SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Other important codes

Prof. Daniele Gorla

ASCII Code

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

ASCII is an acronym for *American Standard Code for Information Interchange*

Born in IBM in 1961, becomes ISO standard (*International Organization for Standardization)* in 1968.

7 bits for codifying all the (capital and non capital) letters of the english alphabeth, decimal digits, punctuation symbols, special characters, …

- the most 3 signifying bits identify the kind (ex.: 000 and 001 are special characters, 011 decimal digits, 100 and 101 capital letters, etc.)
- the remaining 4 bits codify the character in a monotonic way (whenever a natural ordering exists)

  Ex.: *a* preceeds *d* in the alphabeth → ASCII(*a*) < ASCII(*d*)
  1 is less than 5 →ASCII(1) < ASCII(5)

7 bit ASCII Code

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Bits b_4	b_3	b_2	b_1	Row ↓ / Column → 0	1	2	3	4	5	6	7
				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
0	0	0	0	0	NUL	DLE	SP	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	FC	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

NULL

Start/end of Text

Horiz./Vert. Tab

Carriage return

Extended ASCII

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

ASCII's problem: 7 bits → 128 possible codewords

Many extensions to 8 bits (every character is codified by one byte)
→ Probl.: every brand (IBM, Commodore, …) had its own code, not necessarily compatible with the 7 bits standard!!

ISO Standard (8859), made up by different parts:

1. 256 characters for western Europe languages
2. 256 characters for central Europe languages
3. 256 characters for southern Europe languages
4. 256 characters for northern Europe languages
5. 256 characters for slavic languages (cyrillic)
6. 256 characters for arabic
7. 256 characters for greek
8. 256 characters for jewish
9. …

Unicode

Problem for ISO 8859: same code for different characters (of different areas).

1991: Unicode → unique code for all languages (both present and past), ideograms, math and chemical symbols, Braille,…

Originarially at 16 bits, nowadays at 21 bits (many unused sequences).

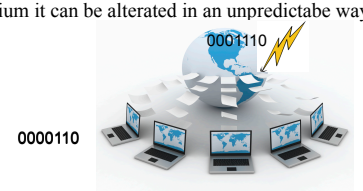
Supported by the main programming platforms and operating systems (Java, XML, Corba,…).

It is *not* a standard but it is continuously updated by the Unicode Consortium.

It also allows for "simplified" versions at 8 or 16 bits, that onlt contain the most frequently used characters.

Error detecting and Correcting Codes

Whatever a sequence of bit represents, when transmitted over a physical medium it can be alterated in an unpredictabe way:



We shall now hint at techniques able to detect and, possibly, correct transmission errors.

OBS.: if $|\{\text{codewords}\}| = |\{\text{messages to be codified}\}|$, then no detection (neither correction) is possible!
→ we need *redundant* codes
(i.e., where $|\{\text{codewords}\}| > |\{\text{messages to be codified}\}|$)

Remark: higher redundancy → higher protection BUT higher cost

Parity bit Code

The simplest error detecting code consists in codifying 2^n messages with n+1 bits → only half of the possible words are codewords!

A sequence w of n bits is codified by the sequence (of n+1 bits) wb, where:

$$b = \begin{cases} 0 & \text{if } w \text{ has an even number of 1s} \\ 1 & \text{otherwise} \end{cases}$$

Every codeword has an even number of 1s → *even parity code*
(we "waste" half of the possible words – those with an odd number of 1s)

We detect 1 error, no correction

There is an odd number of 1s!! ERROR!!! But where??

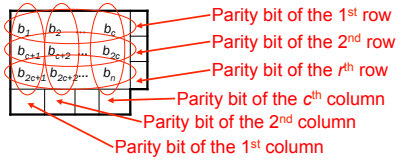
There is an even number of 1s!! Does NOT detect 2 errors!!!

Longitudinal and Vertical Parity Code (1)

Let n be the number of the message bits and let $n = r \times c$.

We represent the message as a table with r rows and c colums, each with its own parity bit → codewords are long $n+r+c$ bits

Mess = $b_1 \ldots b_n$



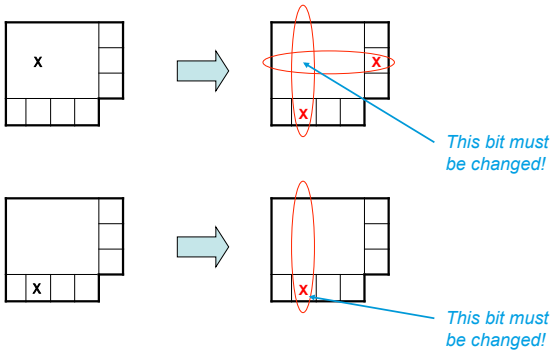
How many redundancy bits?
the best case is when n is a perfect square → $r = c = \sqrt{n}$
the worst case is when n is a prime number → $r = n$ and $c = 1$
Hence, we add a number of redundancy bits (r+c) that varies between $2\sqrt{n}$ and n+1.

