also proves that for a binary channel, parity check codes will suffice.

A partial converse to this (see Ash) can be found from Fano's bound and is:

If we try to transmit information at a rate $R = C + \epsilon$ then p_e is bounded away from zero.

Hence essentially:

the channel capacity is the maximum rate at which reliable transmission of information can take place.

4.11 Entropy of natural language

See Appendix on	"Brown corpus" statistics	Appendix A.3
	Probability distribution of English	Appendix A.4

The Brown Corpus is a selection of 500 two thousand word selections of "natural- language" ranging through English fiction, newspapers, scientific papers, bibliographies, C programs, computer source code, bit map graphics, etc.

Various models of the structure of this corpus have been examined and give

	Entropy	
Model	**94-char**	**26-char**
Equally-likely symbols	$\log_2 94 = 6.55$	$\log_2 26 = 4.70$
Relative frequency model	4.47	4.14
1-memory (Markov)	3.59	3.56
2-memory	2.92	3.30
3-memory	2.33	

Many different texts agree that "English" can be thought of as a 30-memory source with "true entropy" of 1.5 bits/letter. We will accept this value of 1.5 bits/letter as the true entropy value for English.

Chapter 5

Number Theory and Algebra

5.1 Revision of Discrete Mathematics

See section 1.2 for the basics of modular arithmetic and primes.

Division Algorithm

Given $a \in \mathbb{Z}$, $b \in \mathbb{Z}^+$ there exist unique integers q, r with

$$a = bq + r, \qquad 0 \le r < b.$$

This is called the **division algorithm**.

We say that q is the **quotient** and r is the **remainder** when a is divided by b.

For example

12 and 7 have $12 = 7 \cdot 1 + 5$ so $q = 1$, $r = 5$,

-12 and 7 have $-12 = 7 \cdot (-2) + 2$ so $q = -2$, $r = 2$.

The remainder is also denoted by a **mod** b where **mod** is an operator on a and b.

gcd and Euclidean Algorithm

Given $a, b \in \mathbb{Z}$, not both equal to zero, there exists a unique $d \in \mathbb{Z}^+$ called the **greatest common divisor** of a and b and written $d = \gcd(a, b)$, defined by

$d = \gcd(a, b)$ if and only if

(i) $d \mid a,\ d \mid b$ (d is a *common divisor*),
(ii) if $c \mid a$ and $c \mid b$, then $c \mid d$ (d is the greatest such).

For example, $\gcd(12, 18) = 6$. Note that $\gcd(a, 0) = a$ for any nonzero integer a.

The **Euclidean Algorithm** can be used to find $d = \gcd(a, b)$. Moreover, it can be used to find $x, y \in \mathbb{Z}$ such that

$$d = \gcd(a, b) = ax + by.$$

This equation is known as **Bezout's Identity**. An alternative definition of gcd is that $\gcd(a, b)$ is the least positive number in the set $\{ax + by : x, y \in \mathbb{Z}\}$.

We say that a and b are **relatively prime** if and only if $\gcd(a, b) = 1$, which is the same as saying that a and b have no proper factors in common.

For example, 7, 12 are relatively prime as $\gcd(7, 12) = 1$.

Euclidean Algorithm:
divide b into a giving quotient q_1 and remainder r_1, then

divide r_1 into b giving quotient q_2 and remainder r_2, then
divide r_2 into r_1 giving quotient q_3 and remainder r_3
and so on,
until the remainder is zero.

Then d equals the last non-zero remainder and x, y are found by back-substitution.

Example

Find d, x, y with $\gcd(1547, 560) = d = 1547\,x + 560\,y$.
(Note: remainders are underlined so that we can keep track of them.)

$$\begin{aligned}
\underline{1547} &= \underline{560}\cdot 2 + \underline{427}\\
\underline{560} &= \underline{427}\cdot 1 + \underline{133}\\
\underline{427} &= \underline{133}\cdot 3 + \underline{28}\\
\underline{133} &= \underline{28}\cdot 4 + \underline{21}\\
\underline{28} &= \underline{21}\cdot 1 + \underline{7}\\
\underline{21} &= \underline{7}\cdot 3.
\end{aligned}$$

Hence $d = 7$. Then

$$\begin{aligned}
7 &= \underline{28} - \underline{21}\\
&= \underline{28} - (\underline{133} - 4\cdot\underline{28}) = 5\cdot\underline{28} - \underline{133}\\
&= 5(\underline{427} - 3\cdot\underline{133}) - \underline{133} = 5\cdot\underline{427} - 16\cdot\underline{133}\\
&= 5\cdot\underline{427} - 16(\underline{560} - \underline{427}) = 21\cdot\underline{427} - 16\cdot\underline{560}\\
&= 21(\underline{1547} - 2\cdot\underline{560}) - 16\cdot\underline{560}\\
&= 21\cdot\underline{1547} - 58\cdot\underline{560}\\
\text{so}\qquad 7 &= 1547(21) + 560(-58)\\
\text{and}\qquad x &= 21,\ y\ =\ -58.
\end{aligned}$$

Note: the x, y in $d = ax + by$ are not unique since

$$7 = 1547(21 + 80t) + 560(-58 - 221t),\ \text{ for any } t \in \mathbb{Z}.$$

Here $80 = \dfrac{560}{7}$, $\quad 221 = \dfrac{1547}{7}$ and we note that $1547(80t)\ +\ 560(-221t) = 0$.

Extended Euclidean Algorithm:
A more efficient way of finding the numbers in Bezout's Identity is as follows: In looking for $\gcd(a, b)$, assume $a > b > 0$. We make up a table with four columns labelled q_i, r_i, x_i and y_i, where i labels the rows, starting from -1. We set the first (-1) row to be $(0, a, 1, 0)$ and the second (zeroth) to be $(0, b, 0, 1)$. Thus $q_1 = q_2 = 0$, $r_1 = a$ and $r_2 = b$. Then for i from 1 onwards, set q_{i+1} to be the quotient on dividing r_{i-1} by r_i (a divided by b in the first case). Then subtract q_{i+1} times the rest of row i from row $i - 1$ (ignoring the q_j terms). Repeat until we get $r_{n+1} = 0$ for some n (we do not need to find x_{n+1} and y_{n+1}). Then the gcd is r_n and $r_n = ax_n + by_n$, that is the last row *before* r_i was zero gives the gcd, the x and the y. In fact a similar identity holds at each step: $r_i = ax_i + by_i$. This algorithm is called the **extended Euclidean Algorithm**

Example: We find the gcd of 1788 and 1492:

i	q_i	r_i	x_i	y_i
-1	0	1788	1	0
0	0	1492	0	1
1	1	296	1	-1
2	5	12	-5	6
3	24	8	121	-145
4	1	4	-126	151
5	2	0		

So $\gcd(1788, 1492) = 4 = -126 \times 1788 + 151 \times 1492$. Note that $r_i = ax_i + by_i$ at each stage.

Both the Euclidean and extended Euclidean algorithm are computationally fast.

Inverses

Recall that for $a \in \mathbb{Z}_m$, the **inverse** of a, denoted by a^{-1} (if it exists), is the solution to $ax = 1$ in $\mathbb{Z}_m$ if a solution exists.

Now $a^{-1} \in \mathbb{Z}_m$ exists if and only if $\gcd(a, m) = 1$. For small m we can find a^{-1}, if it exists, by trying all possibilities, but more systematically, a^{-1} can be found by the (extended) Euclidean Algorithm for $\gcd(a, m) = 1$ since $1 = ax + my \iff ax = 1 \in \mathbb{Z}_m$.

Hence, division by a is possible if and only if $\gcd(a, m) = 1$.

For example, in $\mathbb{Z}_{12}$

9^{-1} does not exist since $\gcd(9, 12) \neq 1$
5^{-1} does exist since $\gcd(5, 12) = 1$
We find 5^{-1} by the Euclidean Algorithm:

$$\begin{aligned} \underline{12} &= \underline{5} \cdot 2 + \underline{2} \\ \underline{5} &= \underline{2} \cdot 2 + \underline{1} \\ \underline{2} &= \underline{1} \cdot 2 \\ \underline{1} &= \underline{5} - \underline{2} \cdot 2 \\ &= \underline{5} - 2(\underline{12} - \underline{5} \cdot 2) \\ &= \underline{5} \cdot 5 - 2 \cdot \underline{12} \end{aligned}$$

so $5x = 1$ when $x = 5$ and hence $5^{-1} = 5$ in $\mathbb{Z}_{12}$.
Alternatively 5^{-1} in $\mathbb{Z}_{12}$ can be found by trying each of $x = 0, 1, \cdots, 11$ in turn.

An equation like $ax = b$ in $\mathbb{Z}_m$ has solutions for given a and b if and only if $\gcd(a, m) = d \mid b$. If $\gcd(a, m) = d \mid b$, then the equation has d solutions.
For example, in $\mathbb{Z}_{12}$ the equation $9x = 6$ has solutions since $\gcd(9, 12) = 3$ and $3 \mid 6$. By trial and error we find that the solutions are $x = 2$, 6, 10 in $\mathbb{Z}_{12}$: a systematic way to solve these equation is taught in MATH1081.

All elements in the set $\mathbb{Z}_m - \{0\} = \{1, 2, 3, \cdots, m-1\}$ have inverses
if and only if $\gcd(a, m) = 1$ for all $a \in \{1, 2, 3, \cdots, m-1\}$
if and only if m is a prime.

In this case every non-zero element has an inverse and hence division by multiplying by inverses is possible.

Some people write

$$a/b = a \div b \qquad \text{for} \quad ab^{-1} \in \mathbb{Z}_m, \qquad \text{but we will NOT do this.}$$

A set in which every nonzero element has an inverse and the operations $+$, $-$, $\cdot$ and multiplication by inverses all obey the usual rules (closure, associativity, commutativity, distributivity) is called a **field**.

So $\mathbb{Z}_m$ is a field if and only if m is prime and we usually write $\mathbb{Z}_p$ in this case.

Chinese Remainder Theorem

We often find situations where a problem is hard to solve modulo a composite number, but much easier to solve modulo the prime factors. In many cases we can then use the following result to bootstrap from the factors to the product:

Theorem 5.1 : Chinese Remainder Theorem *Let p and q be coprime. Then for any m_1 and m_2, the coupled equations*

$$x \equiv m_1 \pmod{p}, \qquad x \equiv m_2 \pmod{q}$$

have a unique common solution modulo pq given by

$$x = m_1 + ps(m_2 - m_1) \pmod{pq},$$

where s is the inverse of p modulo q.

This is actually easy to prove: there are integers s and t such that $ps + qt = 1$. Then $x = m_2ps + m_1qt$ is one common solution to the given equations and

$$x = m_2ps + m_1qt = m_2ps + m_1(1 - ps) = m_1 + ps(m_2 - m_1).$$

Given any second solution y, $x - y$ is zero modulo both p and q and so is multiple of pq since p and q are coprime.

5.2 Number Theory Results (no proofs given)

Given $m \in \mathbb{Z}^+$, the set of invertible elements in $\mathbb{Z}_m$ is denoted by

$$\mathbb{U}_m = \{a \in \mathbb{Z}_m : \gcd(a, m) = 1\}$$

and its elements are called **units** in $\mathbb{Z}_m$. For example, $\mathbb{Z}_{12} = \{0, 1, 2, \cdots, 11\}$ has units $\mathbb{U}_{12} = \{1, 5, 7, 11\}$.

If p is prime, then $\mathbb{U}_p = \mathbb{Z}_p - \{0\}$.

Euler's **phi-function** is defined by

$$\begin{aligned} \phi(m) &= |\mathbb{U}_m| \\ &= \#\,\text{units} \in \mathbb{Z}_m \\ &= \#\,\{a : 1 \le a < m, \quad \gcd(a, m) = 1\} \\ \text{e.g.} \quad \phi(12) &= 4 \\ \text{Note} \quad \phi(p) &= p - 1 \qquad \text{for any prime } p. \end{aligned}$$

It can be shown that

Theorem 5.2 : Formula for $\phi(m)$

1. *If $gcd(m,n) = 1$, then $\phi(mn) = \phi(m)\phi(n)$.*
2. *For a prime p and $\alpha \in \mathbb{Z}^+$, we have $\phi(p^\alpha) = p^\alpha - p^{\alpha-1}$.*
3. *Hence, if $m = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_r^{\alpha_r}$ is the prime factorization of m, then*

$$\begin{aligned}\phi(m) &= (p_1^{\alpha_1} - p_1^{\alpha_1 - 1})(p_2^{\alpha_2} - p_2^{\alpha_2 - 1}) \cdots (p_r^{\alpha_r} - p_r^{\alpha_r - 1}) \\ &= p_1^{\alpha_1}(1 - \frac{1}{p_1})\, p_2^{\alpha_2}(1 - \frac{1}{p_2}) \cdots p_r^{\alpha_r}(1 - \frac{1}{p_r}) \\ &= m \prod_{p|m}(1 - \frac{1}{p})\end{aligned}$$

For example,

$$\begin{aligned}\phi(12) = \phi(2^2 \cdot 3) &= (2^2 - 2^1)(3^1 - 3^0) \\ &= 2 \cdot 2 = 4 \\ &= 12\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right).\end{aligned}$$

Theorem 5.3 : Primitive Element Theorem

Given a prime p, there exists $g \in \mathbb{U}_p$ such that

$$\mathbb{U}_p = \{g^0 = 1,\ g,\ g^2, \cdots, g^{p-2}\} \qquad \textit{and} \qquad g^{p-1} = 1\,.$$

Here, g is a **generator** of $\mathbb{U}_p$ and is called a **primitive element** of $\mathbb{U}_p$ as well.

Hence for a prime p and a primitive element g of $\mathbb{U}_p$,

$$\mathbb{Z}_p = \{0, 1, g, g^2, \cdots, g^{p-2}\}$$

For example,

$$\begin{aligned}\mathbb{Z}_5 &= \{0, 1, 2, 2^2 = 4,\ 2^3 = 3\} \ \text{ and } \ 2^4 = 1 \\ \mathbb{Z}_7 &= \{0, 1, 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5\} \ \text{ and } \ 3^6 = 1\end{aligned}$$

Finding a primitive element is usually done by trial and error starting with 2, then 3, and so on. For example, for $\mathbb{Z}_7$

try $g = 2$
$2^2 \equiv 4,\ 2^3 \equiv 8 \equiv 1,\ 2^4 \equiv 16 \equiv 2, \ldots$ (repeating)
and we don't get everything
but trying $g = 3$
$3^2 \equiv 9 \equiv 2,\ 3^3 \equiv 2 \times 3 \equiv 6,\ 3^4 \equiv 6 \times 3 \equiv 18 \equiv 4,$
$3^5 \equiv 4 \times 3 \equiv 12 \equiv 5,\ 3^6 \equiv 5 \times 3 \equiv 15 \equiv 1$
and we do generate everything.

We note that, if a primitive element exists, then all elements and hence all primitive elements are powers of it, and in fact:

Theorem 5.4 : Primitive Powers
If g is primitive in $\mathbb{Z}_p$, then g^k is primitive if and only if $\gcd(k, p-1) = 1$ *and hence there are $\phi(p-1)$ primitive elements in $\mathbb{Z}_p$.*

For example, in $\mathbb{Z}_7$ we have $\phi(6) = 2$ and then 3, $3^5 = 5$ are all the primitive elements.

For a with $\gcd(a, m) = 1$, (that is, $a \in \mathbb{U}_m$) we call the *least positive integer* k for which

$$a^k \equiv 1 \,(\text{mod } m) \qquad (\text{that is, } a^k = 1 \in \mathbb{U}_m)$$

the **order** of a and write $k = \text{ord}_m(a)$.

For example, from above

$$\text{ord}_7(2) = 3 \quad \text{and} \quad \text{ord}_7(3) = 6$$

Similarly $\text{ord}_{12}(5) = 2$ since $5^1 \neq 1$ and $5^2 = 1 \in \mathbb{Z}_{12}$.

Note that g is primitive in $\mathbb{Z}_p$ if and only if $\text{ord}_p(g) = p - 1$.

Theorem 5.5 : Euler's Theorem
If $\gcd(a, m) = 1$ *(that is, $a \in \mathbb{U}_m$), then*

$$a^{\phi(m)} \equiv 1 \,(\textit{mod } m) \qquad (\textit{that is, } a^{\phi(m)} = 1 \textit{ in } \mathbb{Z}_m).$$

Theorem 5.6 : Corollary
If $\gcd(a, m) = 1$, *then* $\text{ord}_m(a) \mid \phi(m)$.

For example,

$$\begin{aligned} \text{ord}_7(2) &= 3 \mid 6 = \phi(7) \\ \text{ord}_{12}(5) &= 2 \mid 4 = \phi(12) \end{aligned}$$

Here, statements like $a = b|c$ means that a and b are equal to each other and (both) divide c.

Theorem 5.7 : Fermat's Little Theorem
For prime p and any $a \in \mathbb{Z}$

$$a^p \equiv a(\textit{mod } p)$$

Proof: If $p \mid a$, then $a \equiv 0 \,(\text{mod } p)$ and both sides $= 0$. If $p \nmid a$, then $\gcd(a, p) = 1$ so $a^{p-1} \equiv 1 \,(\text{mod } p)$ by Euler's Theorem, and then multiply both sides by a. $\square$

5.3 Polynomials over $\mathbb{Z}_p$ (p prime)

An expression

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

where $a_0, a_1, \cdots, a_n \in \mathbb{Z}_p$ and $a_n \neq 0$ is a polynomial with coefficients in $\mathbb{Z}_p$ or a **polynomial over** $\mathbb{Z}_p$ and has degree $\deg(f(x)) = n$. The degree of the zero polynomial is defined to be -1.

If $a_n = 1$ we say that the polynomial is **monic**.

Since $\mathbb{Z}_p$ is a field we can take out a factor a_n from each term and write

$$f(x) = a_n(x^n + b_{n-1} x^{n-1} + \cdots + b_1 x + b_0),$$

where $b_k = a_k a_n^{-1}$. This says that every polynomial can be written as a constant times a monic polynomial.

We denote the set of all polynomials over a field $\mathbb{F}$ by $\mathbb{F}[x]$: for example, $\mathbb{Z}_2[x]$.

Similarly to before, we write $g(x) \mid f(x)$ if and only if there exists $h(x) \in \mathbb{Z}_p[x]$ with $f(x) = g(x)h(x)$.

The Division Algorithm holds for polynomials over a field. That is, given $f(x)$ and $g(x)$, there exists $q(x)$ and $r(x)$, where $\deg(r(x)) < \deg(g(x))$ and $f(x) = g(x)q(x) + r(x)$. The division used is "polynomial long division".
For example, to divide $x^2 + 2x + 1$ into $x^4 + 2x^3 + 2x^2$ over $\mathbb{Z}_3$,

$$\begin{array}{r|l}
 & \quad x^2 \; + \; 1 \\ \hline
x^2 + 2x + 1 & x^4 + 2x^3 + 2x^2 \\
 & \underline{x^4 + 2x^3 + \; x^2} \\
 & \qquad\qquad x^2 \\
 & \qquad\qquad \underline{x^2 + 2x + 1} \\
 & \qquad\qquad\qquad x + 2
\end{array}$$

so $$(x^4 + 2x^3 + 2x^2) = (x^2 + 2x + 1)(x^2 + 1) + (x + 2).$$

A polynomial $f(x)$ is **irreducible** over field $\mathbb{F}$ if and only if for any polynomial $g(x) \in \mathbb{F}[x]$, if $g(x) \mid f(x)$, then either

$$\begin{array}{rrcll}
 & \deg(g(x)) & = & 0 & \text{(that is, } g(x) \text{ is a nonzero constant polynomial)} \\
\text{or} & \deg(g(x)) & = & \deg(f(x)) & \text{(that is, } g(x) \text{ is a constant multiple of } f(x)).
\end{array}$$

The term "irreducible" is used rather than "prime" in this context.
Note that being irreducible or not *depends on the field you are considering.*

There is a unique (up to order of factors) representation of any polynomial $f(x)$ as a constant multiple of a product of powers of monic irreducibles called the **factorization** of $f(x)$. That is, given $f(x)$, there exist monic irreducible polynomials $p_1(x), \cdots, p_r(x) \in \mathbb{Z}_p[x]$ with $\alpha_1, \ldots, \alpha_r \in \mathbb{Z}^+$ and $a_n \in \mathbb{Z}_p$ with

$$f(x) = a_n (p_1(x))^{\alpha_1} \cdots (p_r(x))^{\alpha_r}.$$

A unique monic **greatest common divisor** exists for any $f(x)$, $g(x)$ (not both identically zero), and there exists $a(x), b(x)$ with

$$d(x) = \gcd(f(x), g(x)) = a(x)f(x) + b(x)g(x).$$

These are found using the polynomial **Euclidean Algorithm**: the **extended Euclidean Algorithm** can also be used.

For example, over $\mathbb{Z}_3$ consider $f(x) = x^4 + 2x^3 + 2x^2$, and $g(x) = x^2 + 2x + 1$.
We already saw that $\underline{f(x)} = \underline{g(x)}(x^2 + 1) + \underline{(x + 2)}$.

Next, $\underline{g(x)} = x^2 + 2x + 1 = \underline{(x + 2)}(x) + \underline{1}$. The $\underline{1}$ is a constant, so is the last non-zero remainder. Hence

$$\begin{array}{rcll}
\underline{1} & = & \underline{g(x)} - \underline{(x + 2)}x & \\
 & = & \underline{g(x)} + 2x(\underline{f(x)} + 2(x^2 + 1)\underline{g(x)}) & (\mathbb{Z}_3 \text{ remember}) \\
 & = & (x^3 + x + 1)\underline{g(x)} + 2x\underline{f(x)} &
\end{array}$$

so $\gcd(f(x), g(x)) = d(x) = 1$ and $a(x) = 2x, \quad b(x) = x^3 + x + 1$.

The **factor theorem** and **remainder theorem** also hold in $\mathbb{Z}_p[x]$. So the remainder when $f(x)$ is divided by $x - a$ is $f(a)$, and so $x - a$ divides $f(x)$ if and only if $f(a) = 0$.

Remainders modulo a polynomial

If $m(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ is a **monic** polynomial in $\mathbb{Z}_p[x]$, then the set of remainders that result when polynomials are divided by $m(x)$ is denoted by

$$\mathbb{Z}_p[x]/\langle m(x)\rangle = \left\{b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1x + b_0 \ : \ b_{n-1}, b_{n-2}, \cdots, b_0 \in \mathbb{Z}_p\right\}$$

and arithmetic is done modulo both $m(x)$ and p.

Here $\langle m(x)\rangle$ denotes all multiples of $m(x)$ and $\mathbb{Z}_p[x]/\langle m(x)\rangle$ is the notation for the polynomials over $\mathbb{Z}_p[x]$ modulo $m(x)$.

For example, in $\mathbb{Z}_3[x]$

$$\begin{aligned} m(x) &= x^2 + 2x + 1 \quad \text{is monic, so} \\ \mathbb{Z}_3[x]/\langle m(x)\rangle &= \{ax + b : a, b \in \mathbb{Z}_3\} \qquad \text{and has 9 elements.} \end{aligned}$$

Now arithmetic is done in this set with $\equiv$ meaning congruent modulo 3 *and* modulo (x^2+2x+1). For example,

$$\begin{aligned} (2x+2) + (2x+1) &\equiv 4x + 3 \equiv x \\ (2x+2)\cdot(2x+1) &\equiv 4x^2 + 6x + 2 \\ &\equiv x^2 + 2 \equiv (x^2 + 2x + 1) + (x + 1) \\ &\equiv x + 1 \end{aligned}$$

Normally we will write $\equiv$ as $=$ when working in $\mathbb{Z}_p[x]/\langle m(x)\rangle$.

5.4 Finite Fields

If $\deg(m(x)) = n$, then $\mathbb{Z}_p[x]/\langle m(x)\rangle$ is a **vector space** of dimension n over $\mathbb{Z}_p$ which also has some multiplicative structure.

Similarly to before, $f(x)^{-1}$ exists in $\mathbb{Z}_p[x]/\langle m(x)\rangle$ if and only if $\gcd(f(x)), m(x)) = 1$.

If the inverse of $f(x)$ exists, then it is found by the polynomial Euclidean Algorithm.

For example, in $\mathbb{Z}_3[x]/\langle m(x)\rangle$ where $m(x) = x^2 + 2x + 1$ to find x^{-1}, if it exists, we apply the Euclidean Algorithm:

$$\begin{aligned} (\underline{x^2 + 2x + 1}) &= \underline{x}(x+2) + 1 \\ \text{so } \gcd(x, x^2 + 2x + 1) &= 1 \text{ and} \\ 1 &= \underline{x}(2x+1) + \underline{(x^2 + 2x + 1)} \\ \text{so } x(2x+1) &\equiv 1 \pmod{x^2 + 2x + 1} \\ \text{and hence} & \\ x^{-1} &= 2x + 1 \text{ in this set.} \end{aligned}$$

However $(x+1)^{-1}$ does not exist since $\gcd(x+1, x^2+2x+1) = x + 1 \neq 1$.

Suppose that $m(x)$ is a monic polynomial. Then
  $\mathbb{Z}_p[x]/\langle m(x)\rangle$ has an inverse for every non-zero element and hence is a **field**
  if and only if $\gcd(g(x), m(x)) = 1$ for all nonzero $g(x)$ with $\deg(g(x)) < \deg(m(x))$
  if and only if $m(x)$ is irreducible (and monic).

Example: Show that $m(x) = x^4 + x + 1$ is irreducible over $\mathbb{Z}_2$ and describe the field $\mathbb{F} = \mathbb{Z}_2[x]/\langle x^4 + x + 1\rangle$.

Now $m(0) = 1,$ and $m(1) = 1$. Hence since $\mathbb{Z}_2 = \{0, 1\}$, so by the Factor Theorem, $f(x)$ does not have any linear factors.

So if $m(x)$ factors, then it must factor into the product of two irreducible monic quadratics. But there are only 4 monic quadratics in $\mathbb{Z}_2[x]$, and only $x^2 + x + 1$ is irreducible. Since $(x^2 + x + 1)^2 = x^4 + x^2 + 1 \neq m(x)$, $m(x)$ must be irreducible over $\mathbb{Z}_2$.

(Alternatively, we could assume $m(x) = (x^2 + ax + b)(x^2 + cx + d)$ and show that there is no solution for the unknowns.)

The set of remainders

$$\mathbb{F} = \{ax^3 + bx^2 + cx + d : a, b, c, d \in \mathbb{Z}_2\}$$

is a vector space of dimension 4 over $\mathbb{Z}_2$ and contains $2^4 = 16$ elements.

We can multiply and reduce mod 2 and mod $x^4 + x + 1$ and also find inverses by Euclidean Algorithm.

For example, over $\mathbb{F}$, $(x^2 + 1)^{-1}$ is found by:

$$\begin{aligned}
\underline{(x^4 + x + 1)} &= \underline{(x^2 + 1)}(x^2 + 1) + x \\
\underline{(x^2 + 1)} &= \underline{x}x + \underline{1} \\
\text{so} \quad \underline{1} &= \underline{(x^2 + 1)} + \underline{x}x \\
&= \underline{(x^2 + 1)} + (\underline{(x^4 + x + 1)} + \underline{(x^2 + 1)}(x^2 + 1))x \\
\text{and} \quad \underline{1} &= \underline{(x^2 + 1)}(x^3 + x + 1) + \underline{(x^4 + x + 1)}x
\end{aligned}$$

Hence $(x^2 + 1)^{-1} = x^3 + x + 1$ in $\mathbb{F}$

The above process is pretty tedious. We will investigate a different way to view this finite field which will make these calculations easier.

Recall how the complex numbers $\mathbb{C}$ are built up from the real numbers $\mathbb{R}$. The polynomial $x^2 + 1 = 0$ is irreducible over the reals, so we introduce a new symbol i which represents a "root" of $x^2 + 1$. Then we perform arithmetic on numbers of the form $a + bi$ where $a, b \in \mathbb{R}$, using the fact that $i^2 = -1$.

We will apply the same trick here.

Let α stand for a "root" (any one) of

$$\begin{aligned}
&m(x) = x^4 + x + 1 \text{ over } \mathbb{Z}_2 \\
&\text{then } \alpha^4 + \alpha + 1 = 0 \\
&\text{so } \alpha^4 = -\alpha - 1 = \alpha + 1 \qquad \text{(as the coefficients are in } \mathbb{Z}_2).
\end{aligned}$$

Using this relationship, any power α^k can then be easily expressed as a linear combination of $\alpha^3, \alpha^2, \alpha, 1$. Since $\mathbb{F}$ is a vector space of dimension 4 over $\mathbb{Z}_2$, it follows that

$$\{\alpha^3, \alpha^2, \alpha, 1\} \text{ is a } \textbf{basis} \text{ for } \mathbb{F}.$$

Hence

$$\begin{aligned}
\mathbb{F} = \mathbb{Z}_2[x]/\langle x^4 + x + 1\rangle &= \{a\alpha^3 + b\alpha^2 + c\alpha + d : a, b, c, d \in \mathbb{Z}_2\} \\
&\qquad \text{where } \alpha^4 = \alpha + 1 \\
&= \mathbb{Z}_2(\alpha) \qquad \text{say.}
\end{aligned}$$

The notation $\mathbb{Z}_2(\alpha)$ means the smallest field containing $\mathbb{Z}_2$ and α.

