Note :- "Single bit errors are most occuring errors".

lets go with "even parity first",

000 0 "even no. of 1's"
001 1
011 0
010 1 Now this all is ["even-parity"].
110 0
111 1
101 0
100 1

Now after increasing the number of bits in some meaningful manner. We now check for the distance b/w any two "valid patterns."

So, the distance becomes ["2"] b/w every two valid patterns.

Now since the ["hamming distance"] for this code is ["2"] now. It can be used for "single-bit error detection.

•——•

"CHAPTER - 34" :-

→ "Hamming - Code" :-

It is also known as [("single-bit "Error Correction - Code")].

So, It means that, if there is "an error" in (1-bit) then it will correct that error.

We are interested in ["1-bit error"] becoz it is the most frequently occurring error & the "probability" that ("many bits") or more than one bit gets corrupted is very less.

let us see how does it work :->

let us first look at the theory behind "Hamming - Codes" :->
let us say, we are sending ["m-bits"] where ["m"] is the no. of bits in the "message to be transmitted".

Now what we do is, to the ["actual-message"] we are adding some ("parity bits") & these "parity - bits" are then used for ["error - correction"] purposes.

To the message to be transmitted ("m-bits") we add ["p"] "parity bits". these ["p" parity bits] added are just "redundant" & they are used only for ("error-correction purposes.").

$$["m+p"] \longrightarrow$$

Now when this ["m+p"] bits received by the "receiver" on the other side, then the various cases, we might have are :->

1> There will be ("no error") at all.
or
2> There will be a ("single-bit error") in any one of the bits. (m+p)

Here our assumption is that, if there is ("an error") then it will be atmost a ["single-bit error."].

$$["m+p"] \longrightarrow \{(m+p)+1\}$$

↓ "1-bit error" in any of these "m+p" bits

↳ "no error at all."

∴ Total number of cases we have to consider is
$[(m+p)+1]$.

P.T.O

Now we should be able to detect each of these "(m+p+1)" ca
separately.

We are going to detect these ("m+p+1") cases separately using ["p-bits"].

So, using ["p bits"], we have to get those many combinations as there are cases.

So using "p-bits", we get ["2^p combinations"].

& now these 2^p combinations should be able to handle all these (m+p+1) cases

$$\therefore \quad \boxed{2^p \geq (m+p+1)}$$

So this is the "condition" to detect & correct "1-bit error".

Let us now understand it with the help of an example (Numerical)

Let us assume that, we are sending a message consisting of ["4-bits"]

Now to these ["4-bits"], just to add the "correction part", we are adding ["p" number of bits].

$$4 \text{ bits} + p \longrightarrow \underbrace{(4+p)} + 1$$

↳ "No corruption at all"

↓

("1 bit error in any of these p+4-bits").

∴ we need to handle [" (4+p) +1 "] number of cases, so as to correct a '1-bit error'"

So in total, we have to handle (p+5) cases.

& we can handle these (p+5") cases using "p" parity bits.

Now according to the condition:

$$\boxed{2^p \geq p+5}$$

Now let us see the smallest value of ["p"] which satisfies this ("inequality condition").

The reason is simple, we need to add as less "number of parity-bits as possible.

if we put ["p = 3"] this "inequality" is satisfied as:->

$\Rightarrow$ ("$2^3 \geq 3+5$")

$\Rightarrow$ ["$8 \geq 8$"] condition is satisfied ∴ ["p = 3"]

Hence we will be sending ["4-message-bits"] and ["3-parity bits"] to detect/correct "1 bit error" when the message is ["4-bits long"].

In total we will be sending ["4+3 = 7" bits] as a message towards the "receiver".

["7-bits"] (so "error" can occur in any of the "7 bits")

("1, 2, 3, 4, 5, 6, 7")

or

there will be ["no "error"] at all.

("0").

∴ In total we have to handle these "8" cases.

Now let us take an example & find out how to set the "parity bits"

we know that, if "m = 4" the "p = 3"
where "m" is the no. of "message bits" & "p" is the no. of "parity bits".
So in total we have to send ["7"-bits] for a ["4-bit message"] ("4 msg bits & "3 parity bits")

let us say the no. we are sending as a message is ["5"]

("m = 0101").

& in order to correct a "single bit error", we are adding ("3-parity bits").

