MODULO 1. CONCETTI DI BASE DELL'ICT

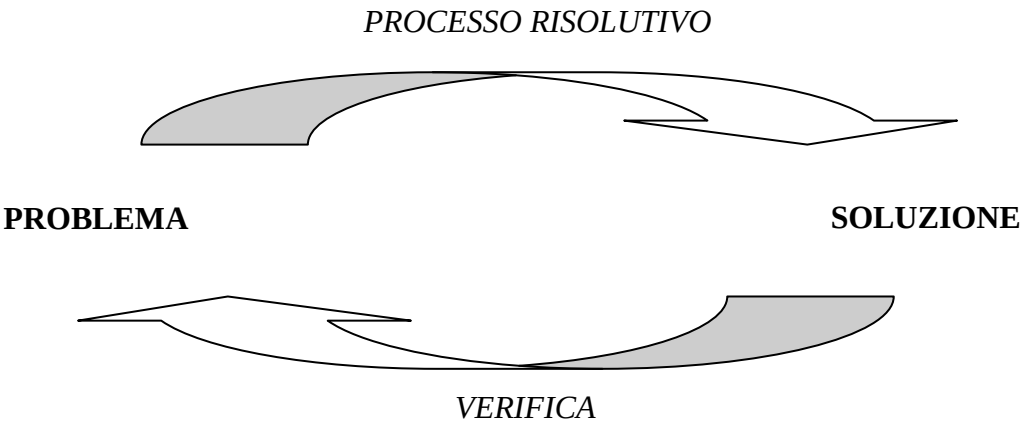
UNITA' DI APPRENDIMENTO 0: DAL PROBLEMA AL PROGRAMMA.

0.1 L'ALGORITMO

Un **problema** è un quesito (ossia una domanda) che attende una risposta detta **soluzione** del problema.

Per **processo risolutivo** si intende un insieme di passi da compiere per giungere alla soluzione del problema. Naturalmente deve esistere un modo di controllare l'esattezza della soluzione trovata al problema alla fine del processo risolutivo, ossia deve essere possibile effettuare la **verifica** della soluzione.

Schematicamente:



Per potere tentare di trovare una soluzione ad un problema quest'ultimo deve essere **correttamente formulato** ossia deve accadere che:

a) non deve apparire evidente che il problema non abbia soluzione;
b) deve esistere un criterio per la verifica del corretto raggiungimento degli obiettivi finali;
c) i dati iniziali devono essere completi.

Le figure coinvolte nel processo risolutivo sono due: il **risolutore** e **l'esecutore.**

IL RISOLUTORE è colui che definisce il processo risolutivo per risolvere il problema *Al risolutore spetta l'attività creativa, il processo risolutivo non dipende solo da capacità tecniche ma anche dalla cultura e dall'intuizione del risolutore.*

L'ESECUTORE è colui il quale esegue il processo risolutivo. L'esecutore può essere anche una persona. *All'esecutore spetta l'attività esecutiva del procedimento, ossia seguire il processo risolutivo descritto dal risolutore per giungere concretamente alla soluzione del problema*

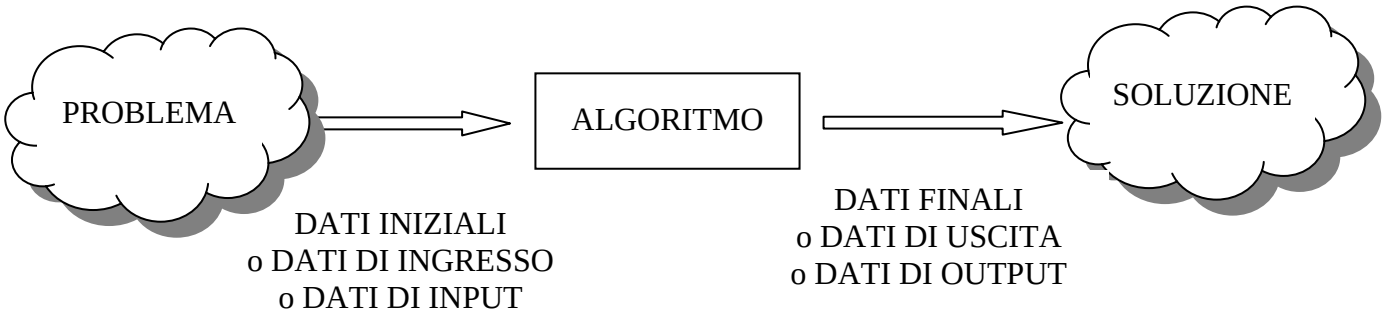
Per giungere alla **risoluzione di un problema** occorre effettuarne una sua corretta **formalizzazione** ossia:

- individuare tutte le informazioni di partenza ossia i dati iniziali;
- individuare tutte le informazioni di arrivo ossia lo scopo da raggiungere, la soluzione del problema (i dati finali);
- individuare il processo risolutivo per giungere dalle informazioni di partenza a quelle di arrivo;
- verificare la soluzione trovata.

Compito dell'**algoritmo** (che deriva dal nome di "al-Khwarizmi", importante matematico arabo del nono secolo al quale si deve l'uso del sistema di notazione decimale posizionale) è quello di formalizzare una serie di "passi" da percorrere o una serie di "azioni" da svolgere o una serie di "istruzioni" da eseguire per ottenere i risultati voluti e risolvere un problema (più precisamente una classe o tipologia di problemi) seguendo un processo risolutivo.

Definizione Con il termine ***algoritmo*** si intende la descrizione di un *processo risolutivo* che contiene la sequenza finita, ordinata ed univocamente interpretabile di *azioni elementari* da eseguire (dette *istruzioni* o *passi* dell'algoritmo) per risolvere una determinata classe di problemi.

