



Codifica dell'informazione



Introduzione
all'informatica

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Analogico e digitale

- Una grandezza (fisica o astratta) può essere rappresentata in due forme
 - Analogica:** insieme di valori **continuo** (*denso e "senza buchi"*)
 - Digitale** (o numerica): insieme di valori **discreto** (*tutti i punti sono isolati*)



www.ce.unipr.it/people/tomamic

2/62

Approssimazione discreta

- Alcune informazioni sono intrinsecamente discrete
 - Informazioni "artificiali", es. un testo scritto
 - Scala atomica o subatomica ...
- Molte grandezze fisiche hanno forma continua
 - Per loro elaborazione al calcolatore: rappresentazione digitale
 - Approssimazione** del valore analogico
 - Errore dipende dalla precisione della rappresentazione digitale scelta

Codice

- Sistema basato su simboli, che permette la rappresentazione dell'informazione
- Simbolo:** elemento atomico
- Alfabeto:** insieme dei simboli possibili (A)
- Cardinalità** del codice: numero di simboli dell'alfabeto
- Stringa:** sequenza di simboli ($s \in A^*$)
- Linguaggio:** insieme stringhe ben formate ($L \subseteq A^*$)

www.ce.unipr.it/people/tomamic

3/62

www.ce.unipr.it/people/tomamic

4/62



Codice posizionale

- Un numero naturale può essere rappresentato con una notazione posizionale
- $N = c_0 \cdot \text{base}^0 + c_1 \cdot \text{base}^1 + \dots + c_n \cdot \text{base}^n$
- Sistemi di numerazione posizionali di uso comune
 - Decimale (base 10; c: 0-9)
 - Binario (base 2; c: 0-1)
 - Esadecimale (base 16; c: 0-9, A-F)



Numeri binari

Codice binario

- Base 2; c: 0-1
- Informazione digitale nei calcolatori rappresentata con una sequenza di 0 e 1
 - Konrad Zuse, ~1940
- Ogni elemento di una sequenza binaria viene detto **bit**
- Una sequenza di 8 bit viene detta **byte**

A photograph of handwritten binary addition on lined paper. The problem is 0111 + 1110 = 10101. A pen is shown pointing to the result.

$$\begin{array}{r}
 0111 \\
 1110 \\
 \hline
 10101
 \end{array}$$

Codifica dell'informazione

- Codifica: regole di corrispondenza per passare da un certo codice ad un altro
- Corrispondenza biunivoca
 - Tra una stringa di un codice
 - E una stringa di un altro codice
- Ad una certa stringa in un alfabeto ricco di simboli, corrisponde una stringa più lunga in un alfabeto più ridotto

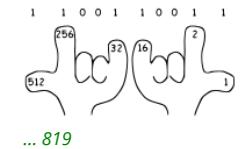
Codifica decimale → binaria

- (1) Dividere il numero decimale per 2
- (2) Assegnare il resto come valore del bit (*loop*)
- Ossia continuare a dividere per 2 il quoziente, finché non si annulla
- Es.: $35_{10} = 00100011_2$

n	n/B	n%B	peso
35	17	1	2^0
17	8	1	2^1
8	4	0	2^2
4	2	0	2^3
2	1	0	2^4
1	0	1	2^5

Numeri naturali

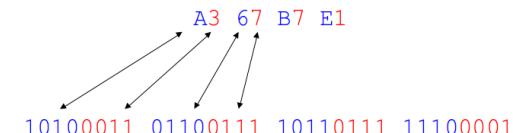
- Rappresentare un numero naturale **N** in forma binaria
- Occorrono **K** bit, t.c. $2^K > N$
- Es. 4 bit per numeri naturali da 0 a 15
- Un calcolatore assegna un numero fisso di bit per diversi tipi di informazione
 - Casi di valori non rappresentabili
 - Overflow, underflow**



Esadecimale (Hex)

Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex	Dec
0000 0000	00	000	0001 0000	10	016	0010 0000	20	032
0000 0001	01	001	0001 0001	11	017	0010 0001	21	033
0000 0010	02	002	0001 0010	12	018	0010 0010	22	034
0000 0011	03	003	0001 0011	13	019	0010 0011	23	035
0000 0100	04	004	0001 0100	14	020	0010 0100	24	036
0000 0101	05	005	0001 0101	15	021	0010 0101	25	037
0000 0110	06	006	0001 0110	16	022	0010 0110	26	038
0000 0111	07	007	0001 0111	17	023	0010 0111	27	039
0000 1000	08	008	0001 1000	18	024	0010 1000	28	040
0000 1001	09	009	0001 1001	19	025	0010 1001	29	041
0000 1010	0A	010	0001 1010	1A	026	0010 1010	2A	042
0000 1011	0B	011	0001 1011	1B	027	0010 1011	2B	043
0000 1100	0C	012	0001 1100	1C	028	0010 1100	2C	044
0000 1101	0D	013	0001 1101	1D	029	0010 1101	2D	045
0000 1110	0E	014	0001 1110	1E	030	0010 1110	2E	046
0000 1111	0F	015	0001 1111	1F	031	0010 1111	2F	047

Bin ↔ Hex



- Ogni gruppo di 4 bit: 16 configurazioni diverse ($2^4 = 16$)
- Ciascuna combinazione corrisponde ad uno dei 16 simboli esadecimali

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Somma e sottrazione

$$\begin{array}{r} 1 \ 1 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 + \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 = \\ \hline 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \end{array}$$

BINARY

$$\begin{array}{r} 0 \ 1 \ 0 \\ 1 \ 1 \ 1 \ 0 - \\ 0 \ 1 \ 0 \ 1 = \\ \hline 1 \ 0 \ 0 \ 1 \end{array}$$

BINARY

Attenzione a riporto e prestito (in alto)

Moltiplicazione

$$\begin{array}{r} 1 \ 0 \ 1 \ 1 \times \\ 1 \ 1 \ 0 \ 1 = \\ \hline 1 \ 0 \ 1 \ 1 + \\ 0 \ 0 \ 0 \ 0 \\ \hline 0 \ 1 \ 0 \ 1 \ 1 + \\ 1 \ 0 \ 1 \ 1 \\ \hline 1 \ 1 \ 0 \ 1 \ 1 \ 1 + \\ 1 \ 0 \ 1 \ 1 \\ \hline 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \end{array}$$

BINARY

Divisione

$$\begin{array}{r} 1 \ 0 \ 1 \ 1 \ 0 \ 1 : 1 \ 1 \\ 0 \ 0 \quad \cdots \cdots \\ \hline 0 \ 1 \ 1 \ 1 \ 1 \\ 1 \ 0 \ 1 \quad \cdots \cdots \\ 1 \ 1 \\ \hline 1 \ 0 \ 1 \quad \cdots \cdots \\ 1 \ 1 \\ \hline 1 \ 0 \ 0 \quad \cdots \cdots \\ 1 \ 1 \\ \hline 0 \ 0 \end{array}$$

BINARY

Numeri interi

- Occorre rappresentare anche i numeri negativi
 - Necessario riservare un bit per il segno
 - Ovvero, si dimezza il massimo modulo ammesso
- Modulo e segno**
 - Il primo bit indica il segno
 - 0 positivo, 1 negativo

Complemento a due

- Rappresentazione alternativa, *diversa da modulo e segno!*
- Numero negativo, ottenuto dal suo opposto positivo
 - Complemento il numero (cambio gli 1 con 0 e viceversa)
 - Sommo 1
- Anche così, il primo bit indica il segno
 - 0 positivo, 1 negativo
- Attenzione:** bisogna conoscere codifica e num bit
 - Esempi seguenti: ogni intero con segno memorizzato in un singolo byte

Binary	Hex	Decimal	
		US	S
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	-8
1001	9	9	-7
1010	A	10	-6
1011	B	11	-5
1100	C	12	-4
1101	D	13	-3
1110	E	14	-2
1111	F	15	-1

Es. numero intero

- Avendo un byte, +35 è in binario: **00100011**
- Numero -35, in modulo e segno: **10100011**
- Numero -35, in complemento a due: **11011101**

0 0 1 0 0 0 1 1 -

1 1 0 1 1 1 0 0 +
1 =

1 1 0 1 1 1 0 1
-: complemento semplice, bit a bit

BINARY

Somma con segno

- Sommare 12 e -35 su 8 bit, modulo e segno
 - Sottrazione tra 35 e 12
 - Cambio di segno
- Stessa operazione, complemento due
 - Semplice somma: **12 + -35 = -23**

0 0 0 0 1 1 0 0 +
1 1 0 1 1 1 0 1 =

1 1 1 0 1 0 0 1

BINARY

Numeri reali

- Insieme continuo, per grandezze analogiche
 - Rappresentabili solo in modo approssimato
- Parte frazionaria:
 - $F = c_{-1} \cdot \text{base}^{-1} + \dots + c_{-n} \cdot \text{base}^{-n}$
- Due rappresentazioni *alternative*
 - **Virgola fissa:** segno, parte intera, parte decimale
 - **Virgola mobile:** segno, mantissa, esponente

Parte frazionaria in binario

- Moltiplicare la parte frazionaria per 2
 - Assegnare la parte intera del risultato come valore del bit (*loop*)
 - Ossia: continuare a moltiplicare per 2 la parte frazionaria del risultato... finché non si annulla

fract	fract*B	int	peso
0,375	0,750	0	2^{-1}
0,750	1,500	1	2^{-2}
0,500	1,000	1	2^{-3}

 www.ce.unipr.it/people/tomamic

21/61

Virgola fissa

- Numero espresso come: $r = (i, f)$
 - i e f sono in base p
 - i è la parte intera, n_1 bit
 - f è la parte frazionaria, n_2 bit
 - Precisione costante lungo l'asse reale



Virgola mobile

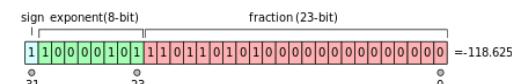
- Numero espresso come: $r = m \cdot b^n$
 - m e n sono in base p
 - m è la mantissa (numero frazionario con segno), n_1 bit
 - b è la base della notazione esponenziale
 - n è la caratteristica (intero), n_2 bit
 - Precisione variabile lungo l'asse reale



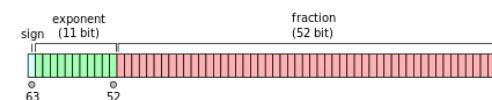
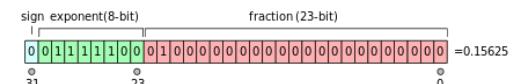
 www.ce.unipr.it/people/tomamic

23/61

IEEE 754



- $118.625 = 11101110.101_2 = 1.1101110101_2 \times 2^6$
 - All'esponente bisogna sommare sempre 127 ($= 2^8 - 1 - 1$)



 www.ce.unipr.it/people/tomamico

24/63

Algebra di Boole

- L'Algebra di Boole è un formalismo che opera su variabili (dette **variabili booleane**)
- Le variabili booleane possono assumere due soli valori: **vero**, **falso**
- Sulle variabili booleane è possibile definire delle funzioni (dette **funzioni booleane**)
- Anche le funzioni booleane possono assumere solo i due valori **vero** e **falso**

Algebra di Boole

Funzione e tabella di verità

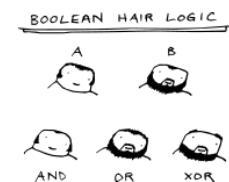
- Una **tabella di verità** permette di definire una **funzione booleana**
- Valore risultante per ciascuna combinazione dei valori in ingresso
- A volte, **specifica incompleta** (certe combinazioni di ingressi non possono verificarsi) → Non è specificato alcun valore

A	B	C	F_1
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Espressione booleana

- Algebra di Boole: basata su un insieme di operatori
- Possono essere combinati in espressioni
- Altra forma di definizione di funzioni booleane
- Es. $F_2(A, B, C) = A \cdot B + C$

Operatore	Simbolo
And	\cdot (\wedge)
Or	$+$ (\vee)
Not	\neg
Xor	\oplus
Nand	\uparrow
Nor	\downarrow



Operatori principali

A	B	$A \cdot B$	$A + B$	$A \oplus B$	$A \uparrow B$	$A \downarrow B$
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	1	0
1	1	1	1	0	0	0

A	$\neg A$
0	1
1	0

Proprietà degli operatori

Proprietà	Not	
Complemento	$\neg \neg A = A$	
Proprietà	And	Or
Commutativa	$A \cdot B = B \cdot A$	$A + B = B + A$
Associativa	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	$(A+B)+C = A+(B+C)$
Distributiva	$A + (B+C) = (A+B) \cdot (A+C)$	$A \cdot (B+C) = (A \cdot B) + (A \cdot C)$
Idempotenza	$A \cdot A = A$	$A + A = A$
Identità	$A \cdot 1 = A$	$A + 0 = A$
Del limite	$A \cdot 0 = 0$	$A + 1 = 1$
Assorbimento	$A \cdot (A+B) = A$	$A + (A \cdot B) = A$
Inverso	$A \cdot \neg A = 0$	$A + \neg A = 1$
De Morgan	$\neg(A \cdot B \cdot C \dots) = \neg A + \neg B + \neg C \dots$	$\neg(A+B+C \dots) = \neg A \cdot \neg B \cdot \neg C \dots$

Attenzione a De Morgan: errore comune!

Forme canoniche

- Somma di Prodotti (SP):** si considerano le righe a 1
 - $F_1(A, B, C) = (\neg A \cdot \neg B \cdot \neg C) + (\neg A \cdot B \cdot C) + (A \cdot \neg B \cdot C) + (A \cdot B \cdot \neg C) + (A \cdot B \cdot C)$
- Prodotto di Somme (PS):** si considerano le righe a 0
 - $F_1(A, B, C) = (A + B + \neg C) \cdot (A + \neg B + C) \cdot (\neg A + B + C)$

A	B	C	F_1	→ Forma canonica...
0	0	0	1	→ SP
0	0	1	0	→ PS
0	1	0	0	→ PS
0	1	1	1	→ SP
1	0	0	0	→ PS
1	0	1	1	→ SP
1	1	0	1	→ SP
1	1	1	1	→ SP

Operazioni bit a bit in Python

```
x, y, z, shift = 0, 0, 0, 0 # some int values
x << shift # x = x * (2^shift)
x >> shift # x = x / (2^shift), con segno
x & y      # AND applicato bit a bit
x | y      # OR applicato bit a bit
x ^ y      # XOR bit a bit
~x         # complemento di ogni bit

z = 0x0B    # hex value (11 dec)
z = 0b1011  # bin value (11 dec)

hex(11)     # '0x0b' (text)
bin(0x0B)   # '0b1011' (text)
```

PYTHON

Da non confondere con operatori logici (and, or, not)



DII

Caratteri e testo

- Necessaria convenzione per codifica numerica (binaria) dei caratteri
- Codifica **ASCII** (American Standard Code for Information Interchange) a 7 bit
 - Caratteri alfanumerici*: lettere maiuscole, minuscole, numeri, spazio
 - Simboli e punteggiatura*: @, #, ...
 - Caratteri di controllo* (non tutti visualizzabili): TAB, LF, CR, BELL ecc.

Caratteri e testo

Tabella ASCII di base

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	(NULL)	32	20	[SPACE]	64	40	@	96	60	'
1	1	(START OF HEADING)	33	21	!	65	41	A	97	61	a
2	2	(START OF TEXT)	34	22	“	66	42	B	98	62	b
3	3	(END OF TEXT)	35	23	#	67	43	C	99	63	c
4	4	(END OF TRANSMISSION)	36	24	\$	68	44	D	100	64	d
5	5	(EOT)	37	25	%	69	45	E	101	65	e
6	6	(ACKNOWLEDGE)	38	26	&	70	46	F	102	66	f
7	7	(BELL)	39	27	,	71	47	G	103	67	g
8	8	(BACKSPACE)	40	28	(72	48	H	104	68	h
9	9	(HORIZONTAL TAB)	41	29)	73	49	I	105	69	i
10	A	(LINE FEED)	42	2A	*	74	4A	K	106	6A	j
11	B	(VERTICAL TAB)	43	2B	+	75	4B	L	107	6B	k
12	C	(FORM FEED)	44	2C	,	76	4C	M	108	6C	l
13	D	(CARRIAGE RETURN)	45	2D	.	77	4D	N	109	6D	m
14	E	(SHIFT OUT)	46	2E	-	78	4E	O	110	6E	n
15	F	(SHIFT IN)	47	2F	/	79	4F	P	111	6F	o
16	10	(DEVICE ESCAPE)	48	30	0	80	50	Q	112	70	p
17	11	(DEVICE CONTROL 1)	49	31	1	81	51	R	113	71	q
18	12	(DEVICE CONTROL 2)	50	32	2	82	52	S	114	72	r
19	13	(DEVICE CONTROL 3)	51	33	3	83	53	T	115	73	s
20	14	(DEVICE CONTROL 4)	52	34	4	84	54	U	116	74	t
21	15	(NEGATIVE ACKNOWLEDGE)	53	35	5	85	55	V	117	75	u
22	16	(SYNCHRONOUS IDLE)	54	36	6	86	56	W	118	76	v
23	17	(END OF TRANS. BLOCK)	55	37	7	87	57	X	119	77	w
24	18	(CANCEL)	56	38	8	88	58	Y	120	78	x
25	19	(END OF MEDIUM)	57	39	9	89	59	Z	121	79	y
26	1A	(SUBSTITUTE)	58	3A	:	90	5A	[122	7A	z
27	1B	(ESCAPE)	59	3B	:	91	5B	\	123	7B	{
28	1C	(FIELD SEPARATOR)	60	3C	<	92	5C	^	124	7C	}
29	1D	(GROUP SEPARATOR)	61	3D	=	93	5D	_	125	7D	j
30	1E	(RECORD SEPARATOR)	62	3E	>	94	5E	~	126	7E	~
31	1F	(UNIT SEPARATOR)	63	3F	?	95	5F	-	127	7F	{DEL}

Interruzione di riga

- Unix: **LF**
 - Multics, Unix etc., Mac OS X, BeOS, Amiga, RISC OS
- Windows: **CR+LF**
 - Most early OSes, DOS, OS/2, Windows, Symbian
- Vecchi Apple: **CR**
 - Commodore machines, Apple II family, Mac OS up to version 9

Tabella ASCII estesa

- Caratteri accentati + caratteri per grafici
 - [Code Page 437](#) per PC (DOS) in Nord America
 - Possibile mischiare testo in inglese e francese (anche se in Francia [CP850](#); ma non assieme greco ([CP737](#)), russo ecc.
- [ISO 8859](#), estensioni standard per ASCII ad 8 bit
 - ISO 8859-1 (o Latin1): Lingue dell'Europa Occidentale
 - ISO 8859-2: Lingue dell'Europa Orientale
 - ISO 8859-5: Alfabeto cirillico
 - ISO 8859-15: Latin1 con simbolo euro (€)



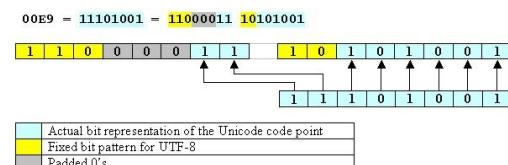
Unicode

- Unicode associa un preciso **code-point** (32 bit) a ciascun simbolo
 - Possibile rappresentare miliardi di simboli
 - Primi 256 code-point = [Latin1](#)
- Attualmente >30 sistemi di scrittura
 - Rappresentazione di [geroglifici](#) e caratteri [cuneiformi](#)
 - Proposta per [Klingon](#) (da Star Trek... rifiutata!)



UTF-8, UTF-16

- Unicode Transformation Format:** codifica di un **code-point** in una sequenza di bit (uno o più **code-unit**)
 - [UTF-32](#) – code-unit di 32-bit, lunghezza fissa
 - [UTF-16](#) – code-unit di 16-bit, lunghezza variabile
 - [UTF-8](#) – code-unit di 8-bit, ma lunghezza variabile (1-4 byte), max compatibilità con ASCII



String	abcdሴݞ
Char Count	7
Character Count	6
UTF-8 Byte Count	11

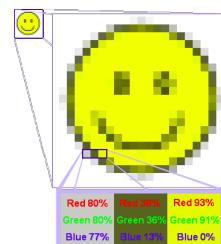


DII

Immagini

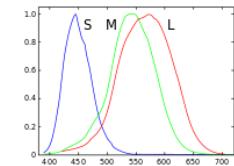
Immagini raster

- **Digitalizzazione:** da immagine a sequenza binaria
- **Immagine raster** suddivisa in una griglia di punti (**pixel**)
 - Ogni pixel descritto da un codice, che ne individua il colore
- **Profondità:** # bit per rappresentare il colore di un pixel
 - 1, 2, 8, 12, 16, 24, 32... bit per pixel (**bpp**)
 - Es. 8 bit per 256 (2^8) possibili colori
 - Colore diretto o indicizzato da una **palette**
- **Risoluzione:** # punti per pollice (**dpi**), come in tipografia
 - Spesso (ma non sempre), risoluzione orizzontale uguale a verticale



Modelli di colore

- Occhio sensibile a variaz. luminosità
 - 6 mln di coni, 125 di bastoncelli
- **RGB:** rosso, verde, blu
 - 8 bit: 3 bit × R e G, 2 × B
 - 24 bit: 8 bit × R, G e B
 - 32 bit: canale alpha
- **YUV:** luminosità, crominanza di R e B
 - Sistema PAL, MPEG
 - TV a colori (compat. B&W)
- **HSB:** tonalità, saturazione e luminosità



Formati di file grafici

- **BMP:** immagine (normalmente) non compressa
- **TIFF, PNG:** comprimono l'immagine, per ridurne l'occupazione, senza deteriorarla (compressione *lossless*)
- **JPEG:** comprime (molto di più), ma deteriora l'immagine (compressione *lossy*)

Formato BMP

FILE INFO HEADER (14)
2 **Tipo** file (= "BM")
4 **Dim.** file (**in byte**)
4 **Riservato**
4 **Offset** immagine (**in byte**)
BITMAP INFO HEADER (40)
4 **Dimensione** struttura
4+4 **Larghezza** e altezza immagine
2 **Piani** (non usato)
2 **# bit per pixel**
4+4 **Compressione** e dim. img (0 senza compressione)
4+4 **Risoluzione** orizz. e vert. (pixel per metro)
4+4 **# colori in palette e # colori importanti**
Palatte (RGBQUAD)
4 **Blue, Green, Red, Riservato**



Es. Redbrick.BMP

BM*	File size	WxH=32x32	Inizio img	Profondità (bpp)	40
0000	42 4d 76 02 00 00 00 00 00 00 00 00 00 00 00 00	76 00 00 00 00 28 00			
0010	00 00 20 00 00 00 00 20 00 00 00 00 00 00 00 00	00 01 00 04 00 00 00 00			
0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 80 00 00 00 00 00 00 00 00 00 00			
0040	00 00 00 80 80 00 80 00 00 00 00 00 00 00 00 00	80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
0050	00 00 80 80 80 00 00 00 00 00 00 00 00 00 00 00	c0 00 00 00 00 ff 00 00 00 00 00 00 00 00 00 ff ff			
0060	00 00 00 ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00	ff 00 00 ff ff ff 00 00 00 00 00 00 00 00 ff ff ff			
0070	00 00 ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
0090	00 00 00 00 00 00 00 00 11 11 01 19 11 01 10 10 09 09				
00a0	01 09 11 11 01 90 11 01 19 09 09 91 11 10 09 11				
00b0	09 11 19 10 90 11 19 01 19 19 10 10 11 10 09 01				
00c0	91 10 91 09 10 10 90 99 11 11 11 11 19 00 09 01				
00d0	91 01 01 19 00 99 11 10 11 91 99 11 09 90 09 91				
00e0	01 11 11 11 91 10 09 19 01 00 11 90 91 10 09 01				
00f0	11 99 10 01 11 11 91 11 11 19 10 11 99 10 09 10				
0100	01 11 11 11 11 19 10 11 09 09 10 19 10 10 10 09 01				
...					

