Error Detection

Error detecting codes enable the detection of errors in data, but do not determine the precise location of the error.

- store a few extra state bits per data word to indicate a necessary condition for the data to be correct
- if data state does not conform to the state bits, then something is wrong
- e.g., represent the correct *parity* (# of 1's) of the data word
- 1-bit parity codes fail if 2 bits are wrong…

1011 1101	0001 0000	1101 0000	1111 0010	1

parity bit is 1:
data should
have an odd
number of 1's

A 1-bit parity code is a *distance-2 code*, in the sense that at least 2 bits must be changed (among the data and parity bits) produce an incorrect but legal pattern. In other words, any two legal patterns are separated by a distance of at least 2.

Parity Bits

Two common schemes (for single parity bits):

- *even parity* 0 parity bit if data contains an even number of 1's
- *odd parity* 0 parity bit if data contains an odd number of 1's

We will apply an even-parity scheme.

1011 1101	0001 0000	1101 0000	1111 0010	1

The parity bit could be stored at any fixed location with respect to the corresponding data bits.

Upon receipt of data and parity bit(s), a check is made to see whether or not they correspond.

Cannot detect errors involving two bit-flips (or any even number of bit-flips).

Hamming (7,4) Code

Richard Hamming (1950) described a method for generating minimum-length error-detecting codes. Here is the (7,4) Hamming code for 4-bit words:

Data bits $d_4d_3d_2d_1$	Parity bits $P_3p_2p_1$
0000	000
0001	011
0010	101
0011	110
0100	110
0101	101
0110	011
0111	000
1000	111
1001	100
1010	010
1011	001
1100	001
1101	010
1110	100
1111	111

Say we receive the data word 0100 and parity bits 011.

Those do not match (see the table).

Therefore, we know an error has occurred. But where?

Hamming (7,4) Code

Suppose the parity bits are correct (011) and the data bits (0100) contain an error:

The received parity bits 011 suggest the data bits should have been 0001 or 0110.

The first case would mean two data bits flipped.

The second would mean that one data bit flipped.

Data bits $d_4d_3d_2d_1$	Parity bits $p_3p_2p_1$
0000	000
0001	011
0010	101
0011	110
0100	110
0101	101
0110	011
0111	000
1000	111
1001	100
1010	010
1011	001
1100	001
1101	010
1110	100
1111	111

Hamming (7,4) Code

Suppose the data bits (0100) are correct and the parity bits (011) contain an error:

The received data bits 0100 would match parity bits 110.

That would mean two parity bits flipped.

If we assume that only one bit flipped, we can conclude the correction is that the data bits should have been 0110.

If we assume that two bits flipped, we have two equally likely candidates for a correction, and no way to determine which would have been the correct choice.

Data bits $d_4d_3d_2d_1$	Parity bits $p_3p_2p_1$
0000	000
0001	011
0010	101
0011	110
0100	110
0101	101
0110	011
0111	000
1000	111
1001	100
1010	010
1011	001
1100	001
1101	010
1110	100
1111	111

Hamming (7,4) Code Details

Hamming codes use extra parity bits, each reflecting the correct parity for a different subset of the bits of the code word.

Parity bits are stored in positions corresponding to powers of 2 (positions 1, 2, 4, 8, etc.).

The encoded data bits are stored in the remaining positions.

Data bits: 1011 $d_4 d_3 d_2 d_1$

Parity bits: 010 $p_3 p_2 p_1$

Hamming encoding:

	b_7	b_6	b_5	b_4	b_3	b_2	b_1
101?1??	d_4	d_3	d_2	p_3	d_1	p_2	p_1

Hamming Code Details

But, how are the parity bits defined?

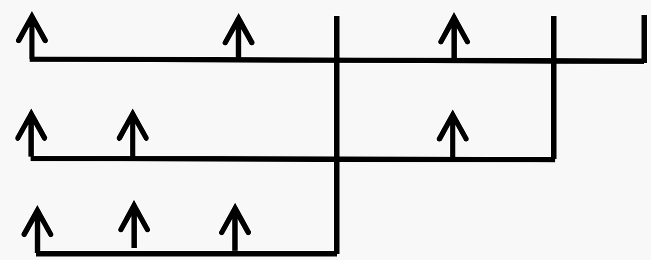
- p_1: all higher bit positions k where the 2^0 bit is set (1)
- p_2: all higher bit positions k where the 2^1 bit is set

…

- p_n: all higher bit positions k where the 2^n bit is set

Hamming encoding:

	b_{111}	b_{110}	b_{101}	b_{100}	b_{11}	b_{10}	b_{1}
101?1??	d_4	d_3	d_2	p_3	d_1	p_2	p_1



This means that each data bit is used to define at least two different parity bits; that redundancy turns out to be valuable.

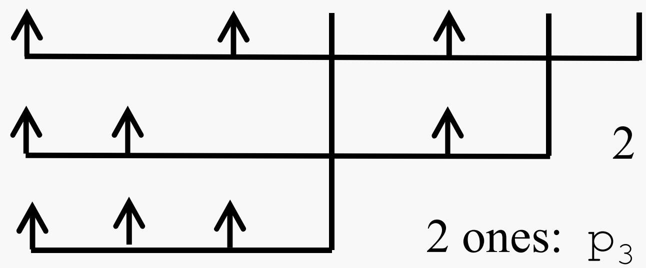
Hamming (7,4) Code Details

So, for our example:

Hamming encoding:

b_{111} b_{110} b_{101} b_{100} b_{11} b_{10} b_{1}

`101?1??` 1 0 1 p_3 1 p_2 p_1



3 ones: p_1 == 1

2 ones: p_2 == 0

2 ones: p_3 == 0

So the Hamming encoding would be :

1010**1**01

Error Correction

A *distance-3* code, like the Hamming (7,4) allows us two choices.

