

# L'ELABORATORE ELETTRONICO

---

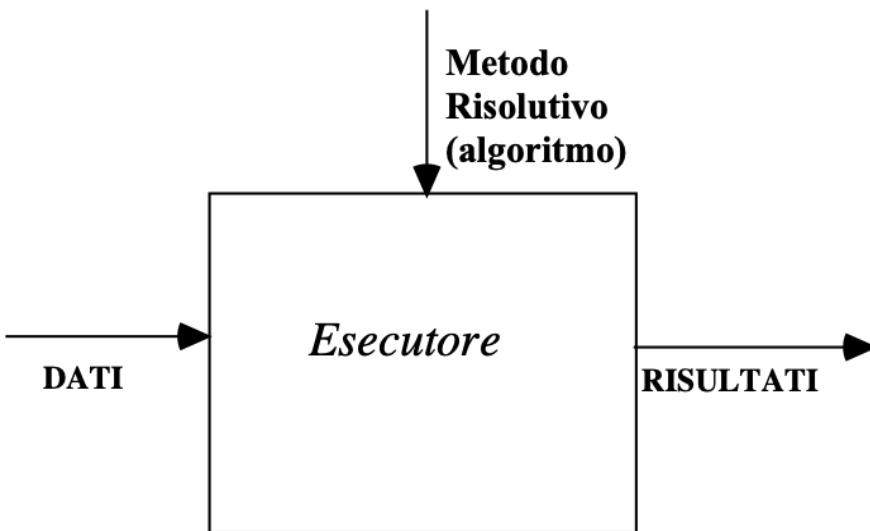
- Il calcolatore elettronico è uno strumento in grado di eseguire insiemi di *azioni* (“*mosse*”) *elementari*
- le azioni vengono eseguite su oggetti (**dati**) per produrre altri oggetti (**risultati**)
- l'esecuzione di azioni viene richiesta all'elaboratore attraverso *frasi* scritte in un qualche *linguaggio* (*istruzioni*)

# ALCUNE DOMANDE FONDAMENTALI

---

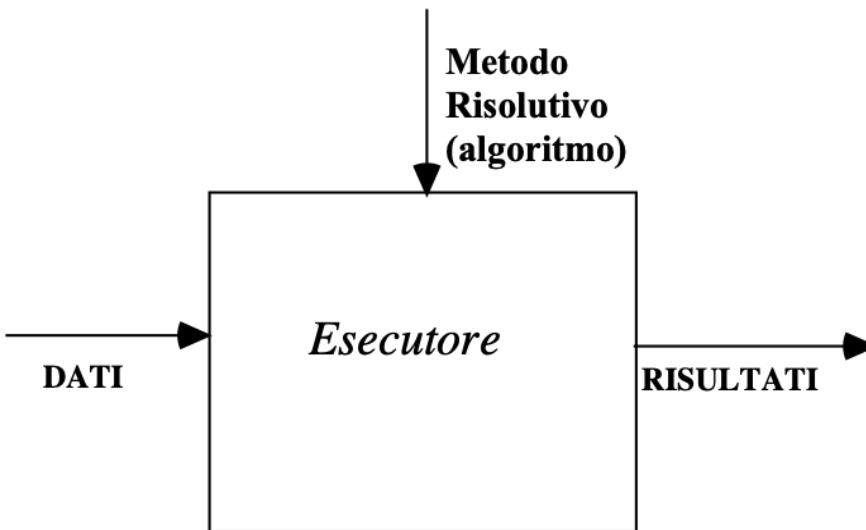
- **Quali istruzioni** esegue un elaboratore?
- **Quali problemi** può risolvere un elaboratore?
- Esistono **problemis che un elaboratore non può risolvere?**
- **Che ruolo ha il linguaggio** di programmazione?

# L'AUTOMA ESECUTORE



Un automa capace di ricevere dall'esterno la descrizione dell'algoritmo richiesto  
cioè  
capace di interpretare un linguaggio (*linguaggio macchina*)

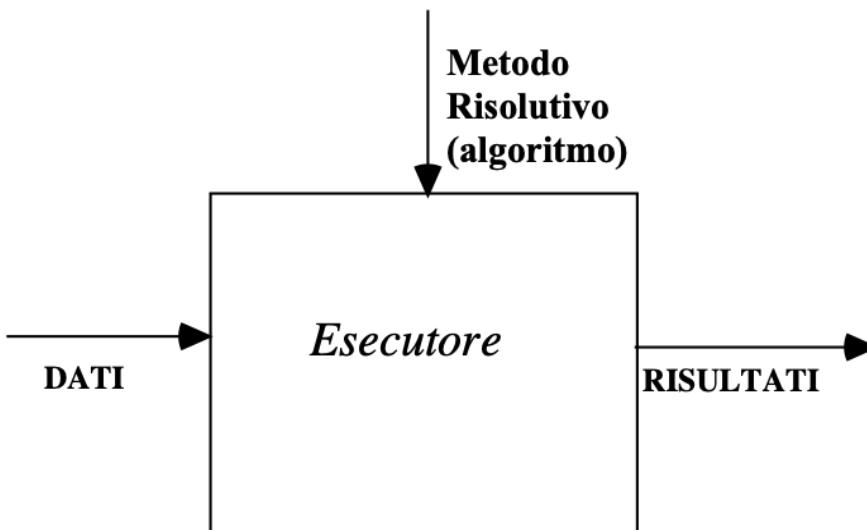
# L'AUTOMA ESECUTORE



## Vincolo di realizzabilità:

- se l'automa è fatto di parti, queste sono in numero finito
- ingresso e uscita devono essere denotabili attraverso un insieme finito di simboli.

# L'AUTOMA ESECUTORE

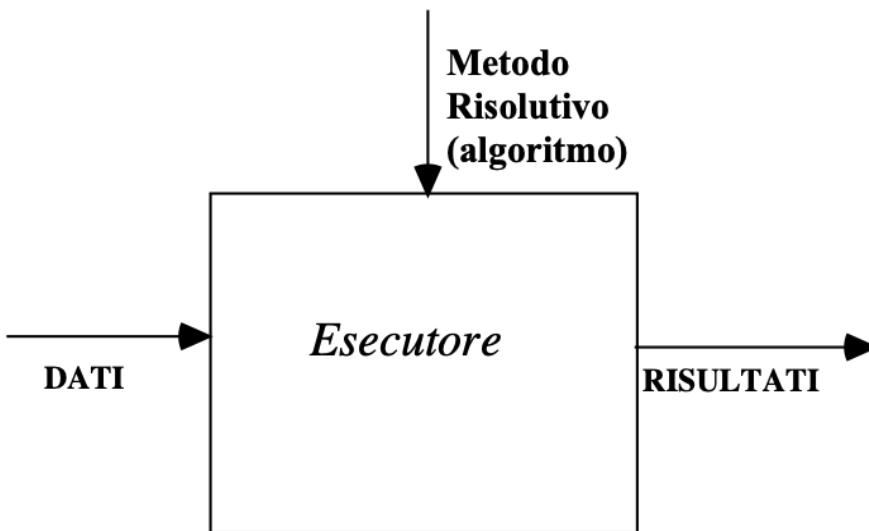


## Realizzazione:

### 1) mediante **congegni meccanici**

- macchina aritmetica (1649) di Blaise Pascal
- macchina analitica di Charles Babbage (1792-1871)