Es. Redbrick.BMP



Grafica vettoriale

- Immagine: insieme di primitive geometriche
 - Linee, poligoni..., colori, sfumature...
 - ▲ Qualità, a varie risoluzioni
 - ▲ Compressione dati
 - ▲ Gestione modifiche
 - ▼ Non intuitiva per alcuni
 - ▼ Possibilmente onerosa
 - ▼ Risorse non note a priori



Grafica vettoriale

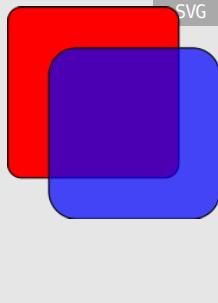
- **Applicazioni:** editoria (DTP), video-editing, architettura, ingegneria, grafica 3D (CAD), font vettoriali (caratteri scalabili in dimensione senza perdere definizione)
 - **Formati esistenti:** PS (PostScript), PDF (Portable Document Format), WMF (Windows MetaFile), DXF (AutoCAD), CDR (CorelDraw), SWF (Flash), SVG (Scalable Vector Graphics, per il web)



Esempio di file SVG

<!-- possibly inside an HTML5 file -->

```
<svg width="800" height="600">  
  
<rect x="80" y="60" width="250" height="250" rx="20"  
fill="#ff0000" stroke="#000000" stroke-width="2" />  
  
<rect x="140" y="120" width="250" height="250" rx="40"  
fill="#0000ff" stroke="#000000" stroke-width="2"  
fill-opacity="0.7" />  
  
</svg>
```



Audio digitale

www.ce.unipr.it/people/tomamic

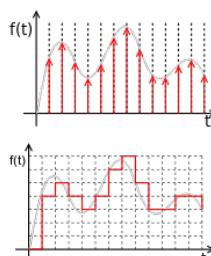
49/62

www.ce.unipr.it/people/tomamic

50/62

Audio digitale

- Grandezza analogica → discretizzazione
 - **Campionamento** (*sampling*) nel tempo
 - **Quantizzazione** (*quantizing*) nelle ampiezze
- Qualità CD: 44 kHz, 16bit
- Spettro udibile: 20-20k Hz, Nyquist-Shannon



Formato WAV

The Canonical WAVE file format

Endian	File offset (bytes)	field name	Field Size (bytes)
big	0	ChunkID	4
little	4	ChunkSize	4
big	8	Format	4
big	12	Subchunk1ID	4
little	16	Subchunk1Size	4
little	20	AudioFormat	2
little	22	NumChannels	2
little	24	SampleRate	4
little	28	ByteRate	4
little	32	BlockAlign	2
little	34	BitsPerSample	2
big	36	Subchunk2ID	4
little	40	Subchunk2Size	4
little	44	Subchunk2Size	4
		data	

The "RIFF" chunk descriptor

The Format of concern here is "WAVE", which requires two sub-chunks: "fmt" and "data".

The "fmt" sub-chunk

describes the format of the sound information in the data sub-chunk.

The "data" sub-chunk

Indicates the size of the sound information and contains the raw sound data.

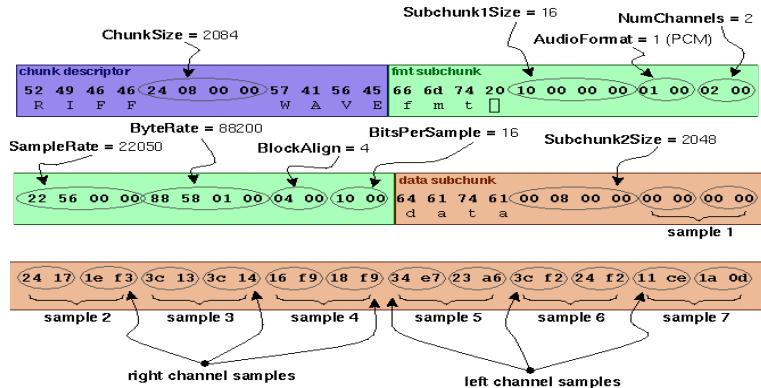
www.ce.unipr.it/people/tomamic

51/62

www.ce.unipr.it/people/tomamic

52/62

Esempio di file WAV



DII

Documenti strutturati

Documenti strutturati

- **Struttura logica**
 - Determina il *ruolo* delle varie parti del testo
 - Titoli, testo, note, etc.
- **Struttura grafica**
 - Assegna una resa grafica ai ruoli
 - Quindi determina la resa grafica del documento nel suo complesso
 - "Stampa" in modo diverso ciò che ha ruolo diverso
- **Word processing:** non tanto *scrivere*, ma *ingegnerizzare informazione*

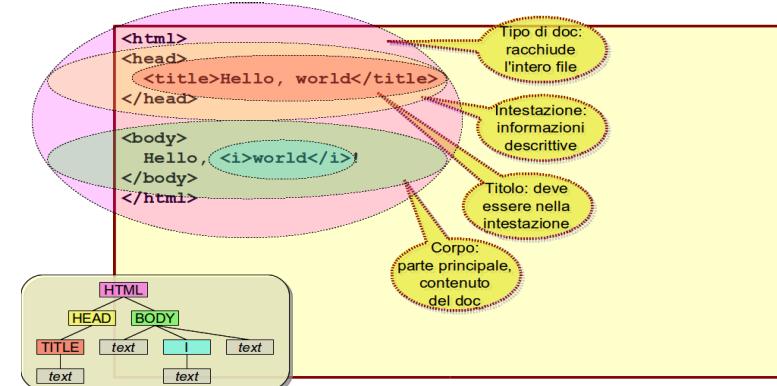
WYSIWYG

- Focus su grafica, si perde di vista la struttura logica
 - Grafica: non con i comandi grafici...
 - Ma definendo gli *stili* delle varie parti di doc, come *ruoli* logici
 - Es. stili di Word/Writer: "*Titolo*", "*Nota in Calce*", "*Intestazione*"
 - Non nomi grafici, ma logici
- In alternativa: editing basato su **comandi** o su **tag**

HyperText Markup Language

- Documenti **strutturati**, standard W3C: <http://www.w3.org/html/>
- HTML dichiara tipi di elementi
 - Paragrafi, titoli, liste, collegamenti ipertestuali, elementi multimediali ecc.
- Tipo di **elemento** descritto da tre parti
 - Tag di apertura, contenuto, tag di chiusura**
 - Bla bla, **** in grassetto. ****, normale.
- Molti tag permettono la definizione di **attributi**
 - UniPR**
 - id e class**: attributi generici per assegnare *ruoli logici*
- Tag semplici non hanno un contenuto
 - **

Anatomia di una pagina



Tag di formattazione testo

```
<p>Questo è un paragrafo.<br />A-capo ma stesso paragrafo.</p>

<p>Testo <strong>in grassetto</strong>, e poi
<em>in corsivo</em>.</p>

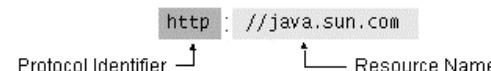
<h1>Il titolo più grande</h1>
...
<h6>Il titolo più piccolo</h6>

<div class="remark">
  Struttura generica di livello blocco,
  con un <span>elemento generico</span> inline.
</div>
```

HTML

Uniform Resource Locator

- Una URL è un riferimento per una risorsa



- Il nome della risorsa dipende interamente dal protocollo. Per HTTP include:
 - Nome dell'host su cui risiede la risorsa
 - Numero di porta cui collegarsi (default = 80)
 - Percorso della risorsa sulla macchina
 - Stringa di query (dopo ?)
 - Frammento: id di un elemento all'interno della risorsa (dopo #)
- <http://www.ietf.org:80/rfc/rfc2732.txt>

Html 5

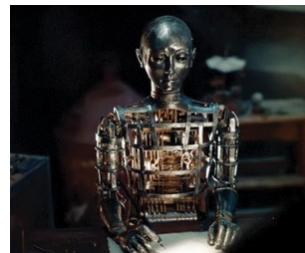
- Nuovi elementi di struttura di Html 5
 - header, main, nav, aside, footer
 - article, section, details, summary
 - figure, figcaption
- Altri nuovi elementi
 - video, audio, canvas, embed
 - mark, menu, command, output, time
 - progress, meter, datalist
- <http://dev.w3.org/html5/html4-differences/>



<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR

Teoria della computazione



Introduzione
all'informatica

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Linguaggi formali

- Presenti in tutte le applicazioni
 - Linguaggi di programmazione
 - Linguaggi di marcatura
 - Interazione uomo macchina
- Fondamentali nel software di sistema
 - Compilatori
 - Interpreti ...
- Paradigmatici nella teoria
 - Molti problemi riconducibili a quello dell'*appartenenza*: una stringa appartiene ad un linguaggio?



Alfabeti e stringhe

- Alfabeto Σ : insieme di simboli
- Stringa s : sequenza di simboli di Σ
 - $s \in \Sigma^*$, insieme di tutte le stringhe
 - ϵ : stringa vuota
 - $|s|$: lunghezza della stringa s
- Linguaggio $L \subseteq \Sigma^*$
 - Sottoinsieme di tutte le stringhe possibili
 - Necessarie regole formali (grammatica) per definire le "*stringhe ben formate*" di L
- Esempio: numeri romani da 1 a 1000
 - Alfabeto {I, V, X, L, C, D, M} + regole...

Concatenazione di stringhe

- Operazione di concatenazione •
 - Propr. associativa: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
 - Non commutativa: $x \cdot y \neq y \cdot x$
 - Σ^* chiuso rispetto a \cdot : $\Sigma^* \times \Sigma^* \rightarrow \Sigma^*$
- Potenza
 - $x^n = x \cdot x \cdot x \cdot x \dots$ (n volte)
- Elemento neutro ϵ
 - Stringa vuota, $\forall x \in \Sigma^*, \epsilon \cdot x = x \cdot \epsilon = x$
- $\langle \Sigma^*, \cdot, \epsilon \rangle$: monoide

Definizione di linguaggi

- Approccio **algebrico**: linguaggio costruito a partire da linguaggi più elementari, con operazioni su linguaggi
- Approccio **generativo**: grammatica, regole per la generazione di stringhe appartenenti al linguaggio
- Approccio **riconoscitivo**: macchina astratta o algoritmo di riconoscimento, per decidere se una stringa appartiene o no al linguaggio



Espressioni regolari

Operazioni su linguaggi

- L_1 ed L_2 linguaggi su Σ^* (due insiemi di stringhe)
- Unione: $L_1 \cup L_2 = \{x \in \Sigma^* | x \in L_1 \vee x \in L_2\}$
- Intersezione: $L_1 \cap L_2 = \{x \in \Sigma^* | x \in L_1 \wedge x \in L_2\}$
- Complementazione: $L_1^c = \{x \in \Sigma^* | x \notin L_1\}$
- Concatenazione o prodotto: $L_1 \cdot L_2 = \{x \in \Sigma^* | x = x_1 \cdot x_2, x_1 \in L_1, x_2 \in L_2\}$
- Potenza: $L^n = L \cdot L^{n-1}, n \geq 1; L^0 = \{\epsilon\}$ per convenzione
 - Concatenazione di n stringhe qualsiasi di L
- Stella di Kleene: $L^* = \cup L^n, n = 0.. \infty$
 - Chiusura riflessiva e transitiva di L rispetto a \cdot
 - Concatenazione arbitraria di stringhe di L

Espressioni regolari

- Dato un alfabeto Σ , chiamiamo **espressione regolare** una stringa r sull'alfabeto $\Sigma \cup \{\+, *, (,), \cdot, \emptyset\}$ t.c.:
 - $r = \emptyset$: linguaggio vuoto; oppure
 - $r \in \Sigma$: linguaggio con un solo simbolo; oppure
 - $r = s + t$: unione dei linguaggi $L(s), L(t)$; oppure
 - $r = s \cdot t$: concatenazione dei linguaggi $L(s), L(t)$; oppure
 - $r = s^*$: chiusura del linguaggio $L(s)$
 - (con s e t espressioni regolari; simbolo \cdot spesso implicito)
- **Linguaggi regolari**: rappresentabili con espressioni regolari ("regex")

Regex nelle applicazioni

- Concatenazione di caratteri: `goal`
- Un carattere qualsiasi: `defin.tely`
- Un car. da un insieme (o no): `[a-z], [^a-z0-9]`
- Unione tra espressioni (opzione): `one|two|three`
- Ripetizioni (0+, 1+, 0-1): `goo*al, go+al, goo?al`
- Sottoespressione: `(left right)*halt`
- In Python: [modulo re](#)

```
>>> text = 'Though not quickly, he run the 5th lap steadily.'
>>> re.findall(r'[a-z]+ly', text)
['quickly', 'steadily']
>>> re.sub(r'([0-9])([a-z]+)', r'\1<sup>2</sup>', text)
Though not quickly, he run the 5<sup>th</sup> lap steadily.
```

PYTHON



Grammatiche di Chomsky

Grammatiche di Chomsky

- Grammatica $G = \langle V_T, V_N, P, S \rangle$
 - V_T : alfabeto finito di simboli **terminali**
 - V_N : ... **non terminali** (variabili, categorie sintattiche)
 - $V = V_T \cup V_N$
 - P : insieme di **produzioni**, relaz. binarie $V^* \rightarrow V_N \rightarrow V^*$
 $\langle \alpha, \beta \rangle \in P$ si indica con $\alpha \rightarrow \beta$
 - $S \in V_N$: **assioma**
- $L(G)$: insieme delle stringhe di terminali ottenibili con finite operazioni di riscrittura
 - Applicazione delle regole di produzione, in vario modo

Linguaggio generato da G

- Derivazione diretta** \Rightarrow^* : riscrittura di una stringa tramite applicazione di una regola di produzione
- Derivazione** \Rightarrow^* : chiusura riflessiva e transitiva della derivazione diretta
- Forma di frase**: stringa x t.c. $x \in V^*$, $S \Rightarrow^* x$
- Linguaggio generato** da G : forme di frase con soli simboli terminali
 - $L(G) = \{x \mid x \in V_T^*, S \Rightarrow^* x\}$
- Equivalenza** tra G_1 e G_2 : $L(G_1) = L(G_2)$

Grammatiche equivalenti

- $G_1 = \langle \{a, b\}, \{S, A\}, P, S \rangle$, con produzioni:
 - $S \rightarrow b$
 - $S \rightarrow aA$
 - $A \rightarrow aS$
 - ... genera il linguaggio $\{a^n b \mid n \text{ pari}\}$
- Anche G_2 , con produzioni:
 - $S \rightarrow Ab \mid b$
 - $A \rightarrow aAa \mid aa$
- Ed anche G_3 :
 - $S \rightarrow Ab$
 - $A \rightarrow Aaa \mid \epsilon$

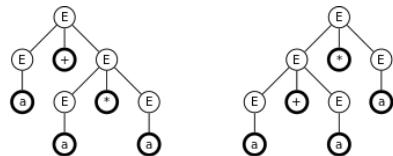
| per raggruppare diverse produzioni di uno stesso non-terminale

Esempio di generazione

- $G = \langle \{a, b, c\}, \{S, B, C\}, P, S \rangle$
 - (1) $S \rightarrow aSBC$
 - (2) $S \rightarrow aBC$
 - (3) $CB \rightarrow BC$
 - (4) $aB \rightarrow ab$
 - (5) $bB \rightarrow bb$
 - (6) $bC \rightarrow bc$
 - (7) $cC \rightarrow cc$
 - ... genera il linguaggio $\{a^n b^n c^n \mid n \geq 1\}$
- Esercizio: provare a generare $aaabbccccc$
 - Soluzione: applicare 1-1-2-3-3-4-5-5-6-7-7

Alberi di derivazione (sintattici)

- Grammatica **ambigua**: due interpretazioni valide per $a + a * a$
 - $V_T = \{a, +, *, (,)\}; V_N = \{E\}$;
 - $E \rightarrow E+E \mid E^*E \mid (E) \mid a$



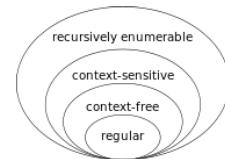
- Grammatica non ambigua (con precedenza tra operatori)

$$\begin{array}{l} E \rightarrow E+T \mid T \\ T \rightarrow T^*F \mid F \\ F \rightarrow (E) \mid a \end{array}$$

Classificazione di Chomsky

- Tipo 0**: grammatiche **ricorsivam. enumerabili** (RE)
 - $\alpha A \beta \rightarrow \gamma$ (*non limitate*)
- Tipo 1**: grammatiche **contestuali** (CS)
 - $\alpha A \beta \rightarrow \alpha \gamma \beta$
- Tipo 2**: grammatiche **non contestuali** (CF)
 - $A \rightarrow \gamma$
- Tipo 3**: grammatiche **regolari** (REG)
 - $A \rightarrow aB$ oppure $A \rightarrow b$
 - Coincide con classe dei linguaggi definiti da **regex**

$$A, B \in V_N; a, b \in V_T; \alpha, \beta, \gamma \in V^*$$



Linguaggi non contestuali

- Controllo di **palindromi**, **bilanciamento di parentesi** e varie **simmetrie**
 - Es.: $\{a^n b^n \mid n \geq 1\}$ gen. da $S \rightarrow aSb \mid ab$ (CF)
 - Ma non: $\{a^n b^n c^n \mid n \geq 1\}$ (CS) (*)
- Linguaggi di programmazione** comuni: grammatiche CF
- Definizione con notazione Extended **Backus-Naur Form** (EBNF)
 - {...}: parte ripetibile (0+), [...]!: parte opzionale,
 - (...): raggruppamento, |: scelta
 - Terminali tra virgolette

(*) Nell'es. visto, sostituire (3) con: (3a) $CB \rightarrow HB$; (3b) $HB \rightarrow HC$; (3c) $HC \rightarrow BC$

Linguaggi non contestuali

Linguaggi LL(1)

- Sottoclasse dei linguaggi CF
- Ogni produzione relativa a stesso non-terminale (a sx)... genera come primo simbolo un terminale diverso
 - No prefissi comuni, no ricorsione sinistra
- Recursive descent parser:** analisi sintattica molto semplice ed efficiente
 - Basta "spiare" il simbolo di input successivo, per capire con certezza quale produzione applicare
- Polish prefix notation**
 - Es.: $* + 1 2 - 3 2 \Rightarrow$ (in forma infissa) $(1 + 2) * (3 - 2)$

```
expr = number | "+" expr expr | "-" expr expr | "*" expr expr | "/" expr expr
number = digit {digit}
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

EBNF

Espressioni infisse

- Grammatica LL(1), ma senza precedenza tra operatori
- Provare a generare $2 + 3 * 3$

```
expr = term {("+" | "-" | "*" | "/") term}
term = number | "(" expr ")" | "-" term
```

EBNF

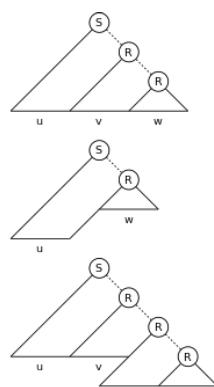
- Grammatica LL(1), con precedenza tra operatori