Nella risoluzione di problemi con l'ausilio del computer il compito del *risolutore* viene svolto dall'**uomo** e quello di *esecutore* dal **computer**



Da subito, rispetto a questa definizione, possiamo precisare che:

(a) GLI ALGORITMI NON APPARTENGONO SOLO ALLA MATEMATICA
(nella vita quotidiana tutti noi eseguiamo operazioni ben note, che devono essere fatte seguendo una precisa sequenza di istruzioni, alla quale possiamo dare la valenza di algoritmo).

Esempi:

(b) NON BASTA UN ELENCO DI ISTRUZIONI PER AVERE UN ALGORITMO

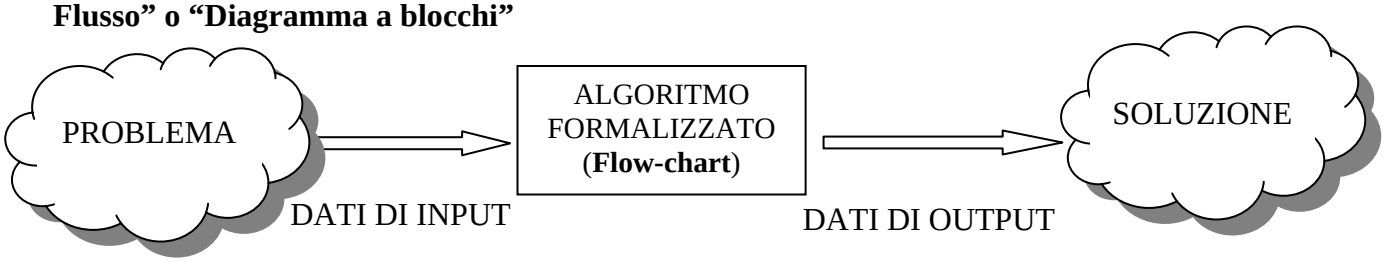
L'algoritmo deve infatti avere le seguenti caratteristiche: essere……..

SEQUENZIALE	Le istruzioni o passi dell'algoritmo vanno eseguite una dopo l'altra secondo la sequenza nella quale sono state stabilite.
NON AMBIGUO ossia UNIVOCO	Ogni passo o azione elementare deve essere univocamente interpretabile dall'esecutore
FINITO	Il processo risolutivo descritto dall'algoritmo deve cioè terminare dopo un numero finito di passi o azioni elementari e deve essere caratterizzato da un punto iniziale e da uno finale.
GENERALE	L'algoritmo va inteso come metodo di risoluzione non di un unico caso ma di una classe di problemi di uno stesso tipo
DETTAGLIATO	Ad ogni passo dell'algoritmo deve corrispondere un'azione elementare che l'esecutore è in grado di compiere
DETERMINISTICO	Deve risolvere il problema in tutti i suoi aspetti e sempre in tutti i casi. L'algoritmo partendo dalle stesse condizioni iniziali deve fornire sempre gli stessi risultati finali.
COMPLETO	L'algoritmo deve considerare tutti i casi possibili che si possono verificare durante l'esecuzione e per ogni caso indicare la soluzione da seguire.
OSSERVABILE NEI RISULTATI	Deve esserci un riscontro oggettivo dei risultati prodotti dall'algoritmo ossia la verifica deve essere testabile
LIMITATO NEL TEMPO	Anche se impiega molto tempo deve comunque avere un punto di arresto
EFFICIENTE	L'esecuzione dell'algoritmo deve comportare l'utilizzo, tendenzialmente, di meno risorse possibili

Per descrivere un algoritmo bisogna necessariamente utilizzare un linguaggio.

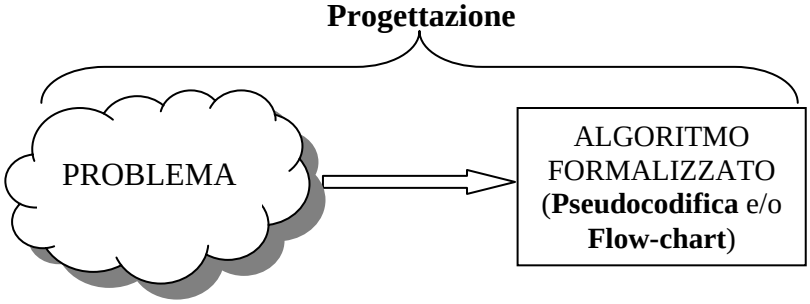
Non è possibile utilizzare il **linguaggio naturale** (quello normalmente parlato) per descrivere un algoritmo perché esso contiene ambiguità e doppi significati che confonderebbero l'esecutore.
Occorre utilizzare il **linguaggio formale** ossia un linguaggio rigorosamente definito in cui ogni simbolo, ogni comando abbia uno ed un solo significato.
Per descrivere un algoritmo, quindi, utilizzeremo due **linguaggi formali (**detti **linguaggi di progetto)**:
(*) il primo basato su una serie di frasi o comandi scritte in italiano che rappresentano le istruzioni dell'algoritmo che chiameremo **"Pseudocodifica"** :

(*) il secondo basato su particolari simboli grafici, connessi fra loro, che rappresentano le diverse azioni elementari descritte nell'algoritmo che prende il nome di **"Flow-chart" o "Diagramma di Flusso" o "Diagramma a blocchi"**

