



- Cosa significa programmare
- Parallelismo con problemi “noti”
- Concetto di algoritmo
  - Problem solving
  - Formalismi
  - Approcci top-down e bottom-up

# Question time!



- Cosa significa per voi programmare?
- Risposte comuni:
  - Scrivere un programma
    - Un po' banale eh?
  - Scrivere del codice
    - Sicuramente ma è la parte meno importante
  - Usare un linguaggio di programmazione
    - Chi siete? Da dove venite? Cosa portate? Dove andate? Un Fiorino!



# Cosa significa programmare?



- Programmare significa...
  - 90% → Risolvere un problema!
  - 2% → Scrivere codice
  - 8% → Capire gli errori che abbiamo fatto
- Percentuali molto indicative (ma non troppo)

- Parallelo con cucina
  - Voglio invitare il nuovo vicino a pranzo
  - Ho già in mente cosa fare
  - Problema: come lo cucino?
  - Soluzione → ricetta



## TORTA FRITTA

- Ingredienti
  - 1 kg Farina o
  - 450 g Latte Intero Caldo
  - 30 g Sale
  - 30 g Lievito di Birra Fresco
  - 1 noce di strutto
  - Strutto per friggere
- Preparazione
  - Sciogliamo 30 g di lievito di birra fresco in 50 g di latte
  - Rovesciamo la farina e versiamo al centro 400 g di latte caldo
  - Aggiungiamo una noce di strutto
  - Mescoliamo con una forchetta
  - Aggiungiamo il lievito e mescoliamo
  - Aggiungiamo 30 g di sale
  - Impastiamo formando un panetto
  - ...

- Caratteristiche ricetta:
  - Ingredienti
  - Come si usano
    - Passi da fare
    - E in che ordine!
  - Scritta in qualche “linguaggio”
- È sufficiente?
  - No
  - Esecuzione!

- Devo risolvere un problema
- Ingredienti
- Individuo ricetta
  - Passi e loro ordine
- Scrivo la ricetta
- La eseguo

- Devo risolvere un problema
- Ingredienti → Dati in ingresso
- Individuo ricetta → Scopro come risolvere
- Passi e loro ordine → **Algoritmo**
- Scrivo la ricetta → Codifica
- La eseguo → Esecuzione

**Punti fondamentali della programmazione!**



- Problema
- Dati
- Individuazione algoritmo
- Codifica
- Esecuzione

- Problema
- Dati
- Individuazione algoritmo
- Codifica
- Esecuzione



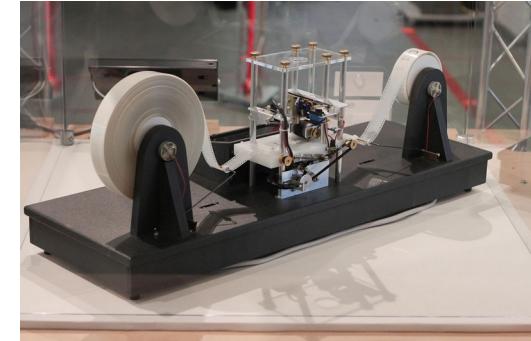
- Tipicamente richiesta committente
  - Nel vostro caso io!
- Classificazione
  - Risolvibili
  - Non risolvibili
  - Non affrontabili



Alan Turing 1912-1954



- Computabilità
  - Possibilità di risolvere un problema in maniera efficace
  - Non ha a che fare con il fatto che si conosca la soluzione
- Problemi risolvibili
  - È possibile arrivare a risolverli tramite un numero finito di passi
- Problemi non risolvibili
  - Non esiste un numero finito di passi che mi portino a risolverli
- Problemi non affrontabili
  - Numero di passi finito ma talmente elevato che non è possibile praticamente risolverli



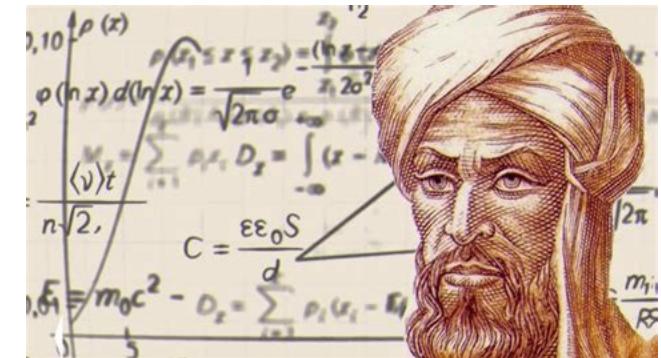
- Problema
- Dati
- Individuazione algoritmo
- Codifica
- Esecuzione

- Tipicamente forniti con il problema
- Influenzano come risolvo il problema?
- Dal punto di vista teorico no
- Dal punto di vista pratico
  - Dipende...
  - La tipologia di dati può influenzare come risolvo il problema, linguaggio di programmazione ecc. ecc.