```
expr = term {("+" | "-") term}
term = factor {"*" | "/" factor}
factor = number | "(" expr ")" | "-" term
```

EBNF

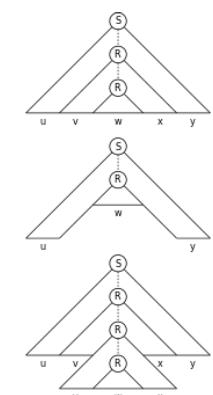
Pumping lemma REG

- Formalmente, $\forall L$ regolare...
 - $\exists k$ t.c. $\forall z \in L$, $|z| \geq k$
 - $\exists u, v, w$, $|uv| \leq k$, $|v| \geq 1$ t.c.
 - $z = uvw$, $uv^iw \in L$, $\forall i \geq 0$
- In ogni stringa abbastanza lunga,
 - c'è una parte che si può ripetere,
 - generando un'altra stringa di L
- Uno stesso non-terminale, per un input abbastanza lungo, deve comparire più volte nell'albero sintattico ...
 - Un Automa a Stati Finiti (*), per un input abbastanza lungo, torna in uno stato già visitato ...



Pumping lemma CF

- Formalmente, $\forall L$ non contestuale...
 - $\exists k$ t.c. $\forall z \in L$, $|z| \geq k$
 - $\exists u, v, w, x, y$, $|vwx| \leq k$, $|vx| \geq 1$ t.c.
 - $z = uvwxy$, $uv^ix^y \in L$, $\forall i \geq 0$
- In ogni stringa abbastanza lunga,
 - ci sono due parti che si possono ripetere assieme, restando in L
- Uno stesso non-terminale, per un input abbastanza lungo, deve comparire più volte nell'albero sintattico ...



Corollari dei due pumping lemma

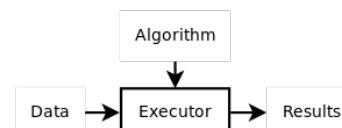
- $\Rightarrow L = \{a^n b^n \mid n \geq 0\}$ non è REG
 - Si prende $a^m b^m$, con $m > k \Rightarrow |uv| < m$, sono tutte a...
- $\Rightarrow L = \{a^n b^n c^n \mid n \geq 0\}$ non è CF
 - Si prende $a^m b^m c^m$, con $m > k \Rightarrow |vwx| < m$
 - Se v ed x con più simboli diversi, uv^2wx^2y con simboli mescolati
 - Se v ed x con un solo simbolo, uv^2wx^2y con numero diverso di a, b, c
 - In entrambi i casi la nuova stringa $z' \notin L$



Automi e calcolo

Automi e calcolo

- **Automa:** macchina astratta
- Riceve dall'esterno i dati ed una descrizione dell'algoritmo richiesto
- Interpreta un linguaggio, detto linguaggio macchina dell'automa
- Vincolo su numero di componenti, finito
- Vincolo su alfabeti di ingresso e di uscita, composti da un numero finito di simboli



Riconoscimento di linguaggi

- Data una stringa x, stabilire se essa appartiene ad L (problema dell'*appartenenza*, o *membership*)
- L. tipo 3: riconosciuti da *macchine a stati finiti (FSM)*
 - Es.: $\{a^n b \mid n \geq 0\}$, gen. da $S \rightarrow aS \mid b$
- L. tipo 2: *automi a pila non deterministici (NPDA)*
 - Es.: $\{a^n b^n \mid n \geq 1\}$ gen. da $S \rightarrow aSb \mid ab$
- L. tipo 1: riconosciuti da *TM "linear bounded"*
 - Es.: $\{a^n b^n c^n \mid n \geq 1\}$
- L. tipo 0: riconosciuti da *macchine di Turing (TM)*
 - Ma, se $x \notin L$, il processo può non terminare!



Macchina a stati finiti (FSM)

- $M = \langle \Sigma, Q, \delta, q_0, F \rangle$
- $\Sigma = \{\sigma_1, \dots, \sigma_n\}$: alfabeto di input
- $Q = \{q_0, \dots, q_n\}$: insieme finito non vuoto di stati
- $F \subseteq Q$: insieme di stati finali
- $q_0 \in Q$: stato iniziale
- $\delta: Q \times \Sigma \rightarrow Q$: funzione di transizione
 - In base allo stato e al simbolo di input attuali ... determina lo stato successivo

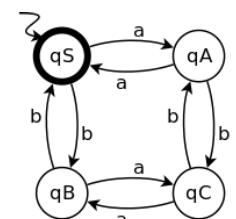
Macchina a stati finiti (FSM)

Linguaggio riconosciuto da FSM

- Funzione di transiz. estesa a stringhe: $\delta: Q \times \Sigma^* \rightarrow Q$
 - $\delta(q, \epsilon) = q$
 - $\delta(q, ax) = \delta(\delta(q, a), x)$, con $a \in \Sigma, x \in \Sigma^*$
- Linguaggio riconosciuto da una macchina M:
 - $L(M) = \{x \in \Sigma^* \mid \delta(q_0, x) \in F\}$
- FSM riconoscono tutti e soli i *linguaggi regolari*
- Rappresentazione della funzione di transizione
 - *Tabella di transizione*
 - *Diagramma degli stati*

Esempio di FSM

- $M = \langle \{a, b\}, \{q_S, q_A, q_B, q_C\}, \delta, q_S, \{q_S\} \rangle$
- | δ | a | b |
|----------|-------|-------|
| q_S | q_A | q_B |
| q_A | q_S | q_C |
| q_B | q_C | q_S |
| q_C | q_B | q_A |
-
- L: stringhe con a e b in numero pari
 - $S \rightarrow aA \mid bB \mid \epsilon$
 - A $\rightarrow aS \mid bC$
 - B $\rightarrow bS \mid aC$
 - C $\rightarrow aB \mid bA$



FSM non deterministica

- $M = \langle \Sigma, Q, \delta_N, q_0, F \rangle$
- $\Sigma = \{\sigma_1, \dots, \sigma_n\}$: alfabeto di input
- $Q = \{q_0, \dots, q_m\}$: insieme finito non vuoto di stati
- $F \subseteq Q$: insieme di stati finali
- $q_0 \in Q$: stato iniziale
- $\delta_N: Q \times \Sigma \rightarrow P(Q)$: funzione di transizione, determina insieme di stati successivi
 - $P(Q)$ è l'insieme delle parti di Q , ossia l'insieme di tutti i possibili sottoinsiemi di Q

Esempio di NFSM

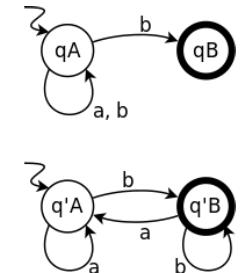
- $M = \langle \{a, b\}, \{q_A, q_B\}, \delta, q_A, \{q_B\} \rangle$

δ	a	b
q_A	$\{q_A\}$	$\{q_A, q_B\}$
q_B	\emptyset	\emptyset

- $M' = \langle \{a, b\}, \{q'_A, q'_B\}, \delta', q'_A, \{q'_B\} \rangle$

δ'	a	b
$\{q_A\}$	$\{q_A\}$	$\{q_A, q_B\}$
$\{q_A, q_B\}$	$\{q_A\}$	$\{q_A, q_B\}$

$$\{q_A\} \leftrightarrow q'_A, \{q_A, q_B\} \leftrightarrow q'_B$$



Equivalenza FSM / NFSM

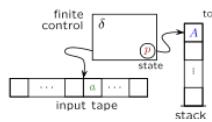
- Per ogni stato / ingresso, definiti più stati successivi
- Non-determinismo: calcolo = albero di computazioni autonome, anziché traiettoria in uno spazio di stati
- Nel casi di FSM, non-determinismo non aggiunge potere computazionale
 - FSM / NFSM: formalismi equivalenti
 - FSM è un caso particolare di NFSM
 - Viceversa, ogni elemento di $P(Q)$ di NFSM diventa uno stato di FSM
 - $P(Q)$ contiene $2^{|Q|}$ elementi



Automa a pila (PDA)

Automa a pila (PDA)

- Simile a FSM, ma dotato di memoria infinita, organizzata a pila
 - Si può accedere solo alla cima della pila
 - Lettura del simbolo in cima
 - Sostituzione simbolo in cima con nuova stringa (anche ϵ)
- In forma non-deterministica, permette di riconoscere i linguaggi non contestuali
- In forma deterministica, riconosce solo i linguaggi non contestuali deterministici (sottoclasse)
 - Base dei comuni linguaggi di programmazione



Definizione di PDA

- $M = \langle \Sigma, \Gamma, z_0, Q, q_0, F, \delta \rangle$
- $\Sigma = \{\sigma_1, \dots, \sigma_n\}$: alfabeto di input
- $\Gamma = \{z_0, \dots, z_m\}$: simboli della pila
- $z_0 \in \Gamma$: simbolo di pila iniziale
- $Q = \{q_0, \dots, q_k\}$: insieme finito non vuoto di stati
- $q_0 \in Q$: stato iniziale; $F \subseteq Q$: insieme di stati finali
- $\delta: Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$: funzione di transizione
 - In base a stato, simbolo di input, simbolo in cima a pila ...
 - Determina stato successivo e simboli inseriti nella pila
 - Per rimuovere il simbolo in cima alla pila, si scrive ϵ

Esempio di PDA

- Automa a pila M che riconosce $L = \{a^n b^n, n \geq 1\}$
- $M = \langle \{a, b\}, \{Z_0, A_0, A\}, Z_0, \{q_0, q_1, q_2\}, q_0, \{q_2\}, \delta \rangle$
 - A serve a ricordare la presenza delle a
 - q_0 : si memorizzano le a
 - q_1 : si confrontano le b con quanto memorizzato
 - q_2 : stato finale

δ	Z_0, a	Z_0, b	A_0, a	A_0, b	A, a	A, b
q_0	A_0, q_0		AA_0, q_0	ϵ, q_2	AA, q_0	ϵ, q_1
q_1			ϵ, q_2		ϵ, q_1	
q_2						

PDA non deterministico (NPDA)

- $A = \langle \Sigma, \Gamma, z_0, Q, q_0, F, \delta_N \rangle$
- $\delta_N: Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Gamma^*)$: funzione di transizione, determina gli stati e i simboli di pila successivi
- Es. $\delta(p, a, A) = \{(q, BA), (r, \epsilon)\}$ (due transizioni)
- Simbolo A in cima alla pila sostituito dalla stringa di caratteri BA, nuovo stato interno q
- Simbolo A in cima alla rimosso (sostituito da ϵ), nuovo stato interno r
- NPDA: maggiore potere computazionale di PDA
 - $L = \{a^n b^n\} \cup \{a^n b^{2n}\}, n \geq 0$
 - $S \rightarrow A|B, A \rightarrow aAb|\epsilon, B \rightarrow aBbb|\epsilon$



Macchina di Turing (TM)



Macchina di Turing (TM)

- Automa con testina di scrittura/lettura su nastro bidirezionale "illimitato"
- Ad ogni passo:
 - Si trova in un certo stato
 - Legge un simbolo dal nastro
- In base alla funzione di transizione (deterministica):
 - Scrive un simbolo sul nastro
 - Sposta la testina di una posizione
 - Cambia lo stato
- Può simulare ogni altro modello di calcolo noto!

TM deterministica

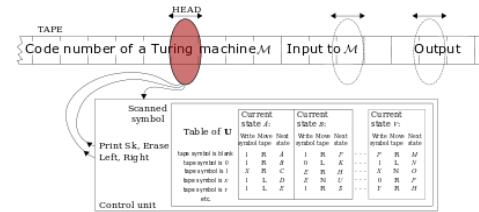
- $M = \langle \Sigma, Q, q_0, F, \delta \rangle$
- $\Sigma = \{\sigma_1, \dots, \sigma_n, b\}$: alfabeto di input (+ **blank**)
- $Q = \{q_0, \dots, q_m\}$: insieme finito non vuoto di stati
- $q_0 \in Q$: stato iniziale
- $F \subseteq Q$: insieme di stati finali
- $\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{d, s, i\}$: funzione di transizione
 - Determina lo stato successivo, il simbolo successivo, lo spostamento della testina

TM non deterministica (NTM)

- $M = \langle \Sigma, Q, q_0, F, \delta_N \rangle$
- $\delta_N: Q \times \Sigma \rightarrow P(Q \times \Sigma \times \{d, s, i\})$: funzione di transizione
 - Determina le configurazioni successive (nuovo stato, simbolo scritto, spostamento della testina)
- Grado di non-determinismo: massimo numero di figli di un nodo dell'albero di computazione
- NTM: **stessa potenza computazionale di TM**
 - Data NTM, M , con grado di non-determinismo d ...
 - $\exists M'$, una equivalente MT, in grado di simulare M
- Ma (finora...) NTM **più efficiente**
 - k passi di M richiedono k^d (asintot.) passi di M'

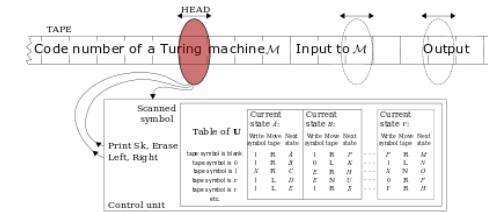
Macchina di Turing universale (UTM)

- UTM realizza la computazione:
 - $(q_0, D_M \# x) \xrightarrow{*} (\alpha, q_f, \beta) \dots$
 - q_0 , stato iniziale; q_f , stato finale
 - $D_M \# x$: input; D_M : descrizione di M (funz. di transizione)
 - $\Leftrightarrow M$ realizza la computazione: $(q_0, x) \xrightarrow{*} (\alpha, q_f, \beta)$
 - q_0 , stato iniziale; q_f , stato finale; x , input
 - Regole di transizione di $M \rightarrow$ Sequenza di quintuple
 - $D_M = d_1 \# d_2 \# \dots \# d_n$
 - $q_i \# \sigma_j \# q_h \# \sigma_k \# t_l \Leftrightarrow \delta(q_i, \sigma_j) = (q_h, \sigma_k, t_l)$



Funzionamento della UTM

- UTM **interpreta** un arbitrario programma su nastro
 - Dato lo stesso input, UTM produce lo stesso output di M
 - Per ogni simbolo letto in x (input di M), scorre la lista di regole, per scegliere la giusta transizione



Calcolabilità

- **Tesi di Church-Turing:** problema intuitivamente calcolabile → esiste TM in grado di calcolarlo
 - TM: formalismo non meno potente di ogni altro modello di calcolo proposto finora
 - Funzioni ricorsive, lambda-calcolo, macchine a registri...
 - Ma esistono **problemi irrisolvibili** (nel caso generale)
 - Attenzione: non si dice niente sulla singola istanza!
 - **Teorema di incompletezza di Gödel**
 - ∀ formalizzazione della matematica che assiomatizza \mathbb{N}
 - $\rightarrow \exists$ proposizione né dimostrabile né confutabile



Calcolabilità



Paradossi classici

- Paradosso del mentitore
 - Questa frase è falsa
 - Epimenide di Creta afferma: "*I cretesi sono bugiardi*"
- Paradosso del barbiere
 - Se un barbiere fa la barba a tutti e soli coloro che non si fanno la barba da soli, chi sbarba il barbiere?
- Paradosso di Russell
 - Consideriamo insiemi di insiemi e supponiamo che un insieme possa contenere se stesso
 - Sia T l'insieme di tutti gli insiemi che non contengono se stessi; possiamo stabilire se T contiene T ?

Problema della terminazione

- Predicato della terminazione, **non calcolabile**
 - $h(D_M, x) = 1$, se M con input x termina
 - $h(D_M, x) = 0$, se M con input x non termina
- Costruiamo per assurdo H , TM che calcola h
- Costruiamo quindi G
 - $g(D_M) = 0$, se $h(D_M, D_M) = 0$
 - Indefinito, altrimenti (ossia G cicla all'infinito, se $h = 1$)
- Ma è assurdo:
 - $g(D_G)$ è indefinita, se $g(D_G) = 0$ (definita)
 - $g(D_G) = 0$ (definita) se $g(D_G)$ è indefinita

M: macchina di Turing; D_M : rappresentazione di M come stringa

Più informalmente...

- Funz. g definita in `paradox.py` (Python è *Turing completo*)

```
from halting import h
def g(file):
    if h(file, file):
        while True: pass
    else:
        return False
g('paradox.py')
```

PYTHON



- Altri problemi indecidibili (corollari)
 - Correttezza: il programma calcola la funzione desiderata?
 - Chiamata: una procedura (o istruzione) sarà eseguita?
 - Equivalenza, ambiguità di grammatiche CF ...

Complessità

Problemi e complessità

- Problemi *non risolvibili*
 - Es. Questa frase è falsa
 - Incompletezza Gödel; indecidibilità terminazione
- Risolvibili
 - *Non trattabili* (costo "esponenziale")
 - Trattabili (costo accettabile, "polinomiale")
- **Calcolabilità**: classificare risolvibili e non risolvibili
- **Complessità**: "facili" e "difficili"



Ricerca lineare

```
def linear_search(v: list, value) -> int:  
    '''v: not necessarily sorted'''  
  
    for i in range(len(v)):  
        if v[i] == value:  
            return i  
  
    return -1
```

PYTHON

Ricerca binaria

```
def binary_search(v: list, value) -> int:  
    '''v: sorted list'''  
  
    begin, end = 0, len(v)  
    while begin < end:  
        middle = (begin + end) // 2  
        if v[middle] > value:  
            end = middle  
        elif v[middle] < value:  
            begin = middle  
        else:  
            return middle  
  
    return -1
```

PYTHON

Costo di un algoritmo

- **Spazio**, memoria richiesta
- **Tempo**, necessario all'esecuzione
- Di solito si contano i cicli, in funzione di **n**
- O i confronti/scambi tra elementi dell'array
 - Array in memoria centrale, accesso lento
 - Altre variabili nei registri del processore
- Test e misure empiriche

Confronto tra algoritmi

- Caso peggiore negli algoritmi di ricerca: elemento non presente
- Ricerca lineare: n confronti
- Ricerca binaria: $\lceil \log_2(n) \rceil$ confronti
 - A ogni iterazione l'insieme è dimezzato
 - Quante volte n deve essere diviso per 2, per arrivare ad 1?
 - $2^k \geq n \rightarrow k \geq \log_2(n)$

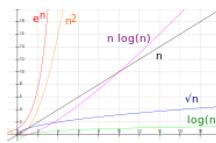
Def. di complessità

- Una funzione $f(n)$ ha **ordine** $O(g(n))$ sse:
 - Esistono due costanti positive c ed m , tali che
 - $|f(n)| \leq c|g(n)| \forall n > m$
- Un algoritmo ha una **complessità** $O(g(n))$ sse:
 - Il tempo di calcolo $t(n)$, sufficiente per eseguire l'algoritmo con ogni istanza(*) di dimensione n , ha ordine $O(g(n))$

(*) *Istanza: insieme di dati su cui è definito il problema; quindi per la complessità conta il caso peggiore*

Analisi asintotica

- Per n abbastanza grande, a meno di una costante moltiplicativa, $f(n)$ non supera in modulo $g(n)$
- Comportamento dell'algoritmo al limite, per dimensione delle istanze tendente all'infinito
- Es. $n = 1\,000\,000$
 - Ricerca lineare: 1'000'000 cicli
 - Ricerca binaria: 20 cicli



Complessità intrinseca

- Limite inferiore di complessità di un problema
- Una funzione $f(n)$ è $\Omega(g(n))$ sse
 - Esistono due costanti positive c e m tali che
 - $|f(n)| \geq c|g(n)| \forall n > m$
- Un problema ha una **delimitazione inferiore** alla complessità $\Omega(g(n))$ sse
 - Per ogni algoritmo risolutore...
 - \exists una istanza (caso peggiore)...
 - per cui il tempo di calcolo $t(n)$ è $\Omega(g(n))$

Algoritmo ottimale

- Algoritmo che risolve un problema P, con le due seguenti condizioni:
 - Costo di esecuzione $O(g(n))$
 - P ha una delimitazione inferiore $\Omega(g(n))$
- Es. L'algoritmo della ricerca binaria è ottimale
 - È dimostrato che $\log_2(n)$ è la complessità intrinseca della ricerca
 - Ma ricerca lineare funziona anche per liste non ordinate!



Algoritmi di ordinamento

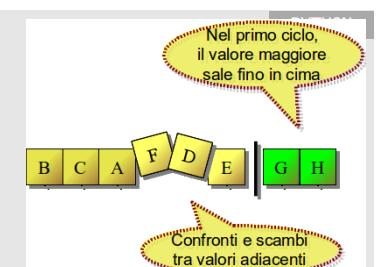
Algoritmi di ordinamento

- Ricerca binaria: importante avere dati ordinati
 - Ordinateur, ordenador
- Algoritmi di ordinamento più semplici hanno complessità n^2
 - Confronto tra ciascun elemento e gli altri
- Algoritmi di ordinamento *divide et impera*
 - Complessità $n \cdot \log_2(n)$
 - Complessità intrinseca

Bubble sort

```
def swap(v: list, i: int, j: int):
    v[i], v[j] = v[j], v[i]

def bubble_sort(v: list):
    end = len(v) - 1
    while end > 0:
        for i in range(end):
            if v[i] > v[i + 1]:
                swap(v, i, i + 1)
        end -= 1
```



Analisi Bubble Sort

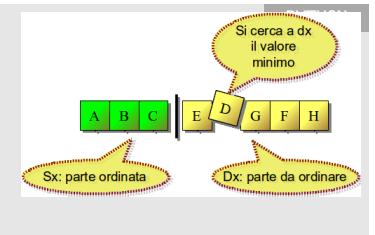
- Gli elementi maggiori salgono rapidamente, *"come bollicine di champagne"*
- Caso peggiore: lista rovesciata
 - Numero di confronti e scambi: $n^2/2$
 - $(n-1)+(n-2)+\dots+2+1 = n(n-1)/2 = n^2/2 - n/2 \approx n^2/2$
 - (Applicata la formula di Gauss per la somma dei primi numeri)
 - Complessità n^2
- Anche in media, circa stessi valori

Selection Sort

```
def selection_sort(v: list):
    for i in range(len(v) - 1):
        min_pos = i

        for j in range(i + 1, len(v)):
            if v[j] < v[min_pos]:
                min_pos = j

        swap(v, pos_min, i)
```



Analisi Selection Sort

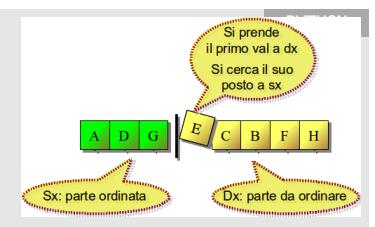
- Ad ogni ciclo principale, si seleziona il valore minore
- Caso peggiore: lista rovesciata
 - Numero di confronti $n \cdot (n-1)/2$; complessità n^2
 - Numero di scambi: $n-1$ scambi
- Anche in media, circa stessi valori

Insertion sort

```
def insertion_sort(v: list):
    for i in range(1, n):
        value = v[i]

        for j in range(i - 1, -1, -1):
            if v[j] <= value: break
            v[j + 1] = v[j]

        v[j + 1] = value
```

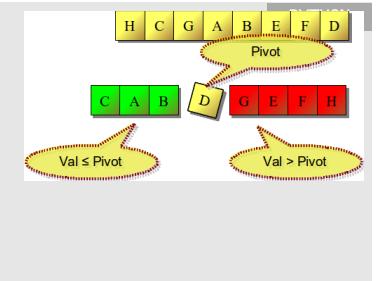


Analisi Insertion Sort

- La prima parte è ordinata, vi si inserisce un elemento alla volta, più facile trovare il posto
- Caso peggiore: lista rovesciata
 - Cicli: $1+2+\dots+(n-1) = n \cdot (n-1)/2$; compl: $O(n^2)$
- In media si scorre solo 1/2 della prima parte
 - In media $n^2/4$ confronti e $n^2/4$ scambi
- Ottimizzazioni
 - Ricerca binaria in parte ordinata, ma scambi
 - Inserimento a coppie, o gruppi

Quick Sort