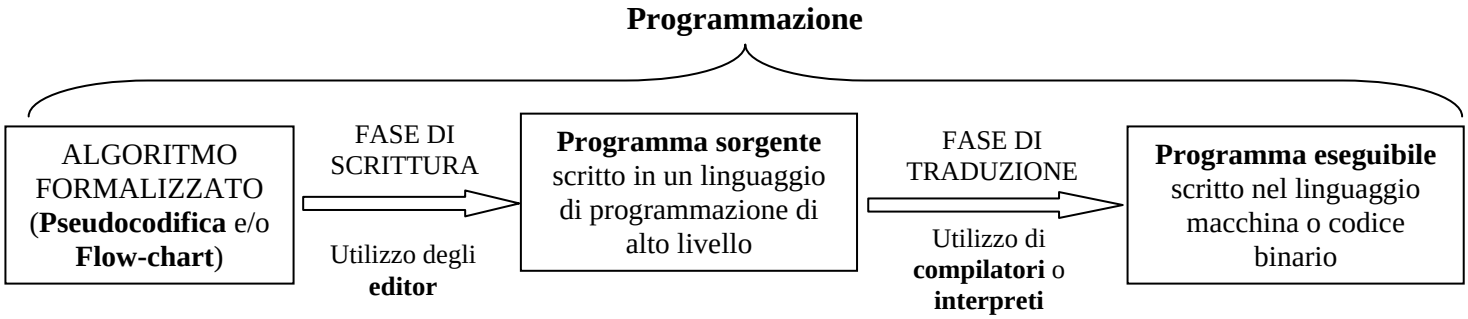


0.2 I LINGUAGGI

Per risolvere un problema attraverso l'uso di un computer abbiamo visto che la prima fase è quella di passare dal **problema** all'**algoritmo formalizzato** (ossia scritto usando la pseudocodifica o utilizzando il flow-chart).
Questa fase prende il nome di **fase di progettazione** o più brevemente **progettazione** dove si esaminano le possibili soluzioni del problema e dopo averle confrontate si sceglie quella considerata migliore.



La fase successiva consiste nel tradurre l'algoritmo formalizzato in una serie di istruzioni scritte in un linguaggio che la macchina (ossia il nostro esecutore) sia in grado di comprendere.
Questa fase prende il nome di **fase di programmazione** o più brevemente **programmazione** e può essere dettagliata come segue:



I **linguaggi di programmazione di alto livello** (esempio C, C++, Visual Basic, Java, etc.) sono linguaggi più vicini al linguaggio naturale le cui istruzioni sono indipendenti dalla macchina e permettono di descrivere l'idea che sta dietro all'operazione.
Quindi essi permettono di scrivere il **programma sorgente** che è la traduzione di un algoritmo formalizzato in un ben determinato linguaggio di programmazione di alto livello.
Ma il programma sorgente non è scritto ancora in un linguaggio comprensibile da un elaboratore.
Un elaboratore riconosce solo istruzioni scritte in codice binario (ossia utilizzando sequenze di 0 ed 1), il cosiddetto **linguaggio macchina,** ossia un linguaggio a basso livello dipendente dal tipo di macchina utilizzato.
Una volta i programmatori immettevano direttamente negli elaboratori le istruzioni in linguaggio macchina. L'operazione era ovviamente laboriosa e complicata, per cui nel tempo si sono sviluppati e diffusi i linguaggio di programmazione (ad alto livello) sempre più vicini al modo di ragionare del'uomo affiancandogli l'utilizzo di particolari programmi "traduttori" (**compilatori** ed **interpreti**) che svolgessero il difficile compito di tradurre ogni istruzione di alto livello in una sequenza binaria.
I **compilatori** traducono l'intero programma sorgente in una volta sola producendo un programma eseguibile ossia il programma in linguaggio macchina.
Gli **interpreti** traducono una istruzione alla volta del programma sorgente e la mandano direttamente in esecuzione.

PREMESSA: In un calcolatore tutte le informazioni (numeriche, caratteri, immagini) devono essere rappresentate in forma binaria o digitale (dall'inglese "digit" che significa "cifra") ossia utilizzando solo delle sequenze che utilizzano i due simboli 0 ed 1

0.3 LA RAPPRESENTAZIONE DEI DATI IN FORMA BINARIA

Sistema di numerazione decimale o in base 10

Il nostro sistema di numerazione viene chiamato **in base 10** o **decimale.**
Tutti i numeri che noi scriviamo sono sequenze finite di cifre scelte all'interno di un **alfabeto** ammesso costituto esattamente da **10 simboli** ossia dalle cifre **0, 1, 2, 3, ..** fino a **9**
Il nostro sistema di numerazione (in base 10) è un sistema di numerazione **posizionale** o **pesato** nel senso che una stessa cifra assume valori diversi ossia *pesi diversi* a seconda della *posizione* che essa occupa all'interno del numero.

Esempio:
*Nel numero **1415** che in base 10 leggiamo **"millequattrocentoquindici"** la prima cifra **"1"** che incontriamo a partire da destra verso sinistra ha un significato diverso dalla seconda cifra **"1"** incontrata. Infatti il primo "1" sta per **decine** mentre il secondo "1" indica le **migliaia.***
*Essi hanno quindi **pesi** diversi ed il peso dipende dalla posizione relativa della cifra. Tali pesi risultano sempre più significativi man mano che procediamo da destra verso sinistra.*

Il numero **1415** può come ben sappiamo essere anche espresso in base 10 con la seguente espressione:

$$\underbrace{1415}_{\text{Notazione compatta}} = \underbrace{1 \times 10^3 + 4 \times 10^2 + 1 \times 10^1 + 5 \times 10^0}_{\text{Notazione espansa}}$$

Le due notazioni prendono rispettivamente il nome di **notazione compatta** e **notazione espansa** del numero dato.