We detect 2 errors, correct 1.

Longitudinal and Vertical Parity Code (2)

SAPIENZA Università di Roma Dipartimento di Informatica

It can correct 1 error:



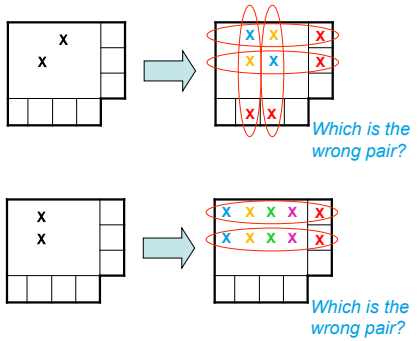
Longitudinal and Vertical Parity Code (3)

SAPIENZA Università di Roma Dipartimento di Informatica

It can detect the presence of 2 errors in the message:

a) Not aligned

b) Aligned



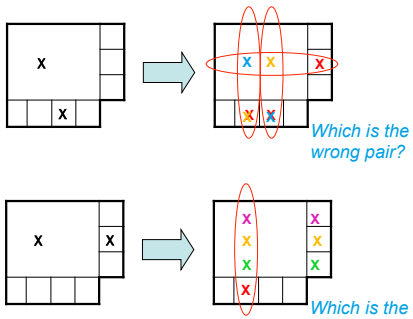
Longitudinal and Vertical Parity Code (4)

SAPIENZA Università di Roma Dipartimento di Informatica

It can detect 2 errors, one in the message and one in the parity bits:

a) Not aligned

b) Aligned



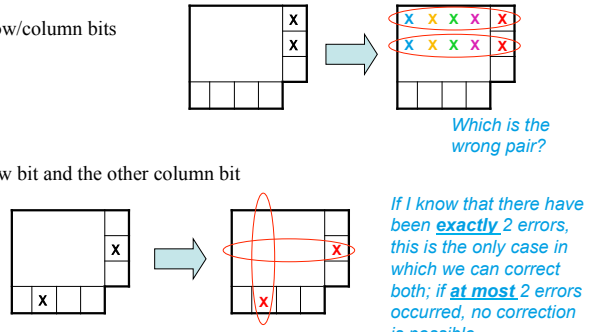
Longitudinal and Vertical Parity Code (5)

SAPIENZA Università di Roma Dipartimento di Informatica

It can detect 2 errors in the parity bits:

a) Both row/column bits

b) One row bit and the other column bit



Hamming Code

It corrects 1 error and detects 2, but with a smaller number of bits (w.r.t. the long. & vert. parity code)

→ it always uses $\log_2 n + 1$ bits, instead of at least $2\sqrt{n}$

Already for $n = 4$ this is better ($\log_2 4 + 1 = 3$, $2\sqrt{n} = 4$)!

Many codes, called *Hamming codes 2^n-to-$(n+1)$*: messages at 2^n bits and $n+1$ parity check bits.

It can be used with arbitrarily long messages

→ if messages include *m bits*, we take the smallest n such that $m \leq 2^n$, that is, we take $n = \lceil \log_2 m \rceil$

→ we put $2^n - m$ meaningless 0s at the beginnig of the message

Hamming Code 4-to-3

Idea: mix control bits (in positions that are a power of 2) and message bits (in the remaining positions):

Mess.: $m_4 m_3 m_2 m_1$

Mess.:	m_4	m_3	m_2		m_1		
Posit.:	7	6	5	4	3	2	1
Contr.:				c_3		c_2	c_1

We check parity of substrings:

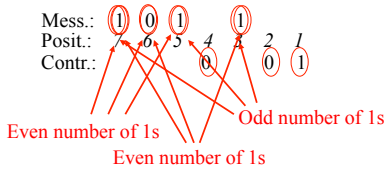
- c_1 checks parity of $m_1 m_2 m_4$;
- c_2 checks parity of $m_1 m_3 m_4$;
- c_3 checks parity of $m_2 m_3 m_4$.

	m_4	m_3	m_2	c_3	m_1	c_2	c_1
c_1	√		√		√		√
c_2	√	√			√	√	
c_3	√	√	√	√			

We set the control bits so that each of these substrings has an even parity (i.e., an even number of 1s)

Example

Find the Hamming 4-to-3 codeword for the message 1011.