# L'AUTOMA ESECUTORE

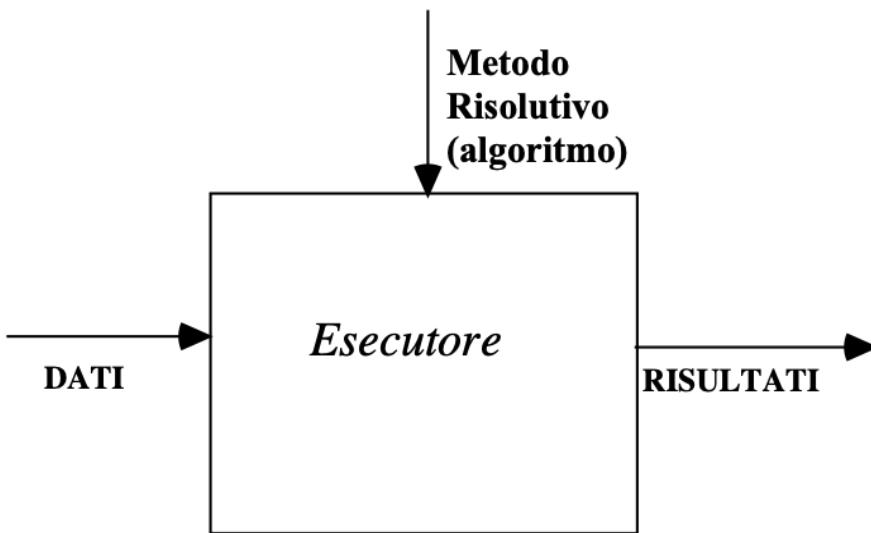


## Realizzazione:

### 2) mediante un **modello matematico**

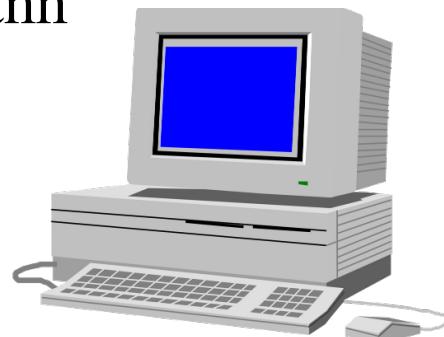
- funzionale (Hilbert, (1842-1943), Church, Kleene)
- **operazionale (Turing, 1912-1954)**
- sistemi di riscrittura (Post, Markov,...).

# L'AUTOMA ESECUTORE



## Realizzazione:

Mediante calcolatore elettronico basato sull'architettura di Von Neumann



# **PERCHÉ I MODELLI MATEMATICI**

---

Macchine diverse potrebbero avere **diversa capacità di risolvere problemi**

**Una macchina potrebbe essere “più potente” di un’altra  
È NECESSARIO SAPERLO**

Se neanche la macchina “più potente” riesce a risolvere un problema, **potrebbero esserci PROBLEMI NON RISOLUBILI**

# GERARCHIA DI MACCHINE

---

- macchine combinatorie
- macchine (automi) a stati finiti
- macchina a stack
- **macchina di Turing**



***TESI DI CHURCH-TURING***

***Non esiste alcun formalismo capace  
di risolvere una classe di problemi  
più ampia della Macchina di Turing***



# Alan Turing (1912-1954)

- Matematico, logico, crittanalista britannico
- Uno dei padri della informatica e uno dei più grandi matematici del Novecento.
- Egli introdusse la *macchina (universale)*, e *il test* che portano il suo nome.
- Fu anche uno dei più brillanti decrittatori che operavano in Inghilterra, durante la seconda guerra mondiale per decifrare i messaggi nemici.
- Omosessuale, morì suicida a 41 anni a causa di una persecuzione omofobica condotta nei suoi confronti.
- In suo onore l' Association for Computer Machinery (ACM) ha creato nel 1966 il Turing Award, massima riconoscenza nel campo dell'informatica.

# Procedura di “calcolo”

---

- Definizione il più possibile precisa e generale del concetto di "procedura di calcolo"
- Alan Turing a partire dal 1930 iniziò ad analizzare le attività rilevanti che un essere umano faceva durante l'esecuzione di un processo di calcolo.
  - utilizza fogli di carta come memoria;
  - legge dati dal foglio di carta;
  - scrive dati a partire da altri dati, eventualmente cancellando dati precedenti;
  - decide cosa scrivere in base a ciò che è presente sul foglio e al proprio stato mentale.

# La Macchina di Turing

---

- Alan Turing propose nel 1936 l'idea di una “macchina immaginaria” che potesse effettuare ogni tipo di calcolo su numeri e simboli.
- Era un modello di calcolatore molto semplice, ma al tempo stesso sufficientemente potente per studiare le proprietà matematiche di ciò che è calcolabile.
- Turing concepì questo dispositivo come modello ideale di calcolatore per studiarne le caratteristiche in termini teorici e non lo costruì fisicamente con dei dispositivi fisici.

# La macchina di Turing

---

- Una Macchina di Turing (MT) è definita da un insieme di regole che ne determinano il comportamento su un nastro di input/output.
- E' costituita da:
  - un **nastro**, che gioca il ruolo del dispositivo di memoria e viene supposto illimitato in entrambe le direzioni; il nastro è suddiviso in celle tutte uguali e ognuna di esse può contenere un simbolo.
  - una **testina** che permette di leggere e scrivere sulle celle del nastro; la testina esplora una cella alla volta, quella cioè che a un dato istante viene a trovarsi in sua corrispondenza. **Il simbolo letto dalla testina può essere cancellato e sostituito con un nuovo simbolo.**
  - un **dispositivo di controllo** che regola il movimento della testina rispetto al nastro, spostandola ogni volta di una cella verso sinistra o verso destra secondo i casi. **Il dispositivo di controllo è dotato di uno stato interno.**
- Il comportamento della TM dipende:
  - dal simbolo letto sul nastro mediante la testina;
  - dallo stato interno in cui si trova il dispositivo di controllo in un dato momento.

