Error Correcting Codes

Components of a computer system perform the following functions:

1. Information processing (logic elements, CPU etc.)
2. Information storage (RAM, registers etc.)
3. Information transmission (interconnects, lines etc.)

All these components are susceptible to failures and therefore can produce erroneous output(s), i.e., some of the output bits are changed. Our concern here is to **detect** an erroneous output and if possible take a **corrective** action.

We achieve this by *encoding* information, i.e., by appending some redundant bits to the information bits in such a manner that the errors can be detected and possibly corrected by special logic circuits. Let us demonstrate the principle by a simple example.
Duplication: Let us consider the case where each information bit x is duplicated (for example transmitted over two lines or stored at two locations in RAM). If only one error is likely to occur, then instead of receiving xx we will receive $x\bar{x}$ or $\bar{x}x$ (but not $\bar{x}\bar{x}$ or xx). Thus, we will immediately know that *an error has occurred* whenever we receive two **non-identical** bits. This is *error detection.* Of course, we will not be able to determine what was transmitted.

Error detection

One of the simplest schemes for single error detection is to use a check bit called *parity bit.* Let $A = a_0, a_1, ..., a_{n-1}$ be an n bit word. A parity bit p is defined as

$p = a_0 \oplus a_1 \oplus a_2 \oplus ... \oplus a_{n-1}$, i.e. $p = 0$ if A has an even number of ones, and $p = 1$ otherwise

where $\oplus$ denotes the Exclusive-OR operation. In this scheme every word A is stored in RAM along along with its parity bit p. On reading RAM, parity is recomputed and checked against stored value of p. An error is said to have occurred if computed(p) $\neq$ stored(p). It is not difficult to see that the above scheme will detect not only a single error but an odd number of errors in the stored word. Note also that the error(s) detected need not be confined to the word A – they could also involve the parity bit itself!

Error correction

The *Hamming distance* between binary vectors $A = \langle a_0, a_1, \ldots, a_{n-1}\rangle$ and $B = \langle b_0, b_1, \ldots, b_{n-1}\rangle$ is the number of bits of A we need to flip to get B. Mathematically,

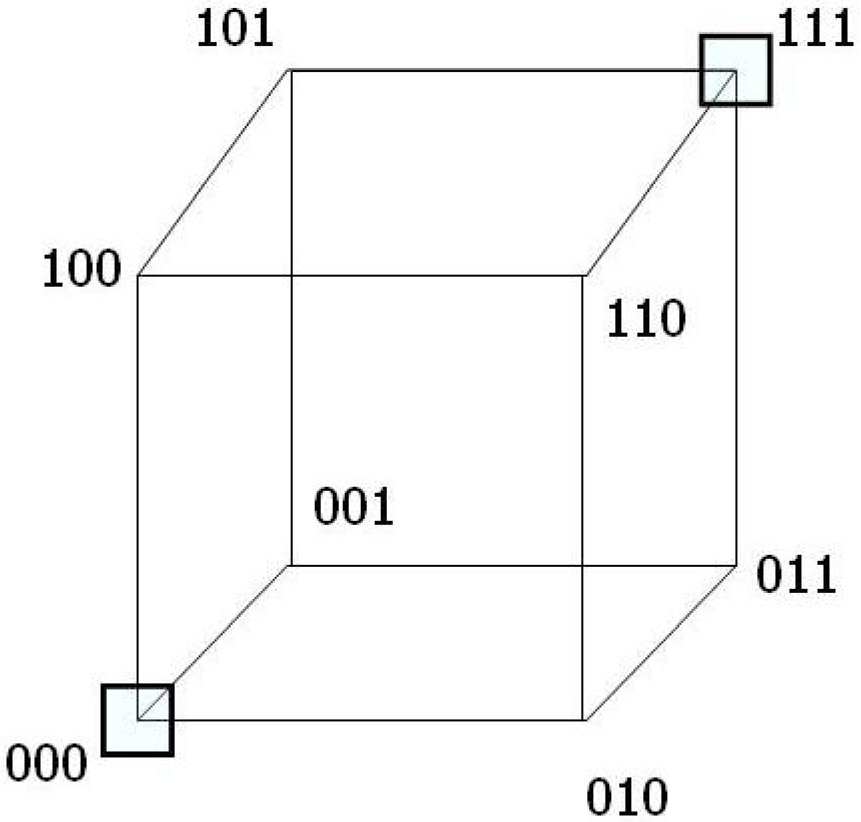
$$H(A, B) = \sum_{i=0}^{i=n-1} (a_i \oplus b_i)$$

.
Example: For $A = (0100)$ and $B = (0010)$, $H(A, B) = 2$

Another way to look at the Hamming distances is that if A and B are nodes of a binary n-cube, then the Hamming distance between them is the minimum number of links traversed to get from node A to B (or from node B to A). It is also evident that two adjacent nodes on the binary n-cube have a Hamming distance of one.

In order to introduce redundancy, we add extra bits to a data word – the result is called a *codeword.* The extra bits are selected in such a way that the **minimum** Hamming distances between all pairs of code words is **maximized**.

Figure 1. The 3-cube



Consider the case of appending a parity bit to each data word. Suppose every code word has an even numbers of 1's in it. From this, we can conclude that the minimum Hamming distance for such a code will be at least two (at least two bits need to be flipped to get from codeword A to codeword B, since flipping one bit makes the number of 1's odd).

Example:
Assume that only a single bit of information is to be transmitted and that 0 is to be encoded as 000 and 1 as 111. We use a 3-cube with a minimum distance of 3 as shown in Figure 1. Thus, if a single-bit error occurs during the transmission of a code word, it changes that code word into a noncode word. The noncode word received would be at a distance of 1 from the original code word and at a distance of 2 from any other code word.

Example:
If 000 is transmitted, a single bit error can change it to 100, 010, or 001. Therefore, if we receive a noncode word with one 1 in it, we change it into 000. On the other hand, if we receive a noncode word with two 1s in it, we change it to 111. This approach results in single-bit error detection and single-bit error correction.

SECDED: Single-bit Error Correction, Double-bit Error Detection (SECDED) is the error correcting code used for standard ECC protected SDRAM.
If a given noncode word is 1 bit away from a code word it can be easily corrected. But, if it differs from code words by 2 bits then the error can be detected, but cannot be corrected, because there are 2 code words with equal Hamming distance away.

Exercises

1. ECC Questions

   (a) (True/False) Parity enables the correction of single-bit errors. Explain your answer.

   (b) What Hamming distance is required to support SECDED?

   (c) Consider full replication (k copies of data stored in multiple locations, $k > 1$). How much error detection and error correction does it provide?

2. I/O Questions

   The IBM Deskstar 75 GXP hard drive has an average seek time of 8.5ms. The rotational speed is 7200 RPM. The controller overhead is about 0.3ms, and tests show the drive can transfer 36 million bytes of data per second. We have a 32MB (32×2^{20} bytes) data file, which is heavily fragmented, so all of its sectors are scattered randomly around the disk.

   (a) How long would it take to read the entire file, assuming the hard drive has 512-byte sectors?

   (b) How long would it take to read the entire file if the hard drive had 4,096-byte sectors instead?

   (c) In what situations would smaller sectors be more desirable than larger ones?

Appendix

Minimum Hamming distance and error detection/correction capability are closely related. If Hamming distance between two codewords is d then no $(d-1)$ errors can transform a codeword into another codeword. Thus all $(d-1)$ errors in such a code (code space) will be detectable.

The following two results can be derived from the above discussion:

Result 1: All errors of d or fewer bits in a codeword can be detected if and only if the minimum Hamming distance of the code is $(d+1)$.

Result 2: All errors of c or fewer bits in a codeword can be corrected if and only if minimum Hamming distance of the code is $(2c+1)$.

This follows from the fact that any codeword with c errors will still be at least distance $(c+1)$ from any other codeword. Thus a scheme which will map erroneous information to the nearest (in the sense of Hamming distance) codeword will cause c errors to be corrected.

The above two results when merged together provide the following result:
Result 3: A code is a c error-correcting OR d error detecting $(d \geq c)$ code if and only if its minimum Hamming distance is $\max((d+1),(2c+1))$.

Result 4: The Common Case: A code is a c error-correcting AND d error detecting $(d \geq c)$ code if and only if its minimum Hamming distance is $(d+c+1)$.

The relation between Hamming distance (H), number of bit errors that can be detected (d) and number of bit errors that can be corrected (c) is given in the following table for Result 4 above. Derive the Result 3 table yourself for practice. You'll just need to use $H = \max((d+1),(2c+1))$.

H	d	c
1	0	0
2	1	0
3	2 1	0 1
4	3 2	0 1
5	4 3 2	0 1 2
6	5 4 3	0 1 2
7	6 5 4 3	0 1 2 3

Hamming Code:
In this scheme, several check bits (k) are generated for a data word of m bits by using multiple parity checks on certain subsets of the data bits. The check bits are then combined with the data bits to form a codeword. Note that for a code to have single error detection and single error correction capability the combination of k check bits must be able to identify any one of

the $(m + k)$ faulty positions in case of single error and also it should be able to indicate that no bit is faulty. Thus the relation between m and k is

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

.