```
def quick_sort(v: list, begin=0, end=len(v)):  
    if end - begin > 1:  
        pivot = v[end - 1]  
        j = begin  
        for i in range(begin, end - 1):  
            if v[i] < pivot:  
                swap(v, i, j)  
                j += 1  
        swap(v, end - 1, j)  
        quick_sort(v, begin, j)  
        quick_sort(v, j + 1, end)
```

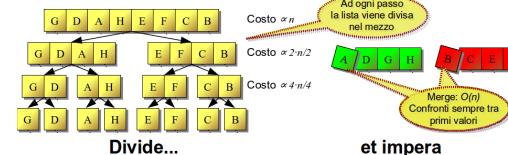


Analisi Quick Sort

- Dato un insieme, sceglie un valore **pivot**
- Crea due sottoinsiemi: $x \leq \text{pivot}$, $x > \text{pivot}$
- Stesso algoritmo sui 2 insiemi (ricorsione)
- Caso peggiore: lista rovesciata, n^2
 - Dipende da scelta pivot, ma esiste sempre
- Caso medio: $n \cdot \log_2(n)$
 - $t(n) = \alpha \cdot n + 2 \cdot t(n/2)$
 - Sostituzione k volte: $t(n) = \alpha \cdot k \cdot n + 2^k t(n/2^k) \dots$

Merge Sort

```
def merge_sort(v, begin=0, end=len(v)):  
    '''In v, sort elements in range(begin, end)'''  
    if end - begin > 1:  
        middle = (begin + end) / 2  
        merge_sort(v, begin, middle)  
        merge_sort(v, middle, end)  
        merge(v, begin, middle, end)
```



Merge, con appoggio

```
def merge(v, begin, middle, end):
    '''Merge two sorted portions of a single list'''
    i1, i2, n = begin, middle, end - begin
    result = []

    for k in range(n):
        if i1 < middle and (i2 >= end or v[i1] <= v[i2]):
            result.append(v[i1])
            i1 += 1
        else:
            result.append(v[i2])
            i2 += 1

    for k in range(n):
        cards[begin + k] = result[k]
```

PYTHON

Analisi Merge Sort

- Simile a Quick Sort, ma non si sceglie pivot
- La fusione ha complessità lineare
- Caso peggiore, caso medio: $n \cdot \log_2(n)$
- **Spazio:** la fusione richiede altra memoria: n
 - Ma si può evitare il costo con spostamenti *in place*
- Accessi sequenziali, buon uso *cache*
- Integraz. con Insertion Sort (Python, Java7)

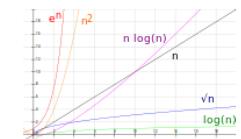
Classi di complessità



DII

Classi di complessità

- Costante: numero op. non dipende da n , dim. istanza
- Sotto-lineare: n^k , $k < 1$; $\log(n)$, ricerca binaria
- Lineare: numero op. $\propto n$, ricerca lineare
- Sovra-lineare: $n \cdot \log(n)$, merge sort
- Polinomiale: n^k , $k \geq 2$, insertion sort
- **Algoritmo efficiente:** fino a classe polinomiale
- **Problema trattabile:** \exists algoritmo efficiente



Complessità esponenziale

- Complessità esponenziale: k^n
 - Es. elenco sottinsiemi, strategia perfetta per scacchi
- Complessità super-esponenziale: $n!$, n^n , ...
 - Es. elenco permutazioni
- **Problemi intrattabili**
 - \nexists algoritmo efficiente
 - Soluzioni non esatte/ottime, euristiche
 - Ma minimi locali...



75/79

Problemi P ed NP

- Problemi **P**: \exists algoritmo *deterministico polinomiale*
- **NP**: \exists algoritmo *non-deterministico polinomiale*
 - Su macchine deterministiche: non noto algoritmo polinomiale per la **ricerca** di una soluzione...
 - Ma algoritmo polinomiale per la **verifica** di una soluzione
- Esempio: fattorizzazione di grandi numeri
 - Ricerca: quali sono i fattori primi di un numero di n cifre?
 - Verifica: è vero che x è divisore di y ?
- **Non è dimostrato che $P \neq NP$, né che $P = NP$**
 - Se $P = NP$, trovare i fattori primi di un numero o verificarli: stessa classe di complessità

76/79

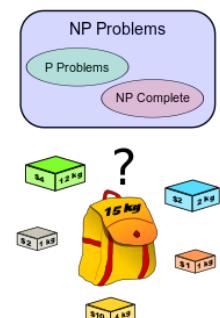
Complessità dei linguaggi

- *Linguaggi di classe P / NP*: stringa x riconosciuta in tempo polinomiale rispetto a $|x|$...
 - **P**: da una macchina di Turing deterministica (*DTM*)
 - **NP**: da una macchina di Turing non-deterministica (*NTM*)
 - Sappiamo che $P \subseteq NP$ (DTM: caso particolare di NTM)
- *Linguaggi di classe EXP*: stringa x riconosciuta in tempo esponenziale rispetto a $|x|$ da una DTM
 - NTM: simulata da DTM in tempo esponenziale
 - Quindi $NP \subseteq EXP$
- $P \subseteq NP \subseteq EXP$

77/79

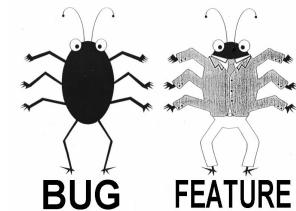
Problemi NP-completi

- Ogni problema NP può essere ricondotto ad un problema **NP-completo** con algoritmo deterministico *polinomiale*
 - *Lower-bound* deterministico esponenziale per uno dei problemi NP-completi? $\Rightarrow P \neq NP$
 - Oppure, *soluzione* con algoritmo deterministico polinomiale? $\Rightarrow P = NP$
- Esempio: **SAT**
 - Data una formula booleana *PdS*, è soddisfacibile?
 - \exists combinazione di input che dà risultato vero?
- Esempio: **Knapsack**
 - \exists combinazione di elementi che realizza utilità $\geq V$, con peso $\leq W$?



78/79

Sviluppo del software



<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR



Introduzione
all'informatica

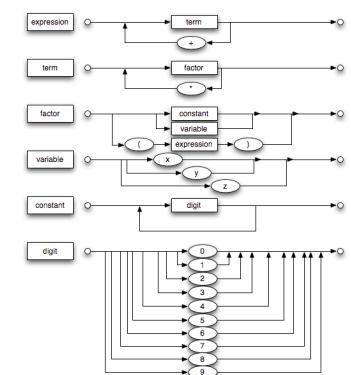
Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Linguaggi di programmazione

- Notazione formale per definire algoritmi
 - Algoritmo**: sequenza di istruzioni per risolvere un dato problema in un tempo finito
- Ogni linguaggio è caratterizzato da:
 - Sintassi**
 - Semantica**

Sintassi

- Insieme di regole formali per scrivere *frasi* ben formate (programmi) in un certo linguaggio
 - Lessico**: parole riservate, operatori, variabili, costanti ecc. (*tokens*)
- Grammatiche non contestuali (...) espresse con notazioni formali:
 - Backus-Naur Form
 - Extended BNF
 - Diagrammi sintattici



Semantica

- Attribuisce un **significato** alle frasi (sintatticamente corrette) costruite nel linguaggio
- Una frase può essere sintatticamente corretta e tuttavia non aver alcun significato
 - Soggetto – predicato – complemento
 - "La mela mangia il bambino"*
 - "Il bambino mangia la mela"*
- Oppure avere un significato diverso da quello previsto...
 - GREEK_PI = 345*

Semantica

- Correttezza sui tipi**
 - Quali tipi di dato possono essere elaborati?
 - Quali operatori applicabili ad ogni dato?
 - Quali regole per definire nuovi tipi e operatori?
- Semantica operazionale**
 - Qual è l'effetto di ogni azione elementare?
 - Qual è l'effetto dell'aggregazione delle azioni?
 - Cioè, qual è l'effetto dell'esecuzione di un certo programma?

Linguaggi di basso livello

- Più orientati alla macchina che ai problemi da trattare
- Linguaggi macchina**: solo operazioni eseguibili direttamente dall'elaboratore
 - Op. molto elementari, diverse per ogni processore, in formato binario
- Linguaggi assembly**: prima evoluzione, codici binari → mnemonici

```
; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

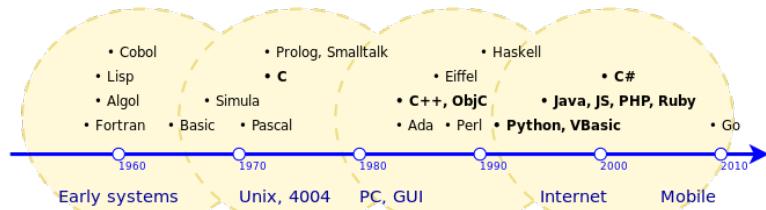
SUB32 PROC    ; procedure begins here
    CMP AX,97   ; compare AX to 97
    JGE DONE    ; if less or equal to 97, jump to DONE
    CMP AX,122  ; compare AX to 122
    JG DONE    ; if greater, jump to DONE
    SUB AX,32   ; subtract 32 from AX
    RET        ; return to main program
SUB32 ENDP    ; procedure ends here
```

FIGURE 17. Assembly language

Linguaggi di alto livello

- Introdotti per facilitare la scrittura dei programmi
- Definizione della soluzione in modo intuitivo
- Con una certa **astrazione** rispetto al calcolatore su cui verranno eseguiti
- Devono essere tradotti in linguaggio macchina

Storia dei linguaggi

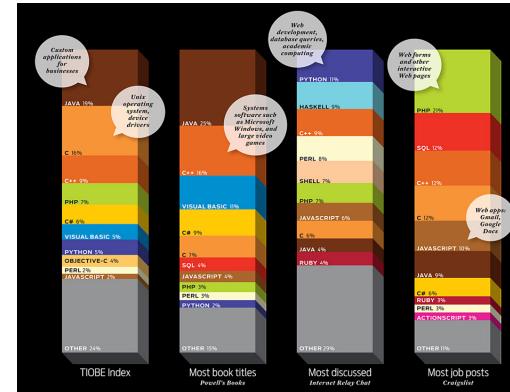


http://www.oreilly.com/news/graphics/prog_lang_poster.pdf

<http://www.leverenz.com/lang/history.html>

http://www.cs.brown.edu/~adj/programming_languages.html

The Top 10 (IEEE Spectrum, 2011)



Paradigmi di sviluppo

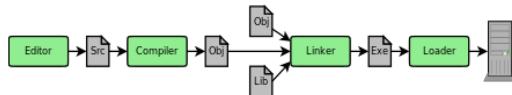
- Forniscono la filosofia e la metodologia con cui si scrivono i programmi
- Definiscono il concetto (astratto) di computazione
- Ogni linguaggio consente (o spinge verso) l'adozione di un particolare paradigma
 - Imperativo / procedurale
 - Orientato agli oggetti
 - Scripting (tipizzazione dinamica, principio *Don't Repeat Yourself, DRY*)
 - Funzionale (funzioni come "cittadini di prima classe")
 - Logico (base di conoscenza + regole di inferenza)

Linguaggi e paradigmi

- Imperativi / procedurali**
 - Cobol, Fortran, Algol, C, Pascal
- Orientati agli oggetti**
 - Simula, Smalltalk, Eiffel, C++, Delphi, Java, C#, VB.NET
- Scripting**
 - Basic, Perl, PHP, Javascript, Python
- Funzionali**
 - Lisp, Scheme, ML, Haskell, Erlang
- Logici**
 - Prolog...

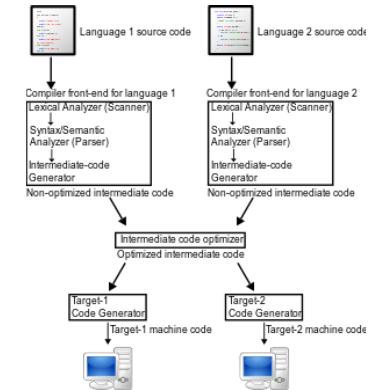
Esecuzione dei programmi

- Linguaggio ad alto livello → passi necessari:
 - Compilazione**, traduzione in linguaggio macchina
 - Collegamento** con librerie di supporto
 - Caricamento** in memoria
- Programmi **compilati**: applicati i 3 passi...
 - A tutto il codice; prima dell'esecuzione
- Programmi **interpretati**: applicati i 3 passi...
 - In sequenza, su ogni istruzione; a tempo di esecuzione



Compilazione

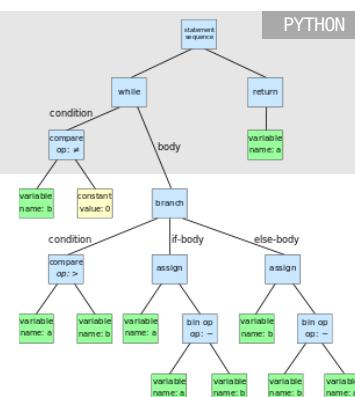
- Traduzione da ling. alto livello a ling. macchina
 - Analisi: lessicale, grammaticale, contestuale
 - Rappresentazione intermedia**: albero sintattico annotato (**AST**)
 - Generazione codice oggetto
- Codice oggetto: non ancora eseguibile
 - Linker, loader



Albero sintattico