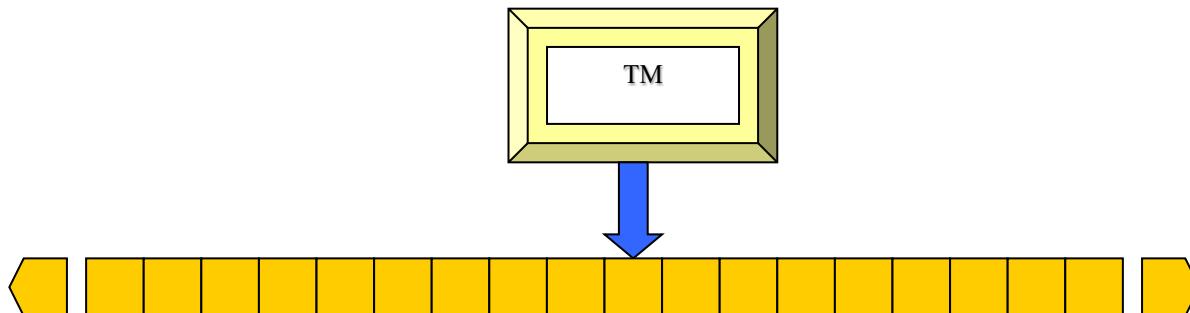
# LA MACCHINA DI TURING

---

La TM è una macchina capace di (in ogni passo e in accordo al suo stato interno):

- *leggere un simbolo dal nastro*
- *scrivere sul nastro un simbolo*
- transitare in un nuovo stato interno specificato
- spostarsi sul nastro di una posizione in una certa direzione (a destra o a sinistra)

Quando raggiunge lo stato HALT, la macchina si ferma



# LA MACCHINA DI TURING E CPU REALI

---

In pratica:

- leggere/scrivere un simbolo dal/sul nastro
- transitare in un nuovo stato interno
- spostarsi sul nastro di una (o più) posizioni

corrisponde a:

- lettura/scrittura memoria RAM/ROM
- nuova configurazione dei registri della CPU
- scelta della cella di memoria su cui operare

# Risolvere problemi con la Macchina di Turing

---

Risolvere un problema con la Macchina di Turing richiede quindi di:

- **definire** una opportuna *rappresentazione dei dati iniziali sul nastro*
- **definire la** *parte di controllo* mediante delle funzioni in modo da **rendere disponibile sul nastro, alla fine,** la *rappresentazione della soluzione*
- In pratica, il comportamento di una MdT può essere descritto da un insieme di **quintuple** della forma: (*stato-corrente, simbolo-letto, nuovo-stato, simbolo-scritto, direzione*)

# Come “calcola” ?

- Inizialmente:
  - Il nastro contiene una sequenza finita di simboli (quelle non vuote) detta stringa di ingresso
  - La MdT si trova nello stato iniziale 0 con la testina sul simbolo più a sinistra sul nastro
- Partendo da questa configurazione iniziale, la MdT effettua una serie di mosse seguendo quanto definito dall’insieme delle sue quintuple
- Se la macchina raggiunge una configurazione in cui non esiste nessuna quintupla che associa un’azione alla coppia **(stato interno, simbolo letto)** termina la computazione
- **Simulatore macchina di Turing (tanti in rete):**  
[http://studiovc.altervista.org/new\\_bin/simulatore\\_macchina\\_di\\_turing-gara\\_nazionale.html?x=4227688](http://studiovc.altervista.org/new_bin/simulatore_macchina_di_turing-gara_nazionale.html?x=4227688)

# Esempio

- Esempio:

**(2, C, 3, D, -)**

**(3, D, 3, D, >)**

			3	3		
...		A	<b>D</b>	B		...

# Esempio

- Una MdT che scrive la sequenza di caratteri CIAO su un nastro vuoto

(0, -, 1, C, >)

(1, -, 2, I, >)

(2, -, 3, A, >)

(3, -, 4, O, >)

0 1 2 3 4

...	C	I	A	O		...
-----	---	---	---	---	--	-----

# Esempio

- Una MdT che modifica una sequenza di A e di B scambiando ogni A con una B e viceversa

**(0,A,0,B,>)**

**(0,B,0,A,>)**

**(0,-,F,-,-)**

	0	0	0	F			
...	B	AB	AB				...

# Esempio

## da Bellini, Guidi “Linguaggio C”, McGraw Hill, 2021:

Realizzare una MdT che riceve in ingresso un numero decimale e restituisce zero se è pari e uno se è dispari.

- Un numero è pari se la sua ultima cifra è pari, altrimenti è dispari.
- La macchina dovrà cancellare tutte le cifre ponendosi nello stato interno 0 se la cifra appena cancellata è pari, nello stato 1 se è invece dispari.
- Alla fine, non appena trova una cella vuota, la macchina scrive 0 se è nello stato 0, cioè se l'ultima cifra cancellata era pari, 1 se è nello stato 1, cioè se l'ultima cifra cancellata era dispari.
- Con lo stato finale F, a cui non corrisponde alcuna istruzione, viene terminata l'elaborazione.

# Esempio (cont.)

da Bellini, Guidi “Linguaggio C”, McGraw Hill, 2021:

- Stati interni: { 0, 1, F }
- Alfabeto: {0,1,2,3,4,5,6,7,8,9,--}
- Regole:

(0,0,0,--,>) (1,0,0,--,>) (0,1,1,--,>) (1,1,1,--,>) (0,2,0,--,>)  
(1,2,0,--,>) (0,3,1,--,>) (1,3,1,--,>) (0,4,0,--,>) (1,4,0,--,>)  
(0,5,1,--,>) (1,5,1,--,>) (0,6,0,--,>) (1,6,0,--,>) (0,7,1,--,>)  
(1,7,1,--,>) (0,8,0,--,>) (1,8,0,--,>) (0,9,1,--,>) (1,9,1,--,>)  
(0,--,F,0,>) (1,--,F,1,>)

- Si noti che non esistono quintuple che abbiano lo stato F come loro primo valore per cui la computazione termina.

# Esempio: Controllo di parità in Base 2

da <https://www.valcon.it/mdt/controllo-di-parita-base-2/>

Aggiunge, a destra, il bit di parità a una sequenza binaria

*NASTRO INIZIALE*

1  
0  
11111  
11011

*NASTRO FINALE*

11  
00  
11111  
110110

## Algoritmo

I bit **0** mantengono lo stato **Pari** oppure **Dispari**

I bit **1** cambiano lo stato da **Dispari** in **Pari**, oppure da **Pari** in **Dispari**

Se termina in **Pari** scrive **0**

Se termina in **Dispari** scrive **1**

# Esempio: Controllo di parità in Base 2 CODICE

- (0,0,P,0,>)
- (0,1,D,1,>)
- (P,0,P,0,>)
- (P,1,D,1,>)
- (P,-,H,0,>)
- (D,0,D,0,>)
- (D,1,P,1,>)
- (D,-,H,1,>)

# MACCHINE SPECIFICHE

---

- Una volta definita la parte di controllo, la Macchina di Turing è capace di risolvere un dato problema (risolubile)..... **ma così facendo, essa è *specifica di quel problema!***
- **Siamo circondati** da macchine specifiche:
  - calcolatrici
  - lavastoviglie,
  - videoregistratori, videogiochi
  - orologi, telecamere, ...