Sistema di numerazione binario o in base 2

Il sistema di numerazione binario o in base 2 è ovviamente anch'esso posizionale ed utilizza come alfabeto due simboli ossia le cifre **0, 1** chiamati **BIT** dall'inglese **binary digit** ossia **cifra binaria.**
Ciò significa che in un calcolatore qualunque numero (intero o con la virgola) viene rappresentato utilizzando stringhe (SEQUENZE) costituite dai soli simboli 0 ed 1.

N.B. Il **BIT** è un sottomultiplo del **BYTE** che è l'unità elementare di memorizzazione di una memoria di massa o centrale, composta da 8 bit. Poiché il byte è un'unità di memorizzazione molto piccola, per comodità, quando si descrive la capacità di memorizzazione di un disco rigido e della memoria principale, si devono usare i suoi multipli.

Nome	Simbolo	Multiplo Sottomultiplo	Fattore di scala	Numero di BYTE	Equivale
BIT	**b**	Sottomultiplo	2^{-3} B	1/8 B	1/8 B
BYTE	**B**	Unità di misura	2^{0} B	1 B	1 B
KILOBYTE	**KB**	Multiplo	2^{10} B	1024 B	1024 B
MEGABYTE	**MB**	Multiplo	2^{20} B	1.048.576 B	1024 KB
GIGABYTE	**GB**	Multiplo	2^{30} B	1.073.741.824 B	1024 MB
TERABYTE	**TB**	Multiplo	2^{40} B	1.099.511.627.776 B	1024 GB
PETABYTE	**PB**	Multiplo	2^{50} B	1.125.899.906.842.624 B	1024 TB
EXABYTE	**EB**	Multiplo	2^{60} B	1.152.921.504.606.846.976 B	1024 PB
ZETTABYTE	**ZB**	Multiplo	2^{70} B	1.180.591.620.717.411.303.424 B	1024 EB
YOTTABYTE	**YB**	Multiplo	2^{80} B	1.208.925.819.614.629.174.706.176 B	1024 ZB

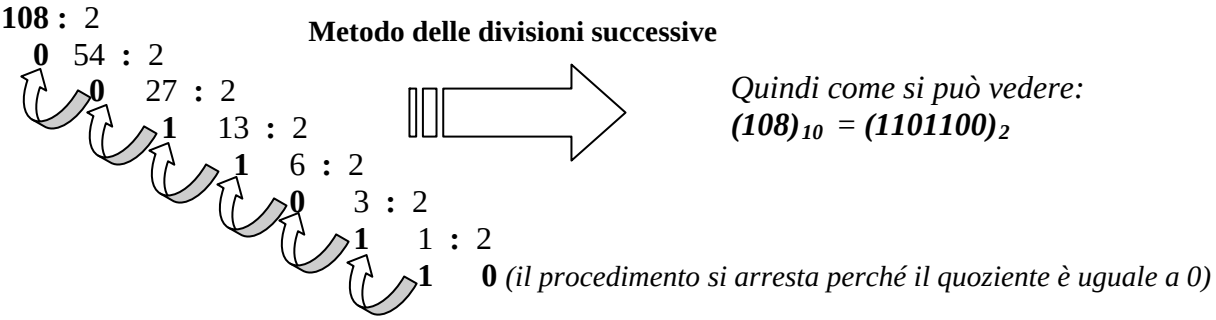
(*) Conversione di un numero da binario a decimale ossia da base 2 a base 10
Si parte dalla notazione compatta del numero in base 2 moltiplicando ciascuna cifra per il peso relativo e sommandone i parziali relativi, se ne calcola il totale.

Esempio
$(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (4 + 0 + 1)_{10} = (5)_{10}$

(*) Conversione di un numero da decimale a binario ossia da base 10 a base 2
Si parte dalla notazione compatta del numero in base 10 **dividendo successivamente per 2** (metodo delle divisioni successive) finchè il quoziente non diventa 0. A questo punto **la sequenza dei resti ottenuti letta dal basso verso l'alto** darà il numero decimale convertito in base 2.

Esempio
Convertire in base 2 il numero decimale $(108)_{10}$

Dobbiamo convertire in base 2 il numero decimale $(108)_{10}$



0.4 LA RAPPRESENTAZIONE DEI CARATTERI

Le informazioni rappresentate da un elaboratore possono essere non solo numeriche (sulle quali ci si possono fare le operazioni aritmetiche) ma anche *alfanumeriche* (numero di targa, codice fiscale, numero di telefono, cap, etc.).
Per rappresentare le informazioni alfanumeriche all'interno di un elaboratore è necessario far corrispondere a ciascuna di esse una determinata sequenza di bit.
Sono nati quindi svariati **codici** che permettono la trasformazione di simboli in sequenze di bit.

IMPORTANTE: Il massimo numero di simboli rappresentabili dipende dal numero di bit messi a disposizione per quel determinato tipo di codice.

Se avessimo un solo bit a disposizione (rappresentato dalla casellina), poiché i valori a disposizione del codice binario sono solamente due (le cifre binarie 0 ed 1), è possibile rappresentare solamente 2 simboli distinti da associare alle seguenti sequenze di bit

bit bit
| 0 | oppure | 1 |

Se avessimo due bit a disposizione (rappresentati da due caselline affiancate), poiché i valori a disposizione del codice binario sono solamente due (le cifre binarie 0 ed 1), è possibile rappresentare solamente 4 simboli distinti da associare alle seguenti sequenze di bit

bit bit
| 0 | 0 | oppure | 0 | 1 | oppure | 1 | 0 | oppure | 1 | 1 |

In generale se abbiamo **n** bit a disposizione il massimo numero di simboli distinti rappresentabili utilizzando le cifre binarie sarà pari a:

(numero di bit a disposizione) **n**

(numero di valori a disposizione) ossia **2**

I codici così creati permettono di associare a ciascuna combinazione distinta di bit una lettera dell'alfabeto, maiuscola, minuscola, un segno di punteggiatura, un segno speciale, una cifra numerica (intesa come carattere) etc.