- Problema
- Dati
- Individuazione algoritmo
- Codifica
- Esecuzione

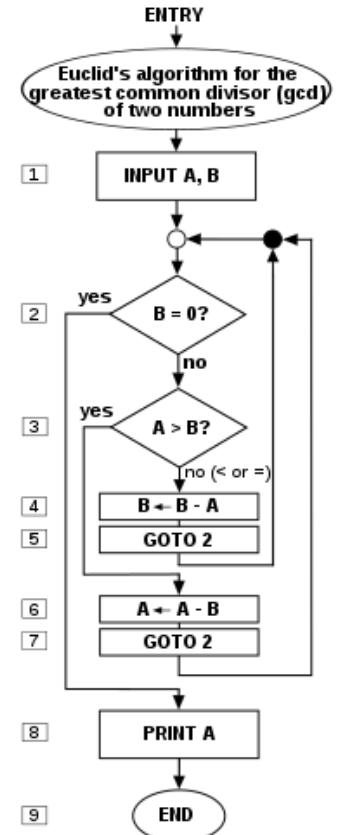


- Dal persiano al-Khwārizmī **خوارزمی**
- Un algoritmo è un metodo generale che risolve con una sequenza finita di mosse un problema dato.
- Mi dice come si risolve un problema:
  - Sequenza finita di azioni non ambigue
  - Però lo devo individuare io!



# Algoritmo

- Insieme finito di passi non ambigui che specificano le operazioni da compiere per risolvere una classe di problemi
  - Passi risolutivi
  - Loro ordine



- Problema: contattare il docente per chiarimenti
    - 1) Trovare mail docente nelle slide
    - 2) Aprire programma gestione mail
    - 3) Scrivere richiesta chiarimento indicando chi siete, quale corso seguite e anche la vostra matricola
    - 4) Premere invio
    - 5) È arrivata risposta?
    - 6) No → ripetere dal punto 5
    - 7) Sì → leggerla
    - 8) Terminare
- Esiste un altro algoritmo che risolve il problema?

- **Finito:** numero finito di passi
- **Non ambiguo:** interpretabile in maniera non equivoca
- **Terminazione:** deve terminare dopo un numero finito di passi
- **Generale:** deve affrontare il problema in generale
- **Deterministico:** a fronte degli stessi dati in ingresso deve fornire la stessa soluzione
- **Eseguibile:** deve essere effettivamente eseguibile
- **Efficiente:** minor numero di passi

- Individuare un algoritmo è la parte chiave del programmare
- L'esecuzione delle azioni nell'ordine specificato dall'algoritmo consente di ottenere a partire dai dati in ingresso i risultati che risolvono il problema
- Trovare un algoritmo è il **punto critico**

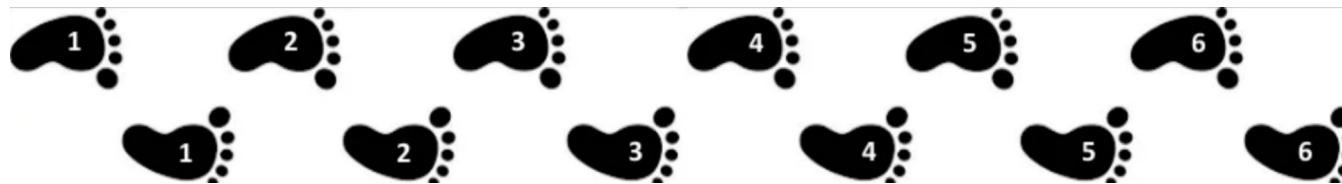
## algorithm

*noun*

Word used by programmers when they do not want to explain what they did.