# MACCHINE SPECIFICHE

---

## Conviene fare macchine specifiche?

- sì, per usi particolari e mercati di massa...
- no, se vogliamo una macchina di uso generale con cui risolvere ogni problema (risolubile)

# MACCHINE UNIVERSALI

---

- Una volta definita la parte di controllo, cioè l'**algoritmo necessario**, la Macchina di Turing è capace di risolvere un dato problema (risolubile)
- Finora, l'algoritmo era ***cablato nella macchina***
- ..e se invece fosse ***sul nastro***, e la macchina se lo andasse a prendere?

# MACCHINA DI TURING UNIVERSALE

---

- Una Macchina di Turing la cui parte di controllo (cioè il cui algoritmo “cablato”) consiste nel leggere *dal nastro una descrizione dell’algoritmo richiesto*
- È una macchina UNIVERSALE: senza modifiche alla sua struttura, può essere istruita per risolvere un qualunque problema (risolubile)
- Una macchina programmabile

# MACCHINA DI TURING UNIVERSALE

---

Ma leggere *dal nastro una descrizione dell'algoritmo* richiesto richiede di:

- saper descrivere tale algoritmo
- il che richiede un qualche *linguaggio*
- e una macchina che lo *interpreti*

Dunque...

- **la Universal Turing Machine (UTM) è *l'interprete di un linguaggio!***

# MACCHINA DI TURING UNIVERSALE

---

Una Universal Turing Machine (UTM)  
*modella il concetto di elaboratore di uso generale* (“general purpose”)

- Una macchina che *va a cercare le “istruzioni” da compiere...* (**fetch**)
- ... le interpreta... (**decode**)
- ... e le esegue (**execute**)

# MACCHINA DI TURING UNIVERSALE E MACCHINA DI VON NEUMAN

---

- Una Universal Turing Machine (UTM) è in grado di *elaborare*
  - prendendo *dati* e *algoritmo dal nastro*
  - e scrivendo **sul nastro** i *risultati*
- Dunque, una UTM opera solo da/verso il **nastro** (astrazione della memoria): **non esiste il concetto di “mondo esterno”**
- **Ingresso/Uscita**

# MACCHINA DI TURING UNIVERSALE E MACCHINA DI VON NEUMAN

---

Dunque,

- la macchina di Von Neumann è modellata dalla UTM per ciò che attiene alla *computazione...*
- ma prevede anche la dimensione dell' *interazione*

# PROBLEMI RISOLUBILI E COMPUTABILITÀ

---

Secondo la Tesi di Church-Turing, **non esiste un formalismo “più potente” di TM**, ossia capace di risolvere una classe più ampia di problemi

Dunque...

se neanche la macchina di Turing riesce a risolvere un problema, **quel problema non è risolubile**

## PROBLEMA RISOLUBILE

Un problema la cui soluzione può essere espressa da una Macchina di Turing o formalismo equivalente

# QUALCHE DEFINIZIONE

---

## FUNZIONE CARATTERISTICA DI UN PROBLEMA

- Dato un **problema P**,  
**l'insieme X dei suoi dati di ingresso,**  
**l'insieme Y delle risposte corrette,**  
si dice ***funzione caratteristica del problema P*** la funzione:

$$f_P: X \rightarrow Y$$

che associa a ogni dato d'ingresso la corrispondente risposta corretta

# QUALCHE DEFINIZIONE

---

## FUNZIONE CARATTERISTICA DI UN PROBLEMA

- Perché questo artificio?
- Perché grazie a questa funzione,  
**decidere *se un problema è risolubile***  
**equivale a chiedersi**  
***se la funzione  $f_P$  è computabile***

D'ora in poi parleremo quindi solo di ***funzioni computabili***, sapendo che ciò equivale a parlare di ***problemri risolubili***

# QUALCHE DEFINIZIONE

---

## FUNZIONE COMPUTABILE

Una funzione  $f: A \rightarrow B$  per la quale esiste una Macchina di Turing che

- data sul nastro una rappresentazione di  $x \in A$

*dopo un numero finito di passi*

- produce sul nastro una rappresentazione del risultato  $f(x) \in B$

# FUNZIONI NON COMPUTABILI

---

È facile dimostrare che **esistono funzioni definibili ma non computabili** e, quindi, **problemi non risolubili**

**ESEMPIO:**

**Problema dell'*halt* della macchina di Turing**

*Dire se una data macchina di Turing  $T$ , con un generico ingresso  $X$ , si ferma oppure no*

# FUNZIONI NON COMPUTABILI

La funzione caratteristica  $f_H$  di questo problema può essere così definita:

$$f_H(m, x) = \begin{cases} 1, & \text{se } m \text{ con ingresso } x \text{ si ferma} \\ 0, & \text{se } m \text{ con ingresso } x \text{ non si ferma} \end{cases}$$

Si può dimostrare che questa funzione è **definita** ma **non computabile**, in quanto tentare di calcolarla conduce a un assurdo.

Osserva che non è possibile semplicemente provare ad eseguire  $m$  con ingresso  $x$  ed aspettare: questo basterebbe per rispondere 1 (quando effettivamente l'esecuzione si ferma), ma quando rispondere 0? Anche dopo milioni di istruzioni eseguite, la terminazione potrebbe essere proprio l'effetto della prossima istruzione, come invece l'esecuzione potrebbe andare avanti all'infinito.

# PROGRAMMAZIONE

---

L'attività con cui si predisponde l'elaboratore a **eseguire un particolare insieme di azioni su particolari dati**, allo scopo di risolvere un problema



# Ada Byron: la prima programmatrice (1815-1852)

**Ada Augusta Byron**, figlia del poeta George Byron, collaborò con Babbage seguendo i progetti della Macchina Analitica e arricchendo con numerosi ed importanti contributi (idea di esecuzione sequenziale e loop) **il concetto di programmazione**



# ADA Byron

- 1833. Incontro con Charles Babbage.
- Primo programma di computer : algoritmo (per il calcolo dei numeri di Bernoulli) codificato per essere elaborato da una macchina (*macchina analitica di Babbage*).
- Parla di variabili, operazioni e stati e si esprime in linguaggio puramente simbolico.
- Strutture cicliche annidate (problemi ricorrenti).
- Uso di un linguaggio molto simile all'attuale assembly

# PROBLEMI DA RISOLVERE

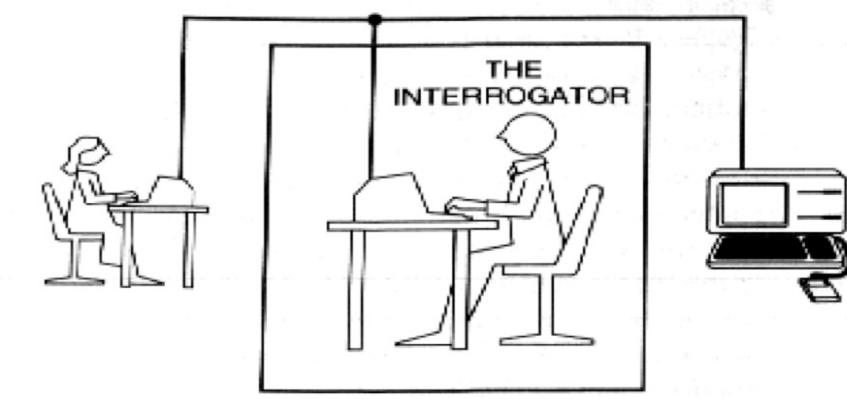
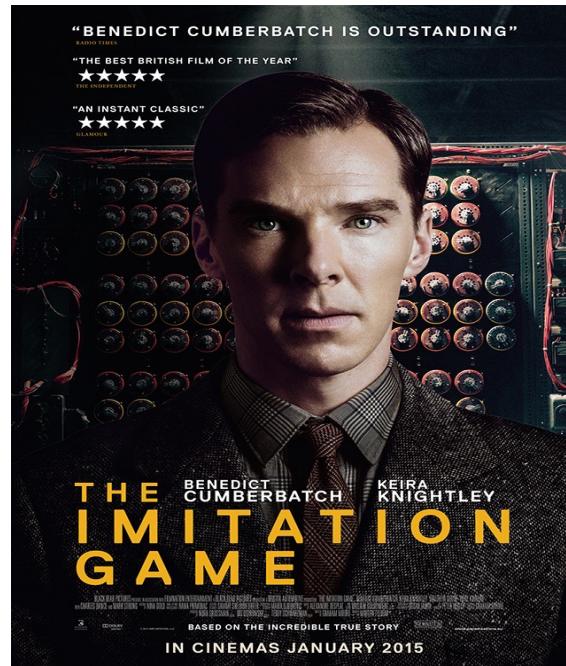
---

I problemi che siamo interessati a risolvere con l'elaboratore sono di **natura molto varia**:

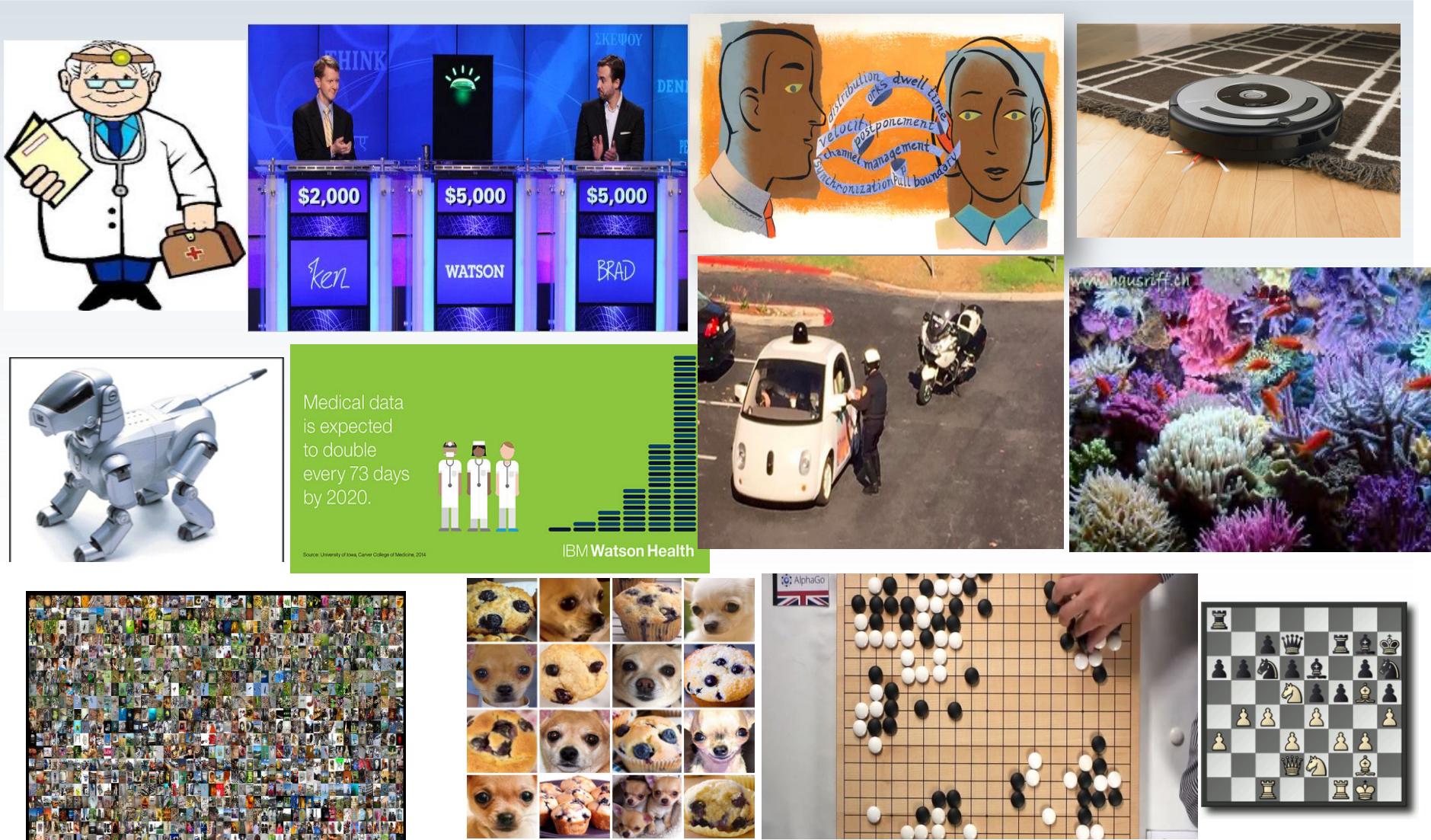
- Dati due numeri trovare il **maggior**e
- Dato un elenco di nomi e relativi numeri di telefono **trovare** il numero di telefono di una determinata persona
- Dati a e b, **risolvere l'equazione**  $ax+b=0$
- Stabilire se una parola viene **alfabeticamente** prima di un'altra
- **Somma** di due numeri interi
- Scrivere tutti gli n per cui l'equazione:  $X^n + Y^n = Z^n$   
ha soluzioni intere positive (“Non esistono soluzioni intere positive se N > 2” - **Teorema di Fermat-Wiles 1995**)
- **Ordinare** una lista di elementi
- Calcolare il **massimo comune divisore** fra due numeri dati
- Calcolare il **massimo** in un insieme
- .....

# Ma anche problemi più complessi... Turing: Le macchine possono pensare? Computing Machinery and Intelligence (1950)

- Problema descritto nei termini di un gioco (gioco dell'imitazione)
- Abbiamo tre protagonisti: Un uomo, una macchina, un interrogante che in una stanza separata deve determinare quale sia l'uomo e quale la macchina.
- *E' possibile che computer digitali con adeguata memoria, in grado di aumentare la loro velocità di azione e adeguatamente programmati possano ingannare l'interrogante comportandosi come un essere umano?*
- Alle origini dell'Intelligenza Artificiale.



# Applicazioni dell'Intelligenza Artificiale



# Sistemi che rispondono a domande in linguaggio naturale

ChatGPT <https://chat.openai.com/chat>. PROVATELO (diverse lingue)!!

Modello del linguaggio naturale per calcolare la probabilità di una frase e fare previsioni nel testo addestrato su 45 terabyte di testo (più di 600 anni per essere letto da un umano).

## Esempi:

"Explain quantum computing in simple terms" →

"Got any creative ideas for a 10 year old's birthday?" →

"How do I make an HTTP request in Javascript?" →

## Capacità

Ricorda precedenti conversazioni

Consente correzioni da parte dell'utente

Addestrato a evitare risposte inopportune

## Limitazioni

Può generare a volte risposte scorrette (allucinazioni..).

Può avolte produrre risposte pericolose o discriminatorie.

Conoscenza limitata ma in continuo aggiornamento.

# Esempio di dialogo (1):

PM: Dovrei fare una presentazione breve delle tue caratteristiche. Una slide. Puoi farla tu?

OpenAI: Mi dispiace, non sono in grado di creare slide visive. Tuttavia, posso fornirti una descrizione di una slide che potresti creare per presentare le mie caratteristiche:

Slide 1: Introduzione a ChatGPT

- Titolo: "ChatGPT: Il modello di linguaggio di OpenAI"
- Immagine rappresentativa del modello, ad esempio il logo di OpenAI
- Sottotitolo: "Un modello di generazione di testo basato su trasformatori"
- Elenca le caratteristiche principali di ChatGPT in un bullet point:
  - Generazione di testo in diversi contesti
  - Comprensione del linguaggio naturale
  - Addestrato su un vasto corpus di testo
  - Utilizzato per automatizzare e velocizzare i processi di generazione di testo
- Invita le persone a provare ChatGPT e vedere come può essere utilizzato per la loro attività.

Spero che questo ti aiuti a creare una slide efficace per presentare le mie caratteristiche.

# RISOLUZIONE DI PROBLEMI

---

- La descrizione del problema non fornisce (in generale) un metodo per risolverlo
  - Affinché un problema sia risolvibile è necessario che la sua definizione sia chiara e completa
- ***Non tutti*** i problemi sono risolvibili attraverso l'uso del calcolatore. Esistono classi di problemi per le quali la soluzione automatica non è proponibile. Ad esempio:
  - se il problema presenta infinite soluzioni
  - per alcuni dei problemi ***non è stato trovato un metodo risolutivo***
  - per alcuni problemi è stato dimostrato che ***non esiste un metodo risolutivo automatizzabile (funzioni non computabili)***

# RISOLUZIONE DI PROBLEMI

---

- Noi ci concentreremo sui problemi che, ragionevolmente,  
***ammettono un metodo risolutivo***  
    → ***funzioni calcolabili***
- Uno degli obiettivi del corso è presentare le ***tecnologie*** e  
le ***metodologie di programmazione***
  - **Tecnologie**: strumenti per lo sviluppo di programmi
  - **Metodologie**: metodi per l'utilizzo corretto ed efficace delle tecnologie di programmazione

# RISOLUZIONE DI PROBLEMI

---

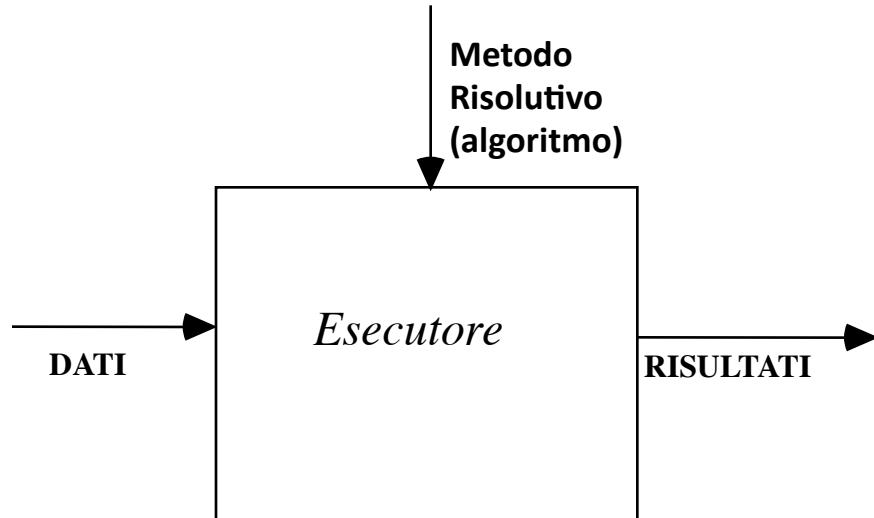
La risoluzione di un problema è il processo che dato un problema e individuato un opportuno metodo risolutivo, trasforma i dati iniziali nei corrispondenti risultati finali

Affinché la risoluzione di un problema possa essere realizzata attraverso l'uso del calcolatore, tale processo deve poter essere definito come *sequenza di azioni elementari*

# ALGORITMO

Un algoritmo è una sequenza **finita** di mosse che risolve ***in un tempo finito*** una classe di problemi

L'esecuzione delle azioni *nell'ordine specificato dall'algoritmo* consente di ottenere, a partire dai dati di ingresso, i risultati che risolvono il problema



## ESECUTORE

una **macchina astratta** capace di **eseguire le azioni** specificate dall'algoritmo

# ALGORITMI: PROPRIETÀ

---

- **Eseguibilità:** ogni azione deve essere *eseguibile* dall'esecutore *in un tempo finito*.
- **Non-ambiguità:** ogni azione deve essere *univocamente interpretabile* dall'esecutore.
- **Finitezza:** il numero totale di azioni da eseguire, per ogni insieme di dati di ingresso, deve essere finito

# ALGORITMI: PROPRIETÀ (2)

---

**Quindi, l'algoritmo deve:**

- essere *applicabile a qualsiasi insieme di dati di ingresso* appartenenti al **dominio di definizione** dell'algoritmo.
- essere costituito da operazioni appartenenti ad un determinato **insieme di operazioni fondamentali**.
- essere costituito da **regole non ambigue**, cioè interpretabili in modo **univoco** qualunque sia l'esecutore (persona o “macchina”) che le legge.

# ALGORITMI E PROGRAMMI

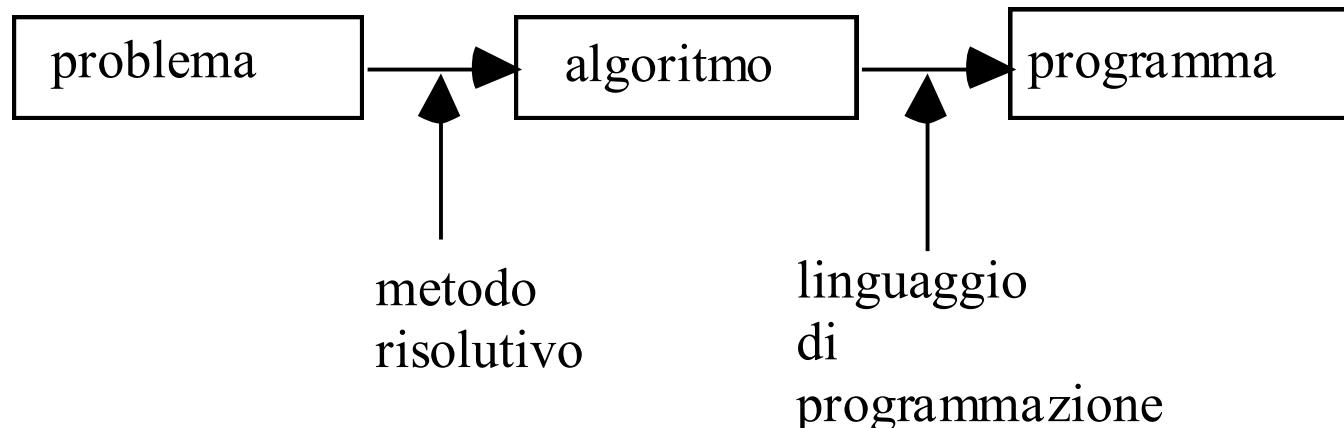
---

- Ogni elaboratore è una macchina in grado di eseguire azioni elementari su oggetti detti **DATI**.
- L'esecuzione delle azioni è richiesta all'elaboratore tramite comandi elementari chiamati **ISTRUZIONI** espresse attraverso un opportuno formalismo: il **LINGUAGGIO di PROGRAMMAZIONE**
- La formulazione testuale di un algoritmo in un linguaggio comprensibile a un elaboratore è detta **PROGRAMMA**.
- **Un programma è un testo** scritto in accordo alla **sintassi** e alla **semantica** di un linguaggio di programmazione.

# ALGORITMO & PROGRAMMA

Passi per la risoluzione di un problema:

- individuazione di un procedimento risolutivo
- scomposizione del procedimento in un insieme ordinato di azioni → **ALGORITMO**
- rappresentazione dei dati e dell'algoritmo attraverso un formalismo comprensibile dal calcolatore → **LINGUAGGIO DI PROGRAMMAZIONE**



# UN ESEMPIO DI PROGRAMMA (in linguaggio C)

---

```
main()  {
    int A, B;
    printf("Immettere due numeri: ");
    scanf("%d %d", &A, &B);
    printf("Somma: %d\n", A+B);
}
```

# ALGORITMI: ESEMPI

---

- **Soluzione dell'equazione  $ax+b=0$**

- *leggi i valori di a e b*
- *calcola  $-b$*
- *dividi quello che hai ottenuto per a e chiama x il risultato*
- *stampa x*

- **Calcolo del massimo di un insieme**

- *Leggi i valori dell'insieme Ins*
- *Scegli un elemento come massimo provvisorio max*
- *Per ogni elemento i dell'insieme: se  $i > max$  eleggi i come nuovo massimo provvisorio max*
- *Il risultato è max*

NOTA: si utilizzano **VARIABILI**, ossia nomi simbolici usati nell'algoritmo per denotare dati

# ALGORITMI: ESEMPI

---

**Stabilire se una parola  $P$  viene alfabeticamente prima di una parola  $Q$**

- leggi  $P$  e  $Q$
- ripeti quanto segue:
  - se prima lettera di  $P$  < prima lettera di  $Q$
  - allora scrivi vero
  - altrimenti se prima lettera  $P$  > prima lettera di  $Q$
  - allora scrivi falso
  - altrimenti (le prime lettere sono =)
  - togli da  $P$  e  $Q$  la prima lettera
- fino a quando hai trovato le prime lettere di  $P$  e  $Q$  diverse
- Nota: funziona solo con parole  $P$  e  $Q$  di uguale lunghezza e con parole diverse
- Esercizio proposto: rilassare tali condizioni

# ALGORITMI: ESEMPI

---

• **Somma degli elementi dispari di un insieme:** Detto  $INS$  l'insieme di elementi (letto) considero un elemento  $X$  di  $INS$  alla volta senza ripetizioni. Se  $X$  è dispari, sommo  $X$  a un valore  $S$  inizialmente posto uguale a  $0$ . Se  $X$  è pari non compio alcuna azione. Termino quando ho considerato tutti gli elementi di  $INS$  e stampo il risultato contenuto in  $S$ .

• **Somma di due numeri:  $X + Y = Z$**  (ovvero incrementare il valore di  $Z$ , inizialmente posto uguale a  $X$  per  $Y$  volte) con un esecutore che non sa calcolare espressioni direttamente, ma sa solo decrementare, incrementare e confrontare valori.