```
while b != 0:
    if a > b:
        a = a - b
    else:
        b = b - a
return a
```

Algoritmo di Euclide per MCD



Collegamento

- Il **linker** collega diversi moduli oggetto
 - Simboli irrisolti → riferimenti esterni
 - Il collegamento può essere statico o dinamico
- Collegamento statico**
 - Libreria inclusa nel file oggetto, eseguibile stand-alone
 - Dimensioni maggiori, ma possibile includere solo funzionalità utilizzate
- Collegamento dinamico**
 - Librerie condivise da diverse applicazioni
 - Installazione ed aggiornamento unici
 - Caricate in memoria una sola volta

Caricamento

- Il **loader** carica in memoria un programma rilocabile
 - Risolti tutti gli indirizzi relativi (variabili, salti ecc.)
 - Caricati eventuali programmi di supporto
- Rilocazione statica**: indirizzi logici trasformati in indirizzi assoluti
- Rilocazione dinamica**: indirizzi logici mantenuti nel programma in esecuzione
 - Programma compilato: indirizzamento relativo
 - Tramite **registro base**: locazione del codice e dei dati in memoria (reg. CS e DS su x86)
 - Memory Management Unit** in S.O.

Codice gestito

- Compilazione in **codice intermedio**
 - Bytecode (Java), Common Intermediate Lang. (.NET), ...
 - Python: compilato per una macchina virtuale (file .pyc), ma in modo trasparente
- Esecuzione su una **macchina virtuale**, che gestisce la memoria (garbage collection)
 - Java Virtual Machine, Common Language Runtime, ...
 - Spesso compilazione "al volo" (*Just In Time*) in codice nativo
- Garbage collection** possibile anche per codice nativo
 - Linguaggio *Go*, "*smart pointers*" in C++, estensioni...

Distruzione degli oggetti

- La creazione di un oggetto richiede memoria per conservare lo stato dell'oggetto
- Quando un oggetto non serve più è necessario liberare questa memoria
- Vari linguaggi offrono un **garbage collector** per gestire in modo automatico la restituzione della memoria
- Quando un oggetto non serve più, il garbage collector lo distrugge
- Vantaggi
 - Non è possibile dimenticare di liberare la memoria (*memory leak*)
 - Non è possibile liberare della memoria che dovrà essere utilizzata in seguito (*dangling pointer*)
- Svantaggi
 - Il garbage collector decide autonomamente quando liberare la memoria
 - Liberare e compattare la memoria richiede del calcolo

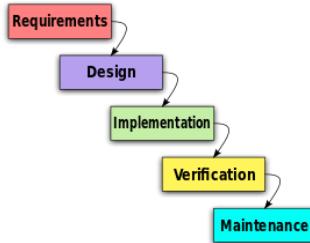
Liberazione automatica della memoria

- Garbage collection**: raccolta della spazzatura
- Non c'è bisogno di allocare o deallocare la memoria esplicitamente
 - La memoria è allocata alla creazione di un oggetto (con **new**)
 - Viene liberata quando non più utilizzata
 - Oggetti sempre in **heap**
 - Costruttori utili per l'inizializzazione
- L'algoritmo usato per la **garbage collection** è diverso sulle varie VM
 - Reference counting**: idea di base, ma cicli...
 - Generational garbage collection**: per oggetti creati di recente
 - Mark & sweep**: per oggetti più duraturi



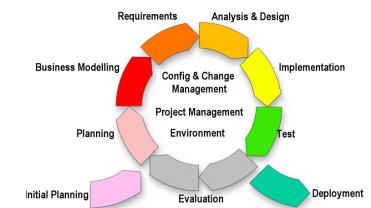
Ciclo di vita del software

- **Analisi**
 - Modello, requisiti, fattibilità
- **Progetto e implementazione**
 - Componenti architetturali, dettaglio classi
- **Collaudo**
 - Rispetto dei requisiti, qualità del sw
- **Rilascio e manutenzione**
 - 70-75% del costo totale (fonti DoD USA, HP)
 - Anomalie ed errori (manutenzione *correttiva*, ~20%)
 - Prestazioni, qualità, funzionalità (m. *perfettiva*, ~60%)
 - Mutamenti dell'ambiente (m. *adattativa*, ~20%)



Evoluzione di un sistema sw

- Evoluzione ineliminabile per molti sistemi
 - Non noti o non colti correttamente i requisiti
 - Cambiano le condizioni operative ...
- Sviluppo iterativo e metodologie agili
 - Rilascio frequente ed incrementale
 - <http://agilemanifesto.org/>



Sviluppo ad oggetti

Approccio allo sviluppo

- Programmazione procedurale: **top-down**
 - Analisi problema x trovare algoritmo risolutore
 - **Scomposizione** problema in problemi più semplici (*loop*)
- OOP: top-down...
 - Analisi problema e descrizione degli oggetti che ne fanno parte (astrazioni generali e coese)
 - Scomposizione oggetti in parti più semplici (e messaggi/servizi scambiati) (*loop*)
- OOP: ... e **bottom-up** (riuso)
 - Definizione e riuso oggetti relativamente semplici
 - **Composizione** in oggetti più complessi (*loop*)

Modularità e astrazione

- Modularità e testabilità
 - Oggetto, completo di dati ed operazioni, inserito facilmente tra gli altri componenti di un sistema
 - Codice di un oggetto scritto e mantenuto indipendentemente dal resto
 - Più facile isolare e risolvere i problemi
- Information-hiding e **astrazione**
 - Interazione solo con i metodi di un oggetto (black-box)
 - Dettagli interni dei dati e dell'implementazione nascosti al mondo esterno (**incapsulamento**)
 - Oggetto: tipo di dato astratto (ADT)

Riuso e sostituibilità

- Riuso del codice
 - Oggetto già esistente → encapsulato in altri oggetti, per altri programmi (**composizione**)...
 - Anche se sviluppato (progetto/implem./test/debug) da altri, specialisti di un certo dominio
 - Oggetti di nuovo tipo creati come estensione di tipi esistenti (**ereditarietà**)
 - Sistemi generici basati su astrazioni (**generalizzazione**), adattati implementando date interfacce
- Sostituibilità
 - Oggetti di tipo diverso possono fornire gli stessi metodi, ed essere gestiti in maniera astratta (**polimorfismo**)
 - Nuovo oggetto (che implementa i metodi richiesti) inserito in un sistema astratto già esistente

Associazioni e relazioni in UML

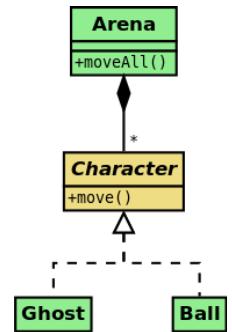
A ————— B	Generic association -- A is associated with B
A —→ B	Directed assoc (has-ref-to) -- Can navigate from A to B
A ————— ◊ B	Aggregation -- A is contained without belonging to B
A ————— ♦ B	Composition -- A is contained and is owned by B
A ——> B	Dependence -- A depends (param, var, ret-val) upon B
A —————▷ B	Inheritance (is-a) -- A inherits from B
A —————○ B	Implementation -- A has interface B
A ——>▷ B	Realization -- A is a realization of the abstract class B
A —————→ B	Synch msg -- A sends a synchronous message to B
A —————→ B	Asynch msg -- A sends an asynchronous message to B

Dipendenza e associazione

- **Dipendenza**: un metodo di una classe ha come argomento, valore di ritorno o var locale un'altra classe (con cui non è associata)
- **Associazione**: una classe ha come campo uno o più oggetti di un'altra classe
 - Ognuno dei lati
 - Ha un nome
 - Ha una cardinalità
 - Può o no essere navigabile
 - Un'associazione equivale a codice
 - Solo i lati navigabili implementano l'associazione
 - Memorizzazione in **array** o altra **collezione** (se cardinalità > 1)
 - Controllare il vincolo sulla cardinalità
 - Dipendenze e associazioni limitano la riusabilità perché **accoppiano** le classi

Contenimento

- Associazione che esprime un contenimento fisico, o in generale un legame di tipo **has-a**
- Può essere **composizione** o **aggregazione**
 - **Composizione**: ciclo di vita dell'oggetto contenuto determinato dal contenitore (legame **whole-part**, **owns-a**)
 - **Aggregazione**: oggetto contenuto non rigidamente legato al contenitore
- Spesso nel codice non è chiara la differenza tra contenimento ed un più generica associazione



Composizione

- Idealmente, un oggetto (creato e testato) rappresenta una unità di codice, che altri oggetti possono usare
 - Il riuso in progetti diversi non è semplice da ottenere...
- Inserire un oggetto (**member object**) dentro un'altro
 - Il nuovo oggetto può contenere un certo numero di oggetti di tipo diverso, per realizzare le funzionalità desiderate
 - Relazione **whole-part** o **owns-a**
- Grado elevato di flessibilità
 - Gli oggetti membri sono di solito nascosti
 - Inaccessibili ai programmati che usano l'oggetto
 - Possono essere cambiati senza disturbare il codice esterno

Associazioni e attributi

- Associazioni e attributi del modello generano codice
- Le **associazioni** vengono usate per collegare tra loro le **classi** del modello
- Gli **attributi** sono tipi di base (o **primitivi**)
 - Come **int**, **float** o **string**
 - Classi di base, usate pervasivamente
 - Classi di altre librerie, non evidenziate nel modello

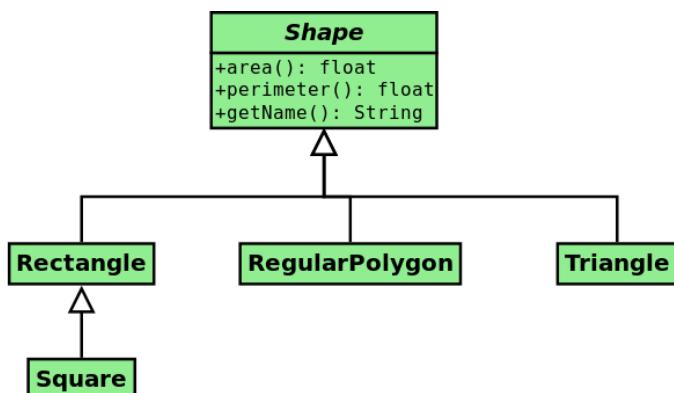
Classe derivata

- Specializzazione: relazione *is-a* tra classe derivata e classe base
- Principio di sostituibilità tra classi
 - È sempre possibile usare una classe derivata al posto di una classe base
- La classe derivata
 - Eredita tutte le caratteristiche **public** della classe base
 - Non può accedere alle caratteristiche **private** della classe base
 - Può dichiarare nuove caratteristiche che non sono visibili dalle classi base (Eckel: *is-like-a*)
- La classe base
 - Può definire delle caratteristiche **protected** a cui solo lei e le classi derivate possono accedere

Classe astratta

- Contiene metodi che possono essere implementati solo dalle sue classi derivate
 - Informazioni non sufficienti nella classe base
 - Meccanismi specifici per implementare un metodo
 - ...
- Una classe astratta non può essere istanziata

Figure geometriche



Ereditarietà e riusabilità

- Strutture dati ed algoritmi possono essere implementati in funzione della classe **Shape**
 - Anzichè di una specifica classe derivata, come **Rectangle**
 - Ad esempio, l'ordinamento può sfruttare il fatto che tutte le figure hanno un area
- Massimizzazione di riuso e flessibilità
 - Dipendenza dalla classe più alta nella gerarchia...
 - Che offre le caratteristiche richieste

A cosa serve l'ereditarietà?

- Due utilizzi principali
 - Modellare il problema (o la soluzione), molto importante nella fase di analisi
 - Massimizzare il riuso, molto importante nella fase di progettazione
- I due utilizzi sono legati perché la prima bozza di un progetto è il modello che analizza il dominio del problema (o della soluzione)
- Eckel: **ereditarietà o composizione?**
 - *"Do I need to upcast?"*

Ereditarietà multipla

- Un quadrato è contemporaneamente...
 - Un rettangolo
 - Un poligono regolare
- La classe **Square** dovrebbe estendere sia **RegularPolygon** che **Rectangle**
 - Quale stato usare?
 - Quale metodi eseguire?
- A volte ereditarietà multipla non ammessa o non conveniente
 - No ambiguità, no "**diamond problem**"

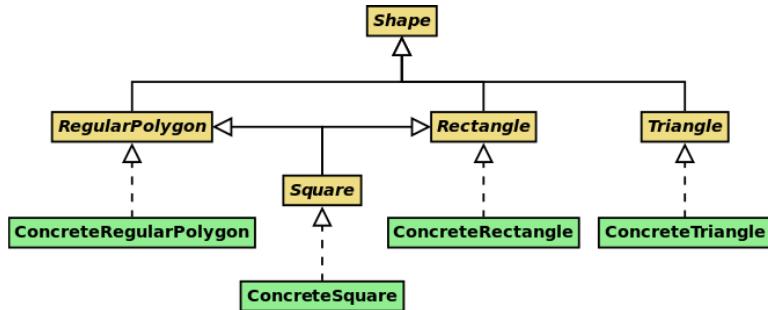
Interfaccia

- Interfaccia di un oggetto: descrizione dei suoi metodi pubblici
 - Modo che ha per interagire con il mondo
 - Servizi che offre agli altri oggetti
- I corpi dei metodi, cioè come i servizi vengono implementati, non sono parte dell'interfaccia
 - L'interfaccia indica cosa un oggetto sa fare e non come lo fa
- Interfaccia di un rettangolo
 - `float area()`
 - `float perimeter()`

Classe astratta pura

- Una interfaccia è una classe astratta pura
 - Tutti i metodi sono astratti
 - Implementazione in classe concreta
 - Libertà su come memorizzare stato ed implementare metodi
- Usando le interfacce
 - Migliore pulizia del modello ed aderenza alla realtà modellata
 - Possibilità di migliorare la riusabilità

Implementazioni di Shape



Framework e librerie

- Ma introdurre un'interfaccia per ogni classe non è sempre l'approccio migliore!
 - Mantenere basso il numero di livelli d'astrazione
 - Può bastare **Shape**, + classi concrete?
- **Libreria**: l'utente usa...
 - Poche e semplici interfacce (astrazioni chiave)
 - Implementazione dietro le quinte (**factory**)
 - È sempre il codice utente che chiama la libreria
- **Framework**: l'utente fornisce...
 - Implementazione di metodi di certe classi astratte
 - Classi che implementano certe interfacce
 - Il framework può chiamare il codice utente

Generalizzazione e riuso

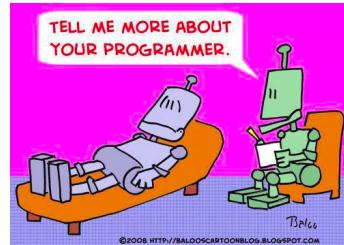
- Riuso tramite **ereditarietà**
 - Aggiungere nuovi servizi a classi già esistenti
 - Si riutilizza il codice della classe esistente
 - Si possono sfruttare anche parti **protected** della classe base
 - Ma si eredita anche l'**interfaccia** pubblica!
- Riuso tramite **generalizzazione**
 - Costruire sistemi che operano su astrazioni di alto livello
 - Specializzati mediante oggetti che implementano date interfacce
 - Si riutilizza un intero sistema, modificandone il comportamento
 - Oggi, composizione e generalizzazione sono i meccanismi di riuso più apprezzati



Specifiche e contratti

Qualità del software

- Le qualità su cui si basa la valutazione di un sistema software possono essere:
 - Interne**, riguardano le caratteristiche legate al **processo** di sviluppo e non sono direttamente visibili agli utenti
 - Esterne**, riguardano le funzionalità fornite dal **prodotto** sw e sono direttamente visibili agli utenti
- Le categorie sono legate:
 - Product quality is process quality*



Qualità esterne

- Correttezza e affidabilità**: il sistema rispetta le specifiche, l'utente può affidarsi al programma
- Robustezza**: il sistema si comporta in modo ragionevole anche fuori dalle specifiche
- Efficienza**: usa bene le risorse di calcolo
- Scalabilità**: migliori prestazioni con più risorse
- Sicurezza**: riservatezza, autenticazione, autorizzazione, accounting
- Facilità d'uso**: interfaccia utente permette di interagire in modo naturale

Qualità interne

- Verificabilità**: sistema basato su modello formale
- Riusabilità**: parti per costruire nuovi sistemi
- Manutenibilità**: riparabilità, evolvibilità (nuove specifiche), adattabilità (cambiamenti ambiente)
- Interoperabilità**: capacità di co-operare con altri sistemi, anche di altri produttori
- Portabilità**: adatto a più piattaforme hw/sw
- Comprendibilità**: codice leggibile, documentato
- Modularità**: interazione tra componenti coesi

Specifiche

- Rispetto a cosa valutiamo **correttezza** o **affidabilità** di un programma?
- Idea del programmatore
 - Non formulata, non documentata
 - Incompleta, mutevole, facilmente dimenticata
- Specifiche (formali o informali)
 - Formulate, scritte, studiate e condivise
→ Parte del progetto e del programma
 - Spec. assiomatiche: espressioni logiche o asserzioni
→ **Precondizioni, postcondizioni e invarianti**



Pre- e post-condizioni

- **Precondizioni**
 - Stabiliscono se è possibile chiamare un metodo
 - Prerequisiti per l'attivazione
- **Postcondizioni**
 - Stabiliscono se il metodo restituisce il valore atteso, cioè se produce l'effetto desiderato
 - ... In relazione ai parametri (che soddisfano le precondizioni)
 - Definiscono il significato del metodo
- **Divisione delle responsabilità** tra moduli
 - Errore del codice *chiamante (client)* se precondizioni non soddisfatte
 - Errore del codice *chiamato (server)*, se postcondizioni non soddisfatte

Responsabilità e contratti

- **Precondizioni + postcondizioni = contratto**
 - ... tra modulo chiamante e modulo chiamato
- Infrazione di un contratto: problema serio
 - Errore rispetto alle specifiche
 - Eccezione e/o terminazione
- No **divisione responsabilità** → sovrapposizioni
 - Tutti i moduli assumono molte responsabilità
 - Programmazione difensiva: tutte le parti del programma controllano tutte le condizioni
 - Grosso programma → ancora più grosso

Esempio di contratto

```
def sqrt(x: float) -> float
```

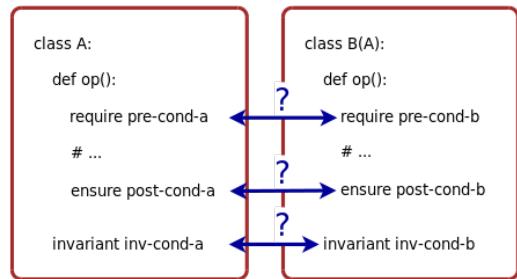
PYTHON

- Precondizioni: $x \geq 0$
- Postcondizioni: $\text{abs}(\text{result} * \text{result} - x) \leq 0.00001$
- Codice chiamante
 - Obblighi: deve passare un numero non negativo
 - Benefici: riceve la radice del numero
- Codice chiamato
 - Obblighi: restituisce un numero r tale che $r * r \approx x$
 - Benefici: può assumere che x non è negativo

Invariante di classe

- Vincolo che deve valere per ogni stato stabile di un oggetto, durante tutto il suo ciclo di vita
- Rafforzamento generale di pre- e post-condizioni
- "Criterio di sanità" dell'oggetto
- Deve essere soddisfatto dal costruttore
- Deve essere mantenuto dai metodi pubblici
- Ma non necessariamente da metodi privati o protetti

Ereditarietà e contratti



- Che relazione c'è tra le asserzioni di una classe e quelle dei suoi discendenti?

Principio di sostituibilità

- Polimorfismo: possibile esecuzione metodo di una sottoclasse, anziché della classe base
 - I metodi delle sottoclassi possono ridefinire i metodi delle classi base... ma non arbitrariamente
- I contratti della sottoclasse devono *rispettare i contratti della classe base* ("sottocontratti")
 - Precondizioni: non devono essere più forti
 - Postcondizioni: non devono essere più deboli
 - Invarianti di classe: non devono essere più deboli

“Require no more, promise no less”

Design by contract

- Paradigma proposto nel linguaggio **Eiffel** (Betrand **Meyer**, 1986)
- Uso di asserzioni in varie fasi di sviluppo
 - Progetto: approccio pragmatico alle specifiche
 - Implementazione: guida per la programmazione
 - Documentazione: interfacce con info aggiuntive
 - Collaudo: DbC delimita i casi da testare (per affidabilità)
 - Manutenzione: DbC fa emergere prima gli errori
 - Uso finale: sollevate eccezioni se violazioni

Asserzioni Python

- Espressioni booleane, simili a predicati matematici
- Esprimono proprietà semantiche di classi e metodi
- Utili per collaudo e debugging, ma anche documentazione
- Violazione → **AssertionError** (e normalmente **abort**, terminazione programma)

`assert age > 0`

PYTHON

Asserzioni e contratti

- Asserzioni in genere utili per:
 - Precondizioni, postcondizioni, invarianti di classe
 - Invarianti interne e di controllo del flusso
- Argomenti di metodi pubblici sbagliati → eccezione
 - `ValueError` o `TypeError`
 - Di solito, asservimenti usate per debug...

Pre- e post-condizioni

```
def sqrt(x: float) -> float:  
    ...  
    Precondition: x >= 0  
    Postcondition: abs(result * result - x) <= 0.0001  
    ...  
    if x < 0 raise ValueError("sqrt: arg < 0")  
    # ...  
  
    assert abs(result * result - x) <= 0.0001  
    return result
```

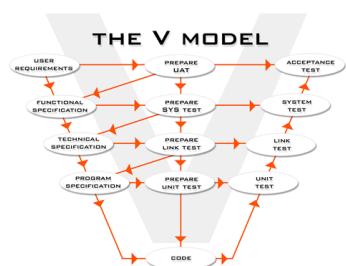
PYTHON

Verifica e validazione



Verifica e validazione

- Mostrare che il sistema...
 - È conforme alle specifiche
 - Soddisfa i bisogni dell'utente
- Comprende revisione e collaudo del sistema
- **Test case**, derivati dalle specifiche



Costo dei bug

- Scovare bug non è un compito facile, e nemmeno una esperienza eccitante...

- Costoso: non è insolito dedicare al testing il 40% del tempo e delle risorse di un progetto

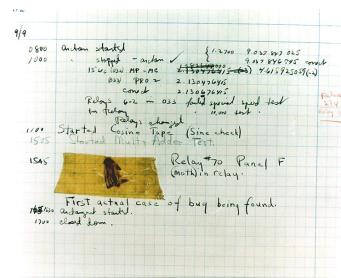
- Far emergere** bug in prime fasi dello sviluppo!

- B. Boehm: se trovare e correggere un problema in fase di specifica dei requisiti costa 1\$...

- \$5 in progetto, \$10 in programmazione,

 - \$20 in unit testing, fino a \$200 dopo consegna

- Alcuni bug possono capitare già a causa di specifiche non ben chiare e capite



Prove formali

- Dimostrazione matematica di un programma: alternativa (~ accademica) al testing

- Annotazione del programma con asserzioni matematiche: comportamento atteso

- Proprietà valide per i vari costrutti del programma

- Prova che post-condizioni verificate, se:

- Precondizioni verificate

 - Programma termina

- Dimostrazioni automatiche

- Se a mano → errori (più che nel programma?)



61/88

Revisione del software

- Analisi del codice (o pseudocodice) per capirne le caratteristiche e le funzionalità

Code walk-through

- Selezione porzioni di codice e valori di input

 - Simulazione su carta comportamento del sistema

Code inspection, più formale e focalizzato

- Uso di variabili non inizializzate

 - Loop infiniti
 - Letture di porzioni di memoria non allocata
 - Rilascio improprio della memoria

Testing

“Le operazioni di testing possono individuare la presenza di errori nel software ma non ne possono dimostrare la correttezza. (E. Dijkstra)”

“Eseguire un programma con l'intento di trovare errori. (Glen Myers, “The art of Software Testing”)”

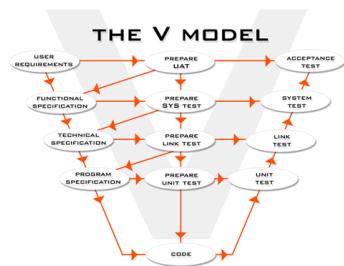
- Verificare sistema in un insieme abbastanza ampio di casi... → plausibile comportamento analogo anche nelle restanti situazioni



63/88

Classificazione dei test

- Tipi di test
 - **White box** (*in the small*)
 - **Black box** (*in the large*)
- Livelli di test
 - *Unit test*
 - *Integration test*
 - *System test*
- Ripetizione di test
 - *Regression test*



Testabilità

- Qualità software che facilitano rilevazione errori
 - **Osservabilità** – Disponibili i risultati dei test
 - **Controllabilità** – Possibilità di impostare ingressi e stato del programma prima di eseguire un test
 - **Decomponibilità** – Programma diviso in parti che possono essere testate individualmente
 - **Comprensibilità** – Si capisce il comportamento corretto (desiderato) del programma
- → Sviluppo per testabilità

White-box testing



White-box testing

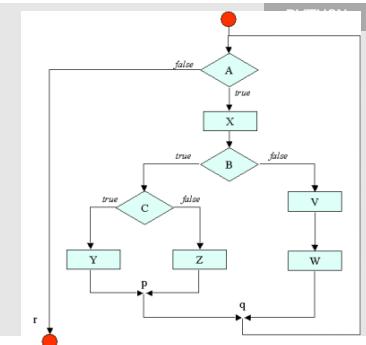
- Test basati sulla conoscenza della struttura interna del codice
- Un errore non può essere scoperto se la parte di codice che lo contiene non viene mai eseguita
- **Statement test**
 - Insieme di test T tali che, eseguendo su tutti i casi di T il programma P , ogni istruzione di P venga eseguita almeno una volta (test utopia?)
 - **Branch test** (copertura delle decisioni)
 - **Branch & condition test** (... condizioni)

Basic path testing

- Scelto insieme minimo di percorsi per coprire tutte le istruzioni e condizioni (**white box**)
 - Tracciare diagramma di flusso
 - Astrarre il diagramma in un grafo di flusso
 - Complessità ciclomatica n = metrica di test
 - Trovare n casi di test che seguono ciascun cammino indipendente
- Cammino: sequenza di comandi, da inizio a fine
- Cammino indipendente: aggiunge almeno una nuova istruzione rispetto ai cammini già identificati

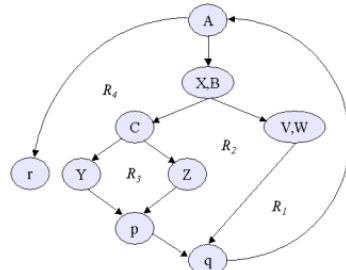
Diagramma di flusso

```
def f():
    // entry
    while a:
        x()
        if b:
            if z: y()
            else z()
        else:
            v()
            w()
        # q
    # exit: r
```



Grafo di flusso

- Piccola astrazione rispetto a diagramma di flusso
- Complessità ciclomatica**, dalla teoria dei grafici:
 - Numero di possibili cammini indipendenti, o...
 - Numero di regioni del grafo di flusso, o...
 - Numero di nodi predicato + 1
- A, r
- A, X, B, C, Y, p, q, A, r
- A, X, B, C, Z, p, q, A, r
- A, X, B, V, W, q, A, r



Black box testing

Black box testing

- Sistema = scatola nera; si verificano le corrispondenze di input e output
 - White-box testing: impossibile per grandi sistemi
 - Test case scelti in base alle specifiche dei requisiti
- Desiderata: trovare errori...
 - Funzionali: otteniamo i risultati attesi per dati input di un metodo?
 - Interfaccia: dati passati correttamente tra i metodi?
 - Efficienza: il metodo è abbastanza veloce?

Partizioni d'equivalenza

- Partizionamento ingressi in **classi di equivalenza**
 - Irrealistico testare tutti i possibili ingressi (es. `sqrt`)
 - Ipotesi: sufficiente testare un solo caso per classe
 - Si includono casi limite e valori non validi
 - Precondizioni: riducono il numero di casi di test

```
def swap_elements(v: list, i: int, j: int):
    ...
    Exchange element i and j in list v
    v: empty, one element, more elements
    i, j: one or both indexes out of range... or both in range: i < j, i > j, i = j
    ...
# ...
```

PYTHON

Regression testing

- Scopo: trovare errori di regressione
 - Errori in un programma che prima era corretto, ed è stato modificato di recente
 - Un errore di regressione è un errore che prima non c'era
- Dopo la modifica di una parte **P** nel programma **Q**
 - Testare che la parte **P** funzioni correttamente
 - Testare che l'intero programma **Q** non sia stato danneggiato dalla modifica



DII

Collaudo in Python

Come collaudare il codice?

- Usare un **debugger** per valutare espressioni in fase di esecuzione
 - Si può decidere cosa valutare a seconda del flusso di esecuzione e dei valori generati, senza ricompilare
- Istruzioni di **stampa** all'interno del programma
 - Valore di espressioni scritto a console o su file di log
- Entrambi gli stili, scarsamente **automatizzati**
 - Necessità di intervento attivo durante l'esecuzione dei test
 - Giudizio dei risultati da parte dell'utente
 - Quali valori analizzare? Sono coerenti?
- Scarsamente **componibili**
 - Difficile controllare molte espressioni nel debugger
 - "*Scroll blindness*": troppe istruzioni di stampa ⇒ codice poco leggibile

Libreria unittest

- I test **unittest** non richiedono continuo intervento o giudizio da parte dell'utente
- Facile eseguire molti test assieme, su un certo progetto
- Come definire un test?
 - Creare una sottoclasse di `unittest.TestCase`
 - Scrivere metodi di test, denominati con prefisso `test`
 - Per controllare la validità di una espressione, usare `assertTrue(bool)`

Esempio di test

- Controllare che una pallina rimbalzi correttamente contro il bordo inferiore

```
import unittest  
  
class SimpleBallTest(unittest.TestCase):  
  
    def test_bounce_down(self):  
        b = Ball(8, 11) # dx = 1, dy = 1, w = 16, h = 12  
        b.move()  
        self.assertTrue(b.position == (9, 10))  
  
if __name__ == '__main__':  
    unittest.main(exit=False)
```

PYTHON

Esecuzione dei test

- Meccanismi per definire i test da eseguire e organizzare i risultati
- Esecuzione di test dalla linea di comando
 - Inclusione dei test di un modulo, di una classe, o metodi di test specifici
 - Implementata anche una semplice forma di **test discovery**

```
python -m unittest test_module1 test_module2  
python -m unittest test_module.TestClass  
python -m unittest test_module.TestClass.test_method  
python -m unittest discover
```

CMD

- Annotazione `@unittest.skip("reason for skipping")`
 - Indica al framework di ignorare un certo metodo di test
 - Si può anche passare un messaggio per documentare la decisione

Controllare le eccezioni

- Mestiere del programmatore
 - Codice che completa correttamente l'esecuzione nei casi normali...
 - Ma che anche in situazioni eccezionali mostra il comportamento atteso
- Come verificare che una eccezione attesa sia effettivamente sollevata?
 - Usare il metodo `assertRaises` direttamente, passando una funzione ed i parametri
 - Oppure creare un `contesto` con `with`
- Esempio: `Ball` solleva effettivamente una eccezione attesa?