I codici standard più conosciuti sono:

(*) **EBCDIC** (*Extended Binary Coded Decimal Interchange Code*): è un codice ad 8 bit che permette di rappresentare massimo $2^8 = 256$ simboli

(*) **ASCII** (*American Standard Code for Information Interchange*): è un codice inizialmente nato a 7 bit ($2^7 = 128$ simboli) poi esteso ad 8 bit ($2^8 = 256$ simboli). Tenendo presente che 8 bit costituiscono 1 BYTE ossia l'unità fondamentale di misura delle capacità della memoria di un elaboratore, ne consegue che un byte riusciamo a rappresentare un carattere.
Applicando il codice ASCII la parola "CANE" in un elaboratore sarà rappresentata dalla sequenza di 4 byte ciascuno rappresentante una singola lettera maiuscola:

C	**A**	**N**	**E**
01000011	01000001	01001110	01000101

(*) **UNICODE**: grazie all'introduzione di memorie sempre più capaci, è stato creato un codice inizialmente a 16 bit ($2^{16} = 65.536$ simboli) che si riteneva sufficiente a rappresentare tutti i simboli di tutte le lingue del mondo, anche di quelle che utilizzano gli ideogrammi. Ora lo standard UNICODE prevede una codifica a 21 bit ($2^{21} = 2.097.152$ simboli) sufficiente a codificare tutti gli scritti del patrimonio storico dell'umanità.

ESEMPIO: Tavola ASCII ad 8 bit

Standard

Byte	Cod.	Char
00000000	0	Null
00000001	1	Start of heading
00000010	2	Start of text
00000011	3	End of text
00000100	4	End of transmit
00000101	5	Enquiry
00000110	6	Acknowledge
00000111	7	Audible bell
00001000	8	Backspace
00001001	9	Horizontal tab
00001010	10	Line feed
00001011	11	Vertical tab
00001100	12	Form Feed
00001101	13	Carriage return
00001110	14	Shift out
00001111	15	Shift in
00010000	16	Data link escape
00010001	17	Device control 1
00010010	18	Device control 2
00010011	19	Device control 3
00010100	20	Device control 4
00010101	21	Neg. acknowledge
00010110	22	Synchronous idle
00010111	23	End trans. block
00011000	24	Cancel
00011001	25	End of medium
00011010	26	Substitution
00011011	27	Escape
00011100	28	File separator
00011101	29	Group separator
00011110	30	Record Separator
00011111	31	Unit separator

Byte	Cod.	Char
00100000	32	Spc
00100001	33	!
00100010	34	"
00100011	35	#
00100100	36	$
00100101	37	%
00100110	38	&
00100111	39	'
00101000	40	(
00101001	41	)
00101010	42	*
00101011	43	+
00101100	44	,
00101101	45	-
00101110	46	.
00101111	47	/
00110000	48	0
00110001	49	1
00110010	50	2
00110011	51	3
00110100	52	4
00110101	53	5
00110110	54	6
00110111	55	7
00111000	56	8
00111001	57	9
00111010	58	:
00111011	59	;
00111100	60	<
00111101	61	=
00111110	62	>
00111111	63	?

Byte	Cod.	Char
01000000	64	@
01000001	65	A
01000010	66	B
01000011	67	C
01000100	68	D
01000101	69	E
01000110	70	F
01000111	71	G
01001000	72	H
01001001	73	I
01001010	74	J
01001011	75	K
01001100	76	L
01001101	77	M
01001110	78	N
01001111	79	O
01010000	80	P
01010001	81	Q
01010010	82	R
01010011	83	S
01010100	84	T
01010101	85	U
01010110	86	V
01010111	87	W
01011000	88	X
01011001	89	Y
01011010	90	Z
01011011	91	[
01011100	92	\
01011101	93	]
01011110	94	^
01011111	95	_

Byte	Cod.	Char
01100000	96	`
01100001	97	a
01100010	98	b
01100011	99	c
01100100	100	d
01100101	101	e
01100110	102	f
01100111	103	g
01101000	104	h
01101001	105	i
01101010	106	j
01101011	107	k
01101100	108	l
01101101	109	m
01101110	110	n
01101111	111	o
01110000	112	p
01110001	113	q
01110010	114	r
01110011	115	s
01110100	116	t
01110101	117	u
01110110	118	v
01110111	119	w
01111000	120	x
01111001	121	y
01111010	122	z
01111011	123	{
01111100	124	|
01111101	125	}
01111110	126	~
01111111	127	Del

Estesa

Byte	Cod.	Char
10000000	128	Ç
10000001	129	ü
10000010	130	é
10000011	131	â
10000100	132	ä
10000101	133	à
10000110	134	å
10000111	135	ç
10001000	136	ê
10001001	137	ë
10001010	138	è
10001011	139	ï
10001100	140	î
10001101	141	ì
10001110	142	Ä
10001111	143	Å
10010000	144	É
10010001	145	æ
10010010	146	Æ
10010011	147	ô
10010100	148	ö
10010101	149	ò
10010110	150	û
10010111	151	ù
10011000	152	ÿ
10011001	153	Ö
10011010	154	Ü
10011011	155	ø
10011100	156	£
10011101	157	Ø
10011110	158	×
10011111	159	ƒ