Hence, the codeword associated to the message 1011 is 1010101.

Correct 1 error with Hamming codes

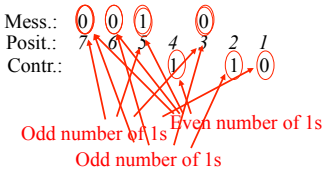
By assuming that at most one error occurs, we can easily identify (and correct) it in the following way:

Check the parity of the substrings $c_1 m_1 m_2 m_4$, $c_2 m_1 m_3 m_4$ e $c_3 m_2 m_3 m_4$ (i.e., the characters in position 1-3-5-7, 2-3-6-7 e 4-5-6-7, respectively):

$c_1 m_1 m_2 m_4$	$c_2 m_1 m_3 m_4$	$c_3 m_2 m_3 m_4$	
"1" odd	"1" odd	"1" odd	**error in m_4** = $\{c_1,m_1,m_2,m_4\} \cap \{c_2,m_1,m_3,m_4\} \cap \{c_3,m_2,m_3,m_4\}$
"1" odd	"1" odd	"1" even	**error in m_1** = $(\{c_1,m_1,m_2,m_4\} \cap \{c_2,m_1,m_3,m_4\}) \setminus \{c_3,m_2,m_3,m_4\}$
"1" odd	"1" even	"1" odd	**error in m_2** = $(\{c_1,m_1,m_2,m_4\} \cap \{c_3,m_2,m_3,m_4\}) \setminus \{c_2,m_1,m_3,m_4\}$
"1" even	"1" odd	"1" odd	**error in m_3** = $(\{c_2,m_1,m_3,m_4\} \cap \{c_3,m_2,m_3,m_4\}) \setminus \{c_1,m_1,m_2,m_4\}$
"1" odd	"1" even	"1" even	**error in c_1** = $\{c_1,m_1,m_2,m_4\} \setminus (\{c_2,m_1,m_3,m_4\} \cup \{c_3,m_2,m_3,m_4\})$
"1" even	"1" odd	"1" even	**error in c_2** = $\{c_2,m_1,m_3,m_4\} \setminus (\{c_1,m_1,m_2,m_4\} \cup \{c_3,m_2,m_3,m_4\})$
"1" even	"1" even	"1" odd	**error in c_3** = $\{c_3,m_2,m_3,m_4\} \setminus (\{c_1,m_1,m_2,m_4\} \cup \{c_2,m_1,m_3,m_4\})$
"1" even	"1" even	"1" even	**no error**

Example

Decide whether 0011010 is a Hamming 4-a-3 codeword; if yes, give the associated message; if no, identify the error (by assuming that there was just one), correct it and give the original message.



- Error in $c_1 m_1 m_2 m_4$;
- Error in $c_2 m_1 m_3 m_4$;
- Error not in $c_3 m_2 m_3 m_4$.

} The wrong bit is m_1

The correct codeword is 0011110, so the original message was 0011.

Detect 2 errors with Hamming Codes

By assuming that there were either 2 or none errors, we can detect these two scenarios, still by checking the parity of the substrings in position 1-3-5-7, 2-3-6-7 e 4-5-6-7 ($c_1 m_1 m_2 m_4$, $c_2 m_1 m_3 m_4$ e $c_3 m_2 m_3 m_4$):

- if they are all parity correct, then there was no error;
- if at least one of these is parity wrong, there were 2 errors, but we're not able to identify the pair of bits to be corrected.

Ex.: the sequence 1110010 is not a Hamming 4-a-3 codeword:
- 1110010 : even 1s
- 1110010 : odd 1s
- 1110010 : odd 1s

With 1 error, we can state that the original codeword was 1010010.

With 2 errors, we cannot univoquely decide the original codeword: it can be 1100110, 0110011 or 1111000.

Detecting 3 errors with Hamming 4-to-3

With 3 errors, it is possible that a codeword turns into another codeword; hence we cannot either detect the presence of errors.

Ex.: if in the codeword 1100110 bits in position 3, 4 and 7 gets corrupted (i.e., if 1100110 becomes 0101010), we obtain a seuqence of bits that is still a Hamming 4-a-3 codeword! Indeed:
- 0101010 has an even number of 1s;
- 0101010 has an even number of 1s;
- 0101010 has an even number of 1s.

To detect/correct more errors, we need different and more sophisticated codes (no more based on parity)