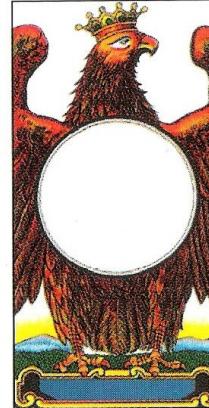
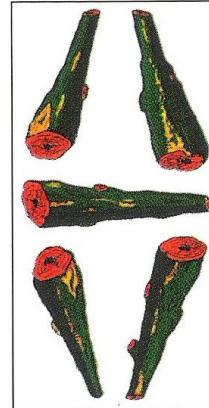
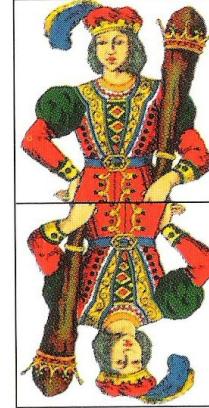
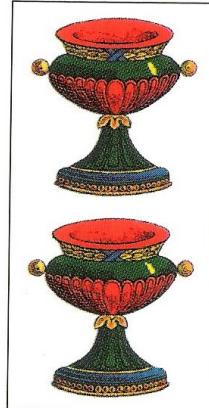
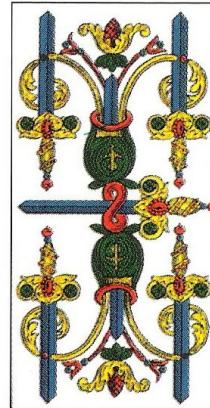
- È la parte difficile...
- Per individuare un algoritmo è necessario:
  - attenta analisi del problema
  - suddivisione in sottoproblemi
  - definizione dei possibili ingressi e uscite
  - definizione delle strutture dati
  - definizione della sequenza di passi risolutiva

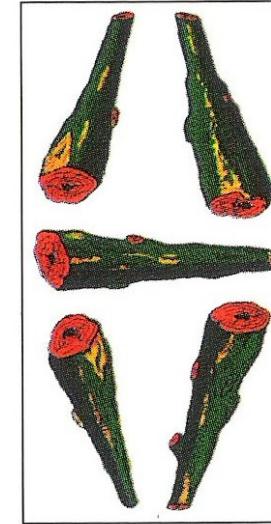
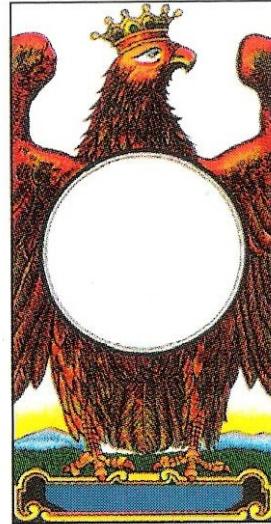
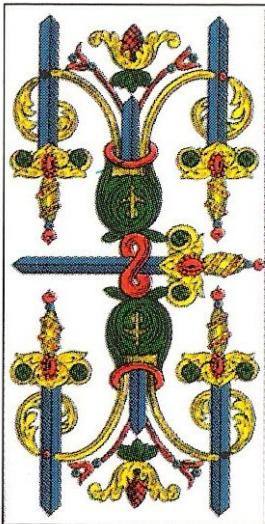


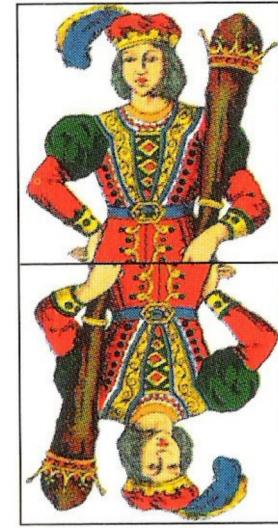
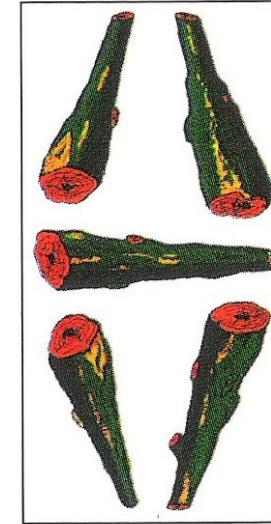
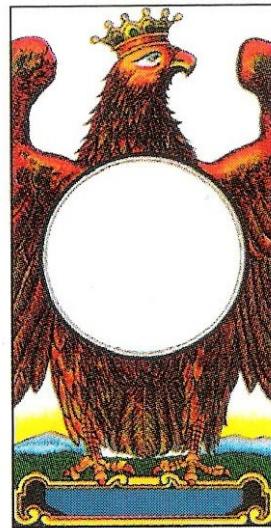
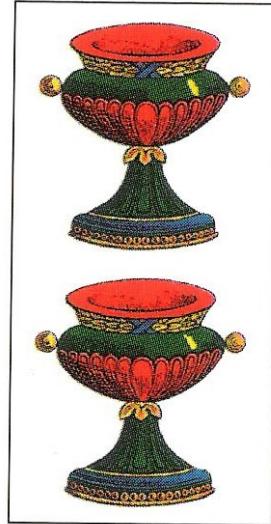
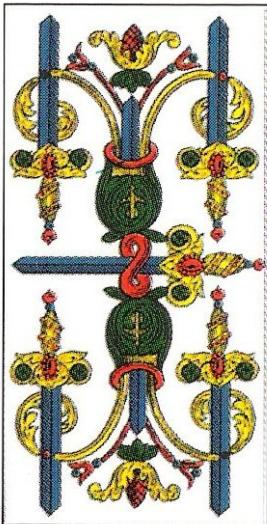
# Esempio