In total we are sending ["7 bits"]

2^0	2^1		2^2			
1	2	3	4	5	6	7

Note:=> At every power of ["2"] try to set the "parity bit".
& at every other location try to include the ["message-bits"].

"2^0"	"2^1"		"2^2"			
"1"	"2"	"3"	"4"	"5"	"6"	"7"
P_1	P_2	m_1	P_3	m_2	m_3	m_4

Now we are supposed to fill in ["P_1"] ["P_2"] & ["P_3"], [m_1], [m_2], [m_3] & [m_4], we already know what to fill.

"1"	"2"	"3"	"4"	"5"	"6"	"7"
P_1	P_2	m_1	P_3	m_2	m_3	m_4
		0		1	0	1

Note:=> By default, we are going to use ["even-parity method"] to fill the "parity-bits".

"C1", "C2", "C3" are the "bits", that the "receiver" will 'calculate' in order to find out the place where the error has occurred.

DATE □□ □□ □□□□

Now we need to find out as which ["parity-bit"] should be assigned what?

For that reason, let us assume ["C1" "C2" & C3"] are the "final things" that the "receiver" will "check". Let us call them as ("check-bits").

	"P3"	"P2"	"P1"	
	"C1"	"C2"	"C3"	
0	"0"	"0"	"0"	→ "No-error"
1	"0"	"0"	"1"	for "every position" there might be an "error" occurring
2	"0"	"1"	"0"	
3	"0"	"1"	"1"	
4	"1"	"0"	"0"	
5	"1"	"0"	"1"	
6	"1"	"1"	"0"	
7	"1"	"1"	"1"	

When we are sending "7 bits", then the "combinations" we might get are there might be "no error" or there might be "an error" at any one of the ("7 bit locations") at pos. no "1", pos. no. "2" ---- at pos. no "7". ∴ we have to handle all these ["8-cases"]

So, for ["every position" there might be an error" occurring.

It is not just the case that "data" might get "corrupted", even the "parity-bit" could get "corrupted". & therefore we would not be able to distinguish whether the ("data has corrupted") or ("parity get corrupted.").

So, we are supposed to find out ["every error,"] "error"] can occur in ["data bits"] or it can even occur in ["parity-bits"].

In ["C3"] check for the "1's" (1, 3, 5, 7)

↓

"P1" will take care of ["3, 5 & 7"] bit positions.

Similarly in ["C2"] check for "1's": (2, 3, 6, 7)

↓

"P2" will take care of [3, 6 & 7] bits,

classmate

It is the responsibility of ["P1"] to check that the parity of bits ("3, 5 & 7") put together will be ["even"]..

Similarly
It is the responsibility of ("P2") to check that the parity of bits ("3, 6 & 7") put together will be ["even"]..

Now let's have the no. of "1's" bits in "C1"

{"4, 5, 6 7"}

So, it is the responsibility of ["Parity bit"] at bit position ["4"] i.e. ["P3"] to check that the parity of bits ("5, 6 & 7") put together will be ["even"]..

So, this is how we are allocating responsibilities to the ("parity bits")

("1, 3, 5, 7") → ("P1's responsibility")

("2, 3, 6, 7") → ("P2's responsibility")

("4, 5, 6, 7") → ("P3's responsibility").

["P1" is present at position no. "1"
"P2" is present at position no. "2"
4
"P3" is present at position no. "4"]

Note:→ ["Check bits"] will be computed at the ("Receiver's end")

Now once we send this "message" to the "other side" i.e. to the receiver, then if any bit get corrupted in bit positions

("1","3","5","7") then the "parity" will change.

Likewise we can even say about ["P2"] & ["P3"].

["P1"] will take care of ("bit positions") ("3", "5" & "7").

P_1	"3"	"5"	"7"
$\boxed{0}$	0	1	1

Now in order to make it ("even parity") [P_1 = "0"].

["P2"] will take care of ("bit-positions") ("3", "6" & "7").

P_2	"3"	"6"	"7"
$\boxed{1}$	0	0	1

Now in order to make it ("even parity") ["P2"] will be set to [1].

["P3"] will take care of ("bit-positions") ("5", "6" & "7").

P_3	"5"	"6"	"7"
$\boxed{0}$	1	0	1

Now in order to make it ("even-parity") ["P3"] will be set to ["0"].

