Lezione 1 - Note

Quanti bit sono necessari per rappresentare *N* informazioni?

Se ho un singolo bit, rappresento 2 informazioni: 0, 1.

Con due bit, 4 informazioni: 00, 01, 10, 11.
Ho due possibilità per il primo bit: 0, 1; per ognuna di quelle ho 2 possibilità per il secondo bit.
Quindi il totale è 2 x 2 = 4.

Con tre bit stessa cosa: 2 (primo bit) x 4 (altri bit) = 8.

Con n bit: 2 x 2 x 2…. n volte, cioè 2^n.

Allora, quanti bit mi servono se ho 100 informazioni da memorizzare?

Proviamo per tentativi. 2^4 = 16, non sono abbastanza. 2^10 = 1024, sono anche troppi.
2^7 = 128 sono abbastanza, mentre 2^6 = 64 no, quindi mi servono 7 bit.

E per non andare a tentativi? Devo trovare n per cui 2^n >= 100.

Elevamento a potenza **a^b = c** ha due operazioni inverse:
⇒ radice b-esima di c è a
⇒ il logaritmo in base a di c è b

Ci serve la seconda per trovare n: **2^n >= 100 ===> n >= logaritmo in base 2 di 100.**

Se non ho log_2 sulla calcolatrice non importa, uso le proprietà dei logaritmi: **log_2(x) = log(x) / log(2).**

Dato che **log(100)/log(2) = 6.64** mi servono **7** bit per **100** informazioni.

Come funzionano le tabelle di verità?

x	y	x e y	x o y	x xor y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Seguo questi passaggi:

- quali sono le variabili? x e y, allora metto le intestazioni delle colonne
- elenco tutte le possibilità per le variabili
- completo le righe una per volta, usando i valori delle variabili nella riga

Conversione di base da 16 a 10 o da 2 a 10

Basta usare il significato della notazione posizionale.

125_10 = 1 x 10^2 + 2 x 10^1 + 5 x 10^0

125_16 = 1 x 16^2 + 2 x 16^1 + 5 x 16^0 = 1 x 256 + 2 x 16 + 5 x 1 = 256 + 32 + 5 = 293_10

BA_16 = 11 x 16^1 + 10 x 16^0 = 11 x 16 + 10 x 1 = 176 + 10 = 186_10

BD_16 = 11 x 16^1 + 13 x 16^0 = 11 x 16 + 13 x 1 = 176 + 13 = 189_10

Conversione di base da 10 a 2

Posso usare il "metodo delle carte":

47 - 32 = 15 - 8 = 7 - 4 = 3 - 2 = 1 - 1 = 0

32	16	8	4	2	1
1	0	1	1	1	1

In alternativa, il metodo della divisione ripetuta:

1789 : 2	=	
894 : 2	=	resto 1
447 : 2	=	resto 0
223 : 2	=	resto 1
111 : 2	=	resto 1
55 : 2	=	resto 1
27 : 2	=	resto 1
13 : 2	=	resto 1
6 : 2	=	resto 1
3 : 2	=	resto 0
1 : 2	=	resto 1
0	=	resto 1

risultato: 11011111101 (resti dal basso in alto)

Perché funziona? Proviamo a pensare di fare le stesse operazioni, ma tutto in base 2:

11011111101 : 10_2	=	(nota che 10_2 = 1x2 + 0x1 = 2)
1101111110	=	resto 1

Dividere per 2 in base 2 è come dividere per 10 in base 10 (o per 16 in base 16): "spostare la virgola". Quindi dividendo per 2 rivelo le cifre in base 2!

Stessa cosa con la base 16. Il motivo per cui dividere per la base è spostare la virgola è questo:

abcd : 10_16	=	(nota che 10_16 = 1x16 + 0x1 = 16)
abc	=	resto d = 13

(a x 16^3 + b x 16^2 + c x 16^1 + d x 16^0) : 16 = a x 16^2 + b x 16^1 + c x 16^0 resto d

In alternativa, meglio convertire prima in base 2 e poi da 2 a 16.

Conversione tra basi 2 e 16?

Non conviene passare per la base 10, perché 16 = 2^4.

B_2 = 1011_2

10111011_2 = BB_16

1 x 2^7 + 0 x 2^6 + 1 x 2^5 + 1 x 2^4 + 1 x 2^3 + 0 x 2^2 + 1 x 2^1 + 1 x 2^0 ⇒ 2^4 x (1 x 2^3 + 0 x 2^2 + 1 x 2^1 + 1 x 2^0) +

01011001_2 = (0101)(1001) = 59_16

1001 = 8+1 = 9
8421

(0011)(0101) = 35_16

Operazione logiche tra numeri binari?

1101 0110 AND 0111 0101?

1101 0110 AND
0111 0101 =
0101 0100

1101 0110 XOR 0111 0101?

1101 0110 XOR
0111 0101 =
1010 0011

NOT 0111 0101?

0111 0101
1000 1010

0010 0110 OR
1101 1001 =
1111 1111

0010 0110 OR
0010 0110 =
0010 0110

0010 0110 XOR
0010 0110 =
0000 0000