Now, let's try finding powers of α, using the relation $\alpha^4 = \alpha + 1$ and working mod 2.

$$\begin{aligned}
\alpha^0 &= 1 & \alpha^7 &= \alpha^4+\alpha^3 = \alpha^3+\alpha+1\\
\alpha^1 &= \alpha & \alpha^8 &= \alpha^4+\alpha^2+\alpha = \alpha^2+1\\
\alpha^2 &= \alpha^2 & \alpha^9 &= \alpha^3+\alpha\\
\alpha^3 &= \alpha^3 & \alpha^{10} &= \alpha^4+\alpha^2 = \alpha^2+\alpha+1\\
\alpha^4 &= \alpha+1 & \alpha^{11} &= \alpha^3+\alpha^2+\alpha\\
\alpha^5 &= \alpha^2+\alpha & \alpha^{12} &= \alpha^4+\alpha^3+\alpha^2 = \alpha^3+\alpha^2+\alpha+1\\
\alpha^6 &= \alpha^3+\alpha^2 & \alpha^{13} &= \alpha^4+\alpha^3+\alpha^2+\alpha = \alpha^3+\alpha^2+1
\end{aligned}$$

and finally

$$\begin{aligned}
\alpha^{14} &= \alpha^4+\alpha^3+\alpha = \alpha^3+1\\
\alpha^{15} &= \alpha^4+\alpha = 1
\end{aligned}$$

So we get 15 different elements of $\mathbb{Z}_2(\alpha)$ as powers of α, which together with 0 give all 16 elements of $\mathbb{Z}_2(\alpha)$. That is, α generates $\mathbb{Z}_2(\alpha)-\{0\}$, so α is a **primitive element** of $\mathbb{Z}_2(\alpha)$.

Hence

$$\begin{aligned}
F = \mathbb{Z}_2(\alpha) &= \{0,1,\alpha,\alpha^2,\cdots,\alpha^{14}\},\\
&\qquad \text{where } \alpha^{15}=1 \text{ and } \alpha^4=\alpha+1.
\end{aligned}$$

Arithmetic is now very easy using this table of powers of α.

We use powers of α when multiplying and linear combinations of $\alpha^3,\alpha^2,\alpha,1$ when adding.

For example from the above table,

$$\begin{aligned}
(\alpha^3+\alpha+1)(\alpha^3+\alpha^2+1) &= \alpha^7\alpha^{13} = \alpha^{20} = \alpha^5 = \alpha^2+\alpha\\
(\alpha^2+1)^{-1} &= (\alpha^8)^{-1} = \alpha^{-8} = \alpha^{15-8} = \alpha^7\\
&= \alpha^3+\alpha+1.
\end{aligned}$$

Theorem 5.8 : Finite Field Theorem

If p is prime, $m(x)$ a monic irreducible in $\mathbb{Z}_p[x]$ of degree n, and α denotes a root of $m(x)=0$, then

1. $\mathbb{F}=\mathbb{Z}_p[x]/\langle m(x)\rangle$ *is a field,*
2. $\mathbb{F}$ *is a vector space of dimension n over $\mathbb{Z}_p$,*
3. $\mathbb{F}$ *has p^n elements,*
4. $\{\alpha^{n-1},\alpha^{n-2},\cdots,\alpha,1\}$ *is a basis for $\mathbb{F}$,*
5. $\mathbb{F}=\mathbb{Z}_p(\alpha)$, *i.e. the smallest field containing $\mathbb{Z}_p$ and α,*
6. *there exists a primitive element γ of order p^n-1 for which $\mathbb{F}=\{0,1,\gamma,\gamma^2,\ldots,\gamma^{p^n-2}\}$,*

if a field $\mathbb{F}$ has a finite number of elements, then $|\mathbb{F}|=p^n$ where p is prime, and $\mathbb{F}$ is isomorphic to $\mathbb{Z}_p[x]/\langle m(x)\rangle$. Hence ALL fields with p^n elements are isomorphic to one another.

Notes:

1. The "essentially unique" field of order $q=p^n$ is denoted by $\mathbb{F}_q$ or GF(q) (for Galois field).
2. Although any two fields with q elements are isomorphic, the isomorphism is rarely obvious and their arithmetic may appear to be quite different.
3. A field with p^n elements is said to have **characteristic** p and **prime field** $\mathbb{Z}_p$.

4. If γ is primitive in $\mathrm{GF}(p^n)$, then there are $\phi(p^n-1)$ primitive elements, namely all the γ^k having $\gcd(k, p^n-1) = 1$.

   For example in the GF(16) described above $\gamma = \alpha$ is primitive and $p^n - 1 = 15$ so a complete list of primitive elements is $\alpha, \alpha^2, \alpha^4, \alpha^7, \alpha^8, \alpha^{11}, \alpha^{13}, \alpha^{14}$.

   We see that there are $\phi(15) = (3-1)(5-1) = 8$ primitive elements in GF(16).

5. The *order* $\mathrm{ord}(\beta)$ of an element $\beta \in \mathbb{Z}_p(\alpha)$ is defined to be the smallest positive integer k such that $\beta^k = 1$. For example, in the GF(16) described above, α has order 15 and α^5 has order 3. The order of an element of $\mathrm{GF}(q)$ is always a factor of $q-1$.

Further example: Consider $\mathbb{Z}_3$ and the polynomial $m(x) = x^2 + 1$.

Then $m(0) = 1,\ m(1) = 2,\ m(2) = 2$, so $m(x)$ has no zeros, so no linear factors and hence is irreducible. (If a quadratic or cubic has factors, then it must have *linear* factors. But note that this is *not true* for polynomials of higher degree.)

We use $m(x)$ to construct GF(9).

Let α be a root of $m(x) = x^2+1$. Then $\alpha^2 + 1 = 0$ so $\alpha^2 = -1 = 2$. (Here α is the equivalent of "i"over $\mathbb{Z}_3$.)

Powers of α give

$$\begin{aligned}
\alpha^0 &= 1 \\
\alpha^1 &= \alpha \\
\alpha^2 &= 2 \\
\alpha^3 &= 2\alpha \\
\alpha^4 &= 2\alpha^2 = 1
\end{aligned}$$

and we see $\mathrm{ord}(\alpha) = 4 \neq 8$ and so α is not primitive.

Now, try $\gamma = \alpha + 1$ and see how we go:

$$\begin{aligned}
\gamma^0 &= 1 \\
\gamma^1 &= \alpha + 1 \\
\gamma^2 &= \alpha^2 + 2\alpha + 1 = 2\alpha \\
\gamma^3 &= 2\alpha^2 + 2\alpha = 2\alpha + 1 \\
\gamma^4 &= 2\alpha^2 + 1 = 2 \quad (= -1) \\
\gamma^5 &= -\gamma = 2\alpha + 2 \\
\gamma^6 &= -\gamma^2 = \alpha \\
\gamma^7 &= -\gamma^3 = \alpha + 2 \\
\gamma^8 &= -\gamma^4 = 1
\end{aligned}$$

and so $\gamma = \alpha + 1$ is primitive and generates GF(9).

We have a γ^k power representation and an $a\alpha + b$ linear combination representation for each element, and we use both representations to do arithmetic.

For example,

$$\begin{aligned}
\frac{\gamma^2+\gamma^3}{\gamma+\gamma^7} &= \frac{2\alpha + (2\alpha+1)}{(\alpha+1)+(\alpha+2)} = \frac{\alpha+1}{2\alpha} \\
= \frac{\gamma}{\gamma^2} &= \gamma^{-1} = \gamma^7 = \alpha + 2.
\end{aligned}$$

Since γ is primitive, the $\phi(9-1) = \phi(8) = 4$ primitive elements are

$$\gamma,\ \gamma^3 = 2\alpha+1,\ \gamma^5 = 2\alpha+2,\ \gamma^7 = \alpha+2.$$

5.5 Minimal and primitive polynomials

Let p be a prime. Fermat's Little Theorem for primes says that for all $a \in \mathbb{Z}$,

$$a^p \equiv a(\operatorname{mod} p).$$

Hence $x^p - x$ has p distinct roots in $\mathbb{Z}_p$, and so

$$x^p - x = x(x-1)(x-2)\,\cdots\,(x-(p-1))$$

is its factorization over $\mathbb{Z}_p$.

In the same way, for $q = p^n$ we get

$$x^q - x = x(x-1)(x-\gamma)(x-\gamma^2)\ \cdots\ (x-\gamma^{q-2})$$

which is the factorization over $\mathrm{GF}(q)$, where γ is primitive in $\mathrm{GF}(q)$.

Given any element β of $\mathrm{GF}(q)$ where $q = p^n$, there exists a unique monic polynomial in $\mathbb{Z}_p[x]$ of minimum degree satisfied by β.

This is called the **minimal polynomial** of β and it must be irreducible of degree $\leq n$.

Fact: if $f(x) \in \mathbb{Z}_p[x]$ is any polynomial with $f(\beta) = 0$, then the minimal polynomial of β divides $f(x)$ (with no remainder).

If β happens to be primitive in $\mathrm{GF}(q)$ then the minimal polynomial of β is called a **primitive** polynomial.

The degree of a primitive polynomial must be n.

Primitive polynomials are the most useful sort of irreducible polynomials for constructing finite fields and also for later applications.

Now consider the binomial expansion, where p is prime:

$$(a+b)^p = a^p + \binom{p}{1} a^{p-1}b + \ \cdots\ + \binom{p}{p-1} ab^{p-1} + b^p\ .$$

Since

$$\binom{p}{k} = \frac{p(p-1)\ \cdots\ (p-k+1)}{k(k-1)\ \cdots\ 1}$$

we see that for $k = 1, 2, \cdots, p-1$ the prime p in the numerator cannot be cancelled out. Hence

$$p \,\Big|\, \binom{p}{k} \quad \text{for} \quad k = 1, 2, \ldots, p-1.$$

Hence $(a+b)^p = a^p + b^p$ in $\mathbb{Z}_p$ (a very useful identity!).

Now if β is a root of $g(x) = \sum b_i x^i$, $\quad b_i \in \mathbb{Z}_p$, then

$$\begin{aligned}(g(x))^p &= \left(\sum b_i x^i\right)^p = \sum b_i^p (x^p)^i \qquad \text{(by above)}\\ &= \sum b_i (x^p)^i \qquad \text{(by Fermat)}\\ &= g(x^p)\end{aligned}$$

and so we also have

$$g(\beta) = 0 \quad \Rightarrow \quad g(\beta^p) = 0.$$

So if β has minimal polynomial $g(x)$ then $g(x)$ also has roots

$$\beta^p,\ (\beta^p)^p = \beta^{p^2},\ \beta^{p^3}, \cdots$$

It can be shown these are the **only** roots and thus

$$g(x) = (x-\beta)(x-\beta^p)(x-\beta^{p^2})(x-\beta^{p^3})\cdots$$

continuing as long as the terms are distinct.

Examples: The arithmetic is done using the previous tables.

1. $f(x) = x^4 + x + 1$ over $\mathbb{Z}_2$ is primitive since its root α is primitive.
2. $f(x) = x^2 + 1$ over $\mathbb{Z}_3$ is irreducible but not primitive.
   $\gamma = \alpha + 1$ was primitive where $\alpha^2 + 1 = 0$, so let's find the minimal polynomial for γ. The minimal polynomial equals

$$\begin{aligned} & (x-\gamma)(x-\gamma^3) && \text{(stop here as } \gamma^9 = \gamma) \\ = \ & x^2 - (\gamma+\gamma^3)x + \gamma\gamma^3 \\ = \ & x^2 + x + 2 && (\text{as } \gamma + \gamma^3 = (\alpha+1) + (2\alpha+1) = 2 = -1) \end{aligned}$$

   Hence $x^2 + x + 2$ is primitive over $\mathbb{Z}_3$.

   As an exercise construct $\mathbb{Z}_3[x]/\langle x^2+x+2\rangle$ showing the powers of the root γ of x^2+x+2 over $\mathbb{Z}_3$.

3. In $GF(16) = \mathbb{Z}_2(\alpha)$ with $\alpha^4 = \alpha + 1$, $\alpha^{15} = 1$, where α is primitive, then

   α^5 has minimal polynomial

$$\begin{aligned} & (x-\alpha^5)(x-\alpha^{10}) && \text{stop here as } \alpha^{20} = \alpha^5 \\ & = x^2 + (\alpha^5 + \alpha^{10})x + \alpha^{15} \\ & = x^2 + x + 1 \end{aligned}$$

   This is irreducible, but not primitive as degree is only 2 not 4.

   α^3 has minimal polynomial

$$\begin{aligned} & (x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^{24}) && \text{stop as } \alpha^{48} = \alpha^3 \\ = \ & x^4 + (\alpha^3 + \alpha^6 + \alpha^{12} + \alpha^{24})x^3 \\ & +(\alpha^3\alpha^6 + \alpha^3\alpha^{12} + \alpha^3\alpha^{24} + \alpha^6\alpha^{12} + \alpha^6\alpha^{24} + \alpha^{12}\alpha^{24})x^2 \\ & +(\alpha^3\alpha^6\alpha^{12} + \alpha^3\alpha^6\alpha^{24} + \alpha^3\alpha^{12}\alpha^{24} + \alpha^6\alpha^{12}\alpha^{24})x \\ & +\alpha^3\alpha^6\alpha^{12}\alpha^{24} \\ = \ & x^4 + x^3 + x^2 + x + 1 \quad \text{(after considerable algebra)} \end{aligned}$$

   This is irreducible but not primitive as α^3 is not primitive.

Note:

1. In $GF(16) = \mathbb{Z}_2(\alpha)$ with $\alpha^4 = \alpha + 1$, $\alpha^{15} = 1$
   α has minimal polynomial

$$(x-\alpha)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8) = x^4 + x + 1.$$

   α is primitive and so are α^2, α^4, α^8.

The other primitive elements are

$$\alpha^7,\ \alpha^{14},\ \alpha^{28} = \alpha^{13},\ \alpha^{56} = \alpha^{11}$$

and their minimal polynomial is

$$(x - \alpha^7)(x - \alpha^{14})(x - \alpha^{13})(x - \alpha^{11}) = x^4 + x^3 + 1$$

hence there are *exactly 2* primitive polynomials of degree 4 over $\mathbb{Z}_2$.

In general there are $\phi(p^n - 1)$ primitive elements in $\mathrm{GF}(p^n)$. Hence there are $\phi(p^n - 1)/n$ primitive polynomials of degree n over $\mathbb{Z}_p$, as each primitive polynomial has n distinct roots.

2. There is another useful result:

   $x^{p^n} - x$ is the product of **all** irreducible polynomials over $\mathbb{Z}_p$ whose *degrees are a factor of* n.

5.6 Primality Testing

In our cryptography section we will often need to know if a number is prime or composite.

How can we check a given number is prime? How can we generate a large prime number?

5.6.1 Trial division

It is obvious if a number is divisible by

2	—	it is even, that is, it ends in $0, 2, 4, 6, 8$
5	—	it ends in 0 or 5
3	—	its decimal digits sum to a number which is divisible by 3

Then trial divide n by each prime number up to $\sqrt{n}$ and if none divide it, then n is prime, see question 1.

This is a legitimate strategy for small n, but hopelessly slow for large n, as the running time is $O(\sqrt{n})$.

5.6.2 Pseudo-prime test

Euler's theorem says that when n is prime, if $\gcd(a, n) = 1$, then $a^{n-1} \equiv 1 \pmod{n}$.

Hence, using the contrapositive:

If there exists a with $\gcd(a, n) = 1$ and $a^{n-1} \not\equiv 1 \pmod{n}$ then n is composite.

This can be used as a test for primality:

Given $a \in \mathbb{Z}^+$:

if $\gcd(a, n) \neq 1$	then n is composite (and you have a factor)
if $\gcd(a, n) = 1$	and $a^{n-1} \not\equiv 1 \pmod{n}$ then n is composite and *fails the test*
otherwise	$\gcd(a, n) = 1$ and $a^{n-1} \equiv 1 \pmod{n}$ and n *passes the test*

and is called a **pseudo-prime to base** a.

Unfortunately, as the name indicates, not all pseudo-primes are in fact primes. For example, $n = 91$ has $\gcd(3, 91) = 1$ and $3^{90} \equiv 1 \pmod{91}$, so 91 is pseudo-prime base 3. However, $91 = 7 \times 13$, so 91 is in fact composite.

If we had tested 91 base 2 we would have deduced that it is composite since $2^{90} \equiv 64 \not\equiv 1 \pmod{91}$.

How do we find such high powers using a hand-held calculator?

In the above example, write the exponent 90 as a sum of powers of 2 viz. $90 = 64+16+8+2$. Find 3 to the power of a power of 2 by squaring and reducing mod 91

in $\mathbb{Z}_{91}$	$3^2 = 9$	$3^8 \equiv 81^2 \equiv 6561 \equiv 9$	$3^{32} \equiv 9$
	$3^4 = 81$	$3^{16} \equiv 81$	$3^{64} \equiv 81$

Finally combine as $3^{90} = 3^{64}3^{16}3^8 3^2 \equiv 81 \times 81 \times 9 \times 9 \equiv 1$.

None of this arithmetic overflows a calculator's 8-10 digit accuracy. Remember that all calculations are integers, so if you get a float or Engineering notation number you will have rounding errors.

We noticed that the composite number 91 is a pseudo-prime base 3 but not base 2.

Now we have seen this we might think that all we have to do is to test for pseudo-primality to lots of different bases.

Unfortunately, there are numbers which are pseudo-prime to very many different bases, but are in fact composite.

A **Carmichael number** is a number which is pseudo-prime to all bases which are prime to the number. That is, if $\gcd(a, n) = 1$, then $a^{n-1} \equiv 1 \pmod{n}$. (If you picked a base with $\gcd(a, n) \neq 1$ of course you know n is composite).

The smallest Carmichael number is $n = 561 = 3 \times 11 \times 17$ and no-one would ever think it prime as it has such small factors. However, large Carmichael numbers are far more difficult to recognise. Alford, Granville & Pomerance proved in 1994 that there are infinitely many Carmichael numbers.

5.6.3 Lucas test

This depends on:

Theorem 5.9 : Lucas' Theorem

Given $n \in \mathbb{Z}^+$, suppose that there exists $a \in \mathbb{Z}^+$ such that $gcd(a, n) = 1$ and $a^{n-1} \equiv 1 \pmod n$ and $a^{\frac{n-1}{p}} \not\equiv 1 \pmod n$ for all primes $p \mid n-1$.

Then n is prime.

This can be restated as:

If there exists an a with $gcd(a, n) = 1$ and $ord_n(a) = n - 1$, then n is prime.

This test *always* works, but it assumes you know the prime factors of $n - 1$, and (as we see later) this is harder than knowing a number is prime (in general).

The Lucas test is only useful for an n where $n - 1$ easily factors.

5.6.4 Miller-Rabin test

Lemma 5.10 *Let p be prime. Then if $a^2 = 1 \pmod p$, $a \equiv \pm 1 \pmod p$*

Proof: We have $a^2 - 1 = (a + 1)(a - 1) \equiv 0 \pmod p$, so p divides one of $a - 1$ or $a + 1$. □

Now let $n = 2^s t + 1$ where t is odd and suppose $\gcd(a, n) = 1$. If n is prime, then $a^{2^s t} \equiv 1 \pmod n$, so by our lemma either $a^t \equiv 1 \pmod n$ or there is an r with $0 \le r < s$ for which $a^{2^r t} \equiv -1 \pmod n$.

Conversely, if there is an integer a with $1 \le a \le n - 1$ and either

- $a^t \equiv 1 \pmod n$, or
- there exists r with $0 \le r < s$ and $a^{2^r t} \equiv -1 \pmod n$

then n is called a **strong pseudo-prime base** a, otherwise a is a **witness** to n being composite.

Fact: If n is odd and composite, then it is a strong pseudo-prime to base a for *at most 25%* of the bases a with $1 \le a \le n - 1$, in other words, at least 75% of bases a are witnesses.

This gives rise to the **Miller-Rabin probabilistic primality test**, given in pseudocode below. The output is either the message *probably prime* or the message *composite*.

INPUT: an odd number n to be tested for primality.

```
    Write $n = 2^s t + 1$ where $t$ is odd;
    Choose $a \in \{1, \ldots, n-1\}$ randomly;
    if $a^t \equiv 1 \pmod n$, then
        return probably prime;
    else
        for $r$ from 0 to $s - 1$ do
            if $a^{2^r t} \equiv -1 \pmod n$, then
                return probably prime;
            end if;
        end do;
    return composite; (witness found)
    end if;
END.
```

The probability that a composite number passes k independent tests is thus less than $(\frac{1}{4})^k$. Using Bayes' rule (see question 74) we can show that for a fixed n, the probability that a number is composite given that it passes k tests (for large k) is a constant multiple of $(\frac{1}{4})^k$, which can be made arbitrarily small by making k large enough.

So repeated Miller-Rabin tests either *prove* that n is composite by finding a witness, or suggest that n is *probably prime* with an error probability a constant multiple of $(\frac{1}{4})^k$. This is usually good enough in practice.

Notes:

1. In practice it has been found that the theoretically derived 1/4 is more like 1/1000 for most cases, so n is even more likely to be prime than the above estimate.

2. Miller-Rabin is a $O(k(\log n)^3)$ test and so it takes time which is polynomial in the number of digits of n. This is called a **polynomial time** algorithm.

3. It is known for all $n < 3.4 \times 10^{14}$ that if n passes the Miller-Rabin test for $a = 2, 3, 5, 7, 11, 13, 17$ then n is **definitely prime**.

   This can be put on a programmable calculator without much difficulty.

4. Provided an unproved result known as the Extended Riemann Hypothesis is true, then, if n passes the Miller-Rabin test for all values of $a = 2, 3, 4, \ldots, \lfloor 2(\log_2 n)^2 \rfloor$ then n is **definitely prime**.

   This test is $O((\log n)^5)$, so is also a polynomial time algorithm.

5. There are also primality tests based on elliptic curve arithmetic which do not rely on any unproved assumptions and give a definite proof that a number is or is not prime. However, these tests are much slower than the Miller-Rabin test and are not polynomial time algorithms.

6. In August 2002 a new primality test was discovered by Agrawal, Kayal and Saxena, a group of Indian Computer Science researchers. This AKS algorithm proves a number definitely is or is not prime in polynomial time. The original paper is in *Annals of Mathematics* **160** (2004), pp. 781–793, available in the library, and the algorithm itself is quite simple — you should be able to follow it. (The notation $q^k \parallel n$ in the paper means q^k is the largest power of q dividing n.)

   Unfortunately, for any reasonably sized numbers, the AKS test is far slower than the elliptic curve method and so is not of any practical use. The original algorithm was theoretically $O((\log n)^{12+\epsilon})$ for some small ϵ, and so much slower than the Miller-Rabin test. Improvements by Lenstra and Pomeranz (2005) have brought this down to $O((\log n)^6)$.

7. The largest prime found to date (found in December 2018) is $2^{82,589,933}1$ This number has 24,862,048 digits. (See `http://www.utm.edu/research/primes/` for the latest information.)

5.6.5 Prime Number Generation

In real world situations, prime number generators work like the following:

(a) Generate a random number of the appropriate size (see later on how to do this).

(b) If it is even, add 1 to make it odd.

(c) Trial divide by all "small" primes (e.g. trial dividing by all primes < 256 eliminates 80% of the composites).

(d) Perform the Miller-Rabin test 5 times for relatively small values of 'a'.

(e) Repeat (a) to (d) until you get a probable prime.

5.7 Factoring

If we know n is composite, sometimes (but not always) we need to find its factors. This is a much harder problem than showing n is prime or composite. Factors of 2 are easily accounted for, so we really only need to consider odd numbers.

5.7.1 Trial division

Trial dividing by all primes $2, 3, 5, \ldots, \lfloor\sqrt{n}\rfloor$ finds the smallest factor p.
Repeat for n/p until n is completely factored.
This is very inefficient, but is an adequate method for finding small factors of n.

5.7.2 Fermat factorization

An odd number n factors as $n = ab$, where $a \geq b$ and both of course odd, if and only if n has a representation of the form

$$n = t^2 - s^2 \quad = (t+s)(t-s) \quad \text{where } t = \frac{1}{2}(a+b), \;\; s = \frac{1}{2}(a-b).$$

If a and b are close, then s is small. We assume $s \neq 0$, otherwise n is a square and we factorise $\sqrt{n}$ instead.
Now $s > 0$, so we must have $t > \sqrt{n}$,
so we start with $t = \lceil\sqrt{n}\,\rceil$ and loop
increasing t by 1 each time and testing to see if $t^2 - n$ is a perfect square s^2.
If it is, then we have factored n.

Note that $(t+1)^2 - n = (t^2 - n) + 2t + 1$ so the next value can be obtained by adding $2t + 1$ to the existing $t^2 - n$ value.
Also, checking to see if a number is a square is computationally easy. On a calculator simply use the square root button and see if you get an integer.
All this means that we are factoring by use solely of square roots and addition/subtraction.
Example:

For $n = 9869$, $\sqrt{n} = 99.3$ so

t	$2t+1$	$t^2 - n$	$\sqrt{\ } \in \mathbb{Z}?$
100	201	131	$\times$
101	203	332	$\times$
102	205	535	$\times$
103	207	740	$\times$
104	209	947	$\times$
105		1156	34

so $t = 105$ $\quad s = 34$
$a = 139$ $\quad b = 71$
and $9869 = 71 \times 139$.

Note the last 2 digits in the decimal form of squares can only be 00, $e1$, $e4$, $e9$, 25, $d6$ where e is even, d is odd, so this simplifies checking for squares by hand.

Fermat factorization is useless if $\frac{1}{2}(a+b) = t$ is much larger than $\sqrt{n} = \sqrt{ab}$ and so is only useful if you have some information suggesting a and b are close.

5.7.3 Pollard's ρ method (1975)

The basic idea of this method is the same as for the famous birthday problem: *if you have a large enough set of integers there is a very high probability that at least two of them are congruent modulo a pre-determined number.*

Given n to be factorised, suppose p is the smallest factor. Pollard's method (1975) involves generating a random set of integers (in some way) and looks for two x and y, say, that are not congruent modulo n, but are congruent modulo p. Their difference $|x-y|$ is smaller than n and a multiple of p and hence a factor of n.

We do not know p of course, but we can check for $|x-y|$ being a factor of n with the Euclidean algorithm by calculating $\gcd(|x-y|, n)$.

In practice, we start with a map $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ which mixes up the elements of $\mathbb{Z}_n$ and is not one-to-one.

Starting with some x_0, iterate $x_{i+1} \equiv f(x_i) \pmod{n}$ to produce a sequence $x_0, x_1, x_2, \ldots$, hoping to find values which are different mod n but the same mod k where $k \mid n$.

These are detected by finding $\gcd(x_i - x_j, n)$ for suitable i, j, using the (relatively fast) Euclidean Algorithm.

If this greatest common denominator is not 1, then we have found a factor of n.

There are a variety of ways of systematically choosing the i, j for taking the differences, the easiest being to take $j = 2i$.

Example: For $n = 91643$ with $f(x) = x^2 - 1,\ x_0 = 3$ we iterate $x_{i+1} \equiv x_i^2 - 1 \pmod{91643}$

$x_0 \equiv 3$		$\gcd(x_{2i} - x_i, n)$
$x_1 \equiv 8$		
$x_2 \equiv 63$	$x_2 - x_1 \equiv 55$	1
$x_3 \equiv 3968$		
$x_4 \equiv 74070$	$x_4 - x_2 \equiv 74007$	1
$x_5 \equiv 65061$		
$x_6 \equiv 35193$	$x_6 - x_3 \equiv 31225$	1
$x_7 \equiv 83746$		
$x_8 \equiv 45368$	$x_8 - x_4 \equiv 62941$	113

and we get $n = 91643 = 113 \times 811$. Both are primes so we have factored it.

Note: this is a "statistical" method with expected time to find the smallest factor p being $O(\sqrt{p})$.

5.7.4 Quadratic sieve method

The **Quadratic Sieve** method (Pomerance, 1981) uses ideas from both Fermat's and Pollard's methods.

Rather than trying to find t and s with $n = t^2 - s^2$, we try to find t, s such that $t^2 \equiv s^2 \pmod{n}$, since then $n \mid (t^2 - s^2) = (t-s)(t+s)$ and finding $\gcd(n, t \pm s)$ gives a factor of n.

Unfortunately, most of the time this gcd is 1 and so many pairs t, s must be tried. The problem is how to find efficiently find suitable t, s.

(Fermat's method simply started near the middle and worked up 1 at a time, which means a long search in general.)

The quadratic sieve method is much more sophisticated and searches through far fewer pairs t, s.

Choose large B_1, B_2, satisfying certain rules that we will not go into, and choose a small m such that $x^2 \equiv mn \pmod{p}$ can be solved for lots of small primes.

Form $P = \{p_0 = -1, p_1, p_2, \ldots, p_k\}$ consisting of -1 and **all** the primes $p_i \le B_1$ for which $x^2 \equiv mn \pmod{p_i}$ can be solved.

Sieve Step

Collect a large set of relations (in a systematic way)

$$v^2 \equiv q_1 \cdot q_2 \prod_{i=0}^{k} p_i^{e_i} \pmod{n}$$

where either $q_i = 1$ or q_i is a prime between B_1 and B_2.

Matrix Step

Find linear dependencies over $\mathbb{Z}_2$ among the vectors $(e_0, e_1, \ldots, e_k)$, after processing to remove the q_i's. Each dependency corresponds to a set of solutions

$$w_j^2 \equiv \prod_{i=0}^{k} p_i^{e_{ij}} \pmod{n} \qquad \text{for which}$$

$$\sum_j (e_{0j}, e_{1j}, \ldots, e_{kj}) = (2a_0, 2a_1, \ldots, 2a_k), \qquad a_i \in \mathbb{Z}.$$

Consequently, letting

$$t \equiv \prod_j w_j \pmod{n} \quad \text{and} \quad s \equiv \prod_{i=0}^{k} p_i^{a_i} \qquad \pmod{n}$$

we have a solution satisfying

$$t^2 \equiv s^2 \pmod{n}$$

and hence can factor n (or at least start to factor it).

Comments

The sieving step can be done in parallel or at different times and places with different ranges of values. Only the matrix step, which is actually just row reduction over $\mathbb{Z}_2$, has to be done all at once.

In 1977 when the RSA cryptographic system was described by Martin Gardner in the Scientific American a \$100 prize was offered for the first person to crack his example cipher. Cracking the cipher could be achieved by simply factoring a given 129 digit number.

At the time they estimated it would take 40×10^{15} years to factor.