Byte	Cod.	Char
10100000	160	á
10100001	161	í
10100010	162	ó
10100011	163	ú
10100100	164	ñ
10100101	165	Ñ
10100110	166	ª
10100111	167	º
10101000	168	¿
10101001	169	®
10101010	170	¬
10101011	171	½
10101100	172	¼
10101101	173	¡
10101110	174	«
10101111	175	»
10110000	176	▒
10110001	177	▒
10110010	178	▒
10110011	179	¦
10110100	180	¦
10110101	181	Á
10110110	182	Â
10110111	183	À
10111000	184	©
10111001	185	¦
10111010	186	¦
10111011	187	+
10111100	188	+
10111101	189	¢
10111110	190	¥
10111111	191	+

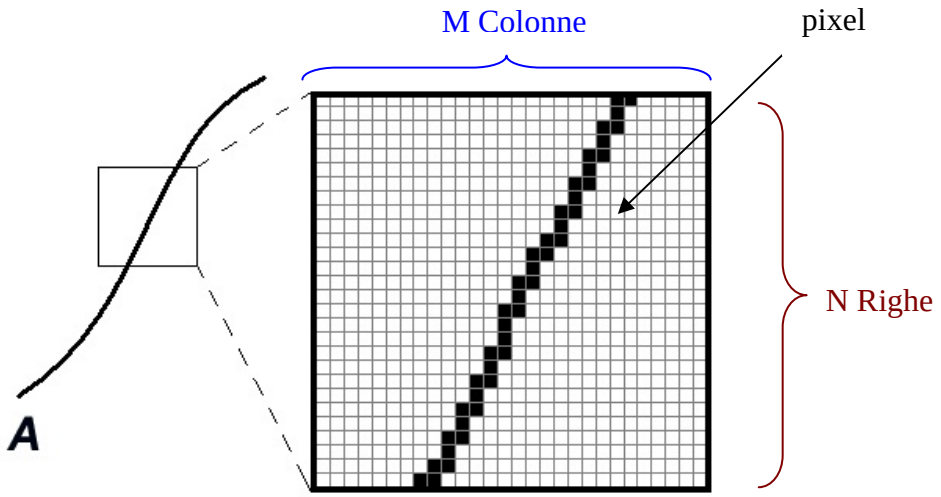
Byte	Cod.	Char
11000000	192	+
11000001	193	-
11000010	194	-
11000011	195	+
11000100	196	-
11000101	197	+
11000110	198	ã
11000111	199	Ã
11001000	200	+
11001001	201	+
11001010	202	-
11001011	203	-
11001100	204	¦
11001101	205	-
11001110	206	+
11001111	207	¤
11010000	208	ð
11010001	209	Ð
11010010	210	Ê
11010011	211	Ë
11010100	212	È
11010101	213	i
11010110	214	Í
11010111	215	Î
11011000	216	Ï
11011001	217	+
11011010	218	+
11011011	219	▒
11011100	220	▒
11011101	221	¦
11011110	222	Ì
11011111	223	▒

Byte	Cod.	Char
11100000	224	Ó
11100001	225	ß
11100010	226	Ô
11100011	227	Ò
11100100	228	õ
11100101	229	Õ
11100110	230	µ
11100111	231	þ
11101000	232	Þ
11101001	233	Ú
11101010	234	Û
11101011	235	Ù
11101100	236	ý
11101101	237	Ý
11101110	238	¯
11101111	239	´
11110000	240	-
11110001	241	±
11110010	242	‗
11110011	243	¾
11110100	244	¶
11110101	245	§
11110110	246	÷
11110111	247	¸
11111000	248	º
11111001	249	¨
11111010	250	·
11111011	251	¹
11111100	252	³
11111101	253	²
11111110	254	▒
11111111	255	

0.5 LA RAPPRESENTAZIONE DI UN'IMMAGINE DIGITALE

Un'immagine per essere interpretata ed elaborata da un computer deve essere *digitalizzata* ossia codificata opportunamente attraverso il codice binario dove ogni cosa è rappresentata da combinazioni di 0 ed 1.

Le immagini digitali sono essenzialmente di due tipi:
(*) **BITMAP**: l'immagine bitmap (dette anche immagini **raster** ossia "griglia") è rappresentata da una matrice con N righe ed M colonne di punti detti *pixel* (dall'inglese **pic**ture **el**ement).
Quindi ogni punto (come le caselle della battaglia navale, ma anche come le celle di un foglio elettronico) è individuato dalla posizione specifica (caratterizzata dalla coppia di valori costituita da numero di riga e numero di colonna) e dall'informazione che ne descrive il valore cromatico.
Quest'ultimo valore che indica il numero di colori o di livelli di grigio possibili per quel pixel (detto anche *profondità*) dipende da quanti bit sono utilizzati a tale scopo:
Esempio: un'immagine con 1 bit per pixel avrà massimo due combinazioni (0 ed 1) e quindi potrà massimo rappresentare due colori (bianco o nero), mentre per un'immagine con 4 bit per pixel si potranno rappresentare al massimo 2^4 = 16 colori o 16 livelli di grigio, per un'immagine con 8 bit per pixel si potranno rappresentare al massimo 2^8 = 256 colori o 256 livelli di grigio e così via.