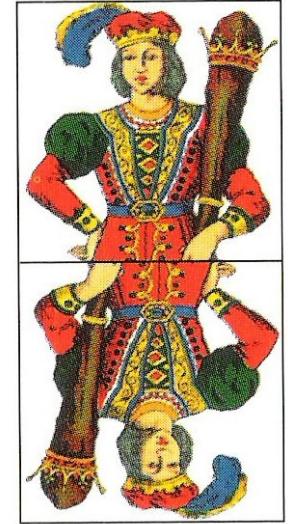
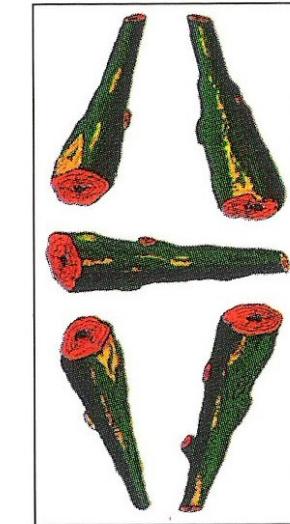
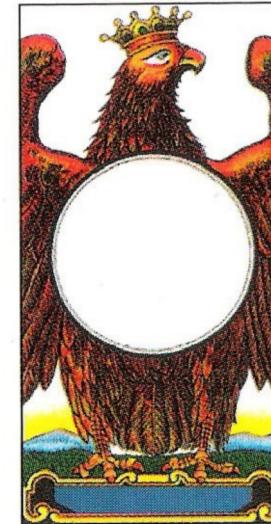
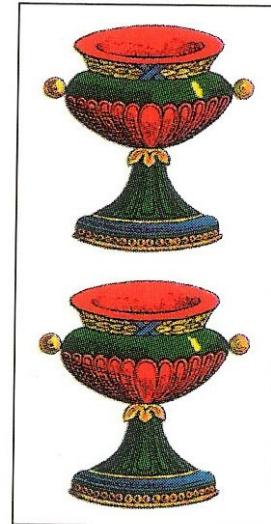
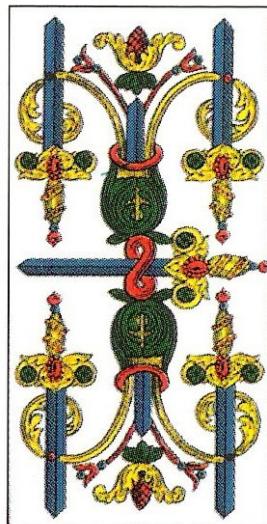


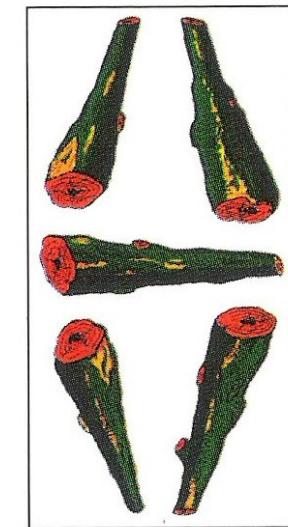
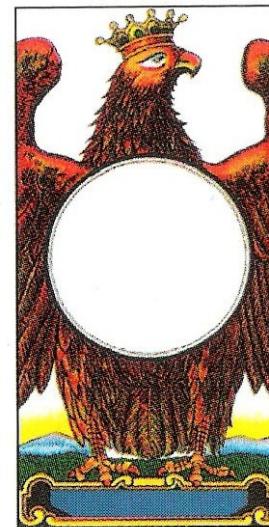
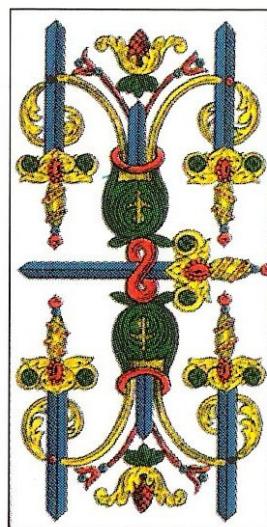
- Problema: ordinare le carte
- Approccio tipico: “insertion sort”
- Parto da sinistra verso destra e inserisco le carte che incontro nella posizione corretta