From August 1993 to April 1994 the quadratic sieve was applied to this problem and the number factored. The successful team used

$$m = 5, \quad B_1 = 16,333,609. \quad B_2 = 2^{30} \quad \text{giving } |P| = 524,339.$$

The sieving was done by emailing parts of the problem out over the internet (including the software) and getting the relations back by email.

Over 600 individuals using over 1600 computers participated. The computers ranged from 386's to Crays and the software was designed to run in idle time.

The final matrix step involved a $569,466 \times 524,339$ sparse matrix with entries in $\mathbb{Z}_2$.

Over 60 hours on a massively parallel MasPar MP-1 with 1 Gbyte core and lots of disk space resulted in getting the answer. The message was

"The Magic Words are Squeamish Ossifrage".

This held the record for the largest computation ever done publicly at the time — an estimated 10^{17} computer instructions.

(Since then many more problems have been solved by using the internet as a massively parallel computer and the size of this computation far exceeded.)

The speed of the quadratic sieve method is $e^{O(\sqrt{\log n \log\log n})}$. It is the fastest known method for numbers having less than 110 digits.

A related method is the **number field sieve method**. It is the fastest known method for factoring large numbers and has speed $e^{O((\log n)^{1/3}(\log\log n)^{2/3})}$.

RSA Laboratories (`http://www.rsa.com`) ran a factoring challenge from 1991 to 2007 to factor the product of two random primes. The largest product that had been factored by the time the challenge was terminated had 200 decimal digits (663 bits). Factoring such numbers is important for the security of RSA encryption, as we will see later. For more information about the RSA factoring challenge see
`http://en.wikipedia.org/wiki/RSA_Factoring_Challenge`

5.7.5 Shor's Algorithm

In 1994 Peter Shor published an algorithm that promises a highly efficient method of factorization on a quantum computer. The details of how the properties of a quantum computer speeds up the algorithm are beyond the remit of this course, but the basic idea is quite simple.

Suppose we are trying to factor the odd composite number n. Let a be an integer whose order in $\mathbb{Z}_n$ is $2k$, i.e. is *even*. Then we know that $a^{2k} \equiv 1 \pmod n$, and so

$$n \mid (a^k + 1)(a^k - 1).$$

As $2k$ is the order of a, we know that $a^k \not\equiv 1 \pmod n$, so n does not divide $a^k - 1$. If n does not divide $a^k + 1$, each prime factor of n must divide either $a^k + 1$ or $a^k - 1$. So if we calculate $\gcd(n, a^k \pm 1)$, we must find non-trivial factors of n. We reduce mod n first of course: if $p \mid n$ and $a^k \equiv r \pmod n$, then $p \mid (a^k \pm 1)$ iff $p \mid (r \pm 1)$.

The hard part of Shor's algorithm is finding the integer a of even order. In practice this is done by picking a random $a < n$ that is coprime to n — obviously if we pick a and it is not coprime to n we have a factor. The quantum part of the calculation is then involved in finding the order of a. If the order is odd, or $n \mid a^k + 1$, we pick another a and start again.

Shor's algorithm (with the quantum speed up for finding the order) is polynomial time in the number of digits: a huge improvement on the exponential time classical algorithms, but of course requires actually having a quantum computer.

5.8 Random Number Generation

In our cryptography section we will need single random integers as well as sequences of random integers. Generating random numbers is vitally important[1] in other scientific applications as well e.g. for simulations and code testing and is a field of active research.

How are they generated?

[1] The late Robert R. Coveyou once titled an article "The generation of random numbers is too important to be left to chance".

5.8.1 Truly random processes

Truly random proceses are all physical processes such as particle counts from radioactive decay, output from zener diodes, and so on. These are difficult to incorporate into computing applications. The web site `www.random.org` claims to generate true random numbers from atmospheric noise.

Even these physical processes are not always truly random but follow some probability distribution and so have to be processed further.

Numbers generated by some mathematical or computing process (with the aim of producing "random numbers") are called **pseudo-random numbers**.

5.8.2 Middle Squares Method

The **middle squares method** was introduced by John von Neumann in 1949. To generate a sequences on n digit numbers, start with such a number, the **seed**, x_0. Square it and (if necessary) pad out with leading zeros to a $2n$ digit number. Then x_1 is the middle n digits of the square. Repeat.

I mention this method as it is quite handy for creating sequences with a calculator, but it is not useful for any serious applications (as von Neumann knew). It tends to repeat quickly for one thing (try it with $x_0 = 2100$ or 3792 for 4 digit numbers).

5.8.3 Linear congruential

Given a, b, m and seed x_0, iterate $x_{i+1} \equiv ax_i + b \pmod m$ to produce the sequence $x_1, x_2, \ldots$

This method for generating pseudo-random numbers is often used in practice. For example, NAG's pseudo-random number generator uses

$$a = 13^{13}, \quad b = 0, \quad m = 2^{59}, \qquad x_0 = 123456789 \times (2^{32} + 1)$$

or an x_0 of the user's choice, and Maple uses

$$a = 427419669081, \quad b = 0, \quad m = 999999999989, \quad x_0 = 1,$$

where x_0 can also be reset.

These produce sequences with good properties, but, as with any bounded iterative computing process that uses a fixed number of previous result, it is eventually periodic. NAG's generator has period 2^{57} which is quite large and so is useful in most simulation situations.

In many typical applications, the seed is generated on the fly from internal registers in the computer e.g. the users ID, local time, the time since the computer was last booted etc.

(See Knuth Art of Computer Programming Vol 2 for analysis of the period of the linear congruential random number generator.)

Unfortunately this method is useless in any cryptographic process, since:

If m is known, then the three consecutive values x_{i-1}, x_i, x_{i+1} give enough linear congruence equations to solve to give a, b.

Even if m is unknown, then given enough consecutive terms, all of a, b and m can be found.

5.8.4 Polynomial congruential and LFSR

Given a prime p and $a_0, a_1, \ldots, a_{n-1} \in \mathbb{Z}_p$ and seeds $x_0, x_1, \ldots, x_{n-1}$, iterate

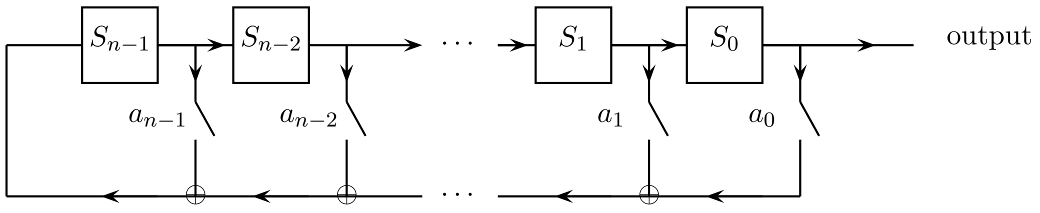
$$x_{i+n} \equiv a_{n-1}x_{i+n-1} + a_{n-2}x_{i+n-2} + \ldots + a_0 x_i \pmod p$$

This "recurrence relation" or "difference equation" has characteristic polynomial

$$f(r) = r^n - a_{n-1}r^{n-1} - a_{n-2}r^{n-2} - \ldots - a_0.$$

The sequence generated by this recurrence has the maximal possible period length, which is $p^n - 1$, provided $f(r)$ is a *primitive polynomial* over $\mathbb{Z}_p$.

When $p = 2$, such sequences can be generated by **linear feedback shift registers** (LFSRs) and so are often used in practice also.



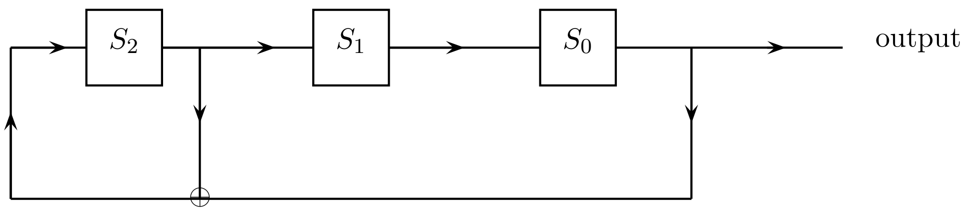
Here, the switches are closed if and only if $a_i = 1$ and $\oplus$ indicates mod 2 addition (performed instantly).

Initial values $x_0, \dots, x_{n-1}$ are placed in the shift registers $s_0, \dots, s_{n-1}$
then the machine is allowed to produce output one bit per clock cycle,
and the stored values move following the arrows to the next shift register once per clock cycle.

Example: the polynomial $x^3 + x^2 + 1$ is primitive over $\mathbb{Z}_2$ and corresponds to the recurrence

$$x_{i+3} = x_{i+2} + x_i.$$

This means that the $s_2 + s_0$ values are fed back into s_2 at the next cycle.



Now, starting with say $x_0 = 1$, $x_1 = 0$, $x_2 = 0$ we have the states

	S_2	S_1	S_0	output
initial	0	0	1	
last $s_2 + s_0 \to$	1	0	0	1
	1	1	0	0
	1	1	1	0
	0	1	1	1
	1	0	1	1
	0	1	0	1
	0	0	1	0

This then cycles and has cycle length $= 7 = 2^3 - 1$.

All non zero patterns of 3 bits are produced equally often as consecutive triples in the output.

In general, since there are $\phi(2^n - 1)/n$ different primitive polynomials of degree n over $\mathbb{Z}_2$, there are that many different LFSRs of length n.

Again these pseudo-random generators are insecure, since knowing $2n+1$ consecutive terms means you can find $a_0, \dots, a_{n-1}$.

5.8.5 Non-linear feedback shift registers

Given $x_0, \dots, x_{n-1}$ and some non-linear function f of n variables
iterate

$$x_{i+n} = f(x_i, x_{i+1}, \dots, x_{i+n-1}).$$

Not much is known theoretically about these except in some special cases.

For example, it is known that the N-LFSR given by

$$x_0 \equiv 2 \pmod 4 \quad \text{and}$$

$$x_{i+1} \equiv x_i(x_i + 1) \pmod{2^k}, \quad k > 2$$

is efficient and has long period.

5.8.6 Cryptographic generators

In cryptography, a function E_k depending on a key k is used to encrypt messages. (See Chapter 7.) We can use these functions to build cryptographic random number generators as follows: take some cryptographic function $E_k(x)$ depending on a key k and take a seed x_0 and iterate $x_{i+1} = E_k(x_i)$, or iterate $x_{i+1} = E_{x_i}(c)$ for some constant c.

These are as secure as the cryptographic function, but their randomness properties are usually unknown.

One exception is the **Blum, Blum, Shub** (quadratic) random bit generator (1986).
Let $n = pq$ where p, q are primes $\equiv 3 \pmod 4$ (these are called **Blum primes**) and take seed x_0.
Generate $x_0, x_1, \ldots, x_k$ by $x_{i+1} \equiv x_i^2 \pmod n$
then output the corresponding *least significant* bit in *reverse* order. That is, the output equals
$b_0, b_1, \ldots, b_k$ where $b_j = 1$, if and only if x_{k-j} is odd.

[Alternatively use $x_j \equiv x_0^{2^j \pmod{(p-1)(q-1)}} \pmod n$ for any order].

This is a provably secure and random generator of bits, but is very slow.

5.8.7 Multiplexing of sequences

Multiplexing means combining two or more sequences of pseudo-random numbers, to both increase the period and make the combined sequence more secure.

For example, suppose
$x_0, x_1, \ldots$ is the output of a maximal length n LFSR and

$y_0, y_1, \ldots$ is the output of a maximal length m LFSR where $2^m - 1 \le n$.
Form the binary number

$$N_t = y_t + 2\, y_{t+1} + \ldots + 2^{m-1}\, y_{t+m-1} \ .$$

Then $1 \le N_t \le 2^m - 1$. For a fixed one-one function

$$f : \{1, 2, 3, \ldots, 2^m - 1\} \longrightarrow \mathbb{Z}_n$$

form the new sequence $z_0, z_1, \ldots$ by

$$z_t = x_{t+f(N_t)}.$$

If $\gcd(m, n) = 1$, then this has period $(2^m - 1)(2^n - 1)$ and needs more than $2n(2^m - 1)$ consecutive terms before it can be predicted.
This is quite a good pseudo-random number generator.

Chapter 6

Algebraic Coding

In Chapter 2 we considered codes designed to detect and correct errors, including the Hamming Codes. There are many other families of codes which can be constructed using more complicated mathematics: we could generalise all that work from $\mathbb{Z}_p$ to $\mathrm{GF}(p^k)$, for example.

Instead, we will look at an important family of codes called **BCH codes** which arise when we look at Hamming Codes from an algebraic point of view. These codes were discovered independently by Bose and Ray-Chaudhuri (1960) and by Hocquenghem (1959).

BCH codes have good error correcting properties when the length is not too great and they can be encoded and decoded relatively easily. A special subset of BCH codes are the **Reed-Solomon** codes which are used in many digital communications and storage devices (e.g. DVD players, digital televisions, satellite communications, mobile phones, high-speed modems and QR codes). See Bose's book for more details.

We then close the chapter by considering **cyclic codes**, which have very nice structural properties. In particular, encoding and decoding using cyclic codes can be performed efficiently using linear switching circuits. There are two very special cyclic codes called **Golay codes** which we construct. One is a binary code, one is a ternary code and they are both perfect.

6.1 Hamming codes revisited

Consider the Hamming (15,11) code rearranged in the following very particular standard form (which is no longer directly related to binary numbers):

$$H = \begin{pmatrix} 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \end{pmatrix}$$

Recall the field $GF(16) = \mathbb{Z}_2[x]/\langle x^4+x+1\rangle$ studied in Section 5.4. Notice that the columns of H are given by the coordinate vectors of $1, \alpha, \alpha^2, \dots, \alpha^{14}$ with respect to the ordered basis $\{1, \alpha, \alpha^2, \alpha^3\}$. For example, $\alpha^5 = \alpha + \alpha^2$ so α^5 has coordinate vector $\begin{pmatrix}0\\1\\1\\0\end{pmatrix}$, which is the sixth column of H.

By a slight abuse of notation, we will replace each column of H by the element of F that it represents. Hence we write

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \cdots & \alpha^{14} \end{pmatrix}.$$

This gives us a shorthand for H. Since the first column of H corresponds to α^0 and, more generally, the jth column of H corresponds to α^{j-1}, we speak of "the column of H corresponding to α^j".

Now let $\mathbf{c} = (c_0, c_1, \ldots, c_{14}) \in \mathcal{C}$ be a codeword. (Again, notice that now we number entries from 0 instead of 1.) The syndrome of $\mathbf{c}$ is

$$S(\mathbf{c}) = H\mathbf{c} = c_0 + c_1\alpha + c_2\alpha^2 + \cdots + c_{14}\alpha^{14} = C(\alpha)$$

where $C(x) = c_0 + c_1x + \cdots + c_{14}x^{14}$ is the polynomial corresponding to $\mathbf{c}$. This allows us to describe the code in terms of polynomials.

We will now generalise these ideas to obtain the BCH (Bose, Chaudhuri & Hocquenghem) codes.

WARNING: in some past exam papers you will see that the bits are given in the opposite order, such as $(c_{14}, c_{13}, \ldots, c_0)$. But we will always order our bits in the order $(c_0, c_1, \ldots, c_{14})$, as described above. This ensures that the check matrix H is in standard form.

6.2 Single error correcting BCH codes

Let $f(x) \in \mathbb{Z}_2[x]$ be a primitive polynomial of degree m with root α. Let $n = 2^m - 1$ and $k = n - m$. Then the matrix

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \end{pmatrix}$$

is the check matrix of a binary Hamming (n, k) code $\mathcal{C}$. (You can check that m, n and k satisfy the sphere-packing bound with $t = 1$.) Note that every binary Hamming (n, k) code with $n = 2^m - 1$ and $k = n - m$ can be obtained in this way.

Let $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathcal{C}$ be a codeword. Because the first m columns of H are the leading columns, the bits $c_0, \ldots, c_{m-1}$ are the check bits and $c_m, \ldots, c_{n-1}$ are the information bits.

As above, the syndrome $S(\mathbf{c})$ can be written as

$$S(\mathbf{c}) = H\mathbf{c} = c_0 + c_1\alpha + c_2\alpha^2 + \cdots + c_{n-1}\alpha^{n-1} = C(\alpha)$$

where $C(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$ is the **codeword polynomial** corresponding to the codeword $\mathbf{c}$. Since $\mathbf{c}$ is a codeword we see that $C(\alpha) = 0$, so α is a root of the polynomial $C(x)$. Therefore the minimal polynomial $M_1(x)$ of α must divide $C(x)$ with no remainder. But $M_1(x) = f(x)$, the polynomial that we first started with.

Encoding: Given the message $(c_m, c_{m+1}, \ldots, c_{n-1})$, form the **information polynomial**

$$I(x) = c_mx^m + c_{m+1}x^{m+1} + \cdots + c_{n-1}x^{n-1}.$$

Using polynomial long division, find the **check polynomial** $R(x)$ of degree at most $m - 1$ such that

$$I(x) \equiv R(x) \pmod{M_1(x)}.$$

That is, $R(x)$ is the remainder when $I(x)$ is divided by the minimal polynomial $M_1(x)$. Then

$$R(x) = c_0 + c_1x + \cdots c_{m-1}x^{m-1}$$

where $c_0, c_1, \ldots, c_{m-1} \in \mathbb{Z}_2$. Finally, the *codeword polynomial* is $C(x) = I(x) + R(x)$ and the codeword is $(c_0, c_1, \ldots, c_{n-1})$. Notice that the first m bits are the check bits and the last k bits are the information bits.

Error Correction: Suppose that $\mathbf{d} = \mathbf{c} + \mathbf{e}_j$ where $\mathbf{e}_j$ is the unit vector with 1 in the α^j position and zeros elsewhere. Writing these vectors as polynomials (with $C(x)$ corresponding to $\mathbf{c}$ and $D(x)$ corresponding to $\mathbf{d}$) and calculating the syndrome of $\mathbf{d}$ gives

$$S(\mathbf{d}) = D(\alpha) = C(\alpha) + \alpha^j = \alpha^j.$$

Therefore the syndrome of $\mathbf{d}$ equals the column α^j of H, which tells us the position of the error. By changing the bit in the α^j position, the error can be corrected. If $D(\alpha) = 0$ then there is no error.

To decode, simply delete the first m bits of the codeword.

Example:

Consider the field $\mathbb{F} = \mathbb{Z}_2[x]/\langle x^3+x+1\rangle$ with 8 elements. Let β be a root of $m_1(x) = x^3+x+1$. Then β is a primitive element of F and the powers of β are as follows:

$$\begin{aligned}
\beta^3 &= 1 + \beta, \\
\beta^4 &= \beta + \beta^2, \\
\beta^5 &= \beta^2 + \beta^3 = 1 + \beta + \beta^2, \\
\beta^6 &= \beta + \beta^2 + \beta^3 = 1 + \beta^2, \\
\beta^7 &= \beta + \beta^3 = 1.
\end{aligned}$$

This gives the matrix

$$H = \begin{pmatrix} 1 & \beta & \beta^2 & \beta^3 & \beta^4 & \beta^5 & \beta^6 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

This is a check matrix for the Hamming (7,4) code, in standard form.

To encode the message 0101 using this code: recall that the information bits are (c_3, c_4, c_5, c_6). This gives the information polynomial $I(x) = x^4 + x^6$. Then polynomial long division shows that

$$x^6 + x^4 = (x^3 + 1)(x^3 + x + 1) + x + 1$$

(check this for yourself!). It follows that the check polynomial is $R(x) = 1 + x$, and so the codeword polynomial is

$$C(x) = I(x) + R(x) = 1 + x + x^4 + x^6.$$

The corresponding codeword is $\mathbf{c} = (1, 1, 0, 0, 1, 0, 1)$.

To correct and decode the received word $\mathbf{d} = 0011011$, let $D(x) = x^2 + x^3 + x^5 + x^6$ be the corresponding polynomial and calculate

$$\begin{aligned}
D(\beta) &= \beta^2 + \beta^3 + \beta^5 + \beta^6 \\
&= \beta^2 + (1 + \beta) + (1 + \beta + \beta^2) + (1 + \beta^2) \\
&= 1 + \beta^2 \\
&= \beta^6.
\end{aligned}$$

Therefore the error lies in the entry of $\mathbf{d}$ corresponding to β^6. (Careful: this is the *last* entry in $\mathbf{d}$ because we start counting from 0, not 1.) We correct to $\mathbf{c} = 0011010$ and decode to 1010.

6.3 Two-error correcting BCH codes

We can extend these ideas further to produce 2-error correcting BCH codes. In the previous section, every codeword gave rise to a polynomial $C(x)$ such that $C(\alpha) = 0$, where α is the primitive element generating the code. If $M_1(x)$ is the minimal polynomial of α then the roots of $M_1(x)$ are all elements of the set

$$\{\alpha,\, \alpha^2,\, \alpha^4,\, \alpha^8, \cdots\}.$$

More generally, if p is a prime and β is a root of $f(x) \in \mathbb{Z}_p[x]$ then so is β^p. So for any finite field $\mathbb{F} = GF(p^k)$ with primitive element α, if α^s is a root of the polynomial $f(x) \in \mathbb{Z}_p[x]$ then so is every element of the set

$$\{\alpha^s,\, \alpha^{ps},\, \alpha^{p^2 s},\, \alpha^{p^3 s}, \ldots\}.$$

(This set is finite since for some t we have $sp^t \equiv s \pmod{p^k - 1}$.) We call the set of indices

$$K_s = \{s,\, ps,\, p^2 s,\, p^3 s, \ldots\}$$

the **cyclotomic coset** containing s. Notice that the degree of the minimal polynomial of α^s is equal to the size of K_s.

Coming back to the BCH code, every α^j with $j \in K_1$ is already a root of $M_1(x)$. We need to choose a different element to help us to produce a two-error correcting BCH code.

Example: For the Hamming (15,11)-code as a BCH code over $GF(16)$, we have $p = 2$ and the cyclotomic coset containing 1 is $K_1 = \{1, 2, 4, 8\}$. So consider α^3 (since 3 is the smallest positive integer not in K_1) and let $M_3(x)$ be the minimal polynomial of α^3. We find that

$$M_3(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^9) = x^4 + x^3 + x^2 + x + 1.$$

(EXERCISE: Check this.) Therefore both α and α^3 are roots of the polynomial

$$\begin{aligned} M(x) &= M_1(x)M_3(x) \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) \\ &= x^8 + x^7 + x^6 + x^4 + 1. \end{aligned}$$

(EXERCISE: Check this.) Since $M(x)$ has degree 8, this will give a code with 8 check bits and 7 information bits.

To encode a message $(c_8, c_9, \ldots, c_{14})$ we form the information polynomial

$$I(x) = c_8 x^8 + \cdots c_{14} x^{14}$$

and calculate its remainder $R(x) \equiv I(x) \pmod{M(x)}$, by long division by $M(x)$. This gives the check polynomial $R(x)$ of degree at most 7. So

$$R(x) = c_0 + c_1 x + \cdots c_7 x^7$$

which tells us the check bits. Then the codeword is $\mathbf{c} = (c_0, c_1, \ldots, c_{14})$.

For example, to encode the message $(1, 0, 1, 1, 0, 1, 1)$, create the information polynomial

$$I(x) = x^8 + x^{10} + x^{11} + x^{13} + x^{14}$$

and calculate that

$$R(x) = x + x^2 + x^4 + x^5 + x^7.$$

Therefore the codeword polynomial is

$$C(x) = x + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{13} + x^{14}$$

and the corresponding codeword is $(0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1)$.

Decoding is similar to the case of one error, but requires more work. As before, given the received word **d** we form the corresponding polynomial $D(x)$ and write

$$D(x) = C(x) + E(x)$$

where $E(x)$ represents the error and $C(\alpha) = C(\alpha^3) = 0$, assuming not more than two errors.

Case I: $D(\alpha) = D(\alpha^3) = 0$.
Then there are no errors, and $E(x)$ is the zero polynomial.

Case II: $D(\alpha) = \alpha^j$ and $D(\alpha^3) = \alpha^{3j} = D(\alpha)^3 \neq 0$ for some j.
Then there is one error and $E(x) = x^j$.

Case III: $D(\alpha) = \alpha^j + \alpha^\ell$ and $D(\alpha^3) \neq D(\alpha)^3$ for some j, ℓ.
Then there are two errors and $E(x) = x^j + x^\ell$.

In this case let $S_1 = D(\alpha)$ and $S_3 = D(\alpha^3)$. Then

$$\begin{aligned} S_3 &= \alpha^{3j} + \alpha^{3\ell} \\ &= (\alpha^j + \alpha^\ell)(\alpha^{2j} + \alpha^j\alpha^\ell + \alpha^{2\ell}) \\ &= S_1(S_1^2 + \alpha^j\alpha^\ell) \end{aligned}$$

Therefore

$$\alpha^j + \alpha^\ell = S_1 \qquad \text{and} \qquad \alpha^j\alpha^\ell = \frac{S_3}{S_1} + S_1^2.$$

Hence α^j and α^ℓ are the roots of the quadratic equation

$$z^2 + S_1 z + \left(\frac{S_3}{S_1} + S_1^2\right) = 0 \quad \text{over} \quad \mathbb{Z}_2(\alpha).$$

This cannot be solved using the standard quadratic formula (since $2a = 0$) or by completing the square.

So we solve it by *trial and error* to find the error locations j and ℓ and correct the 2 errors. This proves that our code is 2-error correcting.

The check matrix of this code is

$$H' = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^{15} & \alpha^{18} & \alpha^{21} & \alpha^{24} & \alpha^{27} & \alpha^{30} & \alpha^{33} & \alpha^{36} & \alpha^{39} & \alpha^{42} \end{pmatrix}$$

Now interpreting this back in bits we have (where the last column bits are shown bracketed)

$$H' = \begin{pmatrix} 1 & 0 & & & 1 \\ 0 & 1 & & & 0 \\ 0 & 0 & & & 0 \\ 0 & 0 & & & 1 \\ 1 & 0 & \ldots & \ldots & 1 \\ 0 & 0 & & & 1 \\ 0 & 0 & & & 1 \\ 0 & 1 & & & 1 \end{pmatrix} \quad \begin{matrix} \left.\vphantom{\begin{matrix}1\\0\\0\\1\end{matrix}}\right\} \alpha^{14} \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \alpha^{42} = \alpha^{12} \end{matrix}$$

which is a 8×15 matrix. Therefore the corresponding code $\mathcal{C}'$ has parameters

$$m = 8,\ n = 15,\ k = 7.$$

Example:

Decode Now suppose $\boldsymbol{c} \rightsquigarrow \boldsymbol{d}$ with errors in location α^3 and α^{10}. Then, (with errors underlined)

$$\boldsymbol{d} = 011\underline{1}1101 \big| 10\underline{0}1011$$

so $D(x) = x + x^2 + x^3 + x^4 + x^5 + x^7 + x^8 + x^{11} + x^{13} + x^{14}$ is the corresponding polynomial.

Then we calculate the syndrome

$$\begin{aligned} H\boldsymbol{d} = S(\boldsymbol{d}) &= \begin{pmatrix} S_1 \\ S_3 \end{pmatrix} = \begin{pmatrix} D(\alpha) \\ D(\alpha^3) \end{pmatrix} \\ &= \begin{pmatrix} \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^7 + \alpha^8 + \alpha^{11} + \alpha^{13} + \alpha^{14} \\ \alpha^3 + \alpha^6 + \alpha^9 + \alpha^{12} + \alpha^{15} + \alpha^{21} + \alpha^{24} + \alpha^{33} + \alpha^{39} + \alpha^{42} \end{pmatrix} \\ &= \begin{pmatrix} \alpha^{12} \\ \alpha^7 \end{pmatrix} \quad \text{using} \quad \mathbb{Z}_2(\alpha) \quad \text{arithmetic.} \end{aligned}$$

Since

$$(\alpha^{12})^3 = \alpha^{36} = \alpha^6 \neq \alpha^7,$$

there are two errors in $\boldsymbol{d}$ (and not just one). Now

$$\frac{S_3}{S_1} + S_1^2 = \frac{\alpha^7}{\alpha^{12}} + \alpha^{24} = \alpha^{13}.$$

So our errors are the zeros of the quadratic

$$x^2 + \alpha^{12}x + \alpha^{13} = 0.$$

Trial and error gives $x = \alpha^3, \alpha^{10}$ and so the errors are located and can be corrected.

6.4 The general case (binary)

Let α be a primitive element of the field $GF(2^r)$ and let $M(x)$ be the least common multiple of the minimal polynomials of α, $\alpha^2, \dots, \alpha^{2t}$ where $2t < 2^r$. Suppose that $M(x)$ has degree $2^r - k$. Then a BCH $(2^r - 1,\, k)$-code $\mathcal{C}$ is the set of polynomials $C(x) \in \mathbb{Z}_2[x]$ of degree at most $2^r - 2$ which are divisible by $M(x)$.