N.B. Le immagini raster possono essere salvate in molti formati, più o meno compressi... (come anche non compressi) I formati ad uso comunque sono i seguenti: BMP, JPEG, PNG, GIF, TIFF, TGA, RAW etc.
Ogni formato ha le sue caratteristiche che lo rendono più o meno adatto ad un determinato uso... ma la caratteristica comune ai formati appena indicati è che appunto si tratta di formati raster... quindi insieme di punti e relativi colori

(*) **VETTORIALI**: le immagini vettoriali sono delle rappresentazioni di entità geometriche come un cerchio, un rettangolo o un segmento. Questi sono rappresentati da formule matematiche (un rettangolo è definito da due punti, un cerchio da un centro e un raggio, una curva da più punti e un'equazione). Sarà il processore ad essere incaricato di "tradurre" queste forme in informazioni interpretabili dalla scheda grafica.
Dato che un'immagine vettoriale è costituita unicamente da entità matematiche, è possibile applicarle facilmente delle trasformazioni geometriche (zoom, linearizzazione,…), mentre un immagine bitmap, fatta di pixel, non potrà subire tali trasformazioni se non perdendo delle informazioni, dette **distorsioni**.

Inoltre, le immagini vettoriali permettono di definire un'immagine con poche informazioni, il che rende i file decisamente poco voluminosi.

Parametro fondamentale per un'immagine è la **risoluzione** che si misura in *pixel per inch* (ossia **ppi**) perché indica il numero di pixel contenuti in un pollice quadrato (1 pollice o inch = 2,54 cm). Maggiore sarà questo numero, maggiore sarà la risoluzione e migliore risulterà la nitidezza del'immagine

0.6 I CONNETTIVI LOGICI

Nel lavoro di programmazione capita spesso di dovere ricorrere ai principi della logica degli enunciati ed occorre conoscere almeno alcuni concetti base dell'**algebra delle preposizioni** detta anche **algebra booleana** o di **Boole** dal matematico inglese George Boole (1815-1864).

Gli oggetti dell'algebra di Boole sono gli **enunciati.**

DEF: Si definisce **enunciato** una **preposizione** che può essere soltanto VERA (in inglese TRUE) oppure FALSA (in inglese FALSE) e non può mai essere né contemporaneamente entrambe le cose né indeterminata.
DEF: Con il termine **valore di verità di enunciato** si intende la sua verità oppure la sua falsità.

Gli enunciati possono essere **semplici** oppure **composti.**
DEF: Un. **enunciato composto** è formato da due o più sottoenunciati collegati tra loro attraverso appositi **connettivi logici.**
La **proprietà fondamentale** di un **enunciato composto** è che il suo valore di verità viene completamente definito dai valori di verità dei suoi sottoenunciati e dal connettivo logico che li unisce.

I principali connettivi logici sono:

A) CONGIUNZIONE logica (AND)

Il connettivo logico **AND** è un operatore **binario** (ossia agisce su due enunciati per crearne un altro) completamente definito dalla seguente **tavola di verità**

p	q	p AND q
V	V	V
V	F	F
F	V	F
F	F	F

Quindi l'**enunciato composto p AND q** risulta **VERO** solo nel caso in cui entrambi gli **enunciati semplici p** e **q** sono **VERI** mentre risulta **FALSO** in tutti gli altri casi.

B) DISGIUNZIONE logica (OR)

Il connettivo logico **OR** è un operatore **binario** (ossia agisce su due enunciati per crearne un altro) completamente definito dalla seguente **tavola di verità**

p	q	p OR q
V	V	V
V	F	V
F	V	V
F	F	F

Quindi l'**enunciato composto p OR q** risulta **FALSO** solo nel caso in cui entrambi gli **enunciati semplici p** e **q** sono **FALSI** mentre risulta **VERO** in tutti gli altri casi (ossia quando almeno uno degli enunciati semplici è VERO).

L'**enunciato composto p OR q** risulta **VERO** solo nel caso in cui i due **enunciati semplici p** e **q** hanno **valori di verità diversi** mentre risulta **FALSO** se i due **enunciati semplici p** e **q** hanno **valori di verità uguali.**

C) NEGAZIONE logica (NOT)
Il connettivo logico **OR** è un operatore **unario** (ossia agisce su un solo enunciato per crearne un altro) completamente definito dalla seguente **tavola di verità**

p	NOT p
V	F
F	V

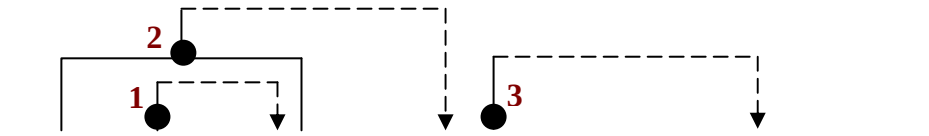
Quindi l'**enunciato composto NOT p** (detto anche **"negazione di p")** che risulta **VERO** se l'**enunciato semplice p** è **FALSO** mentre risulta **FALSO** se l'**enunciato semplice p** è **VERO.**

Le tavole di verità
Combinando in vario modo gli enunciati semplici del tipo p, q, r, etc. ed i connettivi logici AND, OR, e NOT si possono ottenere enunciati molto più complessi.
Il **valore di verità di un enunciato composto** è noto quando si conoscono i valori di verità delle sue variabili. Un modo semplice per conoscerlo è quello che prevede la *costruzione delle tavole di verità.*

Esempio:
Supponiamo di voler conoscere i valori di verità del seguente enunciato composto

NOT (p AND (NOT q))