Even though the message that we have to "transmit" is ("0101") which is [5] in "BCD", we will be sending after the parity bits is.

$$\Rightarrow \left(\begin{array}{ccccccc} P_1 & P_2 & M_1 & P_3 & M_2 & M_3 & M_4 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right)$$

Now let us see how will this be useful in correcting the "error".

Let us see what happens :⇒

$$\left(\begin{array}{ccccccc} \text{'P}_1\text{'} & \text{'P}_2\text{'} & \text{'M}_1\text{'} & \text{'P}_3\text{'} & \text{'M}_2\text{'} & \text{'M}_3\text{'} & \text{'M}_4\text{'} \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right)$$

let us say that the bit "M2" which has to be "1" got corrupted & got converted into "0" as shown below :→

$$\left(0\;1\;0\;0\;\underset{\underset{0}{\downarrow}}{*}\;0\;1\right)$$

So, the receiver will receive ("0100001") instead of receiving ("0100101").

Now let us see how will the receiver calculate the "check-bits"

$$\text{"}C_1\text{"}\quad \text{"}C_2\text{"}\quad \text{"}C_3\text{"} \qquad\qquad \left(\overset{1}{0}\,\overset{2}{1}\,\overset{3}{0}\,\overset{4}{0}\,\overset{5}{0}\,\overset{6}{0}\,\overset{7}{1}\right)$$

$$1 \quad 0 \quad 1$$

["$C_1 = (4,5,6,7)$"]. It has to be even parity but it is odd, hence there has to be something wrong in it.

∴ ["$C_1 = 1$"] to make it "even".

["$C_2 = (2,3,6,7)$."]. It has to be even parity, and after the recepti on of message, it is still "even".

∴ ["$C_2 = 0$"] ("No-error")

["$C_3 = (1,3,5,7)$"]. It has to be even parity but it is odd. Hence there is some error in it

∴ ["$C_3 = 1$"] to make it even.

$$\begin{array}{ccc} \text{"}C_1\text{"} & \text{"}C_2\text{"} & \text{"}C_3\text{"} \\ [1 & 0 & 1] \end{array} \rightarrow \text{"}5\text{"}$$

This means location number "5" has got an error.

∴ At the receivers end we can correct location number

["5"] from ["0 to 1"] to get the original delivered message.
we simply have to ["toggle the bits / corrupted bit"] to get "original message."

[check Bits which say we have an error, Take intersection of them & subtract the bits from the check bit set which indicates no error].

⇨ "CHAPTER-35"

"Problems on Hamming-code" :->

We assume that, we have ["four msg bits"] & ["three parity bits"]. So, in total we have "8-cases" to take care of. Hence we have 3 check bits.

	"P3"	"P2"	"P1"	
	"C1"	"C2"	"C3"	
"0" ←	0	0	0	["No-error"]
"1" ←	0	0	1	→ [["erroneous cases"]]
"2" ←	0	1	0	
"3" ←	0	1	1	
"4" ←	1	0	0	
"5" ←	1	0	1	
"6" ←	1	1	0	
"7" ←	1	1	1	

["P1"] is taking care of ("1", "3", "5", "7")

"Parity bit" at position no. "1" is supposed to take care of bit positions ("3", "5" & "7").

["P2"] is taking care of ("2", "3", "6", "7")

P.T.O

PAGE

"Parity-bit" at "position no. 2" is supposed to take care of {"3", "6" & "7"} bit position.

∴ ["P2"] will take care of ("3" "6" & "7" bit position)

"Parity-bit" at position no. ["4"] i.e ["P3"] is supposed to take care of {"5", "6" & "7"} "bit positions".

∴ ["P3"] will take care of ("5", "6" & "7") bit positions

Let us now assume that, we are supposed to send a "message" ("M" = "1100")

Now we are supposed to have ["3-parity bits"] to be added to the message ["M"],

'P_1'	'P_2'	'M_1'	'P_3'	'M_2'	'M_3'	'M_4'
"1"	"2"	"3"	"4"	"5"	"6"	"7"
$\boxed{0}$	$\boxed{1}$	1	$\boxed{1}$	1	0	0

Let's now see, as what will be the value of parity at position number ["1"]

i.e ("P_1 = ?"). ["3", "5" & "7"] have an ("even parity")

∴ ["$P_1 = 0$"]

~~Let us check for the value to be assigned at~~ Parity bit "P2"

"P2" is supposed to take care of ["3", "6" & "7"] & it has an ("odd parity")

∴ ["$P_2 = 1$"] to make ("3, 6, 7") combined parity ("even").

"P_3" is supposed to take care of [5, 6 & 7] 5 6 & 7 has odd parity. ∴ P_3 will be set to "1" to make it even.

$$\therefore \left["P_3 = 1" \right].$$

Now the "message" that will be transmitted is

"1"	"2"	"3"	"4"	"5"	"6"	"7"
0	1	~~1~~ ↓ 0	1	1	0	0

let us now assume that bit at ("position-3") got corrupted during "transmission" & it now becomes ["0"].

& the receiver has got the message as shown below

1	2	3	4	5	6	7
'0'	'1'	'0'	'1'	'1'	'0'	'0'

Now let us see how the receiver will correct this corrupted message.

"C_1"	"C_2"	"C_3"
0	1	1

[C_3] has to take care of [{1, 3, 5, 7}]. [{1, 3, 5, 7}] has "odd parity"

$$\therefore [C_3 = 1]$$

[C_2] has to take care of [{2, 3, 6, 7}]. [{2, 3, 6, 7}] has "odd parity"

$$\therefore [C_2 = 1]$$

[C_1] has to take care of [{4, 5, 6, 7}]. [{4, 5, 6, 7}] has "even parity"

$$\therefore [C_1 = 0]$$

["0 11" → "3"] hence bit at position ["3"] got corrupted

Now receiver will correct that "bit".

DATE □□□□□□□□

Q:→ let us say we have a "5-bit message"

"M = 5"

Now $2^p \geq M+p+1$

$\Rightarrow 2^p \geq p+6$.

$\therefore \boxed{p=4}$ Now "Message" will have now "5"+"4" bits = "9 bits" in total.

Using 4 bits, we can have "16 combinations", but we need only "10 combinations"

[9 for single-bit error in 9 bits (5 M + 4 P)
& 1 for no-error condition]

"P4"	"P3"	"P2"	"P1"	
"C1"	"C2"	"C3"	"C4"	
0	0	0	0	→ "0"
0	0	0	1	→ "1"
0	0	1	0	→ "2"
0	0	1	1	→ "3"
0	1	0	0	→ "4"
0	1	0	1	→ "5"
0	1	1	0	→ "6"
0	1	1	1	→ "7"
1	0	0	0	→ "8"
1	0	0	1	→ "9"

"C4" = { 1, 3, 5, 7, 9 }

"C3" = { 2, 3, 6, 7 }

"C2" = { 4, 5, 6, 7 }

"C1" = { 8, 9 }

let M = "11011"

"2^0"	"2^1"		"2^2"				"2^3"	
1	2	3	4	5	6	7	8	9
'P1'	'P2'	'M1'	'P3'	'M2'	'M3'	'M4'	'P4'	'M5'
0	0	1	0	1	0	1	1	1

Now let us find out the value of ["P1", "P2", "P3" & "P4"]

["P1"] has to take care of {"3, 5, 7, 9"} which has ("even-parity") already ∴ $\boxed{P_1 = 0}$.

["P2"] has to take care of {"2, 3, 6, 7"} which has ("even-parity") already ∴ $\boxed{P_2 = 0}$.

["P3"] has to take care of {"5, 6, 7"} which has ("even-parity") already ∴ $\boxed{P_3 = 0}$.

["P4"] has to take care of {"9"} which has ("odd-parity") ∴ we need to make it even.

∴ $\boxed{P_4 = 1}$

∴ "Message" that now be ("send") by the sender will be :→

'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
0	0	1	0	1	0	*	1	1

↓
"0"

Let us now assume that ["bit"] at (position number "7") get corrupted. & gets changed into "0" during transmission.

Now the ("message") received by the "receiver" is :→

'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
'0'	'0'	'1'	'0'	'1'	'0'	'0'	'1'	'1'

Let us now see, how ("receiver") is going to make a "correction".

"C1"	"C2"	"C3"	"C4"
0	1	1	1

→ "Error" is at "7th position". Hence receiver can correct it.

— o —