```
def test_out_of_arena(self):
    with self.assertRaises(ValueError):
        b = Ball(-1, -1)
```

PYTHON

80/88

Test parametrizzati

- Ripetere un test con diversi parametri
 - Un test case per ogni gruppo di parametri?
 - In alcune applicazioni, enorme quantità di test!
- Soluzione semplicistica: test contente un ciclo
 - Ad ogni iterazione, preparato un gruppo di parametri diversi
 - Eseguite le istruzioni da testare sui nuovi parametri
 - Problema: il test si blocca al primo errore

 www.ce.unipr.it/people/tomamic

 www.ce.unipr.it/people/tomamic

81/88

Test parametrizzato, semplicistico

```
class ParamBallTest(unittest.TestCase):
    TEST_VALUES = ( (8, 4, 1, 1, 9, 5),
                    (8, 11, 1, 1, 9, 10),
                    (15, 4, 1, 1, 14, 5),
                    (0, 4, -1, 1, 1, 5),
                    (8, 0, 1, -1, 9, 1) )

    def test_move(self):
        w, h = 16, 12
        for param in ParamBallTest.TEST_VALUES:
            x0, y0, dx, dy, x1, y1 = param
            b = Ball(x0, y0, dx, dy, w, h)
            b.move()
            self.assertTrue(b.position == (x1, y1))
```

PYTHON

82/88

Sotto-test, Python 3.4

- Eseguiti tutti i sottotest, anche se uno fallisce

```
class ParamBallTest(unittest.TestCase):
    TEST_VALUES = () # same values...

    def test_move(self):
        w, h = 16, 12
        for param in ParamBallTest.TEST_VALUES:
            with self.subTest(param=param):
                x0, y0, dx, dy, x1, y1 = param
                b = Ball(x0, y0, dx, dy, w, h)
                b.move()
                self.assertTrue(b.position == (x1, y1))
```

PYTHON

 www.ce.unipr.it/people/tomamic

 www.ce.unipr.it/people/tomamic

83/88

Fixture

- Due o più test operano su insiemi di oggetti uguali o simili
 - Questa configurazione iniziale comune si definisce **fixture**
 - Ogni test: operazioni o parametri leggermente diversi sulla fixture e risultati diversi
- Spesso più tempo per scrivere il codice della fixture che i test veri e propri
 - Creazione della fixture facilitata da attenzione in fase di progetto
 - Ancora meglio se il codice della fixture non è ripetuto

Definire una fixture

- Se ci sono diversi test con una fixture comune...
 - Aggiungere dei campi per le varie parti della fixture
 - Inizializzare questi campi, nel metodo **setUp**
 - Liberare eventuali risorse allocate, nel metodo **tearDown**
- Una volta creata la fixture, può essere usata da tutti i test case
 - Aggiungere metodi di test alla classe

Esempio di fixture

- Numerosi metodi di test che operano su stessi dati iniziali
 - Esempio, una combinazione di palline in posizioni predefinite

```
class SimpleBallTest(unittest.TestCase):  
  
    def setUp(self):  
        self.b1 = Ball(8, 4)  
        self.b2 = Ball(4, 8)  
        self.b3 = Ball(12, 2)
```

PYTHON

Test suite

- Meccanismo per **raggruppare** logicamente dei test ed **eseguirli assieme**
- La classe **TestSuite** rappresenta una test suite
 - Lista di classi di test aggiunte con il metodo **addTest**
 - `suite.addTest(unittest.makeSuite(SimpleBallTest))`
- La classe **TestRunner** rappresenta un esecutore di test
 - Per la console, si usa **TextTestRunner**, già inclusa nel framework

```
suite = unittest.TestSuite()  
suite.addTest(SimpleBallTest)  
runner=unittest.TextTestRunner()  
runner.run(suite)
```

PYTHON

Sistemi di elaborazione



<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR



Introduzione all'informatica

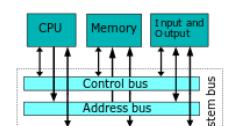
Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Calcolatore

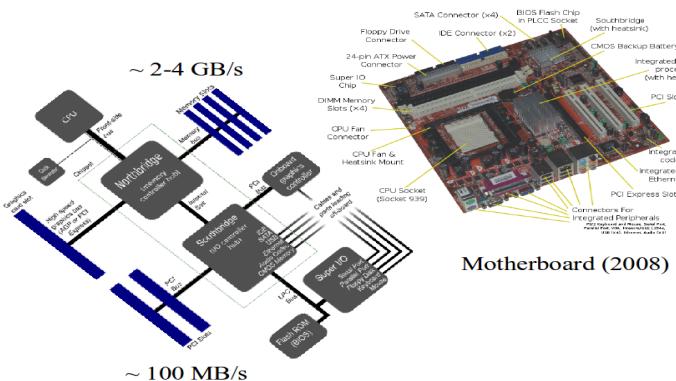
- Macchina programmabile
 - Memorizza ed elabora automaticamente...
 - Attraverso istruzioni di un programma...
 - Informazioni in formato digitale (I/O)
- Diversi modello di calcolo
 - Definizione di operazioni elementari e concetto stesso di algoritmo
 - Definizione di problema risolvibile / irrisolvibile (calcolabilità): dipende dal modello di calcolo
- Macchina di Turing
 - Modello teorico (A. Turing, 1936)
 - Tesi di Church-Turing: problema intuitivamente calcolabile → esiste TM in grado di calcolarlo

Macchina di von Neumann

- 1941, Z3 (Berlino)
 - A relais, programma su nastro esterno
- 1946, ENIAC (Philadelphia)
 - Balistica, meteorologia, reazioni nucleari
 - Programmazione con cavi
- 1948, Manchester "baby"
 - Programma in memoria
- 1951, sistema IAS (Princeton)
 - Dati e programmi in memoria centrale



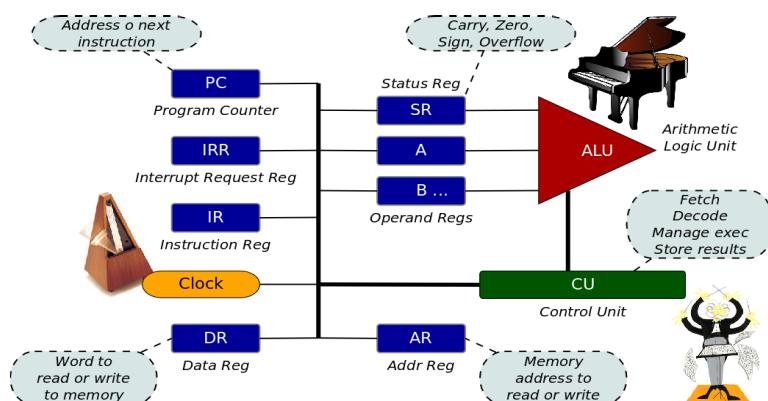
Northbridge e southbridge



Central Processing Unit

- CPU: "cervello" del calcolatore
 - Esegue i programmi
 - Comanda le altre parti del calcolatore
- Composta da due parti:
 - **Control Unit (CU):** interpreta le istruzioni, comanda le altre parti della CPU, controlla il flusso tra CPU e memoria
 - **Arithmetic Logical Unit (ALU):** esegue le operazioni aritmetiche e logiche, esegue i confronti tra dati

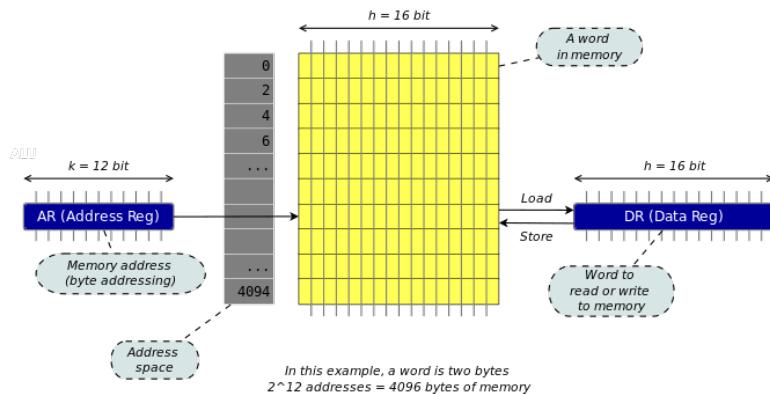
Architettura CPU



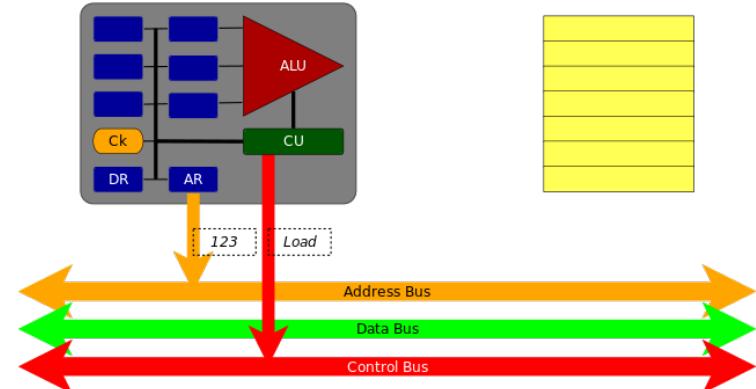
Ciclo principale della CPU

- **Caricamento:** CU preleva l'istruzione dalla locazione di memoria indicata dal registro PC (Program Counter) e la memorizza in IR (Instruction Register)
- **Decodifica:** CU interpreta l'istruzione, legge eventualmente dalla memoria i dati necessari
- **Esecuzione:** CU comanda le parti
- **Memorizzazione:** risultati memorizzati nella memoria centrale o in registri della CPU

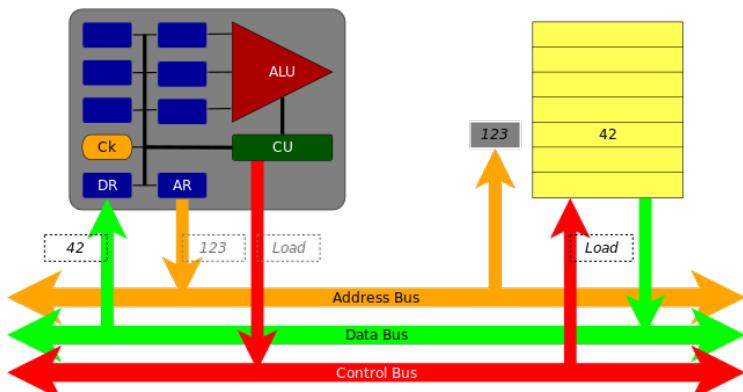
Lettura dalla memoria



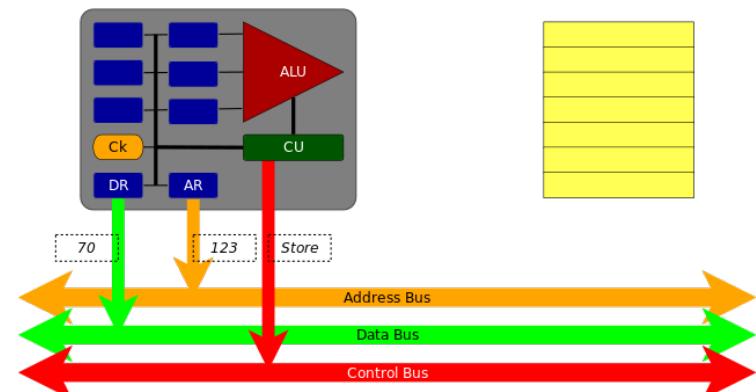
Sequenza di lettura



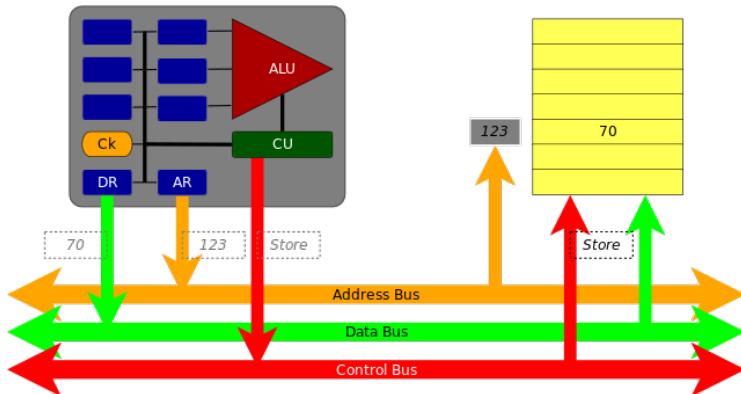
Sequenza di lettura



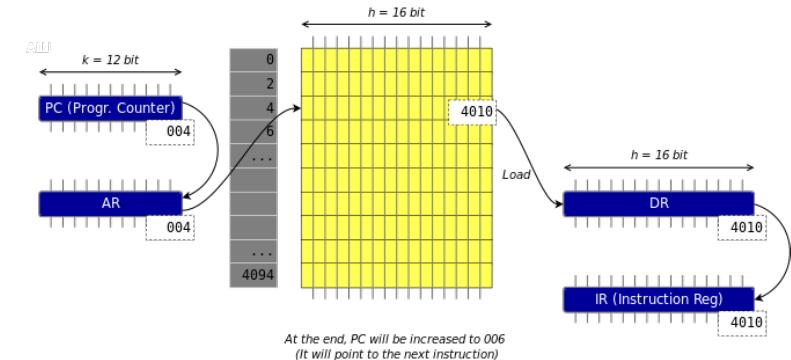
Sequenza di scrittura



Sequenza di scrittura



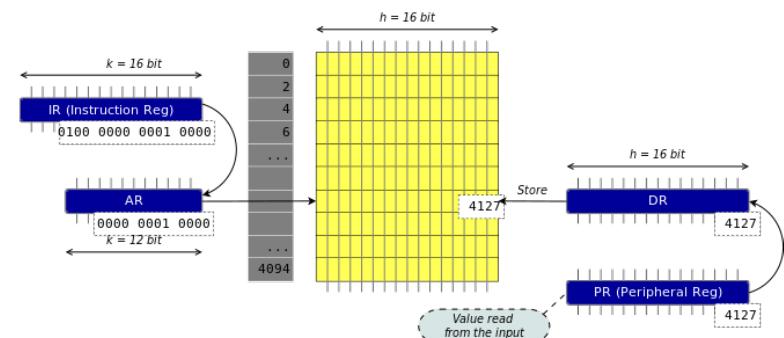
Fetch istruzioni



Interpretazione istruzioni

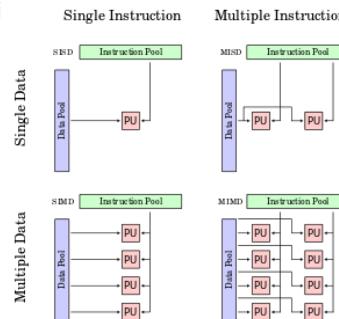
- Al termine della fase di *fetch*, IR contiene l'istruzione da eseguire
 - Codice operativo + operandi**
 - Linguaggio macchina**: il significato dipende dalla CPU
- Nell'esempio: **4 010** (16) = **0100 0000 0001 0000** (2)
 - Codice operativo = **0100**
 - Es. Leggi una parola dal registro delle periferiche...
 - E memorizzala in un indirizzo di memoria (operando)

Esecuzione istruzioni



Classificazione di Flynn

- **Parallelismo:** migliori prestazioni, a parità di velocità sulla singola istruzione
- **SISD:** una operazione alla volta; macchine tradizionali a singolo processore e core
- **MISD:** insolite, per tolleranza ai guasti; sistemi eterogenei, sugli stessi dati, devono dare gli stessi risultati
- **SIMD:** operazioni naturalmente parallelizzabili; unità di calcolo vettoriale e GPU
- **MIMD:** istruzioni diverse su dati diversi; architetture con più core o processori autonomi, sistemi distribuiti



Assembler

- **Linguaggio macchina:** definisce il set di istruzioni comprensibile dalla CPU
 - **CISC:** Complex Instruction Set Computing
 - **RISC:** Reduced instruction set Computing
- Assembler: traduce da **linguaggio assembly** (mnemonico) a linguaggio macchina (mapping 1~1)
- Es. **Assembly x86** → macchina (istruzioni di varia lunghezza)

MOV AH, 11 → 1011 0 100 00001011

ASSEMBLY

- 4 bit di op-code (**1011**), tipo di istruzione
- Bit **w (0)**: operazione a 8 o 16 bit, (0 o 1 resp.)
- 3 bit per registro destinazione (**100**)
- 8 bit di dato per operando: **11₍₁₀₎** = **00001011₍₂₎**



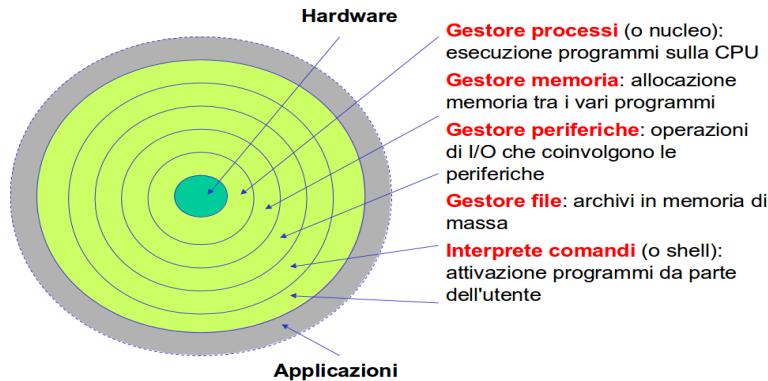
Sistema operativo

Sistema operativo

- Avviato automaticamente all'accensione
- Intermediario tra utente, applicazioni ed hardware
- Funzionalità: a lotti (**batch**), o interattivi
- Accesso utenti: mono-utente, o multi-utente
- Gestione risorse: mono-programmazione, **multi-programmazione** (time-sharing), **multi-elaborazione** (multi-core...)



Sistema operativo a cipolla



Gestione dei processi

- Processo: istanza di un programma, in esecuzione
- S.O. multi-tasking: esegue più programmi in contemporanea, associati a processi
 - Crea e cancella i processi
 - Decide a quale processo assegnare la CPU
 - Sospende e riattiva i processi
- Inoltre offre meccanismi per:
 - Sincronizzazione** e gestione **deadlock**
 - Comunicazione** fra processi
 - Comunicazione con **periferiche**

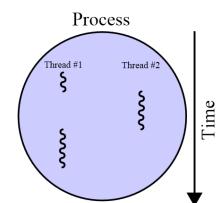
Scheduling

- CPU singola: non reale parallelismo (in un dato istante, un solo processo in esecuzione)
- Time-sharing:** ciclicamente, assegnato il processore per un *time-slice* (intervallo fisso ~10-100ms)
- Se entro l'intervallo il processo termina
 - Avviato nuovo intervallo, eseguito altro processo
- Se non termina o si blocca (attesa risorsa)
 - Processo sospeso, avviato nuovo intervallo, eseguito un altro processo (*preemptive* O.S.)



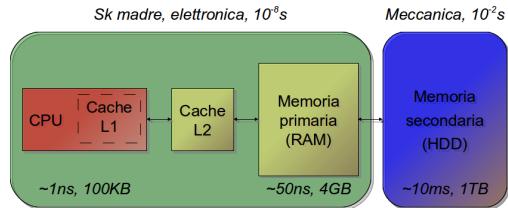
Thread di esecuzione

- Più piccola unità di istruzioni che può essere schedulata dal S.O.
- Possibile dividere un processo in diversi compiti, in esecuzione concorrente
 - Stessa base di codice
 - Condivisione di **heap** e di risorse → **Mutex e semafori**
 - Diverso **stack** e **stato CPU**



Memoria virtuale

- Uno o più programmi: richiesta più memoria di quella disponibile (primaria)
- O.S. cerca di simulare una memoria **grande, economica e veloce**
 - Gestione della gerarchia di memoria primaria/secondaria
 - Libera programma (e programmatore) da dim memoria fisica



Paginazione

- Memoria divisa in pagine di uguale dimensione
- Ad ogni programma in esecuzione assegnato un certo numero di pagine
- Quando l'istruzione da eseguire non è in memoria primaria (**page fault**)...
 - Se necessario, una pagina viene spostata da memoria primaria a secondaria
 - La pagina che contiene l'istruzione viene trasferita in memoria primaria
 - Principio di **località** (dati e riferimenti di solito raggruppati)
- Politica di paginazione
 - Di solito **LRU** (*Least Recently Used*)...
 - Anziché **FIFO** (*First In First Out*)

Gestione periferiche

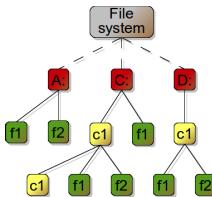
- Procedure I/O ad alto livello per interazione tra programmi e periferiche
- BIOS** (Basic I/O System, firmware)
 - Diagnostica e inizializzazione dispositivi interni (es. controllo memoria)
 - Avvio sistema operativo (da Boot Sector)
- Driver**: programmi software che permettono l'accesso ad una specifica periferica
- In genere periferiche caratterizzate da:
 - Velocità gestione dati << CPU
 - Invio dati saltuario e imprevedibile

Interruzione

- Necessario un meccanismo per:
 - Gestire una periferica mentre la CPU compie altre attività
 - Evitare interrogazioni continue della CPU alle periferiche (**polling**) per disponibilità dati I/O
- Stato normale di elaborazione: la CPU si disinteressa delle periferiche
- Una periferica chiede I/O dati: attiva una linea verso CPU (genera una **interruzione**)
 - Sospensione processo in esecuzione (salvataggio stato dei registri, per poter riprendere l'elaborazione dopo l'interruzione)
 - Eseguito **interrupt handler** (trasferimento dati da/verso periferica ecc.)
 - Riattivazione del processo sospeso

Altre funzionalità

- **File system:** organizzazione logica dei dati in strutture ad albero
- **Shell:** interazione testuale o grafica con l'utente
 - *Gestione utenti:* personalizzazione e protezione degli accessi alle risorse
 - *Esecuzione applicazioni*
- Applicazioni di tipo diverso
 - *Software orizzontale*, di utilizzo generale (es. office)
 - *Software verticale*, per esigenze di settori specifici (es. hotel)
 - *Software personalizzato*, ad hoc per un committente (es. sito web)



Reti di calcolatori

Reti di calcolatori

- Vantaggi
 - Condivisione di dati e risorse
 - Convenienza e crescita graduale
- Classificazione
 - Dimensione
 - Throughput
 - Mezzo fisico
 - Collegamento dati

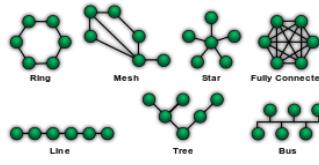


Dimensione e capacità

- **Dimensione:** tipico criterio di classificazione delle reti
 - **LAN:** Local Area Network (ufficio o edificio)
 - **MAN:** Metropolitan Area Network (campus o città)
 - **WAN:** Wide Area Network (rete geografica)
- **Throughput** ("capacità" o "larghezza di banda")
 - Quantità di informazione trasportata nell'unità di tempo (bit/s)
 - Misura principale delle prestazioni di una rete
 - Ma anche *latenza, jitter, probabilità d'errore...*

Collegamento

- **Topologia**
 - *Stella, bus, anello, albero, mesh*



- **Mezzo fisico**

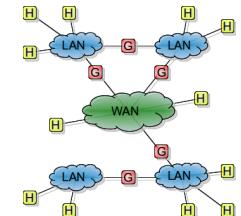
- *Rame*, decine di Mbit/s
- *Fibra ottica*, Tbit/s
- *Senza cavo*, decine di Mbit/s
- *Via satellite*, broadcast 1Gbit/s, condivisione, latenza

- **Data link**, collegamento dati

- Marcatura dei *frame* dati, gestione di *flusso, collisioni, errori*

Internet

- La "rete delle reti": interconnessione tra svariate reti di calcolatori, di dimensione planetaria
- Assegna un *indirizzo* univoco (*indirizzo IP*) ai calcolatori, per l'individuazione globale o locale
- Si basa su protocolli di comunicazione comuni (*stack TCP/IP*) per lo scambio di *pacchetti* di dati tra calcolatori
- *Commutazione di pacchetto* ☎ anziché di *circuito* ☎
 - Per ogni pacchetto viene scelto un percorso



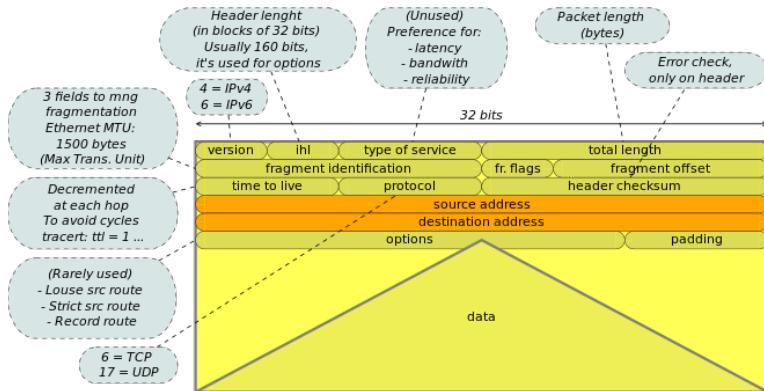
Indirizzo IPv4

- Composto da una sequenza di quattro numeri compresi tra 0 e 255 (4 byte)
 - 160.78.28.83 – Indirizzo pubblico (UniPR)
 - 192.168.1.1 – Indirizzo privato
(10.*.*.* – 172.16-31.*.* – 192.168.*.*)
 - 127.0.0.1 – *Loopback*
- Spazio indirizzi: soli 32 bit → uso indirizzi privati
 - *NAT (Network Address Translator)*: intera rete locale con un unico indirizzo pubblico
 - *Proxy*: intermediario per una certa app., di solito web

Domain Name System

- Il **DNS**: nomi simbolici (domini) → indirizzi IP
 - www.repubblica.it, www.google.com, www.ce.unipr.it
- **Top level domain** (parte finale nome) e indirizzi assegnati da *ICANN/IANA*
 - Indicante il tipo di organizzazione:
com, edu, gov, int, net, mil, org, info, biz, name
 - Indicante la nazione: it, uk, fr, us, eu, ...

Pacchetto IPv4



Protocollo IPv6

- Indirizzi a **128 bit**
 - Numero di indirizzi IPv6 / m² sulla Terra ($6,66 \times 10^{23}$) > Numero di Avogadro!
- Pacchetto con **header molto semplificato**
 - 64 bit in tutto (senza contare gli indirizzi...)
 - Poi header opzionali
- Possibilità di stabilire dei **flussi**
 - Predefinizione di un percorso; via di mezzo tra commutazione di circuito e di pacchetto

Transmission Control Protocol

- IP:** senza connessione e inaffidabile
 - Ciascun pacchetto viaggia in maniera indipendente
 - Senza garanzia di consegna (best effort)
- UDP (User Datagram Protocol):** inaffidabile.
 - Datagrammi dentro pacchetti IP
- TCP:** orientato alla connessione e affidabile
 - Controllata la correttezza dei dati
 - Segmenti** numerati e riordinati
 - Re-invio segmenti persi, eliminazioni duplicati
 - Controllo di congestione del traffico

Modello OSI/ISO

Liv.	Definizione	Descrizione
7	Applicazione	Applicazioni (Web, eMail, Skype...)
6	Presentazione	Standard formato dati (HTML, XML...)
5	Sessione	Protocolli dei servizi: FTP, HTTP, SMTP, RPC, TELNET...
4	Trasporto	Protocolli TCP e UDP
3	Rete	Protocollo IP
2	Colleg. dati	Trasmissione dati dipendente da livello fisico
1	Fisico	Hardware



World Wide Web

- Sistema per condivisione di informazioni ipertestuali
- Uno dei modi più diffusi di utilizzare la rete Internet
- Permette agli utenti di Internet di pubblicare e accedere a documenti **HTML**, raggiungibili ad una certa **URL** via **HTTP**
- Si basa su due programmi: **web server** e **web client (browser)**
- 1990: Tim Berners-Lee @ CERN, x doc scientifici
- 1993: Marc Andreessen @ Mosaic/Netscape, primo browser grafico



World Wide Web



www.ce.unipr.it/people/tomamic

40/53

Form HTML

- Sezione di documento tra tag **form**, contenente:
 - Testo normale e markup
 - Elementi speciali, o **controlli**, che l'utente può "completare"
- Il tag **form** prevede due attributi
 - **action**: url a cui inviare i dati (elaborazione remota su web server)
 - **method**: metodo HTTP da usare nell'invio dei dati del form (**get** o **post**, in accordo con server)