Theorem 6.1 *With $\mathcal{C}$ and $M(x)$ as above,*

(a) the code $\mathcal{C}$ can correct up to t errors,

(b) if a received polynomial $D(x)$ has at most t errors and $S_i = D(\alpha^i)$ for $1 \le i \le 2t$, then the number of errors is equal to the rank v of the matrix

$$\mathbf{S} = \begin{pmatrix} S_1 & S_2 & \cdots & S_t \\ S_2 & S_3 & \cdots & S_{t+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_t & S_{t+1} & \cdots & S_{2t-1} \end{pmatrix}.$$

(c) if there are u errors then the error polynomial is

$$E(x) = x^{j_1} + x^{j_2} + \cdots + x^{j_u}$$

where $\alpha^{j_1}, \dots, \alpha^{j_u}$ are the roots of the polynomial

$$z^u + \sigma_1 z^{u-1} + \cdots + \sigma_{u-1} z + \sigma_u$$

where $\sigma_1, \dots, \sigma_u$ satisfy the matrix equation

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_u \\ S_2 & S_3 & \cdots & S_{u+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_u & S_{u+1} & \cdots & S_{2u-1} \end{pmatrix} \begin{pmatrix} \sigma_u \\ \sigma_{u-1} \\ \vdots \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} S_{u+1} \\ S_{u+2} \\ \vdots \\ S_{2u} \end{pmatrix}.$$

Example: We have looked at the BCH (15,7)-code which corrects up to 2 errors. Now we try to correct up to 3 errors using a BCH code based on the field GF(16) $= \mathbb{Z}_2(\alpha)$ where $\alpha^4 = \alpha + 1$. The cyclotomic cosets for GF(16) are

$$K_1 = \{1, 2, 4, 8\}, \quad K_3 = \{3, 6, 12, 9\}, \quad K_5 = \{5, 10\}, \quad K_7 = \{7, 14, 13, 11\}.$$

We must take the lowest common multiple of the minimal polynomials of $\alpha, \alpha^2, \dots, \alpha^6$. But this will equal $M(x) = M_1(x)M_3(x)M_5(x)$ where $M_5(x)$ is the minimal polynomial of α^5. You can check that $M_5(x) = x^2 + x + 1$, and hence

$$\begin{aligned} M(x) &= M_1(x)M_3(x)M_5(x) \\ &= (x^8 + x^7 + x^6 + x^4 + 1)(x^2 + x + 1) \qquad \text{(from earlier)} \\ &= x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1. \end{aligned}$$

The theorem says that the corresponding BCH code is 3-error correcting. Since $M(x)$ has degree 10, the code has 10 check bits and 5 information bits.

To send the message $(0, 1, 0, 0, 0)$ we form the information polynomial $I(x) = x^{11}$ and calculate the check polynomial using polynomial long division to get

$$R(x) \equiv I(x) \pmod{M(x)} = x + x^2 + x^3 + x^5 + x^6 + x^9.$$

Therefore the codeword polynomial is

$$C(x) = I(x) + R(x) = x + x^2 + x^3 + x^6 + x^9 + x^{11}$$

and the codeword is $(0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0)$.

If the word $\mathbf{d} = (1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0)$ was received, form the corresponding polynomial

$$D(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^9 + x^{11}.$$

Then calculate the following syndromes:

$$\begin{aligned}
S_1 &= D(\alpha) = 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^9 + \alpha^{11} = \alpha,\\
S_2 &= D(\alpha^2) = D(\alpha)^2 = \alpha^2,\\
S_3 &= D(\alpha^3) = 1 + \alpha^3 + \alpha^6 + \alpha^9 + \alpha^{12} + \alpha^{15} + \alpha^{18} + \alpha^{27} + \alpha^{33} = \alpha^{11},\\
S_4 &= (S_2)^2 = \alpha^4,\\
S_5 &= D(\alpha^5) = 1 + \alpha^5 + \alpha^{10} + \alpha^{15} + \alpha^{20} + \alpha^{25} + \alpha^{30} + \alpha^{45} + \alpha^{55} = \alpha^{10},\\
S_6 &= (S_3)^2 = \alpha^7.
\end{aligned}$$

We save ourselves some calculation by using the fact that $S_{2k} = (S_k)^2$ for binary codes.

Now we form the matrix from the theorem and row reduce over $\mathbb{Z}_2$ it to find its rank:

$$\mathbf{S} = \begin{pmatrix} \alpha & \alpha^2 & \alpha^{11} \\ \alpha^2 & \alpha^{11} & \alpha^4 \\ \alpha^{11} & \alpha^4 & \alpha^{10} \end{pmatrix} \to \begin{pmatrix} \alpha & \alpha^2 & \alpha^{11} \\ 0 & \alpha^5 & \alpha^6 \\ 0 & \alpha^6 & \alpha^7 \end{pmatrix} \to \begin{pmatrix} \alpha & \alpha^2 & \alpha^{11} \\ 0 & \alpha^5 & \alpha^6 \\ 0 & 0 & 0 \end{pmatrix}.$$

So the matrix has rank 2, which means that there are two errors.

We proceed as described in the previous section: the errors are in positions α^j and α^ℓ which are the roots of the quadratic $z^2 + S_1 z + (S_3/S_1 + {S_1}^2)$. Now

$$\frac{S_3}{S_1} + {S_1}^2 = \frac{\alpha^{11}}{\alpha} + \alpha^2 = \alpha^{10} + \alpha^2 = \alpha^4$$

so the quadratic is

$$z^2 + \alpha z + \alpha^4 = z^2 + (\alpha^4 + 1)z + \alpha^4 = (z + 1)(z + \alpha^4).$$

Therefore the errors are in positions α^0 and α^4. We correct these errors to obtain the codeword

$\mathbf{c} = (0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0)$, and decoding gives the message 01000.

Correcting more errors: If we wanted to correct 4 errors using GF(16) then we must work with the polynomial $M(x)$ which is the least common multiple of the minimal polynomials of $\alpha, \alpha^2, \ldots, \alpha^8$. Since the cyclotomic cosets are

$$K_1 = \{1, 2, 4, 8\}, \quad K_3 = \{3, 6, 12, 9\}, \quad K_5 = \{5, 10\}, \quad K_7 = \{7, 14, 13, 11\}.$$

we must use $M(x) = M_1(x)M_3(x)M_5(x)M_7(x)$. The minimal polynomial $M_7(x)$ has degree 4 since $|K_7| = 4$, so $M(x)$ has degree 14 and there is only room for 1 information bit.

It is a fact that the lowest common multiples of of the minimal polynomials of *all* elements of $GF(16)$ equals $x^{16} - x$. (This is the same as the product of the distinct minimal polynomials.) Hence

$$M(x) = \frac{x^{15} - 1}{x - 1} = 1 + x + \cdots + x^{13} + x^{14}$$

since $M(x)$ is only missing the minimal polynomials of 0 and 1 (which are x and $x-1$ respectively). There are two possible information polynomials, namely $I(x) = 0$ (the zero polynomial) and $I(x) = x^{14}$, standing for the information words 0 and 1 respectively. You can check that the remainder of $I(x)$ mod $M(x)$ is 0 in the first case and $1 + x + \cdots + x^{13}$ in the second case. Hence the codeword polynomials are

$$C(x) = 0 \quad \text{and} \quad C(x) = 1 + x + \cdots + x^{13} + x^{14}$$

respectively, which means that the two codewords are 000000000000000 and 111111111111111. This code is the **15-repetition code** which we met in Chapter 2.

Thus the family of BCH codes over GF(16) has the Hamming(15,11) code at one extreme and the 15-repetition code at the other extreme. This is the case generally as well.

Next let's consider the BCH codes based on $\mathrm{GF}(32) = \mathbb{Z}_2[x]/\langle x^5 + x^2 + 1\rangle$. First we calculate the cyclotomic cosets of GF(32):

$$\begin{array}{lll} K_1 = \{1, 2, 4, 8, 16\}, & K_3 = \{3, 6, 12, 24, 17\}, & K_5 = \{5, 10, 20, 9, 18\}, \\ K_7 = \{7, 14, 28, 25, 19\}, & K_{11} = \{11, 22, 13, 26, 21\}, & K_{15} = \{15, 30, 29, 27, 23\}. \end{array}$$

For a **1-error correcting code**, we need a polynomial with roots α, α^2. Use $M(x) = M_1(x)$ which has degree 5 (since K_1 has 5 elements). This gives a BCH code with $n = 31$, $m = 5$, $k = 26$. (It is the Hamming(31,26) code.)

For a **2-error correcting code**, we need a polynomial with roots $\alpha, \alpha^2, \ldots, \alpha^4$. Use $M(x) = M_1(x)M_3(x)$ which has degree $5 + 5 = 10$. This gives a BCH code with $n = 31$, $m = 10$, $k = 21$.

For a **3-error correcting code**, we need a polynomial with roots $\alpha, \alpha^2, \ldots, \alpha^6$. Use $M(x) = M_1(x)M_3(x)M_5(x)$ which has degree 15. This gives a BCH code with $n = 31$, $m = 15$, $k = 16$.

For a **4-error correcting code**, we need a polynomial with roots $\alpha, \alpha^2, \ldots, \alpha^8$. Use $M(x) = M_1(x)M_3(x)M_5(x)M_7(x)$ which has degree 20. This gives a BCH code with $n = 31$, $m = 20$, $k = 11$. But notice that α^9 and α^{10} are also roots of $M(x)$, but not α^{11}. This means that the BCH code corresponding to $M_1(x)M_3(x)M_5(x)M_7(x)$ is actually a **5-error correcting code**. So it is better to multiply these polynomials together one by one and then see how many errors each can correct.

Next consider the polynomial $M(x) = M_1(x)M_3(x)M_5(x)M_7(x)M_{11}(x)$. This polynomial has degree 25 and has each of $\alpha, \alpha^2, \ldots, \alpha^{14}$ as roots. Hence it is a **7-error correcting** code with $n = 31$, $m = 25$, $k = 6$.

Finally, the polynomial $M(x) = M_1(x)M_3(x)M_5(x)M_7(x)M_{11}(x)M_{15}(x)$ has degree 30 and contains every power of α as a root (up to α^{31}). Therefore it is a **15-error correcting** code with $n = 31$, $m = 30$ and $k = 1$. Again, it turns out that this code is the 31-repetition code.

Similarly in GF(64) there are BCH codes with the following properties:

57 information digits	and 1-error correcting,
51 information digits	and 2-error correcting,
45 information digits	and 3-error correcting,
39 information digits	and 4-error correcting,
36 information digits	and 5-error correcting,
30 information digits	and 6-error correcting,
24 information digits	and 7-error correcting,
18 information digits	and 10-error correcting,
16 information digits	and 11-error correcting,
10 information digits	and 13-error correcting,
7 information digits	and 15-error correcting,
1 information digit	and 31-error correcting.

6.4.1 Reed-Solomon Codes

(This section is for extra background information.)

The BCH codes we have looked at so far have the symbols of the code word in a prime field (e.g. $\mathbb{Z}_2$) and the errors in an extension field of it (e.g. GF(16)). There is nothing in the way we set up BCH codes that requires this to be the case: we could use any finite field for the symbols of the code, and even chose the polynomial $M(x)$ to have roots in this field.

For example, suppose we wish to encode a stream of 8-bit ASCII characters into a BCH code. An 8-bit character (i.e. a byte) can be easily encoded as an element of the field $\text{GF}(2^8) = \text{GF}(256)$, so we could construct a t error correcting BCH code over GF(256) to do this, and it would be able to correct errors in t bytes. The message would still be sent as binary digits of course, it is just that each block of eight is thought of as an element of GF(256). This code would be particularly useful against burst noise: the code corrects an entire character (byte) that is wrong, so it makes no difference if one bit in the byte is wrong or the whole byte is wrong.

A **Reed-Solomon Code** (I.S. Reed and G. Solomon, 1960) is BCH code where the field of symbols we use is $\text{GF}(p^r)$ for some prime p, and the polynomial $M(x)$ we use to generate the code words also has roots in the same field $\text{GF}(p^r)$. The codewords will therefore consist of a block of $n = p^r - 1$ symbols in $\text{GF}(p^r)$.

Suppose we want a Reed-Solomon code that corrects t errors. Then by the process we use to create BCH codes, the generating polynomial $M(x)$ will be the least common multiple of the minimal polynomials of $\alpha, \alpha^2, \ldots, \alpha^{2t}$ in $\text{GF}(p^r)$ (**not** in $\mathbb{Z}_p$).

But the minimal polynomial of an element β of $\text{GF}(p^r)$ is just $x - \beta$, so

$$M(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t}) \in \text{GF}(p^r)[x].$$

We denote this code by $\text{RS}(n,k)$, where $n = p^r - 1$ and $k = n - 2t$: it has information rate $\dfrac{k}{n}$ obviously.

Consider again encoding 8-bit ASCII characters into GF(256). If we wish to be able to correct, say, 10 errors, we would have the code $\text{RS}(255, 235)$ with 235 data symbols (bytes) and 20 parity symbols. The decoder could then correct up to 10 errors (that is 10 errors *in the bytes*, which could be up to 80 bits in error) with an information rate approximately 0.92.

As mentioned before, Reed-Solomon codes are particularly useful against burst errors, and indeed against a mixture of burst and random errors. In the $\text{RS}(255, 235)$ code mentioned above, a codeword could suffer from, say, 4 random errors in bits (in 4 different bytes) and a burst error 5 bytes (40 bits) long and the code could correct it. For this reason, they are commonly used in data storage (CDs use a variant), QR codes and deep-space communication.

6.5 Cyclic codes.

A linear code $\mathcal{C}$ of length n is **cyclic** if for all codewords

$$c_0c_1\cdots c_{n-1} \in \mathcal{C}$$

we have

$$c_{n-1}c_0c_1\cdots c_{n-2} \in \mathcal{C}.$$

That is, a cyclic shift of the symbols of a codeword produces another codeword.

The coefficients here are in $\mathbb{F} = GF(p^r)$ where p is prime, and $q = p^r$.

Cyclic codes have several advantages when it comes to their implementation. To look further at them, we need some useful algebra.

6.5.1 Rings and Ideals

A **ring** can be thought of a field where the rules of division have been lost, for example, the integers are a ring, as are, say, 2×2 real matrices.

A subset I of a ring R is an **ideal** if for any $x \in I$ and any $r \in R$ then $rx \in I$. For example, the set of even integers is an ideal in the ring of integers, since any multiple of an even integer is even.

An ideal is called **principal** if there is an element $g \in I$ such that I consists all multiples of g, and we write $I = \langle g \rangle$ to show this. A ring is a **principal ideal ring** if all ideals are principal. The Euclidean Algorithm implies that $\mathbb{Z}$ is a principal ideal ring, for example, the set of all even integers is the ideal $\langle 2 \rangle$.

The set $\mathbb{F}[x]/\langle x^n - 1 \rangle = R$ of polynomial remainders modulo $x^n - 1$ over $\mathbb{F}$ is not a field since $x^n - 1$ is clearly not irreducible, but it is a ring. It is in fact a principal ideal ring.

6.5.2 Cyclic codes as ideals

Now, suppose we have a cyclic code $\mathcal{C}$. We identify codewords with elements of $R = \mathbb{F}[x]/\langle x^n - 1 \rangle$ by

$$C(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}.$$

Then

$$\begin{aligned} xC(x) &= xc_0 + c_1 x^2 + \cdots + c_{n-2} x^{n-1} + c_{n-1} x^n \\ &= c_{n-1} + c_0 x + c_1 x^2 + \cdots + c_{n-2} x^{n-1} \end{aligned}$$

so multiplying by x corresponds to a cyclic shift in the codeword.

In this way we view $\mathcal{C}$ as a subset of R. Now as the code is cyclic, $xC(x)$ also corresponds to a code word, and we can extend this result to prove that $\mathcal{C}$ must be an ideal of R. But as R is a principal ideal ring we have

A subset $\mathcal{C}$ of R is a cyclic code if and only if there exists $g(x) \in \mathcal{C}$ with

$$\mathcal{C} = \langle g(x) \rangle = \{a(x)g(x) : a(x) \in R\}.$$

This $g(x)$ is the unique monic polynomial of least degree that lies in $\mathcal{C}$, and it must be a factor of $x^n - 1$ over $\mathbb{F}$. We call $g(x)$ the **generator polynomial** of $\mathcal{C}$, and any factor $g(x)$ of $x^n - 1$ will lead to a cyclic code.

If $g(x) = g_0 + g_1 x + \cdots + g_m x^m$, where $g_m = 1$, then the $k \times n$ matrix G given by

$$G = \begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_m & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & g_2 & \cdots & g_m & 0 & \cdots & 0 \\ \vdots & & \ddots & & \ddots & & \ddots & & \vdots \\ 0 & \cdots & 0 & g_0 & \cdots & \cdots & \cdots & \cdots & g_m \end{pmatrix}$$

is called a **generator matrix** for $\mathcal{C}$ and $\mathcal{C}$ has dimension $k = n - m$. The ith row of G corresponds to the coefficients of $x^{i-1} g(x)$. Since the ideal $\langle g(x) \rangle$ is all multiples of $g(x)$, the code consists of all linear combinations of the rows of G.

Writing $x^n - 1 = g(x)h(x)$ over $\mathbb{F}$ gives the **check polynomial**

$$h(x) = h_0 + h_1 x + \cdots + h_k x^k \quad \text{where} \quad h_k = 1.$$

The name arises from the fact that if $c(x)$ corresponds to a correct code word, then as $c(x) = a(x)g(x)$ for some $a(x)$, $c(x)h(x) = a(x)g(x)h(x) = a(x)(x^n - 1) = 0$ in R. Conversely, if

$c(x)h(x) = 0$ in R, then $c(x)h(x) = a(x)(x^2-1)$ in $\mathbb{F}[x]$ for some $a(x)$, and so $c(x) = a(x)g(x)$ in $\mathbb{F}[x]$.

We also have the $m \times n$ matrix

$$H = \begin{pmatrix} h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & \cdots & 0 \\ 0 & h_k & \cdots & h_1 & h_0 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & \ddots & & \vdots \\ 0 & \cdots & 0 & 0 & h_k & \cdots & h_1 & h_0 \end{pmatrix}$$

as a parity check matrix for $\mathcal{C}$.

Examples:

1. The cyclic codes of length 4 over $\mathbb{Z}_3$.

   We have
   $$x^4 - 1 = (x-1)(x+1)(x^2+1) \quad \text{over} \quad \mathbb{Z}_3$$
   with each factor irreducible.

   Combining the irreducible factors in all possible ways gives 8 different cyclic codes of length 4 over $\mathbb{Z}_3$.

   For example if
   $$\begin{aligned} g(x) &= x^2+1 \\ h(x) &= (x-1)(x+1) = x^2-1 \end{aligned}$$
   then the corresponding cyclic code has
   $$G = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \quad H = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$
   and is $\mathcal{C} = \{0000, 1010, 0101, 1111, 2020, 0202, 2121, 1212, 2222.\}$

   Here $n = 4, k = m = 2$ and there are $3^2 = 9$ elements.

2. Suppose $f(x)$ is primitive of degree r over $\mathbb{Z}_2$.

   That is, $f(x)$ is irreducible and its root α generates $\mathbb{Z}_2[x]/\langle f(x)\rangle = \mathrm{GF}(2^r)$.

   As before $H = (1\ \alpha\ \alpha^2\ \alpha^3\ \cdots\ \alpha^{2^r-2})$ is the parity check matrix of the Hamming (n,k) code (in this particular order). We have $n = 2^r-1, m = r$ and $k = 2^r - r - 1$.

   Then, for a binary vector $\boldsymbol{c}$ of length n,
   $$\begin{aligned} & \boldsymbol{c} \in \text{ Hamming}\,(n,k) \\ \Leftrightarrow\ & C(\alpha) = 0 \quad \text{where} \quad C(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1} \\ \Leftrightarrow\ & C(x) = 0 \quad \text{since} \quad \alpha \quad \text{is primitive} \\ \Leftrightarrow\ & C(x) \quad \text{is a multiple of} \quad f(x) \\ \Leftrightarrow\ & C(x) \ \in\ \langle f(x)\rangle \end{aligned}$$

   Hence this version of Hamming (n,k) is **cyclic** with generator polynomial $f(x)$.

   (In fact all BCH codes are cyclic.)

   For example with $r=4,\ n=15,\ m=4,\ k=11$ we used $f(x) = x^4+x+1$ so
   $$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & & \ddots & & & & & & \ddots & & & & \ddots & & \vdots \\ \vdots & & & \ddots & & & & & & \ddots & & & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

is a generator matrix.

Also $x^{15} - 1 = (x^4 + x + 1)(x^{11} + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1)$ over $\mathbb{Z}_2$, so the check polynomial is

$$h(x) = x^{11} + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1$$

and another Hamming check matrix is the cyclic matrix

$$H' = \begin{pmatrix} 1&0&0&1&1&0&1&0&1&1&1&1&0&0&0 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \end{pmatrix}.$$

6.5.3 Encoding Cyclic codes

There are several ways we could encode messages with cyclic codes. Suppose we have word $\mathbf{w} = w_1 w_2 \dots w_k$ to be encoded with a cyclic(n,k) code $\mathcal{C} = \langle g(x) \rangle$.

The most obvious way of encoding $\mathbf{w}$ is to create a polynomial of degree k

$$w(x) = w_1 + w_2 x + \cdots + w_k x^k$$

with the bits as coefficients and encode it as the polynomial $w(x)g(x)$. This is in $\mathcal{C}$ as $\mathcal{C}$ is an ideal.

The problem with this method is that it is not systematic: we cannot tell the check and information bits apart.

For a systematic encoding, define the polynomial

$$w(x) = w_1 x^{n-1} + \cdots w_k x^{n-k} = \sum_{i=1}^{k} w_i x^{n-i},$$

which we know is in $R = \mathbb{Z}_p[x]/\langle x^n - 1 \rangle$.

Then let $r(x)$ be the remainder on dividing $w(x)$ by $g(x)$: $r(x) \equiv w(x) \pmod{g(x)}$, which has degree at most $m - 1 = n - k - 1$. So $w(x) - r(x)$ is a multiple of $g(x)$, and hence is in $\mathcal{C}$.

Since the information bits are the coefficients of powers x^{n-k} and higher, and the check bits (coefficients of $-r(x)$) are coefficients of powers less than x^{n-k}, this encoding is systematic, so decoding is straightforward.

Error correcting is done using a nearest member strategy.

Alternatively, we could put the generating matrix into standard form, i.e. into row reduced echelon form, and then use the leading columns as the check bits, as we did in chapter 2.

Fortunately, there is a nice result that allows us to do this without any row reduction: it relies on the division algorithm.

Let $R_i(x)$ be the remainder on dividing $x^{n-k+i-1}$ by $g(x)$ for $i = 1, \dots k$ and A be the matrix whose ith row is the coefficients of $R_i(x)$.

Theorem 6.2 *With the notation above, the standard form generating matrix G and associated parity check matrix H of code $\mathcal{C} = \langle g(x) \rangle$ are given by*

$$G = (I_k \mid -A), \qquad H = \left(A^T \mid I_{n-k}\right).$$

Proof: For any polynomial $f(x) \in \mathbb{Z}_p[x]$, let $\mathrm{rem}_g(f(x))$ be the remainder on dividing $f(x)$ by the generating polynomial $g(x)$. Then $f(x) - \mathrm{rem}_g(f(x))$ is a multiple of $g(x)$ and so is in $\mathcal{C}$.

It follows that the k polynomials

$$Q_i(x) = x^{n-k+i-1} - \mathrm{rem}_g(x^{n-k+i-1}), \quad i = 1, \dots k$$

are in $\mathcal{C}$, and in fact must generate $\mathcal{C}$, being independent. Hence the set

$$x^k Q_i(x) = x^{n+i-1} - x^k \operatorname{rem}_g(x^{n-k+i-1})$$

also generates $\mathcal{C}$.

But in $R = \mathbb{Z}_p/\langle x^n - 1\rangle$, $x^{n+i-1} = x^{i-1}$, and thus the set of polynomials

$$G_i(x) = x^{i-1} - x^k \operatorname{rem}_g(x^{n-k+i-1}), \quad i = 1, \dots k$$

generates $\mathcal{C}$. The matrix whose ith row is the coefficients of $G_i(x)$ is the G of the theorem.

The shape of H follows from the work of chapter 2. □

Once again, we decode with a nearest member strategy, see section 2.15.1 on how this may be implemented.

6.6 The Golay codes.

Consider $n = 23$ and $p = 2$. Then the factorization

$$\begin{aligned} x^{23} - 1 = (x - 1)\,&(x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1) \\ &\times (x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1) \end{aligned}$$

yields 8 cyclic codes. In particular let

$$\mathcal{C}_1 = \langle g_1(x)\rangle, \quad \mathcal{C}_2 = \langle g_2(x)\rangle$$

where

$$\begin{aligned} g_1(x) &= x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1 \\ g_2(x) &= x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1. \end{aligned}$$

We notice that $g_2(x) = x^{11} g_1(1/x)$ so $g_2(x)$ is the **reciprocal polynomial** to $g_1(x)$. That is, the coefficients of g_2 are the coefficients of g_1 in the reverse order.

This means $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent codes (you can obtain one from the other by permuting the positions of the entry of each codeword).

Using this fact it can be shown (see for example Hill) that $\mathcal{C}_1$ has minimum distance $d = 7$. Hence $\mathcal{C}_1$ is a binary code with $n = 23,\ k = 12,\ m = 11$ and $d = 7$.

Now as $d = 7$ then $t = 3$, so $\mathcal{C}_1$ is a 3-error correcting code and

$$\begin{aligned} \frac{2^{23}}{\sum_{i=0}^{3} \binom{23}{i}} &= \frac{2^{23}}{1 + 23 + \binom{23}{2} + \binom{23}{3}} = \frac{2^{23}}{2048} \\ &= \frac{2^{23}}{2^{11}} = 2^{12}. \end{aligned}$$

That is, $\mathcal{C}_1$ is a perfect code (see Chapter 2).

This is the **binary Golay code.**

It has the following 12×23 generator matrix G given by

$$\begin{pmatrix} 1&0&1&0&1&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&1&0&1&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&1&0&1&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0\\ &&\vdots&&&&\vdots&&&&\vdots&&&&&\vdots&&&&&\vdots&& \\ 0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1&1 \end{pmatrix}$$

Also $x^{23}-1 = g_1(x)h(x)$ where

$$\begin{aligned} h(x) &= (x-1)g_2(x) \\ &= x^{12}+x^{11}+x^{10}+x^9+x^8+x^5+x^2+1 \end{aligned}$$

so a parity check matrix is the 11×23 matrix

$$H = \begin{pmatrix} 1&1&1&1&1&0&0&1&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0\\ 0&1&1&1&1&1&0&0&1&0&0&1&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&1&1&0&0&1&0&0&1&0&1&0&0&0&0&0&0&0&0\\ &&\vdots&&&&\vdots&&&&\vdots&&&&&\vdots&&&&&\vdots&& \\ 0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&1&0&0&1&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&1&0&0&1&0&1 \end{pmatrix}$$

For the **ternary Golay Code** over $\mathbb{Z}_3$

$$\begin{aligned} x^{11}-1 &= (x-1)(x^5+x^4-x^3+x^2-1) \\ &\quad (x^5-x^3+x^2-x-1) \\ &= (x-1)g_1(x)g_2(x). \end{aligned}$$

Again $g_2(x) = -x^5 g_1(1/x)$, so $g_2(x)$ is minus the reciprocal polynomial, and

$$\mathcal{C}_1 = \langle g_1(x)\rangle \text{ is equivalent to } \mathcal{C}_2 = \langle g_2(x)\rangle,$$

$$\mathcal{C}_1 \text{ has } d=5,\ n=11,\ m=5,\ k=6,\ t=2,$$

$$\frac{3^{11}}{\sum_{i=0}^{2}\binom{11}{i}2^i} = \frac{3^{11}}{1+11\times 2+\binom{11}{2}2^2} = \frac{3^{11}}{243} = \frac{3^{11}}{3^5} = 3^6$$

so this code is also perfect. A generator matrix for the ternary Golay code is the 6×11 matrix

$$G = \begin{pmatrix} -1&0&1&-1&1&1&0&0&0&0&0\\ 0&-1&0&1&-1&1&1&0&0&0&0\\ 0&0&-1&0&1&-1&1&1&0&0&0\\ 0&0&0&-1&0&1&-1&1&1&0&0\\ 0&0&0&0&-1&0&1&-1&1&1&0\\ 0&0&0&0&0&-1&0&1&-1&1&1 \end{pmatrix}$$

and

$$\begin{aligned} h(x) &= (x-1)(x^5-x^3+x^2-x-1) \\ &= x^6-x^5-x^4-x^3+x^2+1 \end{aligned}$$

so a check matrix is the 5×11 matrix

$$H = \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & -1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 & -1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & -1 & -1 & 0 & 1 \end{pmatrix}$$

Notes:

1. We have seen that the following codes are perfect:

   a. the $(2t+1)$-repetition binary codes where $d = 2t+1$,

   b. the radix r Hamming codes, $d = 3$,

   c. the Golay binary code, $n = 23, k = 12, d = 7$,

   d. the Golay ternary code, $n = 11, k = 6, d = 5$.

   The above list in fact contains all the known perfect linear codes.

2. If we extend the binary Golay code as in Chapter 2 we obtain the **extended binary Golay code** G_{24} which is a very interesting code with links to many areas of mathematics.

   Some facts about it are: it has $n = 24,\ m = k = 12,\ d = 8$, rate $R = \frac{1}{2}$.

   It was used by Voyager spacecraft (launched 1977) to send back colour photos of Jupiter and Saturn ($4096 = 2^{12}$ colours were used).

   It is a self dual code, which in coding terminology means that the rows of its generating matrix are orthogonal.

   Its weight numbers are

$$A_0 = A_{24} = 1, \quad A_8 = A_{16} = 759, \quad A_{12} = 2576.$$

   The codewords of weight 8 are a Steiner $S(5, 8, 24)$ system, meaning they form a set of subsets of a set with 24 elements, each subset of size 8, such that every subset of 5 elements is contained in exactly one 8-set.

   It is the only (non-equivalent) code with $n = 24,\ k = 12,\ d = 8$.

   It has a simple decoding algorithm (see Roman).

   Its automorphism group is one of the important sporadic simple groups, the Mathieu group M_{24}.

   It is related to an extremely dense packing of spheres in 24-dimensional space, the Leech packing.

6.7 Where next?

So far our error correcting codes have all been block codes:

A stream of data is broken up into information words of length k, each encoded in turn into a code word of length n, a process that naturally fits with block ciphering techniques. In this scheme there is a one-to-one relationship between information words and codewords.

The disadvantage of these codes is that we need the whole (typically large) codeword before we can start to decode, causing delays. It is possible to create codes that use small information words of course, but good error correcting capabilities require large distances between codewords and thus large blocksizes.

A way around this is to design an encoder with memory — use the previous m information words as well as the current one.

The most important of these types of codes are **c**onvolution codes. Bose's book (chapter 6) discusses them and their coding and decoding strategies. In fact, it is common to combine a convolution and a Reed-Solomon code in what is known as a **c**oncatenated code: these codes can approach the channel capacity given by Shannon's Theorem, theorem 4.6.