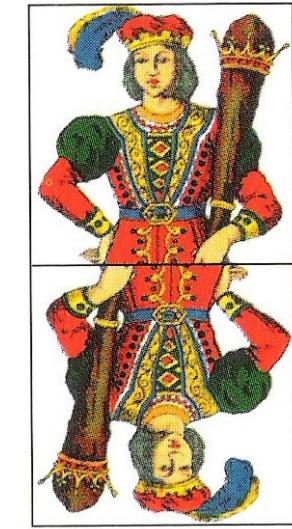
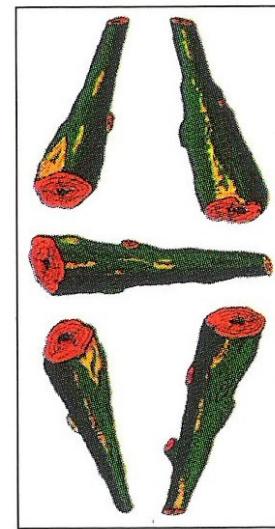
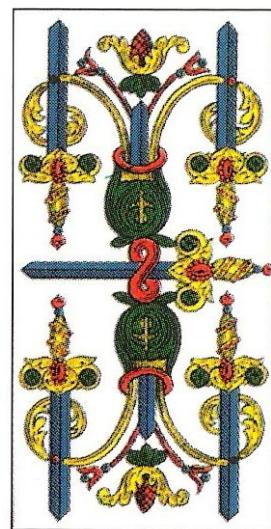
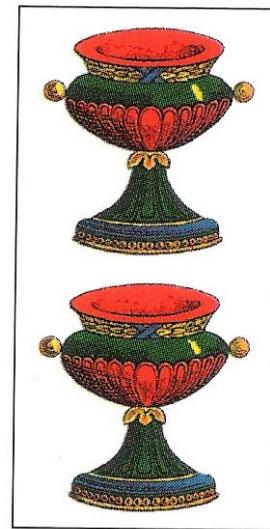
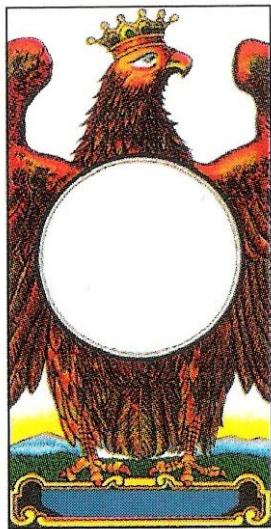


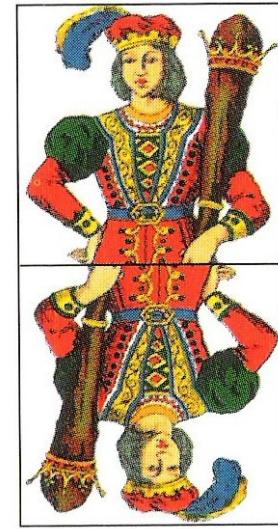
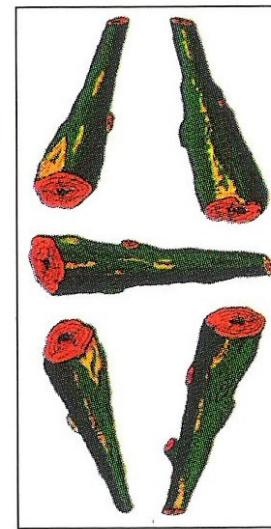
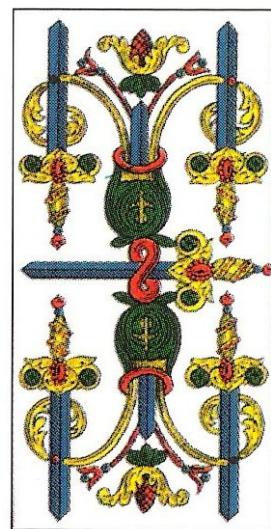
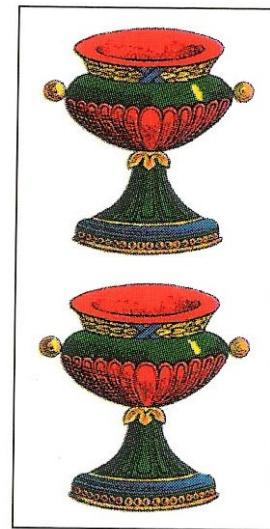
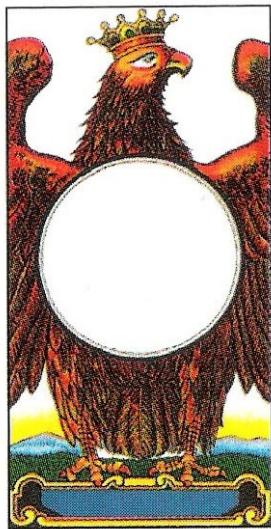