```
<form action="http://xyz.com/script.cgi" method="post">  
...  
</form>
```

HTML



www.ce.unipr.it/people/tomamic

41/53

Tag per i controlli

- Tag principali per **controlli** interattivi
 - **input**: immissione di testo, pulsanti selezionabili, buttoni
 - **select**: menu a discesa e box di selezione
 - **textarea**: area di testo, su più righe
- Ogni controllo di input in un form ha:
 - Un **nome**, definito dall'attributo **name**
 - Un **valore**, che l'utente imposta (immettendo testo o cliccando col mouse)
- Dati inviati come insieme di coppie nome/valore



www.ce.unipr.it/people/tomamic

43/53



42/53

Linguaggio PHP

- **Php: Hypertext Preprocessor** (in origine: *Personal Home Page*)
- Linguaggio scripting lato server, come ASP e JSP
- Frammenti di codice inseriti in pagine HTML
- File PHP eseguiti sul server e poi restituiti al browser come semplice HTML
- Supporta molti DB, spesso si usa MySQL (LAMP)
- Open source
- <http://php.net/manual/it/>
- <http://www.apachefriends.org/it/xampp.html>



Linguaggio JavaScript

- Netscape: interattività ad HTML
 - Reagire ad eventi
 - Controllare dati
 - Modificare elementi HTML
- Completamente diverso da Java
 - Linguaggio interpretato (no compilazione, o *jit*)
 - Tipizzazione dinamica (dati di tipo diverso in una var.)
 - *Object-based* (non ci sono classi)
- Scripting in applicazioni diverse (es. QML)
- <https://developer.mozilla.org/en/JavaScript>
- <http://www.ecmascript.org/>



HyperText Transfer Protocol

- Protocollo di livello 5, sessione
- Sistemi informativi distribuiti, collaborativi, ipertestuali
 - Usato nel World-Wide Web fin dal 1990
 - Semplice: meccanismo richiesta/risposta
 - Flessibile: supporto a vari tipi di dati, estendibile
 - Usa TCP/IP, porta 80... ma richiede solo un livello di trasporto affidabile
 - *Senza connessione*: una richiesta/risposta per ogni connessione (meccanismo base, fino a v1.0)
 - *Senza stato*: lo stato non è conservato tra connessioni diverse (cookie per webapp, ext. v1.0)



Metodo GET

- Recupera l'informazione (in forma di entità) identificata dall'url della richiesta
- Se url = processo di produzione dati...
 - Dati restituiti come corpo della risposta
- Semanticamente modificata in **GET** condizionale se tra gli header:
 - *if-modified-since*, *if-unmodified-since*, ...
 - Evitare spreco di risorse di rete

Metodo POST

- Copre diverse funzioni generali
 - Annotazione di risorse esistenti
 - Invio di messaggi a bulletin board, newsgroup, mailing list, o gruppi di articoli simili
 - Aggiunta di dati ad un database
 - Invio dati (es. form) a processo che li gestisca
- Url = processo che gestirà l'entità acclusa
 - Applicaz. server, o gateway per altri protocolli
 - Entità separata che accetta annotazioni

Esempio di richiesta

```
POST /beta.jsp HTTP/1.1
Referer: http://www.alpha.com/alpha.jsp
Connection: Keep-Alive
User-Agent: Mozilla/4.61
Host: www.alpha.com:80
Cookie: name=value
Accept: image/gif, image/jpeg, */
Accept-Language: en
[blank line here]
selected-item=1234&action=show+details
```

HTTP

```
GET /beta.jsp?selected-item=1234&action=show+details HTTP/1.1
Referer: http://www.alpha.com/alpha.jsp
[...]
If-modified-since: Mon, 10 Jul 2000 22:55:23 GMT
[blank line here]
```

HTTP

Esempio di risposta

```
HTTP/1.1 200 OK
Date: Fri, 12 Nov 2001 11:46:53 GMT
Server: Apache/1.3.3 (Unix)
Last-Modified: Mon, 12 Jul 2000 22:55:23 GMT
Accept-Ranges: bytes
Content-Length: 234
Content-Type: text/html
[blank line here]
<html>
<head><title>Beta page</title></head>
<body>
<h1>Beta page</h1>...
```

HTTP

Codici di stato

- 1xx – Informational (ok, elaborazione in corso)
- 2xx – Success (richiesta ricevuta e accettata)
- 3xx – Redirection (necessarie azioni ulteriori)
 - 301: Perm.; 302: Temp.; 304: Non modificata (→ cache)
- 4xx – Client Error
- 400: Richiesta scorretta; 401: Non autorizzato; 404: Non trovato
- 5xx – Server Error (ma richiesta valida)

Cookie

- Meccanismo generale per memorizzare e recuperare informazioni sul lato client di una connessione HTTP
- Server: quando invia risposta...
 - Può inviare anche info di stato
 - Da memorizzare sul client
 - Specificato il range di url per cui lo stato è valido
 - Può essere specificata una scadenza
- Client: in seguito, per ogni richiesta quel range...
 - Ritrasmessa anche informazione di stato



<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR

HTML e CSS



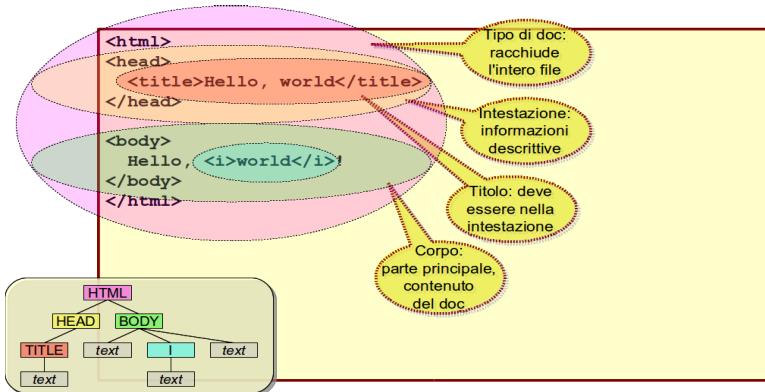
Introduzione
all'informatica

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

HyperText Markup Language

- Documenti **strutturati**, standard W3C: <http://www.w3.org/html/>
- HTML dichiara tipi di elementi
 - Paragrafi, titoli, liste, collegamenti ipertestuali, elementi multimediali ecc.
- Tipo di **elemento** descritto da tre parti
 - **Tag di apertura, contenuto, tag di chiusura**
 - Bla bla, **** in grassetto. ****, normale.
- Molti tag permettono la definizione di **attributi**
 - **UniPR**
 - **id** e **class**: attributi generici per assegnare **ruoli logici**
- Tag semplici non hanno un contenuto
 - ****

Anatomia di una pagina



Tag di formattazione testo

HTML

```
<p>Questo è un paragrafo.<br />A-capo ma stesso paragrafo.</p>

<p>Testo <strong>in grassetto</strong>, e poi
<em>in corsivo</em>.</p>

<h1>Il titolo più grande</h1>
...
<h6>Il titolo più piccolo</h6>

<div class="remark">
    Struttura generica di livello blocco,
    con un <span>elemento generico</span> inline.
</div>
```

Uniform Resource Locator

- Una URL è un riferimento per una risorsa



- Il nome della risorsa dipende interamente dal protocollo. Per HTTP include:
 - Nome dell'host su cui risiede la risorsa
 - Numero di porta cui collegarsi (default = 80)
 - Percorso della risorsa sulla macchina
 - Stringa di query (dopo ?)
 - Frammento: **id** di un elemento all'interno della risorsa (dopo #)
- <http://www.ietf.org:80/rfc/rfc2732.txt>

Collegamenti e citazioni

HTML

```
<p id="par1">Il primo paragrafo. Vai su
<a href="http://www.unipr.it/index.html#news">
UniPR, riquadro News</a>.</p>

<p id="par2">Il secondo paragrafo. Torna al
<a href="#par1">primo paragrafo.</a></p>

<!-- si può attribuire un id ad ogni elemento, ma dev'essere univoco nella pagina --&gt;

&lt;blockquote cite="http://www.faqs.org/"&gt;
&lt;p&gt;The BLOCKQUOTE element contains text quoted
from another source.&lt;/p&gt;
&lt;address&gt;&lt;a href="http://www.faqs.org/"&gt;
Da faqs.org&lt;/a&gt;&lt;/address&gt;
&lt;/blockquote&gt;</pre>
```

Liste

```
<ul>
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ul>

<ol>
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ol>
```

HTML

Tabelle

- Le tabelle HTML servono per incasellare dati (e non per gestire il layout dell'intera pagina!)
- Sono marcate con **<table>**
- <table>** contiene righe di celle, marcate con **<tr>** (dall'alto verso il basso ↓)
- Ogni **<tr>** contiene celle di dati, marcate con **<td>** (da sinistra a destra →)
- Le celle di intestazione sono marcate con **<th>**

Semplice tabella

```
<table>
  <tr>
    <td>northwest</td>
    <td>northeast</td>
  </tr>
  <tr>
    <td>southwest</td>
    <td>southeast</td>
  </tr>
</table>
```

HTML

Tabella con celle unite

```
<table>
<caption><i>A test table with merged cells</i></caption>
<tr>
  <th rowspan="2"></th>
  <th colspan="2">Average</th>
  <th rowspan="2">Red eyes</th>
</tr>
<tr>
  <th>height</th>
  <th>weight</th>
</tr>
<tr><th>Males</th>
  <td>1.9</td><td>0.003</td><td>40%</td></tr>
<tr><th>Females</th>
  <td>1.7</td><td>0.002</td><td>43%</td></tr>
</table>
```

A test table with merged cells		
	Average	Red height weight eyes
Males	1.9	0.003 40%
	Females	1.7 0.002 43%

Tag per computer

- Codice inline: `<code>`
- Blocchi di codice: `<pre><code>`
- Input da tastiera: `<kbd>`
- Variabili e costanti: `<var>`
- Output inline: `<samp>`
- Catture di schermate testuali (blocchi di output) `<pre><samp>`

Tag generici: div, span

- Aggiungere struttura e semantica ai documenti
- `span` raggruppa contenuto *inline*
- `div` raggruppa contenuto di *livello blocco*
- Spesso assegnati attributi `id` e `class`
- Non impongono nessun altro vincolo di presentazione al contenuto
- Elementi da usare assieme a fogli di stile
- Per adattare i doc. html ai vari bisogni e gusti

Id vs. class

- Mentre uno stesso valore di `class` può essere attribuito a molti elementi su una pagina...
- `id` deve essere unico all'interno del documento!
- Non si può applicare uno stesso valore di `id` a più elementi
- Si possono assegnare più classi ad un solo elemento, separate da spazio. Es.
 - `<p class="news gossip">Bla bla.</p>`

Markup semantico

- Documenti privi di markup di presentazione
- Definire un vocabolario di classi semantiche...
 - Da assegnare agli elementi con attributo `class`
 - La cui presentazione può essere specificata in fogli di stile validi per tutto il sito
- Indicazioni di mozilla.org
 - <http://www.mozilla.org/contribute/writing/markup>
- Accessibilità dei contenuti web
 - <http://www.w3.org/TR/WCAG10-HTML-TECHS/>

Html 5

- Nuovi elementi di struttura di Html 5
 - header, main, nav, aside, footer
 - article, section, details, summary
 - figure, figcaption
- Altri nuovi elementi
 - video, audio, canvas, embed
 - mark, menu, command, output, time
 - progress, meter, datalist
- <http://dev.w3.org/html5/html4-differences/>



Cascading Style Sheets

"In principio il web era popolato di semantici tag p ed h1; ma presto arrivarono font, center, color; le tabelle nascoste erano in agguato; era già scoppiata la Guerra dei Browser, tra Netscape e IE."

- Dopo specifiche W3C per HTML 4.0 e stili, tendenza a miglior supporto di standard
- CSS: migliore semantica e lavoro risparmiato
 - Un solo file, esterno ai contenuti, controlla la presentazione di molte pagine web
- <http://www.w3.org/Style/CSS/>



Foglio di stile esterno

- Ideale, soprattutto, per un sito di molte pagine
- Cambiare l'aspetto, modificando un solo file
- Ogni pagina deve essere collegata al file css

```
<head>
...
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

HTML

- Browser formatta secondo mystyle.css
- CSS scritto con comune editor di testo

```
body { background: url("images/back40.gif") }
p { margin-left: 20px }
```

CSS

Stili a cascata

- Stili raccolti a cascata in "foglio di stile virtuale"
- Default del browser (priorità più bassa)
- Foglio di stile esterno (per molti documenti)
- Foglio di stile interno (per un singolo doc.)
- Stile inline (per un singolo elemento, priorità)
- Gli stili non sono HTML, hanno sintassi diversa

Sintassi degli stili

- La più semplice regola css è composta di tre parti: un **selettor**, una **proprietà** ed un **valore**:
 - selector { property: value }
 - Selettor: es. un elemento html da ridefinire
 - Proprietà: aspetto cui assegnare nuovo valore
- Es. body ed elem. contenuti con testo in nero
 - body { color: black }

Attributi e selettori multipli

```
p {  
    text-align: center; /* attributes separated by semicolon */  
    color: red;  
    font-family: "sans serif"; /* string in quotation marks */  
}  
h1, h2, h3, h4, h5, h6 { /* selectors separated by comma */  
    color: rgb(0, 255, 0);  
}  
img {  
    float: right;  
    padding: 5px;  
    border: 1px solid #0000ff;  
    margin: 10px;  
}
```

Selettori di classe e id

```
.important { text-align: center; }  
  
.gossip { display: none; }  
  
#wer345 { font-size: 32px; font-weight: bold; font-style: italic; }
```

CSS

```
<h1 class="important">Centered heading</h1>  
  
<p class="news gossip">Gossip's not displayed.</p>  
  
<h1 id="wer345">Some specific heading</h1>  
  
<p class="important">More text.</p>
```

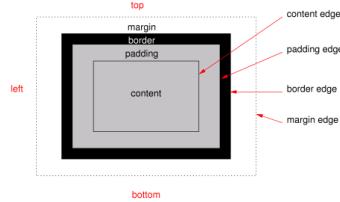
HTML

Colori e dimensioni

- CSS1: 16 colori, come palette VGA di Windows
 - aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow
 - Es. fuchsia == rgb(255, 0, 255) == #FF00FF
- Dimensioni box e testo
 - px: in pixel (fissa rispetto risoluzione monitor)
 - pt: punti tipografici, 1/72 pollice
 - em: rispetto a dimensione font (carattere X)
 - %: rispetto alla dimensione (di font, o spazio) dell'elemento genitore

Box model

- margin, padding, border-width - 1-4 valori
- border-style - none, dotted, dashed, solid ...
- border-color - Colore
- border-radius - 1-4 valori ([CSS3](#))



Sfondo e ombra

- background-color - Colore di sfondo
- background-image - url(...), none
- background-image - linear/radial-gradient(...) ([CSS3](#))
- background-repeat - repeat, repeat-x / -y, no-repeat
- background-attachment - scroll, fixed
- background-position - top left, top center ...
- background-size - Dimensioni immagine ([CSS3](#))
- box-shadow - 2-4 valori (offset-x, offset-y, blur, distanza) + colore ([CSS3](#))