A more recent advance are Turbo Codes, introduced in 1993, which Bose describes as "a quasi-mix of between Block and Convolutional Codes". See his book or

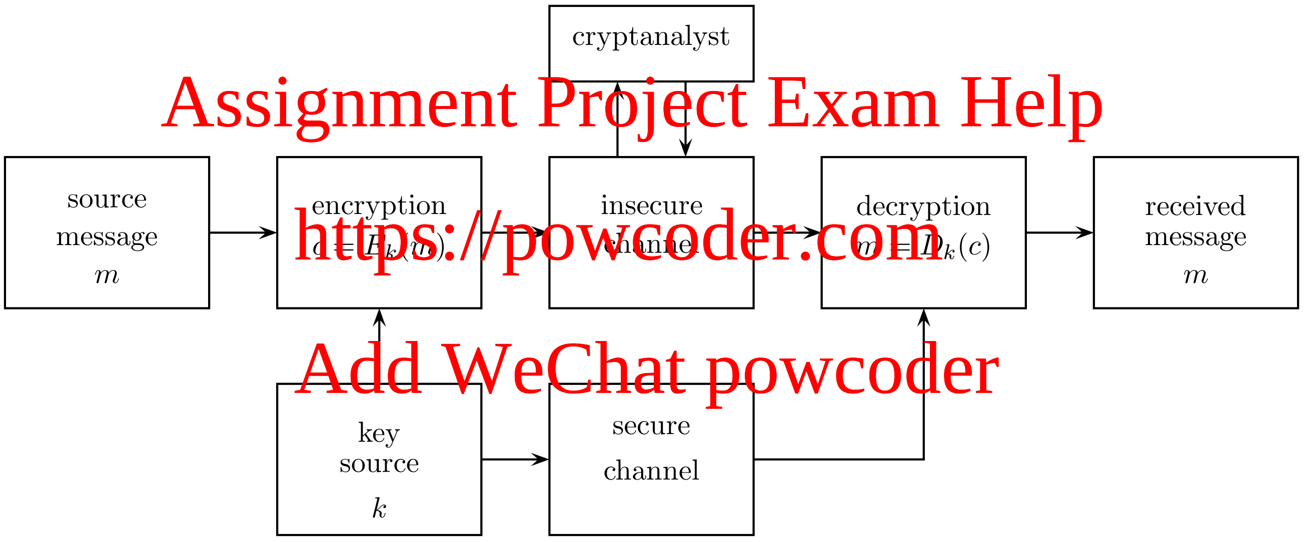
`www.complextoreal.com`

Chapter 7

Cryptography (Ciphers)

We will briefly look at classical cryptography, then more closely at modern cryptography.

Here is a diagram that covers both, more or less.



A **message** m from among a set M of all possible messages is **encrypted** (or enciphered or encoded), using one of a set $\{E_k\}$ of **encryption functions** (or cryptographic transformations)

$$E_k : M \to C$$

into an **encrypted message** $c \in C$, where C is the set of all possible encrypted messages.

The encryption function E_k depends on the particular **key** k chosen from a set K of all possible keys.

The encrypted message c is then transmitted to the receiver, who decrypts it using the decryption function

$$D_k : C \to M$$

using the same key k. Here D_k is the inverse of E_k, which means that

$$D_k(c) = D_k(E_k(m)) = m.$$

We also call m the **plaintext** and c the **cipher text**.

It is assumed that the sender and receiver have the *same* key which has to be transmitted (at some time) by a completely secure method from one to the other.

In practice the set $\{E_k\}$ and the corresponding $\{D_k\}$ are fixed and only the value of k differs from time to time.

The cryptanalyst or listener or spy can take one of a variety of roles, ranging from

passive — simply listening in to c and trying to guess m with or without guessing k,

to

active — intercepts c and retransmits a modified c', or sends c' impersonating the sender, or retransmits c at another time.

Sometimes the cryptanalyst can arrange that the sender sends the cryptanalyst's choice of message m. Then observing the corresponding cipher text c, perhaps the cryptanalyst can find k more easily. Then they can use k on other messages.

Experience has taught us that we need to make sure a crypto-system is **secure by design** rather than **secure by obscurity**. That is, we assume that the cryptanalyst knows everything there is to know about the crypto-system's design ($\{E_k\}$) and knows M, but does not know not the actual key k used. This is often referred to as **Shannon's maxim** "*the enemy knows the system*", after the American engineer and mathematician Claude Shannon (1916-2001).

The cryptanalyst's task is then to **break the code** by finding k.

Clearly the key k is the "key" to the whole system and must be securely kept secret and transmitted in secret and must be the same at both ends. How to achieve this is the **key distribution problem**.

7.1 Some classical cryptosystems:

7.1.1 Caesar cipher

(Named after the Roman dictator Gaius Julius Caesar.)

Cyclicly shift each letter k places forward, e.g. for $k = 2$,

```
plain   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
cipher  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
```

If letters are represented by $\mathbb{Z}_{26}$, then we have, for key k,

$$\begin{aligned} E_k(i) &= i + k \pmod{26} \quad \text{for all } i \\ \text{and} \quad D_k(j) &= j - k \pmod{26} \quad \text{for all } j. \end{aligned}$$

There are 26 keys k, so this cipher is easily broken by simply trying all possible keys until it makes sense. Caesar himself used $k = 3$.

7.1.2 Simple (monoalphabetic) substitution cipher

Replace each letter by another symbol, such as

```
plain   A B C D E F . .  .
cipher  - ; / ' ( + .  .  .
```

More simply, replace each letter by another letter. That is, apply a permutation of $\{$`A`,`B`,...,`Z`$\}$ or of $\mathbb{Z}_{26}$.

Let π be the permutation of $\mathbb{Z}_{26}$ used, then

$$\begin{aligned} E_\pi(i) &= \pi(i) \quad \forall i \\ D_\pi(j) &= \pi^{-1}(j) \quad \forall j \end{aligned}$$

There are $26! \approx 4 \times 10^{26}$ possible keys so it is impossible to try them all.

However, if the plaintext is English, or anything else structured, then a frequency count will give good guesses for the commonly occurring letters. Simply count the frequency of commonly occurring letters in the ciphertext and use the English frequency statistics to guess what they stand for in the plaintext. The rest can be broken by using the redundancy naturally present in English.

Referring to Appendices A.3 and A.4 on letter statistics in English and assuming words are run together without any spaces, we notice that

```
E T A O I N S R H ···
```

are the most common English letters, in that order,

```
TH HE IN ER AN RE ···
```

the most common pairs and

```
THE AND ING ION ···
```

the most common triples.

Since an arbitrary permutation is hard to remember, often the permutation is constructed from a keyword — but then it is even easier to break.

For example, the keyword 'CODEBREAKING' starting at K and with the other letters in order gives the substitution

```
plain   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
cipher  P Q S T U V W X Y Z C O D E B R A K I N G F H J L M
```

7.1.3 Transposition cipher

Mix up the order of the letters by dividing them up into blocks of length r and applying a permutation $\pi \in \mathrm{Sym}(r)$ to the letter order. For example, with blocks of length 5 we could use the permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 4 & 5 & 2 \end{pmatrix} \quad \text{maps 1st letter} \to \text{3rd letter}, \quad \text{2nd} \to \text{1st}, \quad \text{and so on.}$$

```
plain   T H I S I   S A N E X   A M P L E
cipher  H I T I S   A X S N E   M E A P L
```

Frequency counts of pairs or triples and common sense allow these ciphers to be easily broken.

7.1.4 Combined systems

These use both a transposition and a substitution. They are much harder to break by hand but are still breakable using frequency counts.

7.1.5 Polyalphabetic substituion ciphers

The simplest version is the **Vigenère cipher** (1586).

Here, r different Caesar ciphers are applied periodically. The Caesar cipher used is indicated by the letter that ‘A’ maps to and usually the r letters form some easily remembered word — the keyword.

All the Caesar substitution alphabets named ‘A’ to ‘Z’ are shown in a table called the Vigenère table. See Appendix A.5.

For example, if the keyword is ‘code’ then

```
key     c o d e c o d e c o d e c o d
plain   T H I S I S A N E X A M P L E
cipher  V V L W K G D R G L D Q R Z H
```

So the Caesar cipher ‘C’, i.e.

$$\begin{pmatrix} \mathtt{A} & \mathtt{B} & \mathtt{C} & \cdots & \mathtt{Z} \\ \mathtt{C} & \mathtt{D} & \mathtt{E} & \cdots & \mathtt{B} \end{pmatrix}$$

is used for letters $1, 5, 9$ etc. and Caesar cipher ‘O’, i.e.

$$\begin{pmatrix} \mathtt{A} & \mathtt{B} & \mathtt{C} & \cdots & \mathtt{Z} \\ \mathtt{O} & \mathtt{P} & \mathtt{Q} & \cdots & \mathtt{N} \end{pmatrix}$$

for letters $2, 6, 10$ etc.

A more complicated version of this is the **periodic polyalphabetic substitution cipher** where, instead of Caesar ciphers, a set of simple substitution alphabets (permutations) is used periodically.

If the **length** r of the repeating keyword or the length r of the period is known, then both Vigenère and periodic polyalphabetic ciphers are easily broken as it simply becomes a problem of solving r different Caesar or simple substitution ciphers, using

the first substitution cipher for letters $1, r + 1, 2r + 1, \cdots$
the second substitution cipher for letters $2, r + 2, 2r + 2, \cdots$
and so on.

Consecutive pairs and triples of letters can no longer be used to break this code. However, the frequencies of individual letters can be used to break each of the r pieces separately, and by breaking several of the pieces at the same time you can use some limited pairs and triples statistics.

Vigenère ciphers were commonly used in the 17th–19th century until a systematic method was developed in 1863 by the Prussian cryptographer Freidrich Kasiski to find the period **length**. Prior to this, trying all reasonable r was the only way to tackle the problem. Modern methods of cracking Vigenère ciphers typically go back to this brute force method, but naturally use a computer to do it.

7.1.6 Kasiski’s method

This estimates r from the “number of coincidences”.

Let the English letters be represented by $i = 0, 1, 2, \cdots, 25$ and let f_i be the frequency of letter i in some text.

Suppose that we pick two letters at random from the text. Then the probability that they are the same is called the **probability of coincidence**.

Firstly, if the text consists of random letters then

$$\text{the probability of coincidence } = \frac{1}{26} = 0.0385.$$

To see this, choose two positions, then the probability that the letter in the second position equals the letter in the first position is $1/26$.

On the other hand, if the text is English then the letters are not random and we have

$$p_0 = p(\mathtt{A}) = 0.0804, \ldots, p_{25} = p(\mathtt{Z}) = 0.0009$$

(using the values from Appendix A.4).

Now, the probability of coincidence is given by

$$\sum_{i=0}^{25} p_i^2 = 0.0658 \ .$$

To see this, choose a pair of positions randomly. The probability both are '`A`' is p_0^2, and the probability both are '`B`' is p_1^2, and so on. Then sum these probabilities.

From the given text there are f_0 '`A`'s, f_1 '`B`'s etc. Suppose that the ciphertext contains n letters in total. Then we have

$$\binom{f_i}{2} = \frac{1}{2} f_i(f_i - 1)$$

pairs of the letter i, so there are

$$\sum_{i=0}^{25} \frac{1}{2} f_i(f_i - 1)$$

coincidences in total out of

$$\binom{n}{2} = \frac{1}{2} n(n - 1)$$

pairs in total.

Hence we have an estimate of the probability of coincidence given by

$$I_c = \frac{\sum \frac{1}{2} f_i(f_i - 1)}{\frac{1}{2} n(n-1)} = \frac{\left(\sum f_i^2\right) - n}{n^2 - n}$$

which is called the **index of coincidence**, and is independent of r.

Now, if a simple substitution cipher is used, then the letters will simply be permuted and so $\sum f_i^2$ remains unaltered, so we should still have the same $I_c \approx 0.0658$.

However, if r substitutions are used, then I_c will be somewhere between 0.0658 and 0.0385.

To analyse this further, suppose that we write the n letter message in r rows each containing n/r letters. (For simplicity, assume that $r \big| n$ so that n/r is an integer.) Do this by placing the first r letters in the first column in order, the next r letters in the second column in order, and so on.

Each row will have had the same substitutions applied, but different rows will have had different substitutions applied.

Now pick pairs of letters at random and we get either

(a) both in the same row:

Here there are $\dfrac{1}{2}\,\dfrac{n}{r}\left(\dfrac{n}{r} - 1\right)$ choices within a row times r rows, and also the probability of a coincidence is approximately 0.0658.

(b) from different rows:

Now there are $\frac{1}{2}n\left(n-\frac{n}{r}\right)$ choices, and the probability of a coincidence is approximately 0.0385.

Hence we expect the number of coincident pairs to be approximately

$$\frac{1}{2}\,\frac{n}{r}\left(\frac{n}{r}-1\right)\times r\times 0.0658+\frac{1}{2}n\left(n-\frac{n}{r}\right)\times 0.0385$$

and it is also $=\frac{1}{2}n(n-1)I_c$ from the definition of I_c.

Solving for r we get

$$r\approx\frac{0.0273n}{(n-1)I_c-0.0385n+0.0658}$$

and so we can *estimate* r from I_c.

Roughly we have

$$I_c\approx\frac{1}{r}(0.0273)+0.0385\quad\text{as } n\to\infty,$$

and for small r we have the table of values

r	=	1	2	3	4	5	10	∞
I_c	=	.066	.052	.048	.045	.044	.041	.0385

For further evidence that we know r, if we look at the I_c for each of the r rows above we should get approximately 0.0658 for each of them since each is a simple substitution.

Note: This method only works for *very long texts* since it relies on approximations and estimates of probabilities which only hold on average. For examples from problem sheets or exams, you are only given a much shorter text and the estimate of r given by Kasiski's method may not be very accurate.

Even more evidence for period length r can be obtained by looking for repeated pairs, triples and quadruples of letters in the cipher text. It is most likely that these have occurred because the same, say, triple of plain text letters is enciphered by the same part of the periodic substitution.

Hence the positions of the first letters of such triples etc, are likely to differ by a multiple of the keyword length. For example, about every 60 letters you expect to get a '`the`' so, for a period 4 polyalphabetic substitution you would expect to get the '`the`' enciphered by the same code triple every 240 letters.

Note: This is often more useful than I_c, especially for short texts.

7.1.7 Non-periodic polyalphabetic substitutions

By eliminating the periodic nature of the alphabet used we can make the polyalphabetic ciphers much more secure.

For example:

plaintext feedback

```
key     c o d e t h i s i s a n e x a
plain   T H I S I S A N E X A M P L E
cipher  V V L W B Z I F M P A Z T I E
```

ciphertext feedback

```
key     c o d e v v l w d n l j h k l
plain   T H I S I S A N E X A M P L E
cipher  V V L W D N L J H K L V W V P
```

text from some novel or book

```
key     t h e s q u a r e o n t h e h y p o t e n u s e .  .  .
plain   T H I S I S A N E X A M P L E
cipher  M O M K Y M A E I L N F W P L
```

A further historically interesting example are **rotation ciphers**. The rotation cipher R^j is a sliding Caesar cipher, where if the ith letter of the message corresponds to letter k, we replace it by $i + j(k-1) \pmod{26}$. So using R^1 the word `CONVOY` is becomes `CPPYSD`. Alternatively, you can think of R^1 as a substitution code using the alphabet as the keyword: `abcdefgh`....

If we iterate R^1 j times we clearly get R^j, which is also a substitution code of course. For example R^3 can be thought of as using keyword `adgjmpsvybehknqtwzcfilorux`.

Rotation ciphers are very weak, but do have the advantage that for long messages the letter frequencies are flattened (unless $\gcd(j, 26) \neq 1$).

If you follow a rotation cipher with a substition cipher you get a much stronger cipher: these are the basis of the Enigma coding machines used by the Axis powers in WWII.

All these various substitution ciphers can be broken if the key is text in a natural language (e.g. English or German), as there are still underlying probabilities that can be exploited — however large amounts of text (and/or cribs) are needed for this.

Taking this approach to the extreme, we could use as a key a *random sequence of letters*. This is called the **Vernam cipher** (1917) and is also called a **one-time pad cipher**.

The sender and receiver need the same sequence of random letters: preferably only two copies are ever made. The sequence needs to be as long as all the messages that need to be sent, so that the same sequence of key letters is never used more than once. Spies would use a page from their **cipher pad** to encipher a message then destroy it. The only person who could then decipher it was the other holder of the random letters.

This cipher has ultimate security since, given some cipher text, then any plaintext of the same length is just as likely as any other: the only information passed is its length.

However, if the cipher pad is captured then the enemy could send false messages, unless there is some agreement to put a signature or password into the message before enciphering (see section 7.6 on signing messages).

7.2 Types of ciphers

All the classical systems we have considered are **stream ciphers**, as they encipher one stream at a time. There are many other systems which encipher more than one letter at once, such as pairs, triples of letters, and so on.

One such method uses matrices for encoding and decoding, see problems. But if the matrix is small then these only increase the effective size of the alphabet by a small amount, and make them marginally more secure.

In general, a **block cipher** divides the message into blocks of n letters which are enciphered as a whole into another block of n cipher letters, see section 7.2.2.

A **product cipher** is a block cipher that combines both transpositions and substitutions over and over with variations, to increase security.

7.2.1 Stream ciphers

The one-time pad, cipher machines, polyalphabetic ciphers and so on can all be modelled mathematically as follows:

Given plaintext symbols $m_1, m_2, \ldots$ and given a key k, use a fixed algorithm to generate from k a sequence of shifts $s_1, s_2, \ldots$

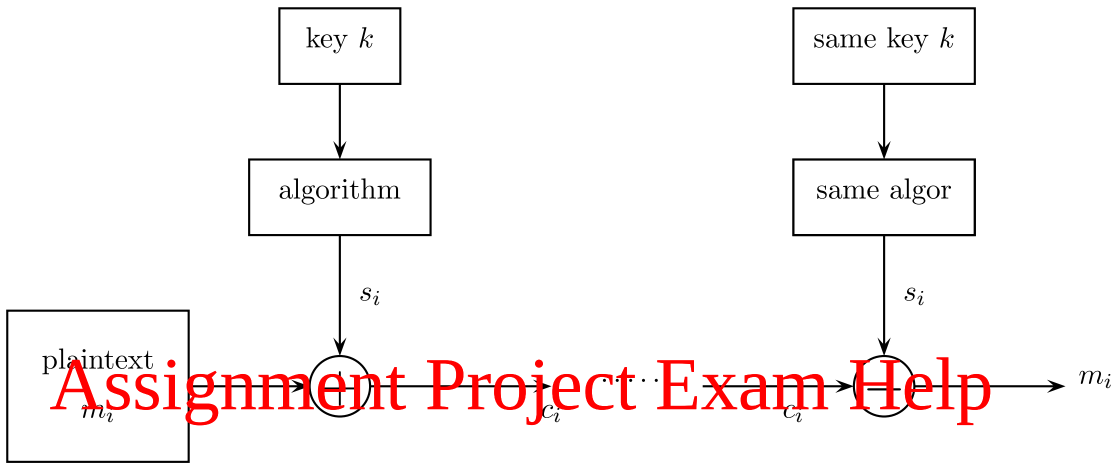
then the ciphertext is the sequence $c_1, c_2, \ldots$ where

$$c_i = m_i + s_i \pmod{26}.$$

This is a **stream cipher** as it enciphers the stream of symbols one at a time.

For the one-time pad $s_1, s_2, \ldots$ are truly random and for other cipher machines they are only pseudo-random.

Any pseudo-random number generator can be used to produce a stream cipher.



With the advent of electronics, this style of cipher took off and is normally used with binary streams — in this case $\oplus$ and $\ominus$ are the same.

Not all pseudo-random number generators are secure for stream ciphers. For example the linear congruential and LFSR type are susceptible to being broken. If the cryptanalyst can obtain some matching m_i and c_i then they can find s_i and hence determine the constants used in the pseudo-random number generator and so break the cipher.

Multiplexed LFSRs have been commonly used in the fairly recent past for stream ciphers as they are much harder to break.

7.2.2 Block and product ciphers

A **block cipher** is a cipher where a block of fixed length n of letters is enciphered as a whole into another block of cipher letters. Effectively they are simple substitutions but with massively large alphabets, as long as n is large. A commonly used block length is 128 bits so the alphabet has 2^{128} characters.

A **product cipher** is one that combines both transpositions and substitutions over and over with variations: each application of the process is usually called a **round**. Preferably the substitutions are also non-linear (unlike all our earlier examples). In such a cipher any plaintext redundancy is *confused* by substitution and *diffused* by permutation.

7.2.3 Encryption Standards

With the rise of computers in the 1960s and 1970s, it became clear that standard protocols were needed. In 1976 the US Government adopted DES (Data Encryption Standard), originally developed by IBM and modified by the NSA (US National Security Agency), and it became a de facto standard. DES is a 64-bit block cipher of the product cipher type and uses a 64-bit

key containing 8 check bits, so it is effectively a 56-bit key cipher, and has 16 rounds. For more information see Salomaa, Section 1.4, and the complete details of the DES system are found at the web site
`http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`
or in Schneier's book. (Rumour has it that the NSA could always break DES in a few minutes using its classified knowledge of the design of DES.)

DES suited the hardware limits of 1977 but by the mid 1990s was dated. It is still used in the form of Triple-DES, where the message is encoded using three applications of DES to each block with different keys, and that still seems to be secure.

IDEA (International Data Encryption Algorithm, 1991) was intended as a replacement for DES, and is an encryption algorithm with a key length of 128 bits, which is over twice as long as DES, and $8\frac{1}{2}$ rounds. The best that has been done to break IDEA is a crack of a 6 round application with chosen plaintext in 2007. IDEA is used in the PGP (Pretty Good Privacy) encryption package (see later), and its patent is owned by Ascom-Tech AG.

In 2002 the US government adopted AES (Advanced Encryption Standard), originally called **Rijndael** after its Belgian inventors Joan Daemen and Vincent Rijmen. AES is a 128-bit block cipher with key of either 128, 192 or 256 bits and is an also an open cipher (not covered by any patent).

AES uses 10, 12 or 14 rounds and is noticeably quicker than DES. AES has a reasonably simple algebraic description which DES does not, and this is part of the reason why it is fast. Unfortunately, this is also considered to be its weakness. There are known attacks that are significantly better than brute force, but they still seem to be computationally infeasible. Wikipedia contains a reasonably detailed description on the AES process.

7.3 One-way, hash and trapdoor functions

We now define some special kinds of functions which are very useful in cryptography.

A **one-way function** f is one for which

1. given x, it is easy to find $f(x)$,
2. given $f(x)$, it is hard to find x.

A **one-way hash function** (or cryptographic hash function) in addition satisfies

3. f takes a message of **any** length and maps it to a **fixed** length value,
4. given x, it is hard to find an x' with $f(x) = f(x')$,
5. it is hard to find *any* pair x, x' with $f(x) = f(x')$.

Here **hard** here means "impractical to find with huge but finite resources and time".

Requirements 4) and 5) are known as **collision resistance**.

One-way hash functions are used in UNIX password encryption and in ATM machines.

Another example of a 1-way function is:

Given a large prime p and primitive element g in $\mathbb{Z}_p$, let

$$f(x) \equiv g^x \pmod{p}.$$

Now, g^x is easy to calculate (in $\mathrm{O}(\log p)$ time).

However, the inverse problem, to find x given g^x, is called the **discrete logarithm problem** and is hard (at the moment): the best known algorithms for solving the discrete logarithm problem are about $O(\exp(c\sqrt{\log p \log\log p})$.

A **trapdoor function** or 1-way trapdoor function f is one for which

1. given x it is easy to find $f(x)$,
2. given $f(x)$ and nothing else, it is hard to find x,
3. given $f(x)$ and some small extra information, it is easy to find x.

An example is the following:

Let p, q be two large Blum primes, so both are $\equiv 3 \pmod 4$. Define $n = pq$. Then

$$f(x) \equiv x^2 \pmod n$$

is a trapdoor function, since:
finding square roots mod n is hard,
but finding square roots mod primes p and q is relatively easy and from these you can easily find a square root mod $n = pq$ by a result known as the Chinese Remainder Theorem.
Hence knowing p and q is the trapdoor.

These types of functions are the tools of modern cryptography.

7.4 Cryptography in the Public Eye

All the classical cryptosystems such as DES, IDEA and similar systems rely on both sender and receiver having the **same secret key** — they are called **symmetric** cryptosystems.

They also need fairly regular changing of the key and this poses many security problems in what is called **key exchange**, that is, how to agree on a common key.

Also complicating the issue is the large number of people who might want to communicate securely and who would then have to share secret keys with one another. This is a key exchange nightmare.

Hence various key exchange protocols and schemes have been developed so that keys can be exchanged in **public**, such as over radio links, over the internet, in the presence of spying and listening devices and so on.

7.4.1 Diffie-Hellman key exchange (1976)

A and B wish to exchange a common key, with people listening in, in such a way that at the end, the listeners do **not** know the common key.

1. A chooses a *large* prime p and a primitive element g of $\mathbb{Z}_p$
   and sends p and g to B.
2. A chooses a secret random number a.
   B chooses a secret random number b.
3. A computes $x \equiv g^a \pmod p$ and sends it to B.
   B computes $y \equiv g^b \pmod p$ and sends it to A.
4. A calculates $y^a \equiv g^{ab} \equiv k \pmod p$
   B calculates $x^b \equiv g^{ab} \equiv k \pmod p$.

5. A and B then communicate in secret, say using IDEA, with key k.

For example (using only a small prime):

A chooses $p = 101, g = 12, a = 8$ and sends to B the information p and g and also the message $x \equiv g^a \equiv 12^8 \equiv 52 \pmod{101}$.

B receives p and g, chooses $b = 19$, and sends back to A the message $y \equiv g^b \equiv 12^{19} \equiv 50 \pmod{101}$.

They then use the common key $k \equiv x^b \equiv 50^8 \equiv y^a \equiv 52^{19} \equiv 58 \pmod{101}$.

To find k, the listener must know either x and b or y and a. To get a or b must solve a discrete logarithm problem, which is hard.

So this method used the one-way function $f(a) \equiv g^a \pmod{101}$.

Diffie-Hellman is "secure" against listeners, but not against active cryptanalysts — for example C can cut the link between A and B before they start, then pretend to A that they are B and vice-versa, in the style

$$A \rightleftarrows C \rightleftarrows B$$

This means not only that they listen in, but they can in fact take part in the communication between A and B without either of them knowing.

The only way to stop this sort of thing happening is if A insists on some sort of **authentication** that B really is who they claim to be and vice versa — see later.

7.4.2 Massey-Omura Protocol

This is another 2-party protocol for key exchange.

Everyone agrees on a fixed public finite field $GF(p^n)$ (or else A sends her own choice of p and n to B publicly).

Each user secretly selects an e with $0 < e < p^n - 1$ and $\gcd(e, p^n - 1) = 1$.

Each user computes $d \equiv e^{-1} \pmod{p^n - 1}$ and also keeps it secret.

A chooses a key k sends B the data k^{e_A} in $GF(p^n)$.

B sends back to A the data $(k^{e_A})^{e_B}$.

A calculates $((k^{e_A})^{e_B})^{d_A} = (k^{e_A d_A})^{e_B} = k^{e_B}$ and returns it to B.

B calculates $(k^{e_B})^{d_B} = k$.

The security here also relies on the discrete logarithm problem.

7.5 Public key cryptography

Diffie and Hellman in 1976 proposed this new form of cryptography in which the encrypting and decrypting keys are different — **asymmetric cryptography**. One of the keys is **public** and the other is **secret** (or **private**) and hence there is no need to exchange keys at all to communicate secretly.

Their method follows the following scheme:

There is a set of encryption functions E_k with encryption keys k, and there is a corresponding set of decryption functions $D_{\overline{k}}$ with decryption keys $\overline{k}$, such that

1. for all k, there exists a $\overline{k}$ such that for all m, $D_{\overline{k}}(E_k(m)) = m$,
2. it is easy to generate a random k and its paired $\overline{k}$,
3. $E_k(m) = c$ is easy to calculate, $D_{\overline{k}}(c) = m$ is easy to calculate,
4. without knowing $\overline{k}$, $E_k(m) = c$ is hard to invert.

Each user of the **public-key cryptosystem** generates their own $k, \overline{k}$ pair,
the functions E and D are public knowledge (for generic keys),
the keys k are openly published as in a phone book of public keys,
the keys $\overline{k}$ are kept secret.

If A wants to send a message m to B then she looks up k_B, B's public key, in the public-key list and sends

$$E_{k_B}(m) = c \quad \text{to} \quad B.$$

Then B reads the message by calculating

$$D_{\overline{k_B}}(c) = m$$

and since B is the only person who knows $\overline{k_B}$ he is the only person who can read it.

To reply with r, B looks up k_A and sends

$$E_{k_A}(r) = d$$

which can be read by A and A only using

$$D_{\overline{k_A}}(d) = r.$$

7.5.1 RSA Scheme (Rivest, Shamir, Adleman 1977)

This is one of the earliest public-key schemes proposed and, unlike other early schemes, has survived the test of time and is now widely used.

Each user does the following:

1. choose 2 random large primes p, q (100 or 200 digits), find $n = pq$ and calculate $\phi(n) = (p-1)(q-1)$.

2. choose a random e between 1 and $\phi(n)$ for which $\gcd(e, \phi(n)) = 1$ and find $d \equiv e^{-1} \pmod{\phi(n)}$, which exists since $\gcd(e, \phi(n)) = 1$.

   Now d has been found, you can forget p, q and $\phi(n)$.