We can use it to reliably determine an error has occurred if no more than 2 received bits have flipped, but not be able to distinguish a 1-bit flip from a 2-bit flip.

We can use to determine a correction, under the assumption that no more than one received bit is incorrect.

We would like to be able to do better than this.

That requires using more parity bits.

The Hamming (8,4) allows us to distinguish 1-bit errors from 2-bit errors. Therefore, it allows to reliably correct errors that involve single bit flips.

The Hamming Code pattern defined earlier can be extended to data of any width, and with some modifications, to support correction of single bit errors.

Hamming (11,7) Code

Suppose have a 7-bit data word and use 4 Hamming parity bits:

0001	P1 : depends on D1, D2, D4, D5
0010	P2 : depends on D1, D3, D4, D6
0011	D1
0100	P3 : depends on D2, D3, D4, D7
0101	D2
0110	D3
0111	D4
1000	P4 : depends on D5, D6, D7
1001	D5
1010	D6
1100	D7

Note that each data bit, Dk, is involved in the definition of at least to parity bits.

And, note that no parity bit checks any other parity bit.

Hamming (11,7) Code

Suppose have a 7-bit data word and use 4 Hamming parity bits:

P1 : depends on D1, D2, D4, D5
P2 : depends on D1, D3, D4, D6
P3 : depends on D2, D3, D4, D7
P4 : depends on D5, D6, D7

Note that each data bit, Dk, is involved in the definition of at least to parity bits.

And, note that no parity bit checks any other parity bit.

Hamming (11,7) Code

P1 : depends on D1, D2, D4, D5
P2 : depends on D1, D3, D4, D6
P3 : depends on D2, D3, D4, D7
P4 : depends on D5, D6, D7

Suppose that a 7-bit value is received and that one data bit, say D4, has flipped and all others are correct.

Then D4 being wrong will cause three parity bits to not match the data: P1 P2 P3

So, we know there's an error…

And, assuming only one bit is involved, we know it must be D4 because that's the only bit involved in all three of the nonmatching parity bits.

And, notice that: 0001 | 0010 | 0100 == 0111

binary indices of the incorrect parity bits

binary index of the bit to be corrected

QTP: what if the flipped data bit were D3?

Hamming (11,7) Code

P1 : depends on D1, D2, D4, D5, D7
P2 : depends on D1, D3, D4, D6, D7
P3 : depends on D2, D3, D4
P4 : depends on D5, D6, D7

What if a single parity bit, say P3, flips?.

Then the other parity bits will all still match the data: P1 P2 P4

So, we know there's an error…

And, assuming only one bit is involved, we know it must be P3 because if any single data bit had flipped, at least two parity bits would have been affected.

Why (11,7) is a Poor Fit

Standard data representations do not map nicely into 11-bit chunks.

More precisely, a 7-bit data chunk is inconvenient.

If we accommodate data chunks that are some number of bytes, and manage the parity bits so that they fit nicely into byte-sized chunks, we can handle the data + parity more efficiently.

For example:

(12,8) 8-bit data chunks and 4-bits of parity, so… 1 byte of parity per 2 bytes of data

(72,64) 9-byte data chunks per 1 byte of parity bits

Hamming (12,8) Code

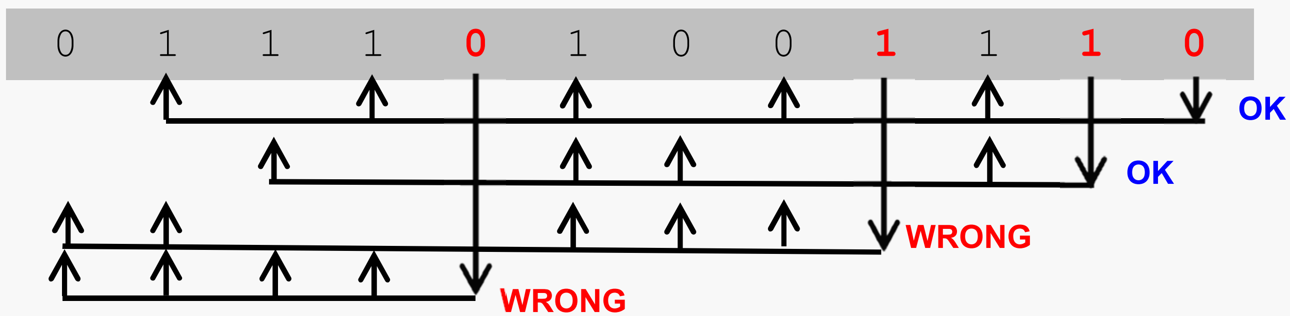
Suppose have an 8-bit data word and use 4 Hamming parity bits:

0001	P1 : depends on D1, D2, D4, D5, D7
0010	P2 : depends on D1, D3, D4, D6, **D7**
0011	D1
0100	P3 : depends on D2, D3, D4, ~~D7~~, D8
0101	D2
0110	D3
0111	D4
1000	P4 : depends on D5, D6, D7, D8
1001	D5
1010	D6
1011	D7
1100	D8

Hamming (12,8) Code Correction

Suppose we receive the bits: 0 1 1 1 **0** 1 0 0 **1** 1 **1** **0**

How can we determine whether it's correct? Check the parity bits and see which, if any are incorrect. If they are all correct, we must assume the string is correct. Of course, it might contain so many errors that we can't even detect their occurrence, but in that case we have a communication channel that's so noisy that we cannot use it reliably.



So, what does that tell us, aside from that something is incorrect? Well, if we assume there's no more than one incorrect bit, we can say that because the incorrect parity bits are in positions 4 (0100) and 8 (1000), the incorrect bit must be in position 12 (1100).