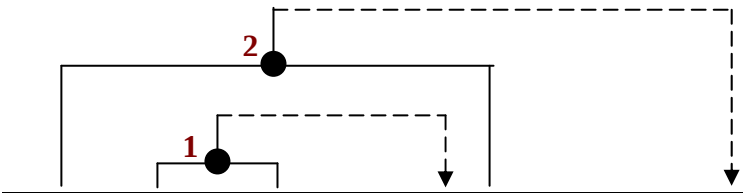
Costruiamo la tavola di verità seguendo i seguenti passi:
*- prima occorre individuare gli enunciati semplici contenuti nell'enunciato composto. Nel nostro caso **p** e **q**;*
*- poi occorre individuare gli enunciati composti partendo dall'enunciato composto più interno fino all'enunciato composto totale. Nel nostro caso **NOT q**, **(p AND (NOT q))** e **NOT(p AND (NOT q))**;*
- disegnare una tabella con tante colonne quanti sono gli elementi individuati nei primi due punti;
- applicare nell'ordine le tavole di verità dei connettivi logici fondamentali AND, OR e NOT.



p	*q*	*NOT q*	*p AND (NOT q)*	*NOT(p AND (NOT q))*
V	V	F	F	V
V	F	V	V	F
F	V	F	F	V
F	F	V	F	V

Esempio:
Supponiamo di voler conoscere i valori di verità del seguente enunciato composto

p OR (q AND r))



p	*q*	*r*	*q AND r*	*p OR (q AND r))*
V	**V**	**V**	**V**	**V**
V	**V**	**F**	**F**	**V**
V	**F**	**V**	**F**	**V**
V	**F**	**F**	**F**	**V**
F	**V**	**V**	**V**	**V**
F	**V**	**F**	**F**	**F**
F	**F**	**V**	**F**	**F**
F	**F**	**F**	**F**	**F**

L'equivalenza logica

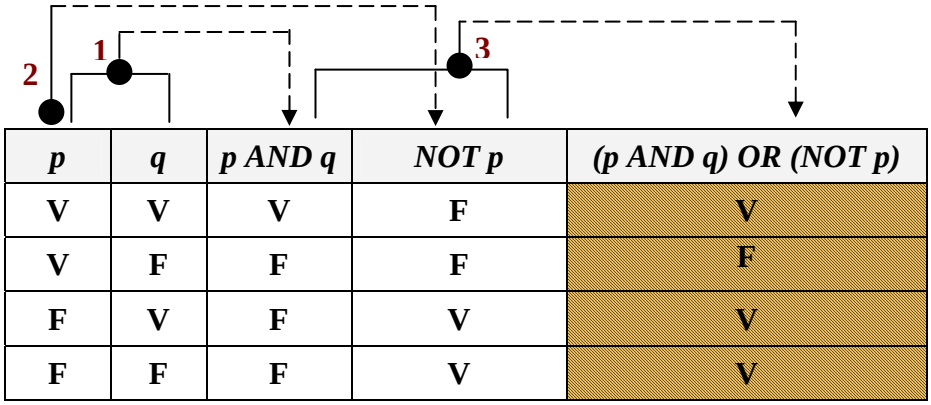
Definizione: Due enunciati semplici o composti sono **equivalenti** se hanno la medesima tavola di verità. L'equivalenza si in dica con il simbolo $\equiv$
Se due enunciati semplici o composti non sono equivalenti si indica con il simbolo $\not\equiv$

Esempio:
Proviamo che le i due enunciati composti

(p AND q) OR (NOT p)e ***(NOT p) OR q***

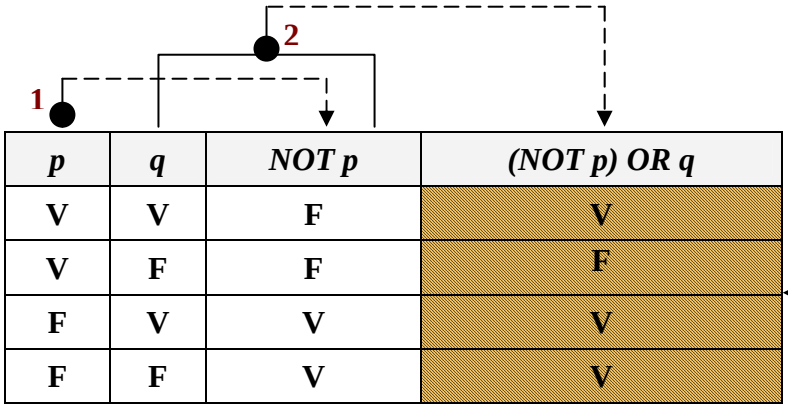
sono equivalenti.
Dobbiamo innanzitutto costruire le due tavole di verità con seguendo i passi prima specificati.

Tavola di verità di: ***(p AND q) OR (NOT p)***



p	*q*	*p AND q*	*NOT p*	*(p AND q) OR (NOT p)*
V	V	V	F	V
V	F	F	F	F
F	V	F	V	V
F	F	F	V	V

Tavola di verità di: ***(NOT p) OR q***



p	*q*	*NOT p*	*(NOT p) OR q*
V	V	F	V
V	F	F	F
F	V	V	V
F	F	V	V

Poichè i due enunciati composti hanno la stessa tavola di verità esse risultano essere equivalenti.

Quindi possiamo scrivere che ***(p AND q) OR (NOT p)*** $\equiv$ ***(NOT p) OR q***

Esercizio:
Siano p = (a > 2) e q = (b = 4) due enunciati semplici.
Dopo avere scritto la tavola di verità degli enunciati composti:
p AND q, p OR q, NOT p, NOT q dire quale valore di verità si ottiene per le seguenti terne di valori:
i) a = 2 , b = 3
ii) a = 3, b = 4
iii) a = 1, b = 4
iv) a = 4, b = 2