- Formalizziamo il problema: “Ordinare le carte”
  - “Parto da sinistra verso destra e inserisco le carte che incontro nella posizione corretta”
- È già un grado di dettaglio sufficiente?
- Esiste la possibilità di suddividerlo in sottoproblemi?

- Parto a considerare le carte da sinistra una alla volta
- La carta in esame è già nella posizione corretta?
  - NO: la scambio
  - SÌ: procedo oltre
- Sono arrivato a considerare tutte le carte?
  - SÌ: termine
  - NO: continuo

Ma è un grado di dettaglio sufficiente?

- La carta in esame è già nella posizione corretta?
  - NO: la scambio
  - SÌ: procedo oltre

Cosa significa posizione corretta?  
Come la scambio?

- La carta in esame è già nella posizione corretta?
  - NO: la scambio
  - SÌ: procedo oltre
- Confronto la carta in esame con quella a sinistra, ha lo stesso valore o superiore?
  - SÌ: procedo oltre
  - NO:
    - Mi sposto a sinistra fino a che non trovo una carta che ha valore inferiore o fino a che non arrivo all'inizio
    - Sposto tutte le carte tra la carta in esame e quelle appena esaminate di una posizione a sinistra
    - Sposto la carta in esame nello "spazio" che ho creato



- Cosa occorre sapere:
  - Affrontare i problemi
  - Analizzare i problemi
  - Modellare i problemi
  - Risolvere i problemi in modo efficace o efficiente o rispettando altri vincoli

- Come si procede?
  - Identificare i problemi principali
  - Decomporli in sottoproblemi
  - Ripetere fino a che si raggiungono problemi elementari
- In pratica si parte dal problema complessivo (alto) e lo si scomponete iterativamente



- Si parte dalle “primitive” del linguaggio (basso)
- Le si assembla in funzioni piú complesse
- Risultato finale → applicazione



- In realtà dipende dal caso d'uso
- Approccio bottom-up permette testing prima
- Top-down più adeguato per problem solving
- Quasi sempre approcci intermedi

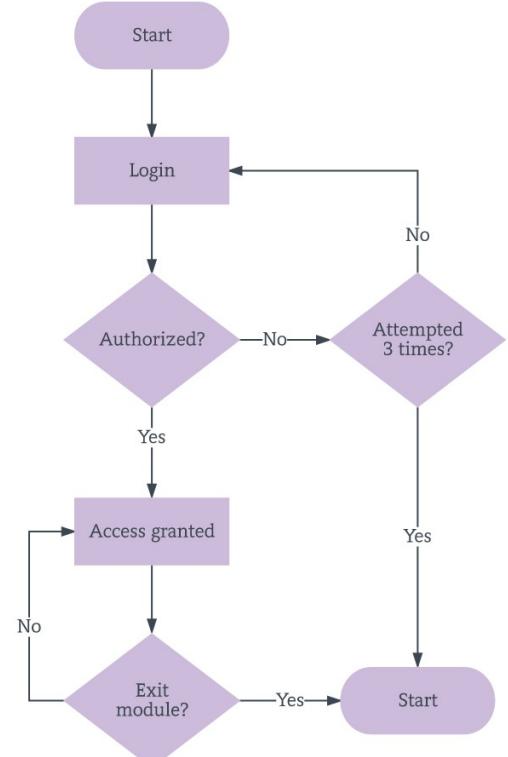
Usate approccio  
TOP-DOWN

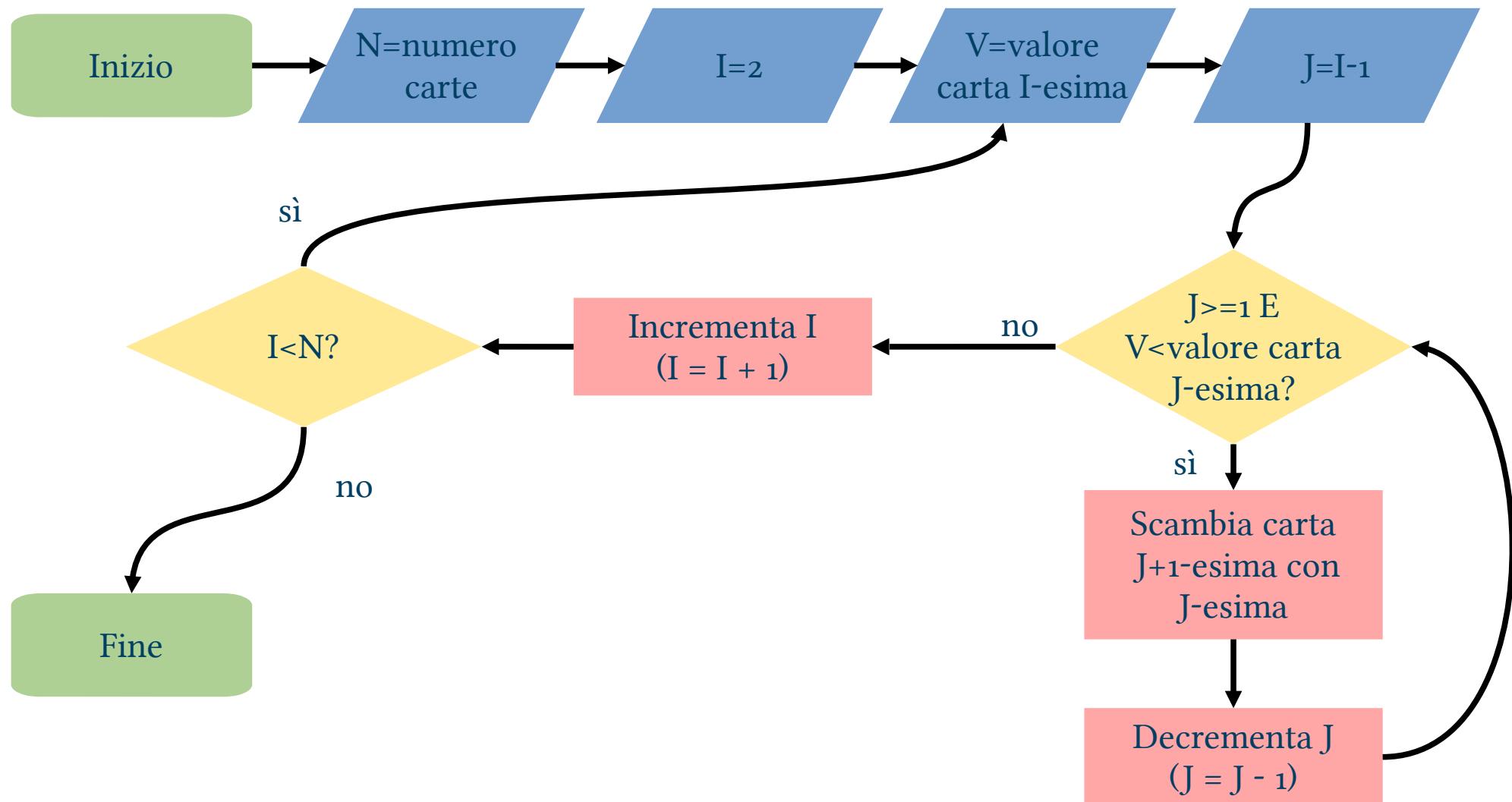
- Le lingue naturali si prestano comunque ad ambiguità
- Esistono metodi più formali per rappresentare un algoritmo
  - Diagrammi di flusso (flow chart)
  - Pseudocodice

# Algoritmi: diagrammi di flusso



- Diagramma costituito da **nodi e archi orientati**
- Ogni nodo corrisponde ad un passo dell'algoritmo
  - Spesso forme differenti in base alla tipologia
    - Assegnamento
    - Calcolo
    - Decisione
    - Inizio/Fine
- Gli archi collegano i passi nell'ordine da seguire





- Linguaggio speciale che codifica i passi
  - Meno soggetto ai limiti dei diagrammi di flusso
- Quasi un linguaggio di programmazione
  - Piú semplice
  - Svincolato (ma non troppo) allo specifico linguaggio di programmazione

$N \leftarrow \text{Length}(\text{Cards})$

For  $I \leftarrow 2$  To  $N$

$V \leftarrow \text{Cards}[I]$

$J \leftarrow I - 1$

    While  $J \geq 1$  AND  $\text{Cards}[J] > V$

$\text{Cards}[J + 1] = \text{Cards}[J]$

$J = J - 1$

- Problema
- Dati
- Individuazione algoritmo
- Codifica
- Esecuzione

- Algoritmo → Codice Sorgente
- Traduco i passi di un algoritmo nelle corrispondenti istruzioni del linguaggio di programmazione
  - Se ho ben strutturato algoritmo quasi 1:1
- Unici problemi:
  - Sintassi

- Problema: calcolare la somma di quattro numeri forniti dall'utente
- Algoritmo:
  - 1) Chiedi all'utente il primo numero e memorizzalo
  - 2) Chiedi secondo numero...
  - 3) Chiedi terzo numero...
  - 4) Chiedi quarto numero...
  - 5) Calcola la loro somma
  - 6) Stampa la loro somma

# Esempio: codifica BASIC

```
10 INPUT "Primo numero: ", A
20 INPUT "Secondo numero: ", B
30 INPUT "Terzo numero numero: ", C
40 INPUT "Quarto numero: ", D
50 S = A + B + C + D
60 PRINT "Il risultato e' ", S
```

# Esempio: codifica Python

```
#!/usr/bin/python

a = int(input("Primo numero: "))
b = int(input("Secondo numero: "))
c = int(input("Terzo numero: "))
d = int(input("Quarto numero: "))

s = a + b + c + d

print "Il risultato e' = ", s
```

# Esempio: codifica C++



```
#include <iostream>

int main(int argc, char **argv){
    int a, b, c, d;

    std::cout << "primo numero: " << std::endl;
    std::cin >> a;
    std::cout << "secondo numero: " << std::endl;
    std::cin >> b;
    std::cout << "terzo numero: " << std::endl;
    std::cin >> c;
    std::cout << "quarto numero: " << std::endl;
    std::cin >> d;

    int s = a + b + c + d;

    std::cout << "Il risultato e' = " << s << std::endl;
    return 0;
}
```

- Testo scritto in accordo alla sintassi e alla semantica di un linguaggio di programmazione
  - Ottengo: Programma Sorgente
  - A seconda del linguaggio utilizzato:
    - Interpretazione
    - Compilazione
    - Situazioni intermedie



- Problema
- Dati
- Individuazione algoritmo
- Codifica
- Esecuzione

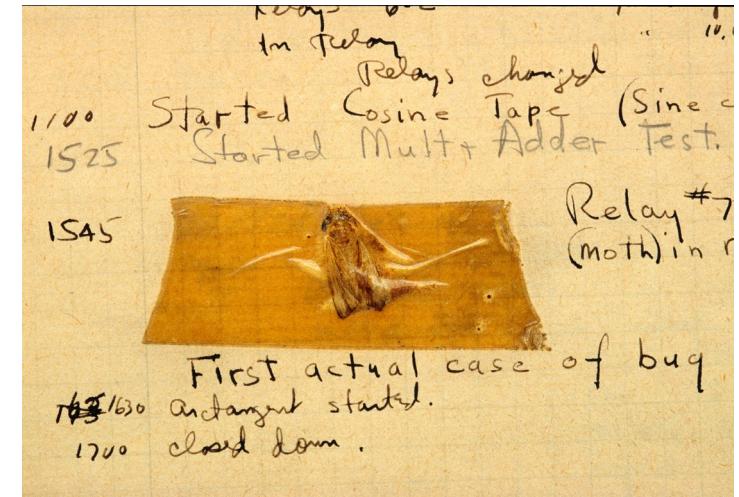


- Fase di esecuzione del programma
  - Nel caso del C deve essere preceduta da compilazione
  - Programma Sorgente → Eseguibile
- Ben difficilmente è la parte finale!
  - Debug



# Debug

- Mediamente 15-50 errori per kLOC (Lines Of Code)
  - Programmatori professionisti
- Debug
  - Rileggere codice
  - Codice ad hoc: esempio stampa
  - Strumenti specifici → debugger



1945

- Chi esegue il programma?
  - Diciamo “il computer”
- E quando esegue cosa fa esattamente?
  - Quello che abbiamo scritto noi!
  - Nessuna possibilità di decisione
  - Computer = mero esecutore di ordini o **automa**

Informatica = Informazione + Automatica

- Quanto visto in queste slide è indipendente dal linguaggio di programmazione
- Infatti prima di programmare:
  - 1) **Devo** capire come risolvere quanto è stato assegnato: “il problema”
  - 2) Una volta che l’ho capito **devo** individuare i passi necessari e il loro ordine: “algoritmo risolutivo”
  - 3) **Solamente** quando ho individuato l’algoritmo risolutivo posso scrivere il codice: “codifica”

