

# Informatica teorica

## Indice

<b>1. Lezione 01 .....</b>	<b>2</b>
1.1. Introduzione .....	2
1.1.1. Definizione .....	2
1.1.2. Cosa e come .....	2
1.2. Richiami matematici .....	2
1.2.1. Funzioni .....	2
<b>2. Lezione 02 .....</b>	<b>4</b>
2.1. Richiami matematici .....	4
2.1.1. Funzioni totali e parziali .....	4
2.1.2. Totalizzare una funzione .....	4
2.1.3. Prodotto cartesiano .....	4
2.1.4. Insieme di funzioni .....	4
2.1.5. Funzione di valutazione .....	5
2.2. Teoria della calcolabilità .....	5
2.2.1. Sistema di calcolo .....	5
2.2.2. Potenza computazionale .....	6

# 1. Lezione 01

## 1.1. Introduzione

### 1.1.1. Definizione

L'**informatica teorica** è quella branca dell'informatica che si "contrappone" all'informatica applicata: in quest'ultima, l'informatica è usata solo come uno *strumento* per gestire l'oggetto del discorso, mentre nella prima l'informatica diventa l'*oggetto* del discorso, di cui ne vengono studiati i fondamenti.

### 1.1.2. Cosa e come

Analizziamo i due aspetti fondamentali che caratterizzano ogni materia:

1. il **cosa**: l'informatica è la scienza che studia l'informazione e la sua elaborazione automatica mediante un sistema di calcolo. Ogni volta che ho un *problema* cerco di risolverlo automaticamente scrivendo un programma. *Posso farlo per ogni problema? Esistono problemi che non sono risolubili?* Possiamo chiamare questo primo aspetto con il nome di **teoria della calcolabilità**, quella branca che studia cosa è calcolabile e cosa non lo è, a prescindere dal costo in termini di risorse che ne deriva. In questa parte utilizzeremo una caratterizzazione molto rigorosa e una definizione di problema il più generale possibile, così che l'analisi non dipenda da fattori tecnologici, storici...
2. il **come**: è relazionato alla **teoria della complessità**, quella branca che studia la quantità di risorse computazionali richieste nella soluzione automatica di un problema. Con *risorsa computazionale* si intende qualsiasi cosa venga consumato durante l'esecuzione di un programma, ad esempio:
  - elettricità;
  - numero di processori;
  - tempo;
  - spazio di memoria.

In questa parte daremo una definizione rigorosa di tempo, spazio e di problema efficientemente risolubile in tempo e spazio, oltre che uno sguardo al famoso problema  $P = NP$ .

Possiamo dire che il *cosa* è uno studio **qualitativo**, mentre il *come* è uno studio **quantitativo**.

Grazie alla teoria della calcolabilità individueremo le funzioni calcolabili, di cui studieremo la complessità.

## 1.2. Richiami matematici

### 1.2.1. Funzioni

Una **funzione** da un insieme  $A$  ad un insieme  $B$  è una *legge*, spesso indicata con  $f$ , che spiega come associare agli elementi di  $A$  un elemento di  $B$ .

Abbiamo due tipi di funzioni:

- **generale**: la funzione è definita in modo generale come  $f : A \rightarrow B$ , in cui  $A$  è detto **dominio** di  $f$  e  $B$  è detto **codominio** di  $f$ ;
- **locale/puntuale**: la funzione riguarda i singoli valori  $a$  e  $b$ :

$$f(a) = b \quad | \quad a \xrightarrow{f} b$$

in cui  $b$  è detta **immagine** di  $a$  rispetto ad  $f$  e  $a$  è detta **controimmagine** di  $b$  rispetto ad  $f$ .

Possiamo categorizzare le funzioni in base ad alcune proprietà:

- **iniettività**: una funzione  $f : A \rightarrow B$  si dice *iniettiva* se e solo se:

$$\forall a_1, a_2 \in A \quad a_1 \neq a_2 \implies f(a_1) \neq f(a_2)$$

In poche parole, non ho *confluenze*, ovvero *elementi diversi finiscono in elementi diversi*.

- **suriettività**: una funzione  $f : A \rightarrow B$  si dice *suriettiva* se e solo se:

$$\forall b \in B \quad \exists a \in A \mid f(a) = b.$$

In poche parole, *ogni elemento del codominio ha almeno una controimmagine*.

Se definiamo l'**insieme immagine**:

$$\text{Im}_f = \{b \in B \mid \exists a \in A \text{ tale che } f(a) = b\} = \{f(a) \mid a \in A\} \subseteq B$$

possiamo dare una definizione alternativa di funzione suriettiva, in particolare una funzione è *suriettiva* se e solo se  $\text{Im}_f = B$ .

Infine, una funzione  $f : A \rightarrow B$  si dice **biiettiva** se e solo se è iniettiva e suriettiva, quindi vale:  
 $\forall b \in B \quad \exists! a \in A \mid f(a) = b$ .

Se  $f : A \rightarrow B$  è una funzione biiettiva, si definisce **inversa** di  $f$  la funzione  $f^{-1} : B \rightarrow A$  tale che:

$$f(a) = b \iff f^{-1}(b) = a.$$

Per definire la funzione inversa  $f^{-1}$ , la funzione  $f$  deve essere biiettiva: se così non fosse, la sua inversa avrebbe problemi di definizione.

Un'operazione definita su funzioni è la **composizione**: date  $f : A \rightarrow B$  e  $g : B \rightarrow C$ , la funzione  $f$  *composto*  $g$  è la funzione  $g \circ f : A \rightarrow C$  definita come  $(g \circ f)(a) = g(f(a))$ .

La composizione *non è commutativa*, quindi  $g \circ f \neq f \circ g$  in generale, ma è *associativa*, quindi  $(f \circ g) \circ h = f \circ (g \circ h)$ .

La composizione *f composto g* la possiamo leggere come *prima agisce f poi agisce g*.

Dato l'insieme  $A$ , la **funzione identità** su  $A$  è la funzione  $i_A : A \rightarrow A$  tale che:

$$i_A(a) = a \quad \forall a \in A,$$

ovvero è una funzione che mappa ogni elemento su se stesso.

Grazie alla funzione identità, possiamo dare una definizione alternativa di funzione inversa: data la funzione  $f : A \rightarrow B$  biiettiva, la sua inversa è l'unica funzione  $f^{-1} : B \rightarrow A$  che soddisfa:

$$f \circ f^{-1} = f^{-1} \circ f = \text{id}_A.$$

Possiamo vedere  $f^{-1}$  come l'unica funzione che ci permette di *tornare indietro*.

## 2. Lezione 02

### 2.1. Richiami matematici

#### 2.1.1. Funzioni totali e parziali

Prima di fare un'ulteriore classificazione per le funzioni, introduciamo la notazione  $f(a) \downarrow$  per indicare che la funzione  $f$  è definita per l'input  $a$ , mentre la notazione  $f(a) \uparrow$  per indicare la situazione opposta.

Ora, data  $f : A \rightarrow B$  diciamo che  $f$  è:

- **totale** se è definita *per ogni elemento*  $a \in A$ , ovvero  $f(a) \downarrow \forall a \in A$ ;
- **parziale** se è definita *per qualche elemento*  $a \in A$ , ovvero  $\exists a \in A \mid f(a) \downarrow$ .

Chiamiamo **dominio** (o *campo*) **di esistenza** di  $f$  l'insieme

$$\text{Dom}_f = \{a \in A \mid f(a) \downarrow\} \subseteq A.$$

Notiamo che:

- $\text{Dom}_f \subsetneq A \implies f$  parziale;
- $\text{Dom}_f = A \implies f$  totale.

#### 2.1.2. Totalizzare una funzione

È possibile *totalizzare* una funzione parziale definendo una funzione a tratti  $\bar{f} : A \rightarrow B \cup \{\perp\}$  tale che

$$\bar{f}(a) = \begin{cases} f(a) & a \in \text{Dom}_f(a) \\ \perp & \text{altrimenti} \end{cases}.$$

Il nuovo simbolo introdotto è il *simbolo di indefinito*, e viene utilizzato per tutti i valori per cui la funzione di partenza  $f$  non è appunto definita.

Da qui in avanti utilizzeremo  $B_\perp$  come abbreviazione di  $B \cup \{\perp\}$ .

#### 2.1.3. Prodotto cartesiano

Chiamiamo **prodotto cartesiano** l'insieme

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\},$$

che rappresenta l'insieme di tutte le *coppie ordinate* di valori in  $A$  e in  $B$ .

In generale, il prodotto cartesiano **non è commutativo**, a meno che  $A = B$ .

Possiamo estendere il concetto di prodotto cartesiano a  $n$ -uple di valori, ovvero

$$A_1 \times \dots \times A_n = \{(a_1, \dots, a_n) \mid a_i \in A_i\}.$$

L'operazione "*opposta*" è effettuata dal **proiettore**  $i$ -esimo: esso è una funzione che estrae l' $i$ -esimo elemento di un tupla, ovvero è una funzione  $\pi_i : A_1 \times \dots \times A_n \rightarrow A_i$  tale che

$$\pi_i(a_1, \dots, a_n) = a_i$$

Per comodità chiameremo  $\underbrace{A \times \dots \times A}_n = A^n$

#### 2.1.4. Insieme di funzioni

L'insieme di tutte le funzioni da  $A$  in  $B$  si indica con

$$B^A = \{f : A \rightarrow B\}.$$

Viene utilizzata questa notazione in quanto la cardinalità di  $B^A$  è esattamente  $|B|^{|A|}$ , se  $A$  e  $B$  sono insiemi finiti.

In questo insieme sono presenti anche tutte le funzioni parziali da  $A$  in  $B$ : mettiamo in evidenza questa proprietà, scrivendo

$$B_{\perp}^A = \{f : A \rightarrow B_{\perp}\}.$$

Sembrano due insiemi diversi ma sono la stessa cosa: nel secondo viene messo in evidenza il fatto che tutte le funzioni che sono presenti sono totali oppure parziali che sono state totalizzate.

### 2.1.5. Funzione di valutazione

Dati  $A, B$  e  $B_{\perp}^A$  si definisce **funzione di valutazione** la funzione

$$\omega : B_{\perp}^A \times A \rightarrow B$$

tale che

$$w(f, a) = f(a).$$

In poche parole, è una funzione che prende una funzione  $f$  e la valuta su un elemento  $a$  del dominio.

Abbiamo due possibili approcci:

- tengo fisso  $a$  e provo tutte le funzioni  $f$ : sto eseguendo un *benchmark*, quest'ultimo rappresentato da  $a$ ;
- tengo fissa  $f$  e provo tutte le  $a$  del dominio: sto ottenendo il grafico di  $f$ .

## 2.2. Teoria della calcolabilità

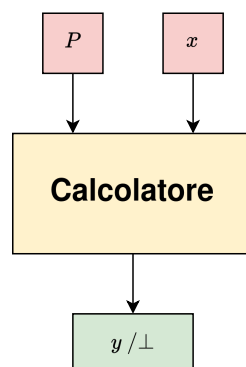
### 2.2.1. Sistema di calcolo

Quello che faremo ora è *modellare teoricamente un sistema di calcolo*.

Un **sistema di calcolo** lo possiamo vedere come una *black-box* che prende un programma  $P$ , una serie di dati  $x$  e calcola il risultato di  $P$  sull'input  $x$ .

La macchina ci restituisce:

- $y$  se è riuscita a calcolare un risultato;
- $\perp$  se è entrata in un loop.



Un sistema di calcolo quindi è una funzione del tipo

$$\mathcal{C} : \text{PROG} \times \text{DATI} \rightarrow \text{DATI}_{\perp}.$$

Come vediamo, è molto simile ad una funzione di valutazione: infatti, la parte dei dati  $x$  la riusciamo a “mappare” sull’input  $a$  del dominio, ma facciamo più fatica con l’insieme dei programmi, questo perché non abbiamo ancora definito cos’è un programma.

Un **programma**  $P$  è una **sequenza di regole** che trasformano un dato di input in uno di output (o in un loop): in poche parole, un programma è una funzione del tipo

$$P : \text{DATI} \longrightarrow \text{DATI}_{\perp},$$

ovvero una funzione che appartiene all’insieme  $\text{DATI}_{\perp}^{\text{DATI}}$ .

Con questa definizione riusciamo a “mappare” l’insieme PROG sull’insieme delle funzioni che ci serviva per definire la funzione di valutazione.

Formalmente, un sistema di calcolo è quindi la funzione

$$\mathcal{C} : \text{DATI}_{\perp}^{\text{DATI}} \times \text{DATI} \longrightarrow \text{DATI}.$$

Con  $\mathcal{C}(P, x)$  indichiamo la funzione calcolata da  $P$  su  $x$ , ovvero la sua **semantica**, il suo *significato*.

Il modello classico che viene considerato quando si parla di calcolatori è quello di **Von Neumann**.

### 2.2.2. Potenza computazionale

Definiamo la **potenza computazionale** di un calcolatore come l’insieme di tutte le funzioni che quel sistema di calcolo può calcolare grazie ai programmi, ovvero

$$F(\mathcal{C}) = \{\mathcal{C}(P, \_) \mid P \in \text{PROG}\} \subseteq \text{DATI}_{\perp}^{\text{DATI}}.$$

In poche parole,  $F(\mathcal{C})$  rappresenta l’insieme di tutte le possibili semantiche di funzioni che sono calcolabili con il sistema  $\mathcal{C}$ .

Stabilire *cosa può fare l’informatica* equivale a studiare quest’ultima inclusione: in particolare, se

- $F(\mathcal{C}) \subsetneq \text{DATI}_{\perp}^{\text{DATI}}$  allora esistono compiti **non automatizzabili**;
- $F(\