3. the public key is the pair $(n, e) = k$, the secret key is the pair $(n, d) = \overline{k}$

The encryption algorithm is (for $0 \le m < n$)

$$E_k(m) \equiv m^e \pmod{n}$$

and decryption is achieved by

$$D_{\overline{k}}(c) \equiv c^d \pmod{n}.$$

This works because

$$D_{\overline{k}}(E_k(m)) \equiv (m^e)^d \equiv m^{ed} \equiv m \pmod{n}$$
$$\text{since } ed \equiv 1 \pmod{\phi(n)} \text{ (by definition), and}$$
$$m^{\phi(n)} \equiv 1 \pmod{n} \text{ by Euler's Theorem.}$$

Note that we have assumed above that $\gcd(m, n) = 1$.
It is left as an exercise to prove that:

if $ed \equiv 1 \pmod{\phi(n)}$, then $m^{ed} \equiv m \pmod{n}$ for **any** m.

We should also note that there are some technical restrictions, such as:

1. p, q should not be too close in size (otherwise Fermat factoring can be applied),
2. $\gcd(p-1, q-1)$ should be small
3. $p \pm 1, q \pm 1$ should have large prime factors or n can be factored by either **Pollard's** $p-1$ **algorithm** or **Williams'** $p+1$ **algorithm**.
4. If d or e is small, then try another e.
5. Neither of the primes p or q should be used repeatedly or they can be found by calculating the gcd of the public keys.

All the encryption and decryption is performed using fairly easy calculations, although with extremely large numbers involved they can become time-consuming.

A cryptanalyst sees $\{E\}$, $\{D\}$, n, e and c. So to invert $c = E_k(m)$ to get m from c means solving $c \equiv x^e \pmod{n}$ for x.

This too is a hard problem, in fact it can be shown to be more or less equivalent to the problem of factoring n.

Example:

Using the following **standard encoding** (chosen to avoid `A` being a small number)

0	1	_	A	B	C	D	E	F	G	H	I	J	K	L
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
15	16	17	18	19	20	21	22	23	24	25	26	27	28

we can turn a letter into a number.

Suppose now, $p = 31, q = 47$ so $n = 1457$ (public)
then $\phi(n) = 1380$.

Suppose $e = 7$ (public) then $d \equiv e^{-1} \equiv 1183 \pmod{1380}$.

We must keep p, q and $\phi(n)$ either secret, or once d is found, they can actually be forgotten.

If someone wanted to send this person a message '`Y`' they would encode, then encipher via

$$\texttt{Y} \to 27 \equiv m \to 27^7 \equiv 914 \pmod{1457}$$

and send them the code $c = 914$. The receiver deciphers via

$$914 \to 914^{1183} \equiv 27 \pmod{1457} \to \texttt{Y}$$

This example is quite unrealistic since

(a) 1457 is easily factored and hence you can easily break the cipher,

(b) the cipher is simply a 29 letter simple alphabetic substitution and so it can be easily broken by statistical methods.

We could improve on (b) as follows, by encoding and then enciphering pairs of letters.

For instance, we encode `UP` $\to 23, 18$ by taking 23 and 18 as digits in a base 29 number, namely

$$23 \times 29 + 18 = 685.$$

Then we encipher as

$$685 \to 685^7 \equiv 1102 \pmod{1457} = c.$$

However, even this is still only a $29^2 = 841$ letter simple alphabetic substitution.

Notes:

1. In real life, say $p, q \approx 100$ digits each, so $n \approx 200$ digits ≈ 600 bits $= 75$ bytes $= 75$ characters.

   Then we break the message up into 75 character blocks treated as 600 bit integers and apply RSA to each in turn. (So we use RSA as a *block cipher*.)

   This is then a $(2^8)^{75} = 2^{600} \approx 10^{200}$ letter substitution alphabet, so statistical methods are useless.

2. The length of RSA keys needs to increase to keep pace with constant improvements in factoring and computer speed. As it stands, 1024-bit key RSA is coming under threat (768-bit was broken in 2009) but 2048-bit key RSA is fine for quite a while.

3. RSA can be slow, since calculating powers mod n for large n needs special large integer packages and lots of arithmetic. However, the computation of the RSA decryption can be sped up using the Chinese Remainder Theorem, theorem 5.1: if we know $m_1 = c^d \pmod{p}$ and $m_2 = c^d \pmod{q}$ we can find $m = c^d \pmod{pq}$.

   The calculations will then take place in the two (smaller) *fields* $\mathbb{Z}_p$ and $\mathbb{Z}_q$, and although we have taken two integer powers this will be a considerable saving. Also, note that by Fermat's little theorem, theorem 5.7, when calulating $c^d \pmod{p}$ we only need to find c^δ, where $\delta \equiv d \pmod{p-1}$, and similarly for $c^d \pmod{q}$.

   For example, with $n = 31 \times 47 = 1457$ from above, using $e = 7$ and $d = 1183$, we encoded `UP` as 685 and encrypted that as 1102. To decode and find $1102^{1183} \pmod{1457}$ would take 16 multiplications in $\mathbb{Z}_{1457}$.

   Now $1183 \pmod{30} = 13$, $1183 \pmod{46} = 33$ and $31^{-1} \pmod{47} = -3$ by the Euclidean algorithm. We find that $m_1 = 1102^{1183} \equiv 17^{13} \equiv 3 \pmod{31}$ with 5 multiplications in $\mathbb{Z}_{31}$ and $m_2 = 1102^{1183} \equiv 21^{33} \equiv 27 \pmod{47}$ with 6 multiplications in $\mathbb{Z}_{47}$. So $m = 3 - 3 \times 31 \times (27 - 3) = -2229 \equiv 685 \pmod{1457}$ as expected.

   Systems that use this method for decoding RSA (such as OpenSSL) will therefore store the private key as p, q, $d \pmod{p-1}$, $d \pmod{q-1}$ and $q^{-1} \pmod{p}$. This is admittedly less secure than just storing n and d, but where speed of decoding is important this may be worth the risk.

   Even with this speed-up, RSA is rarely used in real life to send the *whole* message.

   For example the **PGP** (Pretty Good Privacy) package uses the following method for A sending a message to B:

   a. A randomly generates a 128-bit session key k,

   b. A encrypts the key k as $\mathrm{RSA}_{k_B}(k)$,

   c. A encrypts the message m as $\mathrm{IDEA}_k(m)$,

   d. A sends $(\mathrm{RSA}_{k_B}(k), \mathrm{IDEA}_k(m))$ to B,

   e. B finds $\mathrm{RSA}_{\overline{k_B}}(\mathrm{RSA}_{k_B}(k)) = k$,

   f. B then uses this k to find $\mathrm{IDEA}_k^{-1}(\mathrm{IDEA}_k(m)) = m$.

7.5.2 El Gamal public-key cryptosystem (1985)

Everyone agrees on a fixed public finite field $GF(p^n)$ and primitive element γ (or else A sends her choice of p, n and γ to B in public).

Each user chooses a random integer e such that $0 < e < p^n - 1$ and $\gcd(e, p^n - 1) = 1$. The integer e is the private key and γ^e is the public key.

Now A chooses a random r and calculates $(\gamma^{e_B})^r = \gamma^{e_B r}$, using B's public key γ^{e_B}.
Then A sends the pair $(\gamma^r, m\gamma^{e_B r})$ to B.
Next, B reads this by finding $(\gamma^r)^{e_B}$, inverting it and then finding

$$(\gamma^{e_B r})^{-1} m \gamma^{e_B r} = m.$$

The message m is masked by $\gamma^{e_B r}$ and only B can remove this mask to find m.
The security again relies on the discrete log problem, and is vulnerable to Shor's algorithm.

7.5.3 McEliece Encryption

McEliece Encryption (1978) is a public-key system that relies on the existence of efficiently decodable error-correcting linear codes. As mentioned in chapter 2, decoding general linear binary codes is not easy: in fact it is an NP-hard problem. But there are certain special examples that can be efficiently decoded. We will look at some in the next chapter: McEliece uses **Goppa codes** (which we will not consider).

In McEliece encryption Alice chooses a (large) (n, k)-Goppa code that can correct t errors, with $k \times n$ generator matrix $\mathbf{G}$. McEliece suggested $n = 1024$, $t = 50$, $k = 524$.

Alice then chooses a random $k \times k$ invertible binary matrix $\mathbf{S}$ and an $n \times n$ permutation matrix $\mathbf{P}$, and calculates $\widehat{\mathbf{G}} = \mathbf{SGP}$. Her public key is $(\widehat{\mathbf{G}}, t)$.

If Bob is sending a length k message $\mathbf{m}$ (a binary row vector), he first calculates $\widehat{\mathbf{m}} = \mathbf{m}\widehat{\mathbf{G}}$. He then picks a random set of t errors, a vector $\mathbf{e}_t$, say, and sends $\mathbf{c} = \widehat{\mathbf{m}} + \mathbf{e}_t$.

To decode this, Alice first calculates $\mathbf{cP}^{-1}$. As $\mathbf{P}^{-1}$ is a permutation, this still only has t errors. She decodes this using her efficient algorithm to get the corrected but masked message $\mathbf{mS}$.

Multiplying by $\mathbf{S}^{-1}$ recovers $\mathbf{m}$.

The major practical downside to McEliece is the large size of the public key: nk bits. The message is also expanded by a factor of n/k of course, which is a less serious problem.

To intercept the message by brute force, Eve has to either try all possible t-error correcting linear codes, none of which have efficient decoding techniques, or try all possible matrices $\mathbf{P}$ and $\mathbf{S}$ and hope to find the right Goppa code. Neither strategy in currently practicable.

In 2008 Tanja Lange (Eindhoven University of Technology) and her students announced a crack of the McEliece system with the original parameters, and recommend using, for example, (2960,2288) Goppa codes with $t = 56$ for 128-bit security.

7.6 Digital Signatures

Suppose we have a public key cryptosystem such that for all ciphertexts c,

$$E_k(D_{\overline{k}}(c)) = c.$$

(This is in addition to the usual property that for all messages m,

$$D_{\overline{k}}(E_k(m)) = m$$

as required for encryption.)

RSA is one such system.

Suppose A has keys: k_A public and $\overline{k_A}$ secret and
B has keys: k_B public and $\overline{k_B}$ secret.

(a) A can **encrypt** a message to B by using

$$E_{k_B}(m) = c$$

which B decrypts by

$$D_{\overline{k_B}}(c) = m$$

and *only* B can read anything encrypted by E_{k_B}.

(b) A can **sign** a message to B by using

$$D_{\overline{k_A}}(m) = s$$

which B can read by

$$E_{k_A}(s) = m.$$

Note that, by applying his secret **decrypting** key to the message, A produces a modified message s that *anyone*, including B, can read as they all have access to k_A.

So what is the point?

Well, *only* A could have sent s since only A knows $\overline{k_A}$. Hence this forms a **digital signature**.

So using this public key cryptosystem in reverse allows for digital signatures.

(c) A can **sign and encrypt** a message to B by first signing as only A can, then encrypting so only B can read it via

$$E_{k_B}\left(D_{\overline{k_A}}(m)\right) = t.$$

Then B finds

$$E_{k_A}\left(D_{\overline{k_B}}(t)\right) = m.$$

Next, B can then sign and encrypt a reply to A by

$$E_{k_A}\left(D_{\overline{k_B}}(r)\right) = d$$

which A reads by

$$E_{k_B}\left(D_{\overline{k_A}}(d)\right) = r$$

and *only* A can obtain $D_{\overline{k_A}}(d)$ correctly and *only* B could have sent it since $E_{k_B}(D_{\overline{k_A}}(d))$ is presumably meaningful.

Now suppose that A and B end up in court over a dispute about the messages m and r that they sent one another. Then the Judge J simply requests as follows:

$$B \text{ sends} \quad E_{k_J}\left(D_{\overline{k_A}}(m)\right)$$

that is, the message signed by A and encrypted for J, and

$$A \text{ sends} \quad E_{k_J}\left(D_{\overline{k_B}}(r)\right).$$

This works because J can read both these, since J knows $\overline{k_J}$. Also B cannot forge or interfere with $D_{\overline{k_A}}(m)$ and A cannot forge or interfere with $D_{\overline{k_B}}(r)$, so J can check the signatures of both and adjudicate.

Notes:

1. In real life, the sender does not normally sign the *whole* message but rather signs a separate signature or message authentication which is obtained using some one-way hash function.

   For example, PGP uses a 1-way hash function (e.g. MD5) for signatures which operates on the complete message m and returns a 128-bit hash value.

   This hash value was then signed using RSA or DSS (see below) and sent with the message.

   So PGP signing works as follows:

   A sends to B
   $$(m, \mathrm{RSA}_{\overline{k_A}}(\mathrm{MD5}(m)))$$

   B checks A's signature by
   $$\mathrm{RSA}_{k_A}(\mathrm{RSA}_{\overline{k_A}}(\mathrm{MD5}(m)) = \mathrm{MD5}(m)$$
   and compares it with his calculation of MD5(m).

   PGP signing with encryption takes the form:

   A sends to B
   $$\Big(\mathrm{RSA}_{k_B}(k),\ \mathrm{IDEA}_k\Big((m, \mathrm{RSA}_{\overline{k_A}}(\mathrm{MD5}(m))\Big)\Big)$$

2. Signing authenticates the person sending it and also that the message has not been altered.

7.6.1 Digital Signature Standard (USA 1991)

Unlike RSA, the DSS can only be used to sign messages, not encrypt them.

There is a universal prime q (~ 160 bits) and prime $p \equiv 1 \pmod q$ (~ 512 bits) and universal g such that $g^q \equiv 1 \pmod p$ but $g^i \not\equiv 1 \pmod p$ for any i with $0 < i < q$ (i.e. g has order q) (or else A chooses these and makes them public).

Each user finds a random e, $0 < e < q$ and keeps it secret. Then
$$k \equiv g^e \pmod p \quad \text{is the public key.}$$

For A (with e and k) to sign a message m

A chooses a random x, $0 < x < q$ and finds $y \equiv g^x \pmod p$.

Using a standard hash function called SHA-1 with a 160 bit output, A first calculates SHA-1$(m) = h$ and then calculates
$$\begin{aligned} r &\equiv y \pmod q \\ \text{and} \qquad s &\equiv x^{-1}(h + er) \pmod q \end{aligned}$$

[If r or $s = 0$ then choose another x].

The pair (r, s) of 160 bit integers is the signature.

B verifies this is A's signature by finding $h = $ SHA-1(m) and then calculating
$$u_1 \equiv s^{-1}h \pmod q \quad \text{and} \quad u_2 \equiv s^{-1}r \pmod q$$
then
$$z \equiv g^{u_1}k^{u_2} \pmod p.$$

If $z \pmod q$ and r agree, then the signature is valid.

This works because

$$\begin{aligned} z &\equiv g^{u_1}k^{u_2} \equiv g^{s^{-1}h}(g^e)^{s^{-1}r} \equiv g^{s^{-1}(h+er)} \pmod p \\ &\equiv g^x \pmod p \qquad\qquad (\text{since } g^q \equiv 1 \pmod p \text{ and } xs \equiv h + er \pmod q) \\ &\equiv y \equiv r \pmod q. \end{aligned}$$

The security of this also depends on the discrete log problem.

7.6.2 Key certification

One last problem remains:

How can you be sure that the public key directory has the keys correct? — perhaps k_B is really k_C and C is listening in to all messages intended for B, or when A asks B for his key, C is in between A and B and sends k_C not k_B.

Answer:

You can never be totally sure — you have to trust someone sometime.

One way of trying to solve this is:

Assume the public key directory D is secure, and that the directory publishes their public key k_D widely.

When A wants B's public key she asks D for it. Then D sends a message saying
"k_B is the public key for the person who has been verified to be B"
and signs it using $\overline{k_D}$.

Anyone can read this message and check it is authenticated by D.
In fact B can store $D_{\overline{k}_D}$ (the message about k_B) and send it to anyone who asks for k_B.
That is, he can send a **certified copy** of his public key to A.

This is like a passport, and is as secure as the issuing authority.

Some US states have set up certification schemes.

PGP on the other hand uses a completely different approach called a **key ring**:
people certify other people's keys if they know they are correct. They keep a copy of lots of certificates for their own key and lists of people whose key they accept. These are circulated around and so there is much cross-checking but NO central authority.

B sends A all of B's certificates and A searches them until A finds one she trusts and hence gets k_B.

7.6.3 Zero-knowledge identification protocols

We can use a similar idea to RSA to give a probabilistic method of proving identity without revealing the secret key: a **zero-knowledge identification protocol**.

The simplest scheme is known as the **Fiat-Shamir Protocol** (2001):

A trusted authority, T, publishes the product $n = pq$ of two primes as the basis for verification. We work henceforth in $\mathbb{Z}_n$.

Peggy (the prover) selects a random secret s, $1 \le s \le n-1$ coprime to n and registers $v = s^2$ as public key. Since only T knows p and q, only T could (easily) find s from v.

Suppose Victor (the verifier) wants to check that Peggy is who she says she is, that is, she knows s. Then they run through the following 4 steps until Victor is convinced,

P1 Peggy picks a random r, $1 \le r \le n-1$, and sends $x = r^2$ to Victor.

V1 Victor randomly sends $e \in \{0, 1\}$ to Peggy.

P2 Peggy replies with $y = rs^e$.

V2 Victor compares y^2 to xv^e. If $y = 0$ or the numbers were different Victor would know Peggy is not who she claims to be.

Note that $y^2 = (rs^e)^2 = r^2v^e = xv^e$, so if Peggy knows s she can always provide the correct y: if not then she can only get it right when Victor sends $e = 0$.

If Victor cheats and always sends $e = 1$ (trying to force s to be used), Peggy can counter-cheat and send $x = r^2v^{-1}$, which does not need s. Then when Victor sends $e = 1$, she replies with $y = r$ not rs. When Victor tries to verify, he finds $y^2 = r^2$ and $xv = r^2$, so will believe Peggy.

Peggy cannot use this stategy to deceive Victor if he sends e randomly. If she has sent $x = r^2v^{-1}$ then in order to pass Victor's test when he sends $e = 0$ she would need to send the integer square root of x in $\mathbb{Z}_n$. But she does not know p and q, so she cannot do this.

The point is that after t rounds, Victor knows with probability $1 - 2^{-t}$ that Peggy is who she says she is.

Why is this a zero-knowledge protocol?

Well, in each round two numbers are revealed: $x = r^2$ in (P1) and $y = rs^e$ in step (P2). But we could simulate pairs (x, y) by picking y randomly and defining $x = y^2v^{-e}$ using no new knowledge. Such random pairs are indistiguishable from those generated by the protocol, so knowing x and y gives us zero new knowledge.

7.7 Probabalistic Encryption

7.7.1 Goldwasser-Micali method (public-key)

Some more Number Theory:

If p is an odd prime then unit $x \in \mathbb{U}_p$ is a **square** if and only if there exists $t \in \mathbb{Z}_p$ with $x \equiv t^2 \pmod{p}$.

Then x is square if and only if $x^{\frac{1}{2}(p-1)} \equiv 1 \pmod{p}$ and x is non-square if and only if $x^{\frac{1}{2}(p-1)} \equiv -1 \pmod{p}$. Half the $x \in \mathbb{U}_p$ are squares, half are non-squares.

If $n = pq$ (p, q primes) then x is a square mod n if and only if x is a square mod both p and q. So with $n = pq$, one quarter of $\mathbb{Z}_n$ are squares.

Determining if x is a square mod n is easy if p and q are known but hard otherwise.

For the Goldwasser-Micali cryptosystem, users have a public key (n, y) where $n = pq$ is the product of two primes, y is a non-square mod n and the secret key is (p, q).

For A to send a message m to B, A writes m in binary as $m = m_1m_2m_3\ldots$.
For each i, A chooses a random $r_i \in \mathbb{Z}_n$ and sends to B.

$$c_i \equiv \begin{cases} r_i^2 \pmod{n_B} & \text{if } m_i = 0 \\ y_Br_i^2 \pmod{n_B} & \text{if } m_i = 1 \end{cases}$$

B easily reads this since

$$\begin{aligned} & m_i = 0 \\ & \text{if and only if both } c_i^{\frac{1}{2}(p_B-1)} \equiv 1 \pmod{p_B} \quad \text{and} \quad c_i^{\frac{1}{2}(q_B-1)} \equiv 1 \pmod{q_B} \\ & \text{if and only if } c_i \text{ is a square mod } n_B. \end{aligned}$$

Only B can read this since only B knows p_B, q_B.

If s is the number of zeros in m and t the number of ones in m, then the message m can be enciphered in roughly $(\frac{1}{4}n)^s(\frac{3}{4}n)^t \approx \left(\frac{\sqrt{3}}{4}n\right)^{\text{length}(m)}$ ways, each of which B can uniquely and correctly decipher.

This makes the cryptanalyst's task nearly impossible, unless n can be factored.

This is "probabalistic" due to the randomness in the enciphering. It comes close to being a one-time-pad. Unfortunately the cipher text is much longer than the plain text so this method is highly inefficient.

7.7.2 Blum-Goldwasser method (public-key)

All users have a public key $n = pq$ where p, q are primes $\equiv 3 \pmod 4$ and are secret. Here $k = n$ and $\overline{k} = (p, q)$.

For A to send a t-bit message $m_1m_2\dots m_t$ to B, whose key is $n = n_B$ (with secret $p = p_B, q = q_B$), A chooses a random x_0 and uses the Blum Blum Shub square random bit generator, covered in Section 5.8.6, to produce a sequence $b_0, b_1, \dots b_{t-1}$ of random bits and then forms the t-bit integer $b = b_0b_1\dots b_{t-1}$.

A sends $(x_t, m \oplus b)$ to B where $\oplus$ =bitwise XOR and x_t is the last of the x_i used in the Blum, Blum and Shub algorithm.

Now B can reconstruct b and hence find m by solving backwards from the end:

$$\begin{aligned} x_{t-1}^2 &\equiv x_t \pmod n \quad \text{to get } x_{k-1} \\ x_{t-2}^2 &\equiv x_{t-1} \pmod n \quad \text{to get } x_{k-2} \end{aligned}$$

giving b_1, b_2, etc (b_0 is found from x_t) where $b_j = 1$ if and only if x_{t-j} is odd.

This is provably as secure as factoring and is very close to a one-time-pad. However, it is very slow and is vulnerable to a chosen plaintext attack.

7.8 Recent Developments

7.8.1 Quantum Cryptography

There have been some recent developments in the idea of using quantum-mechanical effects, typically polarisation, spin or entanglement, to set up cryptosystems. The basic idea depends on the fact that if a object that obeys the rules of quantum mechanics is observed, then its properties are altered.

Some experimenters have designed channels capable of transmitting quantum data, for example transmitting individual polarized photons. Such a quantum channel can be set up in such a way as to allow for the perfectly secure transmission of keys.

There are protocols for exchanging individual polarized photons and other data between A and B in such a way that they are able to detect if the exchanged photons have been observed by a cryptanalyst or not. If they have not been observed then they can be used to construct a perfectly secure key. If they have been observed then they are discarded.

The cryptanalyst is still able to completely jam the channel by observing everything, but if they allow any photons through unobserved, then using these protocols A and B are able to agree on a secret key: Singh's book discusses one method.

Such channels have actually been constructed, initially using evacuated tubes but more recently using standard optical cable and some through open air. They still only work over limited distances and at the moment cannot be switched. That is, you need a direct channel or line of sight between A and B.

A related idea is that of quantum computing. A quantum computer manipulates objects called **qubits** to store information, rather than ordinary bits. A qubit can store the superposition of *both* the states 0 and 1 simultaneously. A computer constructed of qubits can then store all the possible states of every bit at the same time. Subjecting the qubit to various transformations allows it to effectively calculate all possible answers to a problem at once, rather than one at a time as on a conventional computer.

A quantum computer is similar to, but far exceeds the capabilities of, a massively parallel conventional computer.

We looked at Shor's algorithm for factoring integers in Section 5.7.5, and commented there on its properties, in particular that it requires quantum computers with massive numbers of qubits (in the billions) so it is not likely to be practical for a long time: no-one has yet built a quantum computer of more than a few qubits, but many are trying, including world-leading group in the UNSW School of Physics.

If a quantum computer of any size can be built, or quantum objects transmitted over any reasonable distance, then cryptography will be revolutionized in a more far-reaching way than it was by the invention of public-key cryptosystems.

There is much interest on cryptosystems that do not rely on quantum computers but are resistant to attacks by them. McEleice encryption, see section 7.5.3, is the current favourite for such a post-quantum cryptosystem. In 2010 three American researchers, Hang Dinh, Christopher Moore and Alexander Russell showed that McEliece is proof against attacks based on Shor's algorithm for factoring integers, so does not have the same vulnerability as RSA or El Gamal. This does not mean other quantum algorithms could not be found that could crack McEliece of course.

If you are interested in this area, there is an enormous amount of information about it on the web. For a start, look at `http://www.qcaustralia.org`

7.8.2 DNA Cryptography

DNA (deoxyribonucleic acid) is the molecule that contains and transmits genetic information in living organisms. It is a double helix of two long strands (polymers) each consisting of a chain of small units (nucleotides). There are four different nucleotides, and so DNA can be considered as a quaternary code. The two strands form complementary pairs, and so there is only really information in one of them, not both.

In 1994 L Adelman (of RSA fame) initiated the field of DNA computing by showing how to use the properties of DNA to find a Hamiltonian path in a directed graph, an important NP-complete problem. DNA cryptography is a recent development along the same lines, using DNA, and techniques designed for creating and manipulating it, for cryptographic purposes.

DNA is very information dense: it is estimated that 1 gramme of DNA could store 10^8 terabytes, compared with 16MB for a silicon chip. This density makes DNA very suitable for the creation of one-time pads: a scheme to do so was devised in 2005, but is currently impractical. DNA computing is also massively parallel and energy efficient: you essentially set up the calculation so that all solutions are attempted by the biological computer, and extract the DNA that corresponds to the solution.

The purpose of DNA cryptography is to use a difficult *biological* problem to hide the message. The difficult problem involved is usually the extraction of the relevent DNA sequence from a large number of masking sequences. The PCR (polymerase chain reaction) process is used to amplify the relevent DNA structure so that it can be analysed. But PCR can only proceed if Bob has the correct secret keys to start and end the amplification process. These secret keys need to be exchanged of course, which leads to the usual key exchange problem.

DNA cryptography is a very new field and as yet has little in the way of theoretical background. However, schemes have been devised to mask messages with both a difficult mathematical problem (e.g. using RSA) and the difficult biological problem of DNA analysis.

7.9 Entropy and Cryptography

Claude Shannon was the first to apply information theory to cryptography.

Suppose that M is the set of (potential) messages/plaintext, K is the set of keys and C is the set of ciphertext.

Then given the ciphertext, if you know the key then you know the plaintext, so the uncertainty in the key must be at least as great as the uncertainty in the plaintext. (The uncertainty in the plaintext may be *strictly less* than the uncertainty in the key, as perhaps the plaintext can be found without finding the key.)

Hence $H(K|C) \geq H(M|C)$ and, in the same way as in section 4.8,

$$H(K) \geq H(K|C).$$

Now the mutual information between the message and cipher is

$$\begin{aligned} I(M,C) &= H(M) - H(M|C) \\ &\geq H(M) - H(K) \quad \text{by the above.} \end{aligned}$$

7.9.1 Perfect Security

A cryptosystem is called **unconditionally secure** or has **perfect security** if there is no information about the message in the ciphertext, that is

$$\text{if and only if} \qquad I(M,C) = 0$$

$$\text{if and only if} \qquad H(M) \leq H(K).$$

This means that there must be more uncertainty about the key than about the message.

Perfect security means that the cryptosystem cannot be broken, no matter how much computing resources or time are used, in contrast to **computational security**, which is the property RSA and McElice have.

It can be shown that perfect security requires:

(a) $|M| = |C| = |K|$,

(b) equally likely keys K,

(c) for all $m \in M$ and $c \in C$, there exists a unique $k \in K$ with $E_k(m) = c$.

The one-time-pad is the *only* cryptosystem that has been proved to be perfectly secure, as we discussed in section 7.1.7.

7.9.2 Random ciphers, unicity distance

Suppose that we have a source alphabet S with q letters and $M = S^n$. That is, all messages have length n. So

$$|M| = |S^n| = q^n.$$

Messages in M are of 2 types:

(a) meaningful ones, which we assume are all equally likely, and

(b) non-meaningful ones, which we assume have zero probability.

Let F_n be the set of all meaningful messages in M. We can think of F_n as a source with $|F_n|$ symbols, all with probability $1/|F_n|$ as they are equally likely. This allows us to define the radix q entropy of the set F_n, which equals

$$H_q(F_n) = -\log_q(1/|F_n|) = \log_q(|F_n|).$$

(Note that we *choose* to work in radix q units, where $|S| = q$.)

Next, let $r_n = \frac{H_q(F_n)}{n}$ be the average entropy per symbol of length n (meaningful) messages in radix q units. Then there are

$$q^{H_q(F_n)} = q^{nr_n}$$

meaningful messages, where r_n is the **information rate** for length n messages. (Note: this is not the same as the concept in chapter 2, where we were looking at the information rate of a *code*, not a message.)

We also let $d_n = 1 - r_n$ and this is often called the **redundancy** of length n messages. (Note that this is also different to our definition in Chapter 2, which was $1/r_n$.)

Now there are q^n messages, with q^{nr_n} being meaningful, so the probability of a message chosen at random being meaningful is $\frac{q^{nr_n}}{q^n} = q^{-n(1-r_n)} = q^{-nd_n}$.

Let K be the set of keys. If all the keys are equally likely then we can think of K as a source with $|K|$ equally-likely symbols. Hence the radix q entropy of K is

$$H_q(K) = \log_q(|K|)$$

and there are $|K| = q^{H_q(K)}$ keys.

If the message m is enciphered as c and then c is deciphered using a **random key** then (assuming a one-one enciphering) a random message results and so the probability of a meaningful decipherment when using random keys is q^{-nd_n}. The total number of keys is $q^{H_q(K)}$, as above.

So given c, if we try *all keys* then the **expected number** of meaningful deciphers is

$$q^{H_q(K)} \cdot q^{-nd_n} = q^{H_q(K)-nd_n}.$$

How is this useful?

Firstly, suppose that the expected number of meaningful deciphers is ≤ 1. Then

$$H_q(K) - nd_n \leq 0, \qquad \text{so} \qquad nd_n \geq H_q(K).$$

Since there *must* be one meaningful decipher (assuming the original m was meaningful), the meaningful decipher must be the **correct** decipherment.