Testo

- color - Colore del testo
- font-size - Dimensione
- vertical-align - Inline, relativo alla riga di testo - top, middle, bottom, baseline, sub, super
- text-decoration - none, underline, line-through, overline
- font-style - normal, italic
- font-weight - normal, bold
- font-family - Lista con priorità di nomi o famiglie di font (serif, sans-serif, cursive, fantasy, monospace)
- text-align - left, right, center, justify

Posizionamento

- position - Posizionamento - relative, absolute, fixed
- float - Elementi flottanti - left, right, none
- z-index - Numero (valori più alti in primo piano)
- overflow - visible, hidden, scroll, auto
- visibility - visible, hidden (occupa spazio)
- display - block, inline, none (non occupa spazio)
- Blocco centrato: margin-left: auto; margin-right: auto; width: 50%;

Basi di dati



<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR

Introduzione
all'informatica

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR



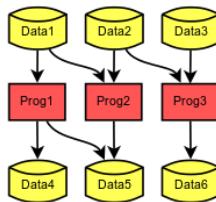
Sistema informativo

Sistema informativo

- Sistema informativo
 - Informazioni di interesse nei processi aziendali
 - Modalità in cui esse sono gestite
 - Risorse coinvolte, sia umane sia tecnologiche
- ICT: insieme di programmi concorrenti
 - Ogni programma opera su un certo insieme di dati
 - Certi dati possono essere condivisi tra i programmi
- Casi semplici: ogni programma gestisce i suoi dati
- Altrimenti: sistema di gestione tra programmi e dati

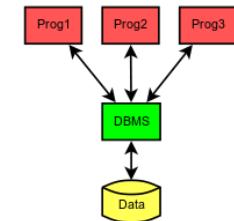
Gestione distinta dei dati

- **Ridondanza:** più copie dello stesso dato
- **Inconsistenza:** copie modificate diversamente
- **Riservatezza:** dati riservati accessibili a persone non autorizzate
- **Integrità:** operazioni sbagliate o incomplete sui dati
- **Concorrenza:** accesso e aggiornamento dati non sincronizzato tra programmi differenti



Gestione condivisa dei dati

- Tutte le azioni sui dati vengono mediate dal **DBMS** (*DataBase Management System*)
- Dati in formato standard, con backup/ripristino
- Controlli su:
 - Ridondanza, consistenza, distribuzione
 - Riservatezza, integrità
 - Accesso concorrente
- Ma risorse hw/sw (stesso DBMS)

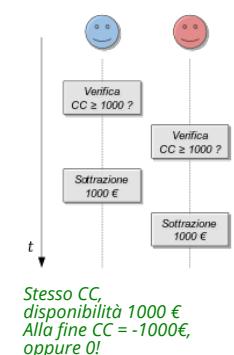


Basi di dati con DBMS

- Da preferire quando:
 - Dati organizzati secondo **modelli** predefiniti
 - **Grandi:** fino e oltre TByte, memoria secondaria
 - **Condivisi:** accesso da app. ed utenti diversi
 - **Persistenti:** tempo di vita > esecuzione app.
- Da evitare quando:
 - Insieme dati piccolo e semplice
 - Poche modifiche nel tempo
 - Non condiviso
 - Prestazioni in tempo reale

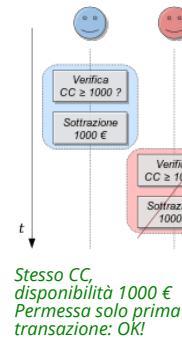
Accessi concorrenti

- Problemi di consistenza dei dati condivisi
- Es. prelievo da un conto corrente come sequenza operazioni
 - Verifica disponibilità
 - Sottrazione importo

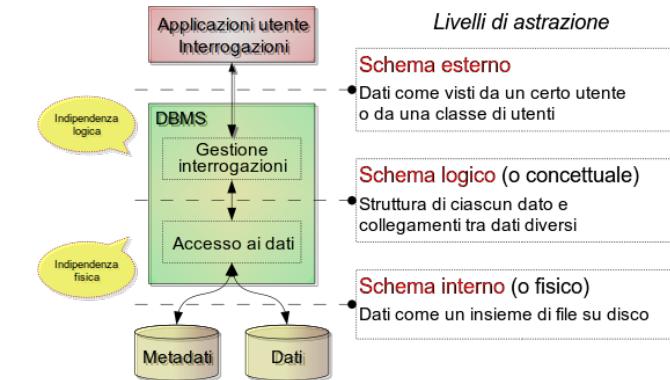


Transazione

- **Insieme di operazioni non decomponibili**
 - Eseguite completamente, prima che stessi dati siano nuovamente disponibili **ACID** (*Atomicity, Consistency, Isolation, Durability*)
- Es. precedente:
 - Verifica disponibilità
 - Sottrazione importo
 - Unica transazione!



Architettura a tre livelli



Linguaggi DDL e DML

- **DDL** (*Data Definition Language*), intensionale
 - Usato dal DBA (amministratore)
 - Definire lo schema dati, secondo il modello concettuale: gerarchico, relazionale ecc.
 - Definire tabelle, campi, chiavi ecc.
- **DML** (*Data Manipulation Language*), estensionale
 - Usato all'interno delle applicazioni
 - Operazioni **CRUD** (*Create, Read, Update, Delete*)
 - SQL (INSERT, SELECT, UPDATE, DELETE)
 - ~ 4 verbi HTTP (POST, GET, PUT, DELETE)

Modelli dei dati

- Caratterizza livello concettuale e esterno DBMS
- Definito da regole precise, per esprimere sia le proprietà statiche che quelle dinamiche dei dati
- Evoluzione dei modelli:
 - **Gerarchico** (anni 1960)
 - **Reticolare** (anni 1970)
 - **Relazionale** (anni 1970)
 - **Object-relational, object-oriented** (anni 1980)



Modello relazionale

Modello relazionale

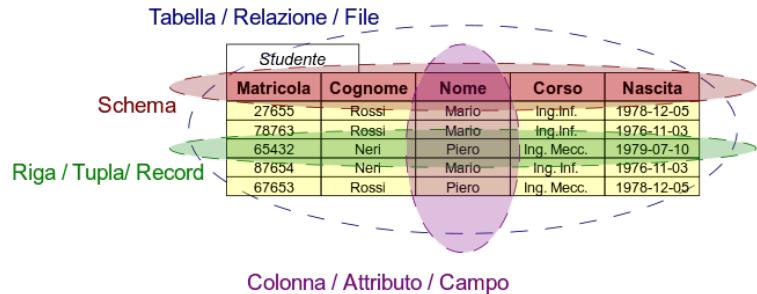
- Codd - 1970; DBMS reali - 1981
- Si basa sul concetto matematico di **relazione**
 - Relazioni rappresentate da familiari **tabelle**
 - Successo anche per **semplicità** di utilizzo
- A ciascun **dominio** è associato un nome (**attributo**), unico nella relazione
 - Il nome "descrive" il ruolo del dominio
 - Attributi usati come intestazioni delle **colonne**
- Informazioni inserite nelle **righe** della tabella

Database universitario

Studente				
Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing.Inf.	1978-12-05
78763	Rossi	Mario	Ing.Inf.	1976-11-03
65432	Neri	Piero	Ing. Mecc.	1979-07-10
87654	Neri	Mario	Ing. Inf.	1976-11-03
67653	Rossi	Piero	Ing. Mecc.	1978-12-05

Esame			Insegnamento		
Studente	Insegnamento	Voto	Codice	Titolo	Docente
78763	04	30	01	Analisi	Chiari
65432	02	24	02	Chimica	Bruni
65432	01	28	04	Chimica	Verdi
27655	01	26			

Terminologia



Dominio di un attributo

- Tuple di una relazione definite dall'insieme dei valori corrispondenti agli attributi
- **Dominio** di un attributo: insieme di tutti e soli i valori che quell'attributo può assumere
- Es. Dominio dei codici fiscali
- Formato dalle stringhe di 16 caratteri che rispettano con precisione le regole di generazione dei codici fiscali

Modello E-R



Modello E-R

- Si creano **associazioni** tra entità **distinte**, tramite condivisione di attributi
 - Le righe di diverse tabelle hanno domini in comune
- Es. Database universitario
 - **Studenti** ed **esami** sono associati tramite gli attributi **matricola** e **studente**
 - **Insegnamenti** ed **esami** sono associati tramite gli attributi **insegnamento** e **codice**
- Semplicità: forza del modello relazionale!

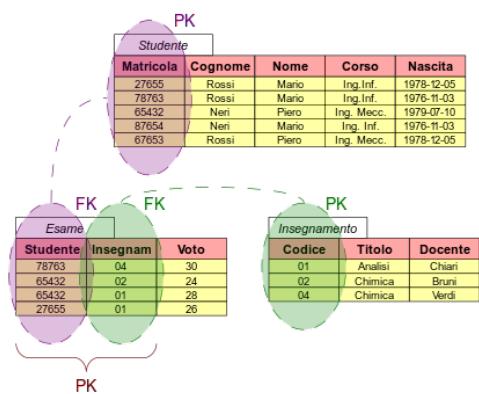
Chiave primaria

- Una tabella (relazione) non dovrebbe contenere due righe identiche
 - Sempre possibile scegliere un sottoinsieme di campi t.c. ...
 - Ciascuna riga della tabella identificata univocamente
- Chiave primaria (primary key, PK)** di una tabella:
 - Minimo sottoinsieme di campi che permette di...
 - Identificare univocamente le righe della tabella

Chiave esterna

- Le informazioni presenti in tabelle diverse possono essere associate tra loro perché tali tabelle hanno dei domini in comune
- Quando il dominio di un campo K che è chiave primaria in una tabella A è presente anche in un'altra tabella B...
- Allora questo campo K è detto **chiave esterna (foreign key, FK)** verso la tabella A

Concetto e tipo di chiave



Chiave candidata

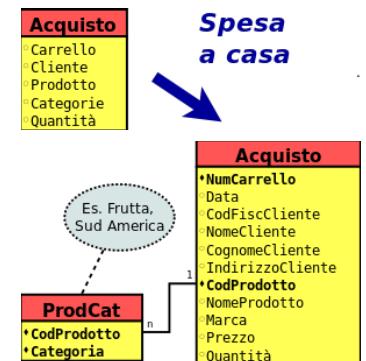
- Le **chiavi candidate** sono gli attributi in una relazione con la proprietà di poter essere la chiave primaria:
 - Tra le chiavi candidate deve essere scelta la chiave primaria
 - Le chiavi escluse si dicono chiavi alternative
- Le righe di una tabella rappresentano "entità" del mondo reale
- La chiave primaria rappresenta il modo con cui è possibile distinguere queste entità

Normalizzazione

- Processo di organizzazione dei dati per evitare **ridondanza**, anomalie, inefficienza
- Stessa informazione in più copie → svantaggi
 - Maggior uso di **memoria**
 - Modifiche ripetute** della stessa informazione
 - Inconsistenza** dei dati, se aggiornati in modo indipendente; la stessa informazione potrebbe assumere valori diversi

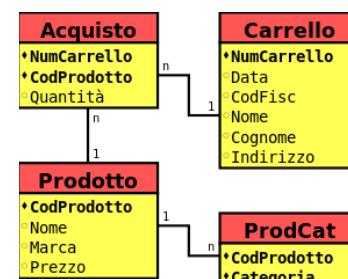
Prima forma normale

- La relazione rispetta il modello relazionale
- Le tuple hanno un numero fisso di attributi definiti su domini elementari
 - Non ci sono **righe uguali**
 - Atomicità: solo **attributi elementari**
 - Non ci sono **attributi ripetitivi**



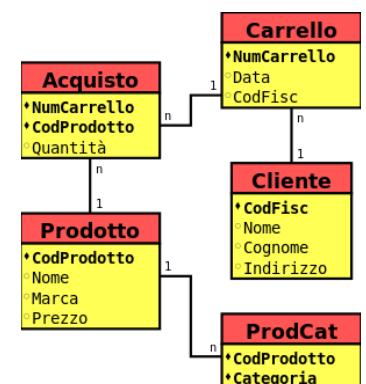
Seconda forma normale

- Non ci sono attributi non-chiave che dipendono **parzialmente** dalla chiave



Terza forma normale

- Non ci sono attributi non-chiave che dipendono **transitivamente** dalla chiave
 - Ossia dipendenti da campi non-chiave



Operatori relazionali

Operatori relazionali

- Base teorica per i linguaggi di interrogazione delle basi di dati relazionali
 - Operano su intere tabelle considerate come insiemi, piuttosto che record per record
 - Prendono in input tabelle
 - Generano in output nuove tabelle
- Operatori
 - **Unione, intersezione, differenza (op. insiemistici)**, applicabili a relazioni definite sugli stessi attributi
 - **Selezione, proiezione (un solo operando)**
 - **Prodotto cartesiano, join (più operandi)**

Operatori insiemistici

Laureati		
Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Quadri		
Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Reparti	
Reparto	Capo
A	Mori
B	Bruni

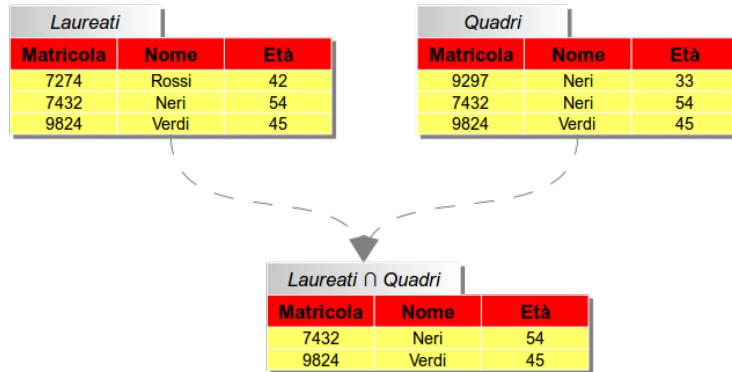
Unione

Laureati		
Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

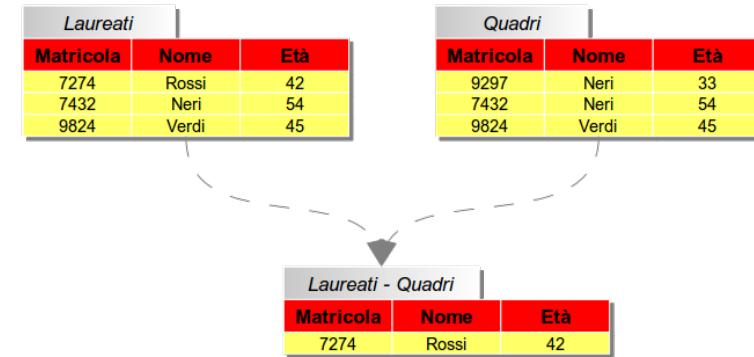
Quadri		
Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Laureati U Quadri		
Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33

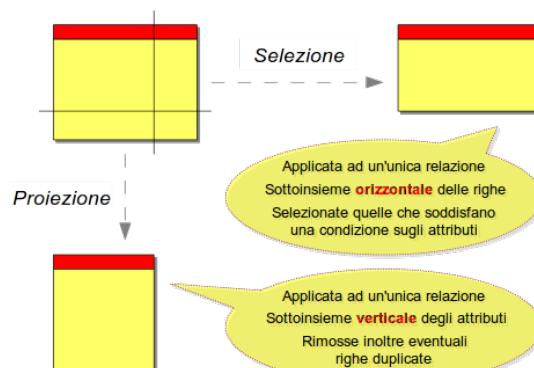
Intersezione



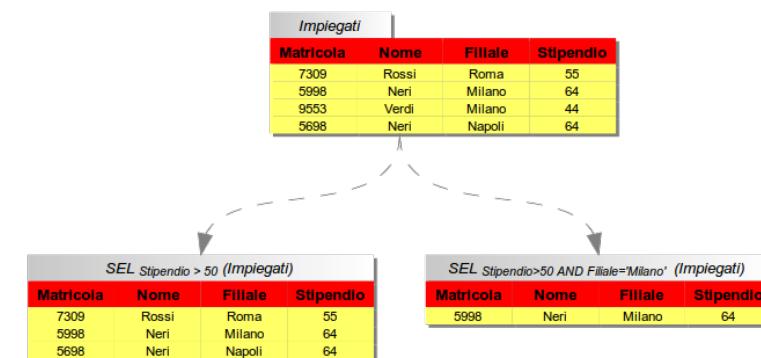
Differenza



Selezione e proiezione



Selezione



Proiezione

Impiegati			
Matricola	Nome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64
9553	Verdi	Milano	44
5698	Neri	Napoli	64

PROJ Matricola, Nome (Impiegati)	
Matricola	Nome
7309	Rossi
5998	Neri
9553	Verdi
5698	Neri

Prodotto cartesiano

Impiegati		Reparti	
Nome	Reparto	Codice	Capo
Rossi	A	A	Mori
Neri	B	A	Bruni
Bianchi	B	B	Chiari

Impiegati x Reparti			
Nome	Reparto	Codice	Capo
Rossi	A	A	Mori
Rossi	A	B	Bruni
Neri	B	A	Mori
Neri	B	B	Bruni
Bianchi	B	A	Mori
Bianchi	B	B	Bruni

Join senza uso di attributi in comune
Risultato con numero di n-uple pari al prodotto delle cardinalità degli operandi
(le n-uple sono tutte combinabili)

Join

Impiegati		Reparti	
Nome	Reparto	Reparto	Capo
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B		

Impiegati		Reparti	
Nome	Reparto	Reparto	Capo
Rossi	A	B	Mori
Neri	B	B	Bruni
Bianchi	B	A	Chiari

Impiegati JOIN Reparti		
Nome	Reparto	Capo
Rossi	A	Mori
Neri	B	Bruni
Bianchi	B	Bruni

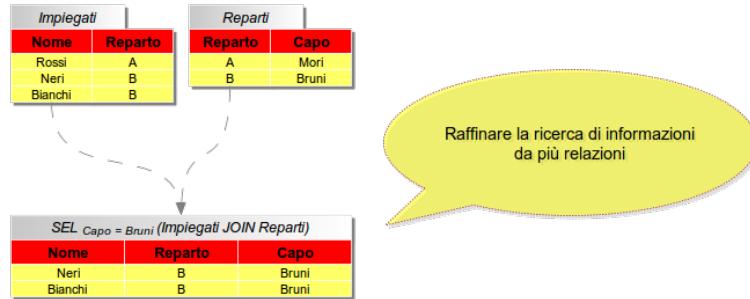
Proiezione + selezione

Impiegati			
Matricola	Nome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64
9553	Verdi	Milano	44
5698	Neri	Napoli	64

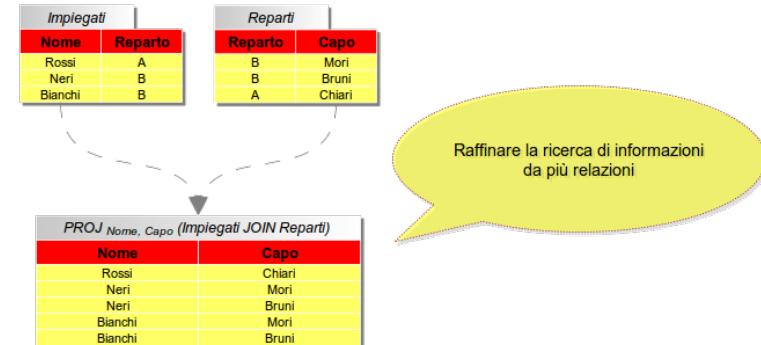
PROJ matricola, Nome (SEL Stipendio > 50 (Impiegati))	
Matricola	Nome
7309	Rossi
5998	Neri
5698	Neri

Raffinare la ricerca di informazioni da una singola relazione

Join + selezione



Join + proiezione



Structured Query Language



Structured Query Language

- **SQL**: riferimento per manipolazione e interrogazione di basi di dati relazionali
- Deriva da una prima proposta di linguaggio di Ibm chiamato *Sequel* (1974)
- Prime implementazioni di Ibm e Oracle (1981)
- Da 1983 "standard di fatto"
- Evoluzione corrispondente ad aggiornamenti delle specifiche (1986, 1989, 1992, 1999...)

Es. Tabelle parentele

```
create table Person (
    Name character(20) primary key,
    Age numeric(3),
    Income numeric(9)
);
create table Paternity (
    Father character(20),
    Child character(20) unique
);
create table Maternity (
    Mother character(20),
    Child character(20) unique
);
```

SQL

Es. Ricerche semplici

- Tell me name and income of people less than 30 yo
- Tell me everything of people less than 30 yo
- Fathers of people earning more than 50

```
select Name, Income from Person where Age < 30
select * from Person where Age < 30
select Paternity.Father
    from Person
    join Paternity
    on Paternity.Child = Person.Name
    where Person.Income > 50
```

SQL

Es. Manipolazione dati

```
insert into Person
    values ('Mario', 25, 52);
insert into Person (Name, Age)
    values ('Pino', 25);
delete from Person
    where Age < 18;
update Person
    set Income = 45
    where Name = 'Piero';
update Person
    set Income = Income * 1.1
    where Age < 30;
```

SQL

Es. Ricerche complesse

- Tell me name, income and fathers' age of people earning more than their father
- Tell me the name of each person's mother and father

```
select C.Name, C.Income, F.Age
    from Person C
    join Paternity P on C.Name = P.Child
    join Person F on F.Name = P.Father
    where C.Income > F.Income;

select Paternity.Child, Father, Mother
    from Paternity
    join Maternity on Paternity.Child = Maternity.Child
```

SQL

Es. Tabella impiegati

```
create table Employee (
    Id character(6) primary key,
    Name character(20) not null,
    Surname character(20) not null,
    Location character(15),
    Salary numeric(9) default 0,
    City character(15),
    foreign key(Location)
        references Department(DepName),
    unique (Surname, Name)
)
```

SQL

Create, select

```
create table Table (
    Attribute Domain [Constraints],
    Attribute Domain [Constraints]
    ...
    [OtherConstraints]
)
select Attribute, Attribute ...
    from Table, Table ...
    [where Conditions]
```

SQL

Insert, update, delete

```
insert into Table [(Attributes)] values(Values)
insert into Table [(Attributes)] select ...
update Table set Attribute =
    <Expression | select ... | null | default>
    [where Condition]
delete from Table [where Condition]
```

SQL

<Domande?>



Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR