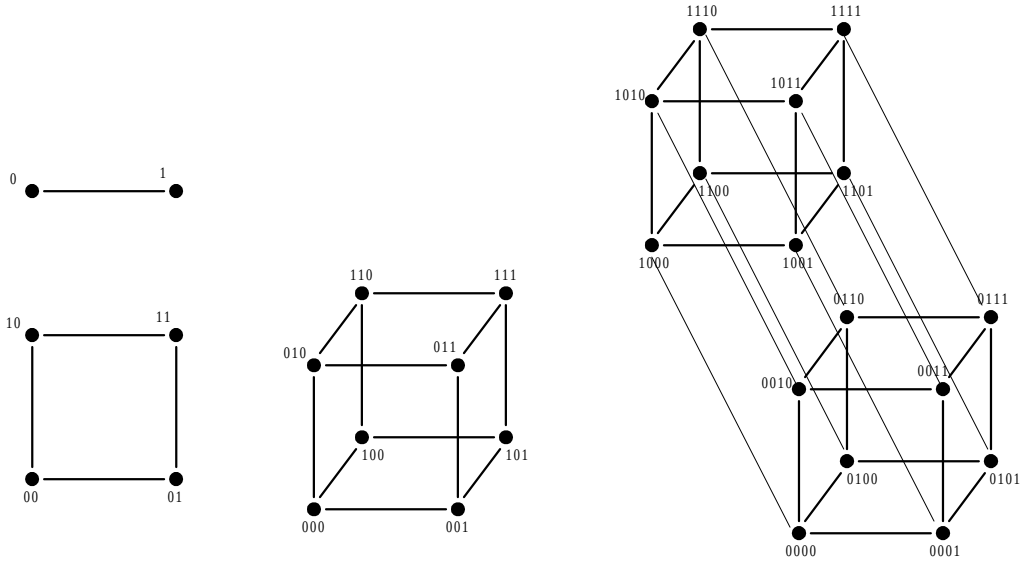
2.12 Code for Error Detection and Correction

To understand error-detecting codes, we need to introduce the concept of Boolean distance, which in term requires the definition of an *n*-cube.

Any *n*-bit string can be visualized as one of the vertices of a binary ***n*-cube**, which is a cube with a total of 2^n vertices, each of which corresponds to a particular string with *n* bits.

Two vertices are connected if and only if the strings to which they correspond differ in one bit.



n-cubes for *n* = 1, 2, 3, and 4.

Hamming distance - the distance between any two vertices is equal to the shortest path between them, which in turn, is equal to the number of bit positions in which they have different binary values.

In general, an ***m*-subcube** of an *n*-cube can be defined as that set of 2^m vertices in which $n - m$ of the variables will have the same value at every vertex, while the remaining *m* variables will take on the 2^m possible combinations of the values 0 and 1.

For example, in the 3-cube, the 2-subcube is defined as that set of $2^m = 2^2 = 4$ vertices in which $n - m = 3 - 2 = 1$ of the variable will have the same value at every vertex (0xx, or 1xx) while the remaining variables will take on the $2^m = 2^2 = 4$ possible combinations (00, 01, 10, and 11)

2.12.1 Error-Detecting Codes

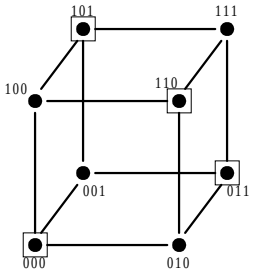
Any given *n*-bit code can be regarded as a subset of all possible *n*-bit strings. Strings included in that particular subset are called **code words**, while strings not included are called **noncode words**.

A code is called an **error-detecting code** if it has the property that certain types of errors will change a code word into a noncode word.

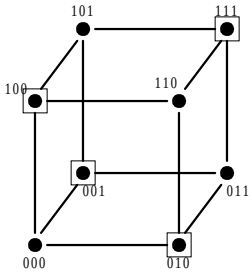
To detect a single-bit error, i.e. to detect a single bit that has been changed from 0 to 1 or vice versa, it is essential that any two code words in the code have a distance ≥ 2. In other words, we must choose code words in such a way that no two vertices that represent code words would be adjacent in its corresponding *n*-cube.