On the other hand, if $nd_n \ll H_q(K)$, then the expected number of meaningful deciphers is large. So, when we obtain any meaningful decipher, the probability that it is the correct decipher is small.

In real life, not all messages are the same length n. Assume that as $n \to \infty$ we have $r_n \to r$, where r is the **true rate** of the language in radix q units. Then $d_n \to d = 1 - r$ and so:

if $n > \dfrac{H_q(K)}{d}$, then we expect a unique meaningful decipher, and

if $n << \dfrac{H_q(K)}{d}$, then the probability a meaningful decipher is correct is small.

We call $n_0 = \left\lceil \dfrac{H_q(K)}{d} \right\rceil$ the **unicity distance** of the cipher.

The unicity distance is the least positive integer n_0 for which, given n_0 letters of ciphertext, we expect to get a *unique* meaningful decipherment when we decipher using all possible keys.

Our methods for breaking classical ciphers such as Vigenère ciphers used this principle: if you find a key that gives a meaningful decipher, then it must have been the correct one.

Normally unicity distance, like entropy, is calculated in binary units. So swapping to bits gives

$$n_0 = \left\lceil \frac{H_q(K)}{d} \right\rceil = \left\lceil \frac{H_q(K)}{1-r} \right\rceil = \left\lceil \frac{H_2(K)}{\log_2 q - R} \right\rceil,$$

where $R = r\log_2 q$ is the rate of the language in bits/character.

For English text, $q = 26$, $R = 1.5$ (as we saw in Section 4.11) so that

$$n_0 = \left\lceil \frac{H_2(K)}{4.7 - 1.5} \right\rceil = \left\lceil \frac{H_2(K)}{3.2} \right\rceil,$$

and if the keys are equally likely (the usual case) then $n_0 = \left\lceil \dfrac{\log_2 |K|}{3.2} \right\rceil$.

Example A simple (monoalphabetic) substitution cipher has 26! keys, so has unicity distance

$$n_0 = \left\lceil \frac{\log_2(26!)}{3.2} \right\rceil = \left\lceil \frac{88.4}{3.2} \right\rceil = 28.$$

So a 28 letter message is the smallest that can be expected to have a unique meaningful decipherment if all possible keys are tried.

Now a message with 28 letters is far too short for statistical methods to be of any use. That is, n_0 is not a practical length, but a theoretical length: remember "expected" means when trying all possible 26! keys.

With $R = 1.5$ and $\log_2 26 = 4.7$ we are saying that in English, out of every 4.7 characters, 3.2 are redundant, i.e. 68% are redundant.

[Some books claim that top cryptanalysts can actually uniquely decipher messages of around 25 letters.]

Consider now what happens with other length messages for this situation with $n_0 = 28$.

With $n = 40$ letters, the expected number of meaningful deciphers is

$$\begin{aligned} q^{H_q(K)-nd} &= 2^{H_2(K)-nD} \qquad \text{(where } D = \log_2 q - R \text{ is the redundancy in bits)} \\ &= 2^{88.4-40\times 3.2} \\ &= 2^{-39.6} \approx 1 \times 10^{-12} \end{aligned}$$

so if you find a meaningful decipher, you can be highly confident it is correct.

With $n = 20$ letters, the expected number of meaningful deciphers is

$$2^{88.4-20\times 3.2} = 2^{24.4} \approx 2 \times 10^{7}$$

so if you find a meaningful decipherment then the chance that it is correct is 1 in 20 million.

MATH3411 Problems

Chapter 1: Introduction

1 **a.** Explain why, if n is not a prime, then n has a prime factor less than or equal to $\sqrt{n}$.

b. Explain why, for $18 \le n \le 400$, $n \ne 361$, that n is prime if and only if n does not have a proper factor 2 or 5 and $\gcd(n, 51051) = 1$.

c. Hence test $n = 323$ and $n = 373$ for primeness.

2 Prove that if $2^n - 1$ is a prime then n is prime and that if $2^n + 1$ is prime then n is a power of 2.

3 **a.** What is the probability of picking a Queen from a standard pack of cards?

b. Suppose you are told that I have picked a face card. Now what is the probability it is a Queen?

c. Suppose instead you are told the card I have is black. Now what is the probability it is a Queen?

d. What do the above tell you about the random events "pick a Queen", "pick a face card" and "pick a black card"?

4 A certain binary information channel is more likely to transmit a 0 as an error for 1 than a 1 as an error for 0. The probability that a 1 is received in error is 0.1 (that is, $P(1 \text{ received} \mid 0 \text{ sent}) = 0.1$) and the probability that a 0 is received in error is 0.2. Write down the conditional probabilities for all four possible situations. If 1s and 0s are sent with equal probability, find the probability of receiving a zero. What is the probability that a zero was sent, given that a zero was received?

5 (For discussion, if you've not heard of it.) The famous and much debated Monty Hall problem is as follows: On a game show, a contestant is presented with three identical doors, behind one of which is a major prize, the others hiding a minor prize. The contestant gets to choose one door. If the contestant picks the door hiding a minor prize, then the host, Monty Hall, opens the door showing the other prize; if the contestant picks the door with the major prize, Monty randomly picks one of the doors hiding a minor prize and opens that. The contestant then has the choice of changing to the other door or not, and wins whatever is behind the door he or she has finally settled on.

The question is: should the contestant change to the other door? Can you prove your answer?

6 Suppose we send a message of n bits in such a way that the probability of an error in any single bit is p and where the errors are assumed to be independent. Use the binomial probability distribution to write down the probability that:

a. k errors occur,

b. an even number of errors occur (including zero errors),

c. show that the answer in (b) can be expressed as

$$\frac{1+(1-2p)^n}{2} \quad .$$

(*Hint: let* $q = 1-p$ *and expand the expression* $\frac{(q+p)^n+(q-p)^n}{2}$.)

7 Taking $p = .001$ and $n = 100$ in Question 6, find the probability that

a. there is no error,

b. there is an undetected error when a simple parity check is used.

8 Check whether the following can be ISBNs, and if not, then assuming it is the check digit that is wrong, alter the check digit so that they are valid ISBNs:

$$\begin{array}{ccccccc} 0 & - & 552 & - & 08638 & - & X \\ 0 & - & 576 & - & 08314 & - & 6 \end{array}$$

9 (A possible telephone number code)

Let $\mathcal{C}$ be the code of all 10 digit (decimal) numbers $\mathbf{x} = x_1x_2\cdots x_{10}$ that satisfy the two check equations

$$\sum_{i=1}^{10} x_i \equiv 0 \pmod{11}, \qquad \sum_{i=1}^{10} ix_i \equiv 0 \pmod{11}.$$

a. Estimate roughly how many such numbers $\mathbf{x}$ there are. (That is, estimate $|\mathcal{C}|$.)

b. Show that this code is single-error correcting and that it can also detect double errors caused by the transposition of two digits.

c. Correct the number $\mathbf{y} = 0680271385$.

10 (Introducing Chapter 4): A bridge deck is a set of 52 distinguishable cards. A bridge hand is any subset containing 13 cards and a bridge deal is any ordered partition of the bridge deck into 4 bridge hands.

a. Describe a simple but inefficient representation of an arbitrary bridge hand by assigning a 6-bit binary number to represent each card. How many bits are needed to describe a deal this way?

b. Show that no binary representation of an arbitrary bridge hand can use fewer than 40 bits. Give a representation using 52 bits.

c. Show that no representation of an arbitrary bridge deal can use fewer than 96 bits. Give a representation using 104 bits. This can be reduced to 101 bits with a little thought, can you see how?

Chapter 2: Error Correcting Codes

11 Using 9-character 8-bit even parity ASCII burst code, encode **HDforall** (don't forget the space).

12 The integers from 0 to 15 inclusive are encoded as four bit binary numbers with one parity check bit at the front, so that, for example, 4 is 10100 and 15 is 01111. A stream of such integers is then encoded into blocks of 4 with one check integer to give overall even parity, similar to the ASCII burst code.

a. Show that the string of integers $1, 13, 2, 6$ has check number 8.

b. The block 10010 11011 01110 00011 00110 is received. Find and correct the error (assuming at most one) and then decode.

13 A code $\mathcal{C}$ consists of all solutions $\mathbf{x} \in \mathbb{Z}_2^5$ (i.e., binary vectors of length 5) of the equation $H\mathbf{x} = 0$ where

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

a. Find all the codewords.

b. Find three linearly independent columns of H.

c. Find three linearly dependent columns of H.

d. Show that no two columns of H are linearly dependent.

e. Show that no four columns of H are linearly independent.

14 For the Hamming (7,4)-code described in lectures:
(a) encode 1010, (b) correct and decode 1001111.

15 Write down the parity check matrix of the Hamming (15,11)-code described in lectures, and hence correct and decode 100101010110110.

16 Using the Hamming (15,11)-code described in lectures:

a. encode 10101010101.

b. correct and decode 011100010111110.

17 What is the probability that a random sequence of $n = 2^m - 1$ '0's and '1's is a code word in a Hamming code?

18 You receive a message which has been encoded using the Hamming (7,4)-code of lectures. The probability of error in any single bit of the message is $p = .001$, and errors in different bits are independent. Find the probability that:

a. there is no error,

b. there are errors and they are correctly corrected.

c. there are errors and they are not correctly corrected,

d. when the code is used to detect but not correct errors (i.e. using a pure error detection strategy), there are undetected errors.

19 Suppose a binary linear code C has parity check matrix H.

a. Prove that $d(C) = w(C)$.

b. Prove that $d = d(C)$ is the smallest integer r for which there are r linearly dependent columns in H modulo 2.

20 Suppose a binary linear code C has parity check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

a. Find its minimum distance d.

b. What are its error correction and detection capabilities?

c. Correct and decode the following received words (if possible), assuming that the first 3 bits are the information bits and we are using a single error correcting strategy.

(i) 010110

(ii) 010001

(iii) 100110

d. Find a standard form parity check matrix H' for the code C, and the corresponding generator matrix G'. Find a generator matrix G in similar fashion from the original parity check matrix H.

e. Explain how to extend the code C to give a new code $\widehat{C}$ with minimum distance $d+1$. (Prove that your new code has minimum distance $d+1$.) Write down a parity check matrix for $\widehat{C}$.

21 We wish to code the four directions N, S, E, W using a binary code.

a. Show that, if we require single error correction, then we need to use messages of at least 5 bits length. Find such a code of length 5.

b. Show that, if we require double error correction, then we need messages of at least 7 bits length. Construct such a code from messages of 8 bits.

***22** Let $r \geq 3$ be an integer and suppose that $\mathbf{x} = x_1 x_2 \cdots x_n$ is a vector in $\mathbb{Z}_r^n$ (so $0 \leq x_i \leq r-1$ for all i).

a. For each nonnegative integer ρ, calculate the number of vectors in $\mathbb{Z}_r^n$ at distance ρ from $\mathbf{x}$.

b. Work out what the sphere-packing bound for t-error correcting radix r codes of length n should be, following the argument from the binary case.

c. Prove the radix r Hamming codes are perfect.

23 **a.** Construct the check matrix of a radix 5 Hamming code with parameters $m = 2$, $n = 6$, using the method given in lectures.

b. Using your check matrix, correct and decode the received word $\mathbf{y} = 410013$.

24 Let C be the code consisting of all vectors $\mathbf{x} = x_1 x_2 x_3 x_4 \in {\mathbb{Z}_5}^4$ satisfying the check equations

$$\begin{aligned} x_1 + x_2 + 3x_3 + 2x_4 &\equiv 0 \pmod{5}, \\ x_1 + 2x_2 + 4x_3 + 3x_4 &\equiv 0 \pmod{5} \end{aligned}$$

a. Assuming that x_3 and x_4 are the information bits, find the codeword which encodes the message 21.

b. Which of the following are valid code word in C?

(1) 1122 (2) 1212 (3) 2323 (4) 4343

Chapter 3: Compression Codes

25 Decide whether the following codes are uniquely decodable, instantaneous or neither.

a. 0, 01, 11, 00 ;

b. 0, 01, 011, 111 ;

c. 0, 01, 001, 0010, 0011 ;

d. 00, 01, 10, 110, 111 .

26 Either prove the following code is uniquely decodable or find an ambiguous concatenated sequence of codewords:

$$\begin{array}{llll} \mathbf{c}_1 = 101, & \mathbf{c}_2 = 0011, & \mathbf{c}_3 = 1001, & \mathbf{c}_4 = 1110 \\ \mathbf{c}_5 = 00001, & \mathbf{c}_6 = 11001, & \mathbf{c}_7 = 11100, & \mathbf{c}_8 = 010100\,. \end{array}$$

(This is more difficult than Q25.)

27 Construct instantaneous codes, or show they cannot exist for the following:

a. radix 2, codeword lengths 1, 2, 3, 3, 3 ;

b. radix 2, codeword lengths 2, 2, 3, 3, 4, 4, 4 ;

c. radix 3, codeword lengths 1, 2, 3, 3, 3, 3, 3, 3, 3 ;

d. radix 3, codeword lengths 1, 1, 2, 2, 3, 3, 3, 3 .

Can any of these be shortened and if so how?

28 What is the minimum radix that would be needed to create a UD-code for the source $S = \{s_1, s_2, s_3, s_4, \ldots, s_8\}$ with respective codeword lengths

a. $1, 1, 2, 2, 2, 3, 3, 4$

b. $2, 2, 2, 4, 4, 4, 4, 5$

29 Find binary Huffman codes and their expected codeword lengths for:

a. $p_1 = 1/2,\ p_2 = 1/3,\ p_3 = 1/6$;

b. $p_1 = 1/3,\ p_2 = 1/4,\ p_3 = 1/5,\ p_4 = 1/6,\ p_5 = 1/20$;

c. $p_1 = 1/2,\ p_2 = 1/4,\ p_3 = 1/8,\ p_4 = 1/16,\ p_5 = 1/16$;

d. $p_1 = 27/40,\ p_2 = 9/40,\ p_3 = 3/40,\ p_4 = 1/40$.

30 A random experiment has seven outcomes with corresponding probabilities

1/3, 1/3, 1/9, 1/9, 1/27, 1/27, 1/27.

The experiment is to be performed once and the outcome transmitted across the country. The telegraph company provides two services. Service 1 transmits binary digits at \$2.00 per digit and service 2 transmits ternary digits $\in \{0, 1, 2\}$ at \$3.25 per digit. You are to select a service and design a code to minimize expected cost.

a. Which service should be selected? What code should be used? What is the expected cost?

b. If the ternary cost is changed, at what new value of the cost would you change your mind?

***31** Prove that the (binary) Huffman code for a 2^n symbol source where each symbol has equal probability is a block code of length n. (Hint: induction.)

***32** Suppose that we have an n symbol source where the ith symbol occurs with frequency f_i, where f_i is the ith Fibonacci number and $f_1 = f_2 = 1$. Describe the standard binary Huffman code for this source. (NOTE: $f_1 + f_2 + \cdots + f_n = f_{n+2} - 1$.)

33 Consider the alphabet $s_1, s_2, \cdots, s_8$ where the symbols occur with probabilities

0.22, 0.20, 0.18, 0.15, 0.10, 0.08, 0.05 and 0.02 respectively.

Code this source with a Huffman code of radix 4 using dummy symbols if necessary. What is the expected codeword length for this coding? Contrast it with the expected codeword length if another Huffman code is constructed by not introducing dummy symbols, but instead combining four symbols at a time as long as possible.

34 Consider the source $S = \{a, b\}$ with probabilities $p_1 = 3/4$ and $p_2 = 1/4$.

a. Find a binary Huffman code for the third extension S^3 of the source S. What is the average code word length per (original) symbol.

b. Encode the message *aababaaaabaa* using this code.

35 Suppose we have two symbols which occur with probabilities $p_1 = 2/3$ and $p_2 = 1/3$. Consider the first, second and third extensions. Find Huffman codes for each extension and calculate the corresponding expected codeword lengths and expected codeword lengths per original symbol.

36 Consider the Markov matrix

$$M = \begin{pmatrix} 1/3 & 1/4 & 1/4 \\ 1/3 & 1/2 & 1/4 \\ 1/3 & 1/4 & 1/2 \end{pmatrix}.$$

a. Show that M has eigenvalues 1, 1/4, 1/12 and find the equilibrium probabilities.

b. Explain why $\lim_{n\to\infty} M^n$ exists and find the limit. What do you notice about the answer?

37 A Markov source on symbols s_1, s_2, s_3 has transition matrix

$$\begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.2 & 0.6 & 0.4 \\ 0.1 & 0.2 & 0.5 \end{pmatrix}.$$

a. Find the equilibrium probability distribution and a Huffman encoding for it. Also find a Huffman code for the Markov source. Compare the expected codeword lengths in the two cases.

b. Encode the string of symbols $s_2s_2s_1s_1s_2s_3s_3$ using the Markov Huffman code.

c. Decode the code string 010001010 which was encoded using the Markov Huffman code.

38 A source has symbols $\{a, b, c, \bullet\}$ where $\bullet$ is the stop symbol. The probabilities of these symbols are $\frac{2}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}$ respectively. Use arithmetic coding to encode the message $bac\bullet$ into a code number.

39 Three symbols s_1, s_2, s_3 and the stop symbol $s_4 = \bullet$ have probabilities $p_1 = 0.4$, $p_2 = 0.3$, $p_3 = 0.2$ and $p_4 = 0.1$.

a. Use arithmetic coding to encode the message $s_2s_1s_3s_1\bullet$.

b. Decode the codeword 0.12345 which was encoded using this arithmetic code.

40 Use the LZ78 algorithm to

a. encode "banana and bee" (including spaces).

b. decode

$$(0,t)(0,o)(0,␣)(0,b)(0,e)(3,o)(0,r)(3,n)(2,t)(3,t)(2,␣)(4,e)$$

Here "␣" denotes a space.

Chapter 4: Information Theory

41 A source S produces the symbols a, b, c, d, e with probabilities 1/3, 1/3, 1/9, 1/9, 1/9. Calculate the entropy of this source in bits.

42 Find the entropies (in appropriate units) for the sources in Questions 29, 33 and 35. Compare your answer with the expected codeword lengths of the Huffman codes you obtained previously.

Calculate the decimal entropy of the source in question 39, and compare to the length of the arithmetically coded message.

43 For the situation in Question 30, suppose the experiment was repeated many times and suitably coded (for long symbol words) before the outcomes were sent. What is the answer in part (a) and (b) now?

44 Find Shannon-Fano codes for the sources in Questions 29, 33 and 35.

45 A source S has 5 symbols $s_1, s_2, \ldots, s_5$ with probabilities $\frac{1}{3}, \frac{1}{4}, \frac{1}{6}, \frac{3}{20}, \frac{1}{10}$ respectively.

a. Calculate the entropy of S in **bits** to three significant figures.

b. Find a **ternary** Shannon-Fano code for S and its expected codeword length.

c. A **binary** Shannon-Fano code is constructed for S^4. Find the lengths of the two longest codewords in this code.

46 Let $H(p)$ be the entropy of a binary source with probability of emitting a 1 equal to p and probability of emitting a 0 equal to $1-p$. Show that on the interval $0 \le p \le 1$, the function $H(p)$ is nonnegative, concave down and has a unique maximum. Find this maximum and where it occurs and sketch the curve for $0 \le p \le 1$.

47 For the Markov sources with transition matrices as in Questions 36 and 37, find the Markov entropy H_M and equilibrium entropy H_E.

***48** In a certain mathematics course, $\frac{3}{4}$ of the students pass, and the rest fail. Of those who pass, 10% own cars while half of the failing students own cars. All of the car owning students live at home, while 40% of those who do not own cars and fail, as well as 40% of those who do not own cars and pass, live at home.

a. Explain why the probability that a student lives at home or not given whether they own a car or not is independent of whether they have failed or not.

b. How much information (in bits) is conveyed about a student's result in the course if you know whether or not they own a car?

c. How much information (in bits) is conveyed about a student's result in the course if you know whether or not they live at home?

d. If a student's result, car owning status and place of residence are transmitted by three binary digits, how much of the total information in the transmission is conveyed by each digit? (Part (a) will be useful for the third digit.)

49 Consider a channel with source symbols $A = \{a_1, a_2\}$ and output symbols $B = \{b_1, b_2\}$, where $P(b_1 \mid a_1) = 0.8$ and $P(b_2 \mid a_2) = 0.6$. Suppose that $P(a_1) = 1/3$.

a. Using Bayes' Law, calculate $P(a_j \mid b_i)$ for all i, j.

b. Calculate the mutual information of the channel.

50 Find the channel capacity for the channel with source symbols $a_1 = 0$ and $a_2 = 1$, and received symbols $b_1 = 0, b_2 = 1$ and $b_3 = ?$, where $P(b_1|a_2) = P(b_2|a_1) = 0$, $P(b_3|a_1) = P(b_3|a_2) = q$ and $P(b_1|a_1) = P(b_2|a_2) = 1 - q$. (This is a special case of the Binary Symmetric Erasure Channel).

51 A channel has source symbols a_1, a_2, a_3 and received symbols b_1, b_2, b_3 and transition probabilities $P(b_i|a_i) = 1/2, P(b_j|a_i) = 1/4$ for all $i \neq j$. Find all the entropies, conditional entropies and the mutual information for the cases

a. $P(a_1) = P(a_2) = P(a_3) = 1/3$,

b. $P(a_1) = 1/2, P(a_2) = 1/3, P(a_3) = 1/6$,

c. What do you guess the channel capacity is and why?

***d.** Prove that your guess in (c) is correct.

52 Consider a channel with source symbols $A = \{a_1, a_2\}$ and output symbols $B = \{b_1, b_2, b_3, b_4\}$ where for some $0 \leq x \leq 1$ we have $P(a_1) = x$, $P(a_2) = 1 - x$ and

$$P(b_1 \mid a_1) = 5/7, \quad P(b_2 \mid a_1) = 2/7, \quad P(b_3 \mid a_2) = 1/10, \quad P(b_4 \mid a_2) = 9/10.$$

a. Calculate $H(A \mid B)$. How do you explain this result?

b. Hence find the capacity of the channel.

53 (For discussion.) Person A thinks of the name of a person who attends UNSW. Person B tries to determine who they are thinking of by asking A questions to which A has to reply simply "yes" or "no". Show that B can determine the name in 15 questions (assuming that they have access to the UNSW student database).

Chapter 5: Number Theory

54 Use the Euclidean Algorithm to find the gcd d of the following pairs a and b and express it in the form $d = xa + yb$ where $x, y \in \mathbb{Z}$:

(a) 324 and 3876, (b) 7412 and 1513, (c) 1024 and 2187.

55 List all the elements of $\mathbb{U}_{24}$ and hence evaluate $\phi(24)$.

Repeat for $\mathbb{U}_{36}$ and $\mathbb{U}_{17}$.

56 Find $\phi(72)$, $\phi(1224)$ and $\phi(561561)$.

57 Draw up addition and multiplication tables for $\mathbb{Z}_5$ and $\mathbb{Z}_6$ and explain why only the first one is a field.

58 Solve the congruence equations or show there is no solution:

(a) $6x \equiv 7 \pmod{17}$, (b) $6x \equiv 9 \pmod{12}$, (c) $11x \equiv 9 \pmod{13}$.

59 Find, if they exist, the inverse of 6 in

(a) $\mathbb{Z}_{11}$, (b) $\mathbb{Z}_{10}$, (c) $\mathbb{Z}_{23}$.

60 Use Euler's Theorem to find

(a) 2^{1001} in $\mathbb{Z}_{17}$, (b) the last two digits of 3^{1001}.

61 Find all the primitive elements for each of $\mathbb{Z}_{11}$ and $\mathbb{Z}_{17}$.

62 Use the polynomial Euclidean Algorithm to find the gcd $d(x)$ of the following pairs of polynomials $f(x)$ and $g(x)$ and express it in the form $d(x) = a(x)f(x) + b(x)g(x)$ where $a(x)$ and $b(x)$ are polynomials:

a. $x^3 + 1$ and $x^2 + 1$ in $\mathbb{Q}[x]$,

b. $x^3 + 1$ and $x^2 + 1$ in $\mathbb{Z}_2[x]$,

c. $x^2 - x + 1$ and $x^3 - x^2 - 1$ in $\mathbb{Z}_3[x]$.

63 Find

a. $x^5 + x^2 + 1 \pmod{x^2 + x + 1}$ in $\mathbb{Z}_2[x]$,

b. $x^5 + x^2 + 1 \pmod{x^2 + x + 1}$ in $\mathbb{Z}_3[x]$.

64 Write down addition and multiplication tables for both $\mathbb{Z}_2[x]/\langle x^2+x+1\rangle$ and $\mathbb{Z}_2[x]/\langle x^2+1\rangle$. Explain why only the first one is a field.

65 Which of the following are fields: $\mathbb{Z}_2[x]/\langle x^4 + x^2 + x + 1\rangle$, $\mathbb{Z}_3[x]/\langle x^4 + x^2 + x + 1\rangle$.

66 Construct the following finite fields, giving the table of powers of a primitive element γ (or α if α is primitive) and the corresponding linear combinations of the appropriate powers of the root α of the defining polynomial:

a. $\mathbb{Z}_2[x]/\langle x^3 + x + 1\rangle$,

b. $\mathbb{Z}_2[x]/\langle x^4 + x^3 + x^2 + x + 1\rangle$,

c. $\mathbb{Z}_3[x]/\langle x^2 + x - 1\rangle$

Also list all the primitive elements in each field.

67 In $GF(16) = \mathbb{Z}_2(\alpha) = \mathbb{Z}_2[x]/\langle x^4 + x + 1\rangle$:

a. find the inverse of $\alpha^3 + \alpha + 1$,

b. evaluate $(\alpha^3 + \alpha + 1)(\alpha + 1)/(\alpha^3 + 1)$,

c. find the minimal polynomial of $\alpha^3 + \alpha + 1$

d. list all the minimal polynomials formed from powers of α.

68 List all the irreducible polynomials in $\mathbb{Z}_2[x]$ of degrees up to 4. Which of these are primitive in the fields they generate?

69 Show that 341 is a pseudo-prime to base 2 but not to base 3.

***70** Let a be an integer coprime to each of 3, 11 and 17.

a. Using Euler's theorem, show that $a^{560} \equiv 1$ modulo 3, 11 and 17.

b. Hence show that 561 is a Carmichael number.

71 Use Lucas's theorem to test the primality of 97.

***72** Suppose n is a pseudo-prime base b and $\gcd(b-1, n) = 1$. Show that $N = \dfrac{b^n - 1}{b - 1}$ is also pseudo-prime base b. Hence show that there are infinitely many pseudo-primes base 2 and infinitely many pseudo-primes base 3.

73 Show that 561 is not a strong pseudo-prime base 2.

***74** Let n be a fixed odd number and assume that

$$P(n \text{ passes } k \text{ Miller-Rabin tests } \mid n \text{ composite}) < 4^{-k}.$$

Use Bayes' rule to show that for large k we have approximately

$$P(n \text{ composite } \mid \text{ passes } k \text{ Miller-Rabin tests }) < 4^{-k}\frac{P(n \text{ composite})}{P(n \text{ prime})}.$$

75 Use Fermat factorization to factor 14647, 83411 and 200819.

76 Use the Pollard-ρ method to factor $n = 8051$, using $f(x) = x^2 + 1$ and $x_0 = 1$. Repeat for $n = 201001$.

77 Let $n = 92131$. Factorise n with Fermat's method and the Pollard-ρ method (start from $x_0 = 2$).

78 Let $N = 24497$.

a. Use Fermat factorisation to show that $N = 187 \times 131$.

b. Apply the Miller-Rabin test with $a = 2$ to give evidence that 131 is prime.

79 A number n is known to be of the form $n = (2^p - 1)(2^q - 1)$, where p, q, $2^p - 1$ and $2^q - 1$ are (unknown) primes with $p < q$. Find a method of factoring n in approximately p steps. Hence factorise 16646017.

80 Consider the LFSR which implements the recurrence

$$x_{i+3} = x_{i+1} + x_i \quad \text{over} \quad \mathbb{Z}_2$$

Let $x_0 = 1$, $x_1 = 1$, $x_2 = 0$.

a. Generate the next 5 pseudo-random bits produced by the LFSR, namely $x_3, \ldots, x_7$.

b. What is the period of this LSFR?

81 Trace the output of the pseudo-random number generators defined by

a. $x_{i+1} \equiv 7x_i + 1 \pmod{18}$, where $x_0 = 1$.

b. $x_{i+4} \equiv x_{i+3} + x_i \pmod 2$, where $x_0 = 1$, $x_1 = x_2 = x_3 = 0$.

Chapter 6: Algebraic Coding

82 Set $m(x) = x^3 + x^2 + 1 \in \mathbb{Z}_2[x]$ and let α be a root of $m(x)$.

a. Check that α is a primitive element of $GF(8)$.

b. What is the minimal polynomial of α?

c. A single error correcting BCH code is constructed over $GF(8)$ with primitive element α.

(i) What is the information rate of this code?

(ii) Encode [0,1,0,1].

(iii) Find the error and decode [1,0,1,1,0,1,1].

83 Set $p(x) = x^4 + x^3 + 1 \in \mathbb{Z}_2[x]$ and let β be a root of $p(x)$.

a. Show that β is a primitive element of $GF(16)$, with minimal polynomial $p(x)$.

b. A single error correcting BCH code is constructed over $GF(16)$ with primitive element β.

(i) What is the information rate of this code?

(ii) Encode [1,0,0,0,0,1,1,1,0,0,1].

(iii) Find the error and decode [0,0,0,0,0,1,1,1,1,0,0,0,1,1,0].

84 Set $q(x) = x^4 + x^3 + x^2 + x + 1$ and $F = \mathbb{Z}_2[x]/\langle q(x)\rangle$ (i.e. $F = GF(16)$).

a. Find a primitive element γ for F and its minimal polynomial.

b. Construct a single error correcting BCH code over F using γ and use it to

(i) encode [1,0,1,0,0,1,1,1,0,0,1],

(ii) find the error and decode [0,0,0,1,0,1,1,1,1,0,0,0,1,1,1].