- Leggi i dati e inseriscili in  $X$  e  $Y$
- poni  $Z = X$
- poni  $U = 0$
- finché  $U$  è diverso da  $Y$ 
  - incrementa  $Z$   $(Z=Z+1)$
  - incrementa  $U$   $(U=U+1)$
- Stampa il risultato  $Z$

# ALGORITMI EQUIVALENTI

---

Due algoritmi si dicono *equivalenti* quando:

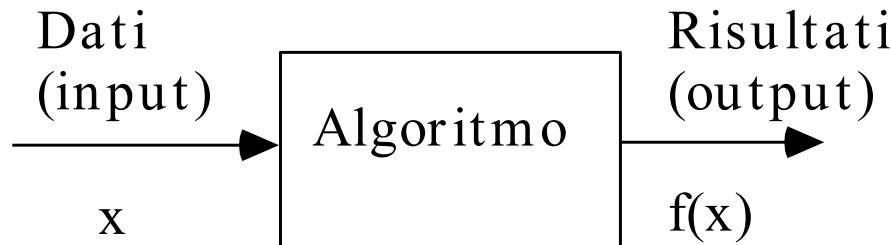
- hanno lo stesso **dominio di ingresso**
- hanno lo stesso **dominio di uscita**
- in corrispondenza degli **stessi valori del dominio di ingresso producono gli stessi valori nel dominio di uscita.**

# ALGORITMI EQUIVALENTI (2)

---

Due algoritmi *equivalenti*

- forniscono lo **stesso risultato**
- ma possono avere **diversa efficienza**
- e possono essere **profondamente diversi!**



# ALGORITMI EQUIVALENTI (3)

---

**ESEMPIO: calcolo del M.C.D. fra due interi  $M$ ,  $N$**

- **Algoritmo 1**

- Leggi  $M$  e  $N$
- Calcola l'insieme  $A$  dei divisori di  $M$
- Calcola l'insieme  $B$  dei divisori di  $N$
- Calcola l'insieme  $C$  dei divisori comuni =  $A \cap B$
- Calcola il massimo Max dell'insieme  $C$
- Stampa Max

- **Algoritmo 2 (di Euclide)**

$$\text{MCD } (M,N) = \begin{cases} M \text{ (oppure } N) & \text{se } M=N \\ \text{MCD } (M-N, N) & \text{se } M>N \\ \text{MCD } (M, N-M) & \text{se } M<N \end{cases}$$

# ALGORITMI EQUIVALENTI (4)

**ESEMPIO: calcolo del M.C.D. fra due interi  $M$ ,  $N$**

## Algoritmo 2 (di Euclide)

- Leggi  $M$  e  $N$
- Finché  $M \neq N$ :
  - se  $M > N$ , sostituisci al corrente valore di  $M$  il valore  $M-N$
  - altrimenti sostituisci al corrente valore di  $N$  il valore  $N-M$
- Il Massimo Comune Divisore è il valore finale ottenuto quando  $M = N$ .
- Stampa  $M$  (o  $N$ ).

**Esempio:**

MCD fra 18 e 12 -> (18-12) e 12 -> 6 e 12 -> 6 e 6 OK risultato 6

$$\text{MCD } (M,N) = \begin{cases} M \text{ (oppure } N) & \text{se } M=N \\ \text{MCD } (M-N, N) & \text{se } M>N \\ \text{MCD } (M, N-M) & \text{se } M<N \end{cases}$$

# L'arte della programmazione (1)

- La soluzione di un problema tramite un programma non si esaurisce nello scrivere il codice in un linguaggio di programmazione. Si passa attraverso diverse fasi.
- **Definizione del problema**
  - ◆ Definizione degli ingressi e delle uscite, delle variabili...
  - ◆ Risoluzione delle ambiguità
  - ◆ Scomposizione in problemi più semplici
- **Definizione dell'algoritmo**
  - ◆ Ad esempio, soluzione in pseudocodice
- **Codifica**
  - ◆ Traduzione dell'algoritmo in un programma (istruzioni in un linguaggio di programmazione)

# L'arte della programmazione (2)

- **Debugging**
  - correzione degli errori sintattici e semantici
- **Validazione**
  - Test sul comportamento del programma anche a fronte di input particolari (es., vettori di dimensione 0 o 1, variabili nulle)
- **Documentazione**
  - Inserimento di commenti nelle varie parti del programma per facilitarne la comprensione.
- **Manutenzione**
  - Modifica del programma nel tempo per soddisfare il cambiamento delle specifiche con cui deve operare.

# Crisi del software

- Progetti oltre il budget
- Progetti oltre i limiti di tempo
- Software di scarsa qualità
- Software che spesso non rispettava i requisiti
- Progetti ingestibili e codice difficile da manutenere

# Diverse interpretazioni di algoritmo nelle soluzioni di problemi

- Tradizionale:
  - Algoritmo = strutture dati e istruzioni
- Intelligenza Artificiale classica- simbolica:
  - Algoritmo = conoscenza + controllo (motore d'inferenza)
- Intelligenza Artificiale e Machine learning (simbolica sub-simbolica, reti neurali)
  - Algoritmo viene generato automaticamente da esempi (esperienza) + apprendimento apartire da una specifica in linguaggio naturale.

Cresce il livello di generalità e “intelligenza” della macchina (si pensi all’AI generativa e chatGPT).

# Consigli per Studenti del Primo Anno di Ingegneria Informatica da parte di ChatGPT

- **IO:** Cosa diresti agli studenti del primo anno dell'Università di Ingegneria informatica su di te? La puoi sintetizzare in qualche slide?
- **ChatGPT:** Purtroppo, non sono in grado di generare direttamente immagini o di creare slide, ma posso fornirti una descrizione testuale di come potrebbe apparire la slide che hai richiesto.

# Slides ChatGPT(1)

## Introduzione

- Benvenuti nuovi studenti di Ingegneria Informatica!
- Siete pronti per un'avventura nel mondo digitale?

## Chi è ChatGPT?

- ChatGPT: Intelligenza Artificiale sviluppata da OpenAI.
- Basata su reti neurali, simula conversazioni umane.

## Applicazioni Pratiche

- Utilizzo di ChatGPT per la scrittura di codice, risoluzione di problemi, generazione di testi e altro ancora.
- Esplorate le molteplici applicazioni per arricchire il vostro apprendimento.

# Slides ChatGPT(2)

## Limiti dell'IA

- L'IA ha limitazioni: non sostituisce l'apprendimento umano e la creatività.
- Utilizzate l'IA come supporto per il vostro apprendimento.

## Programmazione e Coding

- Fondamentale per gli studenti di Ingegneria Informatica.
- Acquisite solide competenze di programmazione per sviluppare software e risolvere problemi.

## Basi Matematiche

- L'Ingegneria Informatica richiede una comprensione approfondita della matematica.
- Studiate algebra, statistica e calcolo per affrontare le sfide tecniche.

# Slides ChatGPT(3)

## Collaborazione e Networking

- Lavoro di squadra e partecipazione a progetti di gruppo.
- Costruire una rete di contatti nel settore per future opportunità.

## Curiosità e Aggiornamenti

- Rimani curioso e aggiornato sulle nuove tecnologie.
- L'Ingegneria Informatica è in continua evoluzione, restate al passo con le tendenze.

## Buona Fortuna!

- In bocca al lupo per il vostro percorso universitario!
- Esplorate, imparate e abbracciate le sfide dell'Ingegneria Informatica.