In general, to maintain a distance of 2 in an arbitrary code, we need an equal number of code words and noncode words. Thus, for an *n*-bit data, usually called **information bits**, we can use (*n*+1)-bit code words by adding one more bit, usually called the **parity bit**.

The value of this parity bit can be set in such a way that the number of 1's in the code word is even for all code words and odd for all noncode words - **even-parity code**.



Even-parity code



Odd-parity code

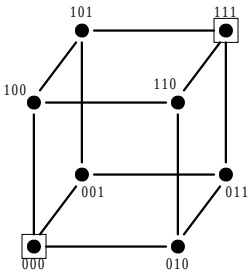
2.12.2 Error-Correcting Codes

To correct a single-bit error, we need more than one parity bit, since this single bit can indicate only that an entire code word is correct or incorrect.

We need an extra $\log_2 n$ parity bits for each n information bits.

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.



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.

Ex. 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 1's in it, we change it to 111.

The code discussed above can also detect, but not correct, cases where there are two single-bit errors.

2.13 Hamming Codes

A general method for constructing error-correcting codes by using a minimum distance of three.
Every integer m there is a $(2^m - 1)$-bit **Hamming code** which contains m parity bits and $2^m - 1 - m$ information bits.
The parity bits are intermixed with the information bits as follows:

If we number the bit positions from 1 to $2^m - 1$, the bits in position 2^k, where $0 \le k \le m - 1$, are the parity bits, and the bits in the remaining positions are information bits.

The value of each parity bit is chosen so that the total number of 1's in a specific group of bit positions is even.

For $m = 3$, we have a $(2^m - 1 =)$ 7-bit Hamming code as shown below:

7	6	5	4	3	2	1	Bit positions
1	1	1	1	0	0	0	Parity group for parity bit 4
1	1	0	0	1	1	0	Parity group for parity bit 2
1	0	1	0	1	0	1	Parity group for parity bit 1
			↑		↑	↑	
			Parity bits				

Parity bits are in position 2^k, where $0 \le k \le m - 1$: $\Rightarrow 2^0, 2^1, 2^2 \Rightarrow 1, 2, 4$.
Information bits in the remaining positions: 3, 5, 6, 7.
For each parity bit in position 2^k, its corresponding group of information bits includes all those bits in the position whose binary representation has a 1 in position 2^k. e.g.

- For parity bit 4=100, therefore positions 5, 6, & 7 are in the group since they all have a 1 in position 4.
- For parity bit 2=010, therefore, positions 3, 6, & 7 are in the group since they all have a 1 in position 2.
- For parity bit 1=001, therefore, positions 3, 5, & 7 are in the group since they all have a 1 in position 1.

The information bits in position 7, whose binary representation is 111, is used to compute the value of the parity bits in positions 4, 2 and 1.
The information bits in position 6, whose binary representation is 110, is used only for computing the value of the parity bits in positions 4 and 2.
The information bits in position 5, whose binary representation is 110, is used only for computing the value of the parity bits in positions 4 and 1.

Example: To send the information bits **1001**, we would send the code word **100***1***1***00*** because the parity bits are *100*.

7	6	5	4	3	2	1	parity bit
1	**0**	**0**	?	**1**	?	?	
1	**0**	**0**	*1*	**1**	?	?	*1*
1	**0**	**0**	*1*	**1**	0	?	*0*
1	**0**	**0**	*1*	**1**	0	*0*	*0*
			↑		↑	↑	

Example: transmit 1010101 but received a single-bit error in position 6, i.e. 1110101.

Count the number of 1 bits in each of the parity groups for 1110101:

7	6	5	4	3	2	1	# 1 bits in group
1	1	1	0	1	0	1	3
1	1	1	0	1	0	1	3
1	1	1	0	1	0	1	4
			↑		↑	↑	
			wrong parity		wrong parity		

Odd number of 1 bits in group 4 and 2. Therefore, the bit error is in position 4+2 = 6.

For m = 4, we have a (2^m-1 =) 15-bit Hamming code with 4 parity bits and 11 information bits as shown below:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit positions
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	Parity group for parity bit 8
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	Parity group for parity bit 4
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	Parity group for parity bit 2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	Parity group for parity bit 1
							↑				↑		↑	↑	

Parity bits