85 If β is as in Question 83, find the minimal polynomial for β^3. Construct a double error correcting code over $GF(16)$ using primitive element β and hence

a. Encode [1,0,1,1,0,1,1].

b. Decode [1,1,1,0,1,1,0,0,0,1,1,0,0,0,1], correcting any errors.

c. Decode [1,1,1,0,1,1,0,0,0,1,1,0,1,0,1], correcting any errors.

d. Decode [1,1,0,0,1,0,0,0,0,0,1,1,0,0,1], correcting any errors.

86 A double error correcting BCH code is based on the field $F = \mathbb{Z}_2[x]/\langle x^4 + x + 1\rangle$.

a. Encode the message [1,0,1,1,0,1,1].

b. Decode the message [0,1,1,1,1,0,0,0,1,1,0,1,0,0,1], correcting any errors.

87 A coder is sending out 15-bit words, which are the coefficients of a polynomial $C(x)$ in $\mathbb{Z}_2[x]$ of degree 14 where $C(\alpha) = C(\alpha^2) = C(\alpha^3) = 0$ with $\alpha^4 + \alpha + 1 = 0$.

You receive $R(x) = 1 + x + x^2 + x^4 + x^8 + x^9 + x^{11} + x^{14}$, and assume that at most two errors were made. What was $C(x)$?

88 A triple error correcting BCH code is constructed over $GF(16)$ with primitive element β, where β is a root of the polynomial $p(x) = x^4 + x^3 + 1$ (as in Question 83).

a. What is the information rate of this code?

b. Decode, with corrections, the message [1,1,0,0,0,0,1,0,0,0,0,1,0,0,1].

c.∗ Decode, with corrections, the message [1,0,1,0,1,0,0,1,0,0,1,0,1,0,1].

89 List all the cyclotomic cosets for the field GF(25).

Chapter 7: Cryptography

90 A message of 30 letters $m_1 m_2 \cdots m_{30}$ has been enciphered by writing the message into the columns of a 6×5 array in the order

$$\begin{pmatrix} m_1 & m_7 & \cdots & m_{25} \\ m_2 & m_8 & \cdots & m_{26} \\ \vdots & \vdots & \ddots & \vdots \\ m_6 & m_{12} & \cdots & m_{30} \end{pmatrix},$$

then permuting the columns of the array and sending the ciphertext by rows (reading across the first row, then the second row and so on). The resulting ciphertext is

```
FSOTU  OHFOI  UIJNP  RPUTM  TELHE  HQYEN
```

a. Decipher the message.

b. Write down the permutation σ of the columns which was used to *encipher* the message (that is, column i becomes column $\sigma(i)$ for $i = 1, 2, \ldots, 5$).

c. Using the same permutation and the same method, encipher the message

```
SELL ALL OIL SHARES BEFORE TAKEOVER
```

(remove the spaces before enciphering).

91 The following has been enciphered using a simple transposition cipher with blocks of length 5. Decipher the message.

```
HSTII  AOSTN   EAHRR  ILTEC   NEOCS   ECROT   EPDQS
```

92 Consider the message

```
ELIMINATE THE PERIODIC
```

a. Encipher the message using plaintext feedback with the keyword FISH. (Remove spaces before enciphering.)

b. Encipher the same message with the same keyword, this time using ciphertext feedback (again removing spaces before enciphering).

c. The following ciphertext was enciphered using either plaintext feedback or ciphertext feedback with the keyword FRED.

```
IZHB  RCJC  HNNI  KBNL  KITZ  PEEK
```

Determine which method was used for enciphering and decipher the message.

93 The following has been enciphered using a Vigenère cipher. It is suspected that the keyword has length 2.

```
NZYN  CYYF  YJYU  CHBW  LMMW  MSMW  LAYK  IXWS  YKUJ  WAJZ
YJMT  UKYV  IFNZ  YDYL  NWLK  IXUC  YQQG  LVZG  LFYS  LDSL
BJYW  BMHV  LWXQ  YSLK  NZYN  CYYF  YJYU  CHBW  LOUK  WGHK
CVYJ  YVOF  VJYS  ESVD
```

a. Calculate the index of coincidence and estimate the keyword length.

b. Find other evidence for a keyword length of 2.

c. Decipher the message, assuming that the keyword has length 2.

The following data may be useful:

letter	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
freq1	0	4	5	0	1	0	0	2	3	1	0	9	4	4	1	0	1	0	1	0	4	2	3	1	21	1
freq2	2	0	1	3	0	5	3	2	0	7	7	2	2	2	1	0	2	0	6	1	2	5	7	2	2	4

In the following questions, the standard encoding of the letters of the alphabet is used (see below). No letter is coded to 0 or 1 as they do not change under some of the arithmetic used. A space is coded as 2 and then the letters in order. The resulting numbers are then treated base 29.

0	1	_	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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

94 The following is an example of a matrix method of enciphering.

Given a message $\mathbf{m} = m_1 m_2 m_3 m_4 \ldots$ and a 2×2 matrix A, the message is enciphered by coding the letters as above and then using the transformation

$$A \begin{pmatrix} m_1 & m_3 & \ldots \\ m_2 & m_4 & \ldots \end{pmatrix} = \begin{pmatrix} c_1 & c_3 & \ldots \\ c_2 & c_4 & \ldots \end{pmatrix} \pmod{29}$$

to give the ciphertext $\mathbf{c} = c_1 c_2 c_3 c_4 \ldots$.

For example, given the matrix

$$A = \begin{pmatrix} 2 & 3 \\ 7 & 8 \end{pmatrix}$$

the message **m**=`NOANSWER` is enciphered as **c**=`WNWB1ZND` via

$$\begin{pmatrix} 2 & 3 \\ 7 & 8 \end{pmatrix} \begin{pmatrix} 16 & 3 & 21 & 7 \\ 17 & 16 & 25 & 20 \end{pmatrix} = \begin{pmatrix} 25 & 25 & 1 & 16 \\ 16 & 4 & 28 & 6 \end{pmatrix}$$

a. Using the matrix A from above, work out how to decipher the ciphertext and then decipher **c**=`LUVBRU`.

b. Using the above scheme but with an unknown matrix A, I sent the ciphertext

c = `ZBXWOPIAIZFUZ0`

(note: the last symbol is zero "0", not capital O). But I made the mistake of starting my message with "HELLO". Using this information, work out in theory how you would decipher this and then do the deciphering.

c. Suppose all we had were the ciphertext (i.e. no crib) in **b**. Would we expect a unique meaningful message if we tried all possible matrices?

95 Using an RSA scheme with $n = 551$ and $e = 55$:

a. Encipher the message `HI` using the coding for letters given above and then the RSA encryption;

b. Find the deciphering exponent d and decipher the ciphertext $302, 241$.

96 Using an RSA scheme with $n = 391$ and $e = 235$, decipher the ciphertext

366, 14, 126, 126, 3, 249, 258, 126, 148, 30, 45, 366, 58, 30

where the letters have been coded as above.

97 Using an RSA scheme with $n = 1147$ and $e = 17$ we can encipher pairs of letters by encoding each as above and then writing the pair as a base 29 number (with a space at the end if there is only one letter) as in the examples

$$\texttt{OK} \to 17, 13 \to 17 \times 29 + 13 = 506 \to 506^{17} \equiv 410 \pmod{1147}$$

$$\texttt{A\_} \to 3, 2 \to 3 \times 29 + 2 = 89 \to 89^{17} \equiv 883 \pmod{1147}.$$

a. Encipher the message `HELLO_`.

b. What is the deciphering exponent?

98 A spy has designed a ciphering scheme as follows. Using an RSA scheme, he encodes a 3 letter key by mapping the first letter to the number a_1 and the second letter to the number a_2 etc using the standard encoding and then replacing (a_1, a_2, a_3) by $29^2 a_1 + 29 a_2 + a_3$.

This key is then encrypted using RSA encryption with $n = 10033$ and encryption exponent 1787. The spy then sends the message consisting of the RSA encoded key and the actual message enciphered by plaintext feedback.

a. Factor $n = 10033$ using Fermat factorisation.

b. Hence find $\phi(n)$, and from this calculate the decryption exponent d for the RSA coding of the key.

c. Now decipher the message

8695IRDBHQIPVPBVKBRQ

sent using this code.

99 In a simplified DSS scheme, the universal primes are $q = 13$, $p = 53$ and we have $g = 15$ of order 13 in $\mathbb{Z}_{53}$.

- **a.** Find Alice's public key if $e = 7$.
- **b.** If Alice picks $x = 5$ and sends a message with hash value $h = 9$, what will her signature be?
- **c.** Bob receives a message with a hash value $h = 10$ and signature $(2, 4)$. Is it genuine?

100 Find the unicity distance for the following ciphers:

- **a.** Vigenère cipher using a random keyword of length r;
- **b.** Vigenère cipher using a keyword of length $2s$ made up of alternately a vowel then a consonant then a vowel etc.
- **c.** Polyalphabetic cipher using a random keyword of length r (here each letter of the keyword corresponds to a Caesar cipher);
- **d.** Transposition of length 10;
- **e.** DES cipher;
- **f.** RSA cipher with a 512-bit key.
- **g.** McEliece encryption using a $(1024, 524)$ Goppa code.

Answers

1 (c) 373 is prime, but $323 = 17 \times 19$. **3** In order $\frac{1}{13}, \frac{1}{3}, \frac{1}{13}$, independent.

7 (a) $.999^{100} = .9047921471$ (b) $(1 + .998^{100})/2 - .9047921471 = .0044912554$.

8 (a) Correct (b) $0 - 576 - 08314 - 3$. **9** (a) roughly 10^8 (c) 0610271385

11 HDforallSYN **12** $2, 11, 12, 3$

13 (a) $\mathbf{x}_1 = 10101$, $\mathbf{x}_2 = 01011$, $\mathbf{x}_1 + \mathbf{x}_2 = 11110$, (b) No column is the sum of the other two: cols 1,2,3. (c) e.g. cols 1, 3, 5. (d) No two columns are equal, and none are zero. (e) H has only three pivot columns, so H has a maximum of three linearly independent columns.

14 (a) 1011010 (b) correct to 0001111, decode as 0111

15 correct: 001010101010110 decode: 11011010110

16 (a) 101101001010101 (b) correct to 010100010111110, decode to 00000111110

17 $2^k/2^n = 2^{-m}$

18 (a) $(1-p)^7 = .99302$. (b) $7p(1-p)^6 = .0069581$. (c) $1 - .99302 - .0069581 = .00002093$. (d) $7p^3(1-p)^4 + 7p^4(1-p)^3 + p^7 = 6.979 \times 10^{-9}$

20 (a) $d = 3$ (b) corrects 1 error (c) (i) is correct, decode as 010 (ii) correct to 011001, decode as 011 (iii) has at least two errors, cannot decode

22 (a) $\binom{n}{\rho}(r-1)^\rho$ (b) $|C| \le \dfrac{r^n}{\displaystyle\sum_{i=0}^{t}\binom{n}{i}(r-1)^i}$.

23 (a) $H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 & 3 & 4 \end{pmatrix}$ (b) correct to 410213, decode to 0213.

24 (a) 0221 (b) words (2) and (4)

25 (a) Neither (b) Uniquely Decodable (c) Neither (d) Instantaneous

26 Not UD: $s_4 s_8 s_2 s_3 = s_7 s_1 s_5 s_6$

27 (a) and (d) have $K > 1$. (b) $\{00, 01, 100, 101, 1100, 1101, 1110\}$. Can shorten 1110 to 111 (c) $\{0, 10, 110, 111, 112, 120, 121, 122, 200\}$. Can shorten 200 to 2.

28 (a) 4, (b) 3

29 (a) $\{1, 00, 01\}$, $L = 1.5$ (b) $\{00, 01, 11, 100, 101\}$, $L = 2.217$ (c) $\{1, 01, 001, 0000, 0001\}$, $L = 1.875$ (d) $\{0, 10, 110, 111\}$, $L = 1.425$.

30 (a) Average cost of binary $= 2.407 \times \$2.00 = \$4.82$. Average cost of ternary $= 1.444 \times \$3.25 = \$4.69$ (b) \$3.33.

32 It's a comma code, with 1 as comma.

33 (a) With dummy symbols, $L = 1.47$; without dummy symbols, $L = 2$.

34 (a) For example $(111) \to 1$, $(112) \to 001$, $(121) \to 010$, $(211) \to 011$, $(122) \to 00000$, $(212) \to 00001$, $(221) \to 00010$, $(222) \to 00011$. The average codeword length is $79/96 \approx 0.823$. (b) with the given coding we get 001 010 1 011

35 $L(S) = 1$, $\frac{1}{2}L(S^2) = 0.9\dot{4}$, $\frac{1}{3}L(S^3) = 0.938$

36 (a) $\mathbf{p} = \frac{1}{11}(3, 4, 4)^T$. (b) Each column of the limit is $\mathbf{p}$.

37 (a) $\mathbf{p} = \frac{1}{17}(6, 7, 4)^T$; $L_M = 1.388$, $L_E = 1.588$. (b) 1010010111. (c) $s_3 s_2 s_2 s_1 s_2$.

38 Any number in $[0.4608, 0.4640)$, e.g. 0.461

39 (a) Any number in $[0.49264, 0.49360)$. The shortest is 0.493. (b) $s_1 s_1 s_3 s_1 s_3 \bullet$.

40 (a) (0,b)(0, a)(0,n)(2, n)(2,␣)(4, d)(0,␣)(1,e)(0,e) (b) to be or not to be

41 $H(S) = 2.113$

42 For Q29: (a) $H(S) = 1.460$; (b) $H(S) = 2.140$; (c) $H(S) = 1.875$; (d) $H(S) = 1.280$. Q33: $H(S) = 1.377$ radix 4 units/symbol. Q35: use $H(S^n) = nH(S)$. Q39: $H(S) = 0.556$ decimal bits/symbol.

43 (a) Average cost of binary is \$4.58; Average cost of ternary is \$4.69. (b) \$3.17.

44 Q29: (a) lengths 1, 2, 3; (b)lengths 2, 2, 3, 3, 5; (c) lengths 1,2,3,4,4; (d) lengths 1,3,4,6.
Q33: radix 4 code, lengths 2,2,2,2,2,2,3,3.
Q35: S^1: lengths 1,2; S^2: lengths 2,3,3,4; S^3: lengths 2,3,3,3,4,4,4,5.

45 (a) 2.20; (b) 1.77; (c) 13 and 14

47 Q37: $H_M = 1.293$, $H_E = 1.548$. Q36: $H_M = 1.523$, $H_E = 1.573$.

48 (b) 0.12, (c) 0.03, (d) 0.811, 0.602, 0.777 respectively

49 (a) $P(a_1 \mid b_1) = 1/2$, $P(a_2 \mid b_1) = 1/2$, $P(a_1 \mid b_2) = 1/7$, $P(a_2 \mid b_2) = 6/7$. (b) $I(A, B) = 0.109$.

50 Channel capacity $1 - q$.

51 (a) $I(A, B) = 0.085$, $H(A, B) = 3.085$, $H(A|B) = 1.5$.
(b) $I(A, B) = 0.077$, $H(A, B) = 2.959$, $H(A|B) = 1.382$.

52 (a) $H(A|B) = 0$ as there is no uncertainty in the input once the output is known. (b) capacity is 1.

54 (a) $12 = 324 \times 12 - 3876$ (b) $17 = 1513 \times 49 - 7412 \times 10$
(c) $1 = 1024 \times 472 - 2187 \times 221$.

55 $\mathbb{U}_{24} = \{1, 5, 7, 11, 13, 17, 19, 23\}$, $\phi(24) = 8$. $\mathbb{U}_{36} = \{1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35\}$, $\mathbb{U}_{17} = \{1, 2, \dots 16\}$.

56 $\phi(72) = 24$, $\phi(1224) = 384$, $\phi(561561) = 253440$

58 (a) $x \equiv 4 \pmod{17}$, (b) No solutions since $\gcd(6, 12) = 6$ is not a factor of 9, (c) $x \equiv 8 \pmod{13}$.

59 (a) $6^{-1} = 2$ in $\mathbb{Z}_{11}$, (b) no inverse in $\mathbb{Z}_{10}$, (c) $6^{-1} = 4$ in $\mathbb{Z}_{23}$ **60** (a) 2 (b) 03.

61 (a) $2, 6, 7, 8$ (b) $3, 5, 6, 7, 10, 11, 12, 14$.

62 (a) gcd $= 1$, $a(x) = \frac{1}{2}(1 + x)$, $b(x) = \frac{1}{2}(1 - x - x^2)$. (b) gcd $= x + 1$, $a(x) = 1$, $b(x) = x$. (c) gcd $= x + 1$, $a(x) = x$, $b(x) = -1$.

63 (a) 1 (b) $x + 2$. **65** Only the second one.

67 (a) $\alpha^2 + 1$ (b) $\alpha^3 + \alpha^2 + \alpha + 1$ (c) $x^4 + x^3 + 1$.

71 Bases 2, 3 do not give any information but base 5 does.

75 $14647 = 151 \times 97$, $83411 = 239 \times 349$ and $200819 = 409 \times 491$.

76 $8051 = 83 \times 97$, $201001 = 19 \times 71 \times 149$. **77** $n = 13 \times 19 \times 373$.

79 Use: $(n - 1)/2^p$ is odd. 127×131071 **80** (a) $0, 1, 0, 1, 1$ (b) 7

81 (a) cycle length 18. (b) cycle length 15.

82 (c) (i) 4/7, (ii) $[1, 0, 0, 0, 1, 0, 1]$, (iii) correct: $[1, 0, 1, 0, 0, 1, 1]$, decode: $[0, 0, 1, 1]$.

83 (b) (i) 11/15, (ii) $[1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1]$
(iii) correct to $[0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0]$, decode as $[0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0]$.

84 (a) E.g. $\gamma = \alpha + 1$, where $q(\alpha) = 0$. The min poly γ is $m(x) = x^4 + x^3 + 1$.
(b) Using the BCH code based on γ: (i) $[0,0,0,1,1,0,1,0,0,1,1,1,0,0,1]$,
(ii) correct to $[0,0,0,1,0,1,1,1,1,0,0,0,1,1,0]$, decode to $[0,1,1,1,1,0,0,0,1,1,0]$

86 (a)] $[0,1,1,0,1,1,0,1,1,0,1,1,0,1,1]$ (b) correct to $[0,1,1,1,1,0,0,0,1,0,0,1,1,0,1]$, decode to $[1,0,0,1,1,0,1]$

87 $C(x) = 1 + x + x^2 + x^4 + x^6 + x^8 + x^9 + x^{11} + x^{14}$.

88 (a) 1/3, (b) correct to $[1,1,0,0,0,0,1,0,1,0,0,1,1,0,1]$, decode to $[0,1,1,0,1]$ (c) correct to $[1,1,1,0,1,0,1,1,0,0,1,0,0,0,1]$, decode to $[1,0,0,0,1]$

90 (a) SHIP EQUIPMENT ON THE FOURTH OF JULY (b) The permutation is $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 1 & 3 \end{pmatrix}$, which can also be written as (12534) in cycle notation.
(c) FSKAL OEERO RLOEU ELVSL TAEBS ALREH

91 THIS IS ANOTHER ARTICLE ON SECRET CODES P Q

92 (a) JTAT MYIF MGHX TXYM DHZK (b) JTAT RGAM VZHQ KDYY YGGA (c) DID YOU GET THE RIGHT ANSWERS (plaintext feedback)

93 (a) index of coincidence $I_c = 0.0602$, estimated keyword length 1.253, suggests keyword length either 1 or 2.
(c) THE VIGENERE CIPHER USES A SERIES OF CAESAR CIPHERS BASED ON THE LETTERS OF A KEYWORD FOR NEARLY THREE HUNDRED YEARS THE VIGENERE CIPHER WAS CONSIDERED UNBREAKABL

94 (a) ATTACK (b) HELLO EVERYBODY **95** (a) 409, 182 (b) OK

96 MERRY CHRISTMAS **97** (a) 1037, 424, 991 (b) 953

98 (a) $n = 10033 = 127 \times 79$; (b) $\phi(n) = 78 \times 126 = 9828$, $d = 11$; key word is ARR, message I AM THE PIRATE KING

99 (a) 42, (b) $(5,1)$, (c) No

100 (a) $\lceil 1.47r \rceil$ (b) $\lceil 2.10s \rceil$ (c) $\lceil 1.47r \rceil$ (d) 7 (e) 18 (f) 160 (g) 167680

APPENDIX

A.1 The International Morse Code

A	di dah	N	dah dit	1	di dah dah dah dah
B	dah di di dit	O	dah dah dah	2	di di dah dah dah
C	dah di dah dit	P	di dah dah dit	3	di di di dah dah
D	dah di dit	Q	dah dah di dah	4	di di di di dah
E	dit	R	di dah dit	5	di di di di dit
F	di di dah dit	S	di di dit	6	dah di di di dit
G	dah dah dit	T	dah	7	dah dah di di dit
H	di di di dit	U	di di dah	8	dah dah dah di dit
I	di dit	V	di di di dah	9	dah dah dah dah dit
J	di dah dah dah	W	di dah dah	0	dah dah dah dah dah
K	dah di dah	X	dah di di dah		
L	di dah di dit	Y	dah di dah dah		
M	dah dah	Z	dah dah di dit		

Period	di dah di dah di dah
Comma	dah dah di di dah dah
Question mark	di di dah dah di dit
Error	di di di di di di di dit
Proceed (K)	dah di dah
Wait (AS)	di dah di di dit
Break (BK)	dah di di di dah di dah
End of message (AR)	di dah di dah dit
End of work (SK)	di di di dah di dah

MORSE CODE TIMING

The basic timing measurement is the dot pulse (di, dit),
all other morse code timings are a function of this unit length:

dot length (di, dit)	one unit
dash length (dah)	three units
pause between elements of one character	one unit
pause between characters	three units
pause between words	seven units

From: J. Maynard, *The Computer and Telecommunications Handbook*, Granada (1984)

A.2 ASCII Code

AMERICAN NATIONAL STANDARD
CODE FOR INFORMATION INTERCHANGE

Recall that the convention for ASCII is to number the bits from the right.

b_7	→					0	0	0	0	1	1	1	1
b_6	→					0	0	1	1	0	0	1	1
b_5	→					0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1		COLUMN								
↓	↓	↓	↓	ROW↓	⟶	0	1	2	3	4	5	6	7
0	0	0	0	0		NUL	DLE	␣	0	@	P	`	p
0	0	0	1	1		SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2		STX	DC2	″	2	B	R	b	r
0	0	1	1	3		ETX	DC3	#	3	C	S	c	s
0	1	0	0	4		EOT	DC4	$	4	D	T	d	t
0	1	0	1	5		ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6		ACK	SYN	&	6	F	V	f	v
0	1	1	1	7		BEL	ETB	'	7	G	W	g	w
1	0	0	0	8		BS	CAN	(	8	H	X	h	x
1	0	0	1	9		HT	EM	)	9	I	Y	i	y
1	0	1	0	10		LF	SUB	*	:	J	Z	j	z
1	0	1	1	11		VT	ESC	+	;	K	[	k	{
1	1	0	0	12		FF	FS	,	<	L	\	l	|
1	1	0	1	13		CR	GS	-	=	M	]	m	}
1	1	1	0	14		SO	RS	.	>	N	ˆ	n	~
1	1	1	1	15		SI	US	/	?	O	_	o	DEL

From: J. Maynard, *The Computer and Telecommunications Handbook*, Granada (1984)

A.3 Letter Statistics from the Brown Corpus

Letter	Prob.(%)	Digram	Prob.(%)	Trigram	Prob.(%)	Tetragram	Prob.(%)
␣	17.41	e␣	3.05	␣th	1.62	␣the	1.25
e	9.76	␣t	2.40	the	1.36	the␣	1.04
t	7.01	th	2.03	he␣	1.32	␣of␣	0.60
a	6.15	he	1.97	␣of	0.63	and␣	0.48
o	5.90	␣a	1.75	of␣	0.60	␣and	0.46
i	5.51	s␣	1.75	ed␣	0.60	␣to␣	0.42
n	5.50	d␣	1.56	␣an	0.59	ing␣	0.40
s	4.97	in	1.44	nd␣	0.57	␣in␣	0.32
r	4.74	t␣	1.38	and	0.55	tion	0.29
h	4.15	n␣	1.28	␣in	0.51	n␣th	0.23
l	3.19	er	1.26	ing	0.50	f␣th	0.21
d	3.05	an	1.18	␣to	0.50	of␣t	0.21
c	2.30	␣o	1.14	to␣	0.46	hat␣	0.20
u	2.10	re	1.10	ng␣	0.44	␣tha	0.20
m	1.87	on	1.00	er␣	0.39	.␣␣␣	0.20
f	1.76	␣s	0.99	in␣	0.38	his␣	0.19
p	1.50	,␣	0.96	is␣	0.37	␣for	0.19
g	1.47	␣i	0.93	ion	0.36	ion␣	0.18
w	1.38	␣w	0.92	␣a␣	0.36	that	0.17
y	1.33	at	0.87	on␣	0.35	␣was	0.17
b	1.10	en	0.86	as␣	0.33	d␣th	0.16
,	0.98	r␣	0.83	␣co	0.32	␣is␣	0.16
.	0.83	y␣	0.82	re␣	0.32	was␣	0.16
v	0.77	nd	0.81	at␣	0.31	t␣th	0.16
k	0.49	.␣	0.81	ent	0.30	atio	0.15
T	0.30	␣h	0.78	e␣t	0.30	␣The	0.15
"	0.29	ed	0.77	tio	0.29	e␣th	0.15
...	...	...	...	...	...	...	...

	Letter	Digram	Trigram	Tetragram
Number of units	94	3410	30249	131517
Entropy (bits/letter)	4.47	3.59	2.92	2.33

From: T.S. Bell, J.G. Cleary and I.H. Witten, *Text Compression*, Prentice-Hall (1990).

A.4 26-letter Frequencies in English

a	0.0804	h	0.0549	o	0.0760	u	0.0271
b	0.0154	i	0.0726	p	0.0200	v	0.0099
c	0.0306	j	0.0016	q	0.0011	w	0.0192
d	0.0399	k	0.0067	r	0.0612	x	0.0019
e	0.1251	l	0.0414	s	0.0654	y	0.0173
f	0.0230	m	0.0253	t	0.0925	z	0.0009
g	0.0196	n	0.0709				

Probability distribution of 1-grams in English language

A.5 Vigenère Table

A	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	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
B	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
C	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
D	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
E	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
F	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
G	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
H	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
I	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
J	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
K	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
L	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
M	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
N	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
O	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
P	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
Q	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
R	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
S	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
T	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
U	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
V	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
W	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
X	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
Y	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
Z	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

Table of Caesar substitution ciphers labeled by what ‘A’ becomes

A.6 Big-O notation

When describing algorithms we usually use **big-O** notation.

A function $f(x)$ is said to be of **order** $g(x)$ if there is a constant C such that $|f(x)| < C\,|g(x)|$ for all sufficiently large x. We write this as $f(x) = O(g(x))$.

Usually, of course, g is chosen to be something very simple.

For example, $3x^5 + x^3\sin(x) - x^2\log(x) = O(x^5)$, $\frac{1}{9}x^2\log(x) + 2x^2 + x - 2 = O(x^2\log(x))$. Essentially, we are just picking out the fastest growing term, remembering always that any power x^q, $q > 0$ grows faster than any logarithm, x^q grows faster than x^p iff $q > p$ and any positive exponential grows faster that any power.

When it comes to algorithms, we usually quote the **time complexity** — a function that is proportional to the running time of the algorithm in terms of the size of its input. For example, the standard multiplication algorithm for two numbers with n decimal digits is $O(n^2)$: double the number of digits and you (roughly) quadruple the time it takes. The Karatsuba algorithm for the same process is $O(n^{\log_2 3})$.

Sometimes we are more interested in **space complexity** — how much memory is taken up by the algorithm. The same notation is used.

A.7 Further Reading

Below are some books in the UNSW Library that are useful for further reading.

1. N. Abrahamson, *Information theory and coding*, McGraw-Hill (1963). PX519.7/9 (for Chapters 3, 4)

2. R. Ash, *Information Theory*, John Wiley (1965), recently reprinted by Dover. PX519.7/12A (for Chapters 3, 4)

3. T.S. Bell, J.G. Cleary and I.H. Witten, *Text Compression*, Prentice-Hall (1990). P005/44 (for Chapter 4)

4. R. Bose *Information theory, coding and cryptography*, Tata McGraw-Hill (2002).

5. G. Brassard, *Modern cryptography*, Springer (1988). P005.82/4 (for Chapter 7)

6. R. W. Hamming, *Coding and information theory*, Prentice-Hall (1986). P003.54/3 (for Chapters 3, 4)

7. R. Hill, *A first course in coding theory*, Clarendon (1986). S005.72/4 (for Chapters 2, 6)

8. V. Pless, *Introduction to the theory of error correcting codes*, Wiley (1982/89). P001.539/23, P005.72/5 (for Chapters 2, 6)

9. O. Pretzel, *Error-Correcting Codes and Finite Fields*, Clarendon (1992). P003.54/20 (for Chapters 2, 5, 6)

10. S. Roman, *Coding and information theory*, Springer (1992). P003.54/19 (for Chapters 2, 3, 4, 5, 6)

11. A. Salomaa, *Public-key cryptography*, Springer (1996). P005.82/11A (for Chapter 7)

12. B. Schneier, *Applied Cryptography*, Wiley (1996). P005.82/10A (for Chapter 7)

13. S. Singh, *The Code Book: The Evolution of Secrecy From Mary Queen of Scots to Quantum Cryptography*, Doubleday (1999). P652.809/2 (for Chapter 7)

14. H. C. A. van Tilborg, *An introduction to cryptology*, Kluwer (1988). P652.8/5 (for Chapter 7)

Singh is a well-written popular science book about the history of cryptography, from the Caesar cipher to the modern day.