This chapter assumes the reader is familiar with the basics of linear algebra.

Mathematics uses the term code to mean something used to send data where the data is recoverable even if the message is somewhat corrupted. This is different to encryption, where the goal is to send data that is private.

### 18.1 Two Simple Codes

We assume the data is given as a binary string.

**Example 18.1. Check-bits**

The check-sum of a string is the sum of all the bits, modulo 2. For example, if the data is 01101, then the check-sum is 1. The message sent is the data with the check-sum appended.

The claim is that: *the receiver is able to detect if exactly one of the bits is changed in transmission*. For, the check-bit is 0 if and only if the original data contains an even number of 1’s. So if exactly one of the data bits is changed, the the check-sum is wrong. Similarly a change in the check-sum is also detectable. But note that there is absolutely no information as to where the error was. And if two of the bits are changed then the error is undetectable.

Check sums are used in many other places. For example, the ISBN number of a book has last digit a check-sum.

**Example 18.2. Repetition Code**

In a repetition code, every bit is sent multiple times. Say every bit is sent three times. This code has the power to detect and correct a single bit error. Indeed, this code can handle multiple errors, as long as they don’t occur twice in the same triple. However, it is inefficient, as what is sent is three times as long as the original data.

### 18.2 Binary Linear Codes

Both of the above are examples of linear codes. In a binary linear code, each sent bit is a linear combination of the data bits. That is, we think of the data \( d \) as a binary vector,
and the code \( e \) that is sent is given by

\[
e = dM
\]

where matrix \( M \), called the **generator matrix**. Arithmetic is modulo 2 (or equivalently, in the group \( \mathbb{Z}_2 \)).

Here are the generator matrices for the above two codes, in the case that \( d \) has three bits:

\[
C_3 = \begin{pmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1
\end{pmatrix}
\quad \text{and} \quad
D_3 = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{pmatrix}
\]

### 18.3 Some Theory

For two vectors of the same length, their **Hamming distance** is the number of places the corresponding entries are different.

Given a code, an important collection is the set of all vectors generated (aka the range of the linear transform or the row-space of the generator matrix). The **distance** of a code is the minimum Hamming distance between any two vectors in it.

**Theorem 18.1**  
(a) A code can detect \( k \) errors if and only if its distance is at least \( k + 1 \).  
(b) A code can correct \( k \) errors if and only if its distance is at least \( 2k + 1 \).

**Proof.** If there are at most \( k \) errors, then the Hamming distance between the sent vector and the received vector is at most \( k \). Thus the received vector can be another possible transmission vector if and only if the distance of the code is \( k \) or less.

Further, to correct the errors, there must be a unique possible sent vector that is closest to the received vector. So any vector can be Hamming distance at most \( k \) away from one sendable vector. If the distance of the code is at least \( 2k + 1 \), then this will be true. If the distance of the code is \( 2k \), say between vectors \( e_1 \) and \( e_2 \), then any vector “half-way” between \( e_1 \) and \( e_2 \) cannot be corrected without the possibility of error. ◊

### 18.4 Hamming Codes

Now, what one would like in a code is a code that (a) is efficient (the sent message is not much longer than the original) and (b) that is easy to decode (if the code has the property that one can correct some number of errors, then these corrections can be done quickly).
One idea is the Hamming code. There are 4 data bits and 3 check bits. The 1st check bit is for the 1,2,4 data bits, the 2nd check bit is for the 1,3,4 data bits, and the 3rd check bit is for 2,3,4 data bits. For example, suppose the data is 1101. Then the first check bit is 1, the second is 0, and the third is 0.

Note that each data bit affects two or more of the check bits. This means that if one of the data bits is changed, at least two of the check-bits are flipped. That is, the distance of this code is at least 3. So by the above theorem, one can correct one error.

But the neat idea of Hamming was how to do the decoding quickly. Specifically, Hamming placed the three check bits in positions 1, 2 and 4 of the resultant code. That is, the generator matrix is:

$$H_4 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

For example, suppose the data is 1101. Then the sent message is 1010101.

To decode, take the received message and verify the check bits. Then:

Construct a number $p$ as follows. Start with $p = 0$. If the first check bit is wrong, add 1 to $p$; if the second check bit is wrong, add 2 to $p$; and if the third check bit is wrong, add 4 to $p$. The final value of $p$ says which bit in the received message was flipped!

For example, suppose that string 1010001 is received. Then the (alleged) data is 1001 and the 1st and 3rd check bits are wrong. That is $p = 1 + 4 = 5$. And, hey presto, that is correct.

Of course, the real question is where did this come from and why does it work. Well, note first that when we write the message as ccDcDDD, the positions of the check bits are powers of 2. Further, the first check bit is dependent on all positions which, written in binary, have a 1 in the last column, namely 1, 3, 5, 7; the second check bit is dependent on all positions which, written in binary, have a 1 in the middle column, namely 2, 3, 6, 7; and the third check bit is dependent on all positions which, written in binary, have a 1 in the first column, namely 4, 5, 6, 7.

In general, one can build a Hamming code with $c$ check bits and $2^c - c - 1$ data bits.

### 18.5 Reed–Muller Codes

We specify a specific family of Reed-Muller codes by the set of strings in it. This is an inductive definition.
Start with $R_1 = \{0, 1\}$. The set $R_{m+1}$ is obtained from the set $R_m$ by taking every string $w$ in $R_i$ and writing down both $ww$ and $w\overline{w}$, where $\overline{w}$ means string $w$ with all the bits flipped.

For example, $R_2$ is all 2-bit strings. The strings in $R_m$ have length $2^{m-1}$ and there are $2^m$ of them. (Check!) Note that the code is closed under flipping all bits; that is, if $w$ is a string in $R_{m-1}$ then so is $\overline{w}$.

The key claim is that: the distance of $R_m$ is at least $2^{m-2}$ for $m \geq 2$. The proof is by induction. True for the base case $m = 2$, since $2^{2-2} = 1$. Assume true for $R_m$. Now consider two strings in $R_{m+1}$, say $x$ and $y$. Note that both halves are themselves strings in $R_m$. So if the first half of $x$ and first half of $y$ are different, then by the IH they differ in at least $2^{m-2}$ bits. Similarly with the second halves. So if the halves both differ, the distance between $x$ and $y$ is at least $2^{m-2} + 2^{m-2} = 2^{m-1}$. If the first half of $x$ is the same as the first half of $y$, then by the way the code is constructed, this means that the second half of $x$ is the bit-flip of the second half of $y$, and so $x$ and $y$ differ in half their length, which is $2^{m-1}$, as required.

It might not be obvious from the above that we can represent as a generator matrix. But it is possible. And again there is a nice decoding algorithm. And by starting with a different set, one can obtain codes with other properties.

**Exercises**

18.1. We saw above the Hamming code with 3 check bits. Explain what the Hamming code with 2 check bits looks like.

18.2. For the Hamming code with 4 check bits, there are 11 data bits. Determine the 4 check bits for
   (a) the data 00000000000
   (b) the data 11111111111
   (c) the data 10101010101

18.3. For the general Hamming code with $k$ check bits, show that the distance is exactly 3.

18.4. Show that the distance of the Reed–Muller code $R_m$ is exactly $2^{m-2}$.

18.5. Consider the Reed–Muller code $R_3$.
   (a) List all strings in $R_3$.
   (b) Provide a suitable generator matrix.
18.6. (a) Consider a code with distance $d$ with $d$ odd. Show that if one appends a parity bit to every string, then the new code has distance $d + 1$.

(b) Give an example that shows that part (a) is not necessarily true if $d$ is even.