Lezione 5 Codici ridondanti e non 2

martedì 3 ottobre 2023 13:54

Torniamo ad un esercizio sula virgola mobile : Convertire 0.3 in virgola mobile:

Si supponga di avere mantissa a 5 bit e esponente a 4 bit

Numero	Moltiplicato per 2	Resto
0.3	0.6	0
0.6	1.2	1
0.2	0.4	0
0.4	0.8	0
0.8	1.6	1

Qui il processo si interrompe che si ripete , quindi il risultato è :

0.01001(si ripetono)

Riportando il risultato nella forma canonica 1.x si ha che :

1,001(periodico)*2^(-2)

Il quale riportando la mantissa a 5 bit diventa :

1,00110**01*2^(-2)

Riassumendo quindi :

N=7
Esponente = -2+7=+5 -> 0101
S=0

Andiamo a vedere ora l'errore assoluto e quello relativo :

$$\varepsilon_A = x - x' \qquad \varepsilon_R = \frac{x - x'}{x}$$

Il primo vale :

0.3-(2^(-2))(1*2^-3+1*2^-4))=0.003125

(qui si prende il valore normalizzato , o meglio la parte della mantissa)

Il secondo invece :

(0.3-0.296875)/0.3 = 0.104

Mentre per quanto riguarda il numero corretto di cifre si fa il log:

-log(0.0104)=2 !!!

Vediamo ora un metodo che serve per costruire codici a distanza h >=3 : **codice di humming:** data una qualunque sequenza di bit da trasmettere, vi si associano n bit di parità , in modo che il messaggio venga codificato.

Posizione cifra		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Dato codificato		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Copertura bit di parità	p1	✓		✓		✓		✓		✓		✓		✓		✓				✓		...
	p2		✓	✓			✓	✓			✓	✓			✓	✓			✓	✓		
	p4				✓	✓	✓	✓					✓	✓	✓	✓					✓	
	p8								✓	✓	✓	✓	✓	✓	✓	✓						
	p16																✓	✓	✓	✓	✓	

Vediamo ora in dettaglio come funziona:

1. Ipotesi fondamentale :
   a. N+1<= 2^n-n
2. Bit di parità o controllo sono nelle posizioni delle potenze di 2 : 1,2,4,8 ecc ecc

Esempio :

1. Codificare 10011010
2. N=8 per trovare k si va a tentativi:
   a. K=3 non rispetta disequazione : 9<=7 FALSO
   b. K=4 rispetta disequazione : 9<=16-4 Vero -> prendiamo questo valore di k
   c. M=n+k -> 8+4=12
3. Si scrive il messaggio nel seguente modo includendo i bit di parità :
   a. **1*0001*1010 [^], dove gli "*" sono i bit di parità nelle posizioni delle potenze di 2, assumendo la cifra meno significativa a sinistra
4. Vediamo la rappresentazione in binario delle configurazioni:

1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100

Dopo mi fermo perché uso 12 bit massimo!

5. Dato che abbiamo 4 bit di parità dobbiamo trovare la parità di tutte le k parole : per ogni bit dobbiamo vedere il bit n-esimo settato ad "1" se presente :
   a. P1 = 1,3,5,7,9,11
      i. Di questa configurazione prendiamo i valori nella [^] corrispondenti , ignorando il primo numero in cui compare :
      ii. Nel nostro caso : 1+0+1+1+1
   b. P2= 2,3,6,7,10,11
      i. Nel nostro caso :1+0+1+0+1
   c. P4 = 4,5,6,7,12
      i. Nel nostro caso : 0+0+1+0
   d. P8 = 8,9,10,11,12 non vado oltre perché ho solo 12 bit
      i. Nel nostro caso : 1+0+1+0
   e. Di ognuna di queste configurazioni facciamo or tra gli addendi : se totale è pari il bit di parità vale 0 , altrimenti 1
      i. In questo esempio ho come sequenza 0110
   f. Assemblo il messaggio finale inserendo al posto di "*" i bit corrispondenti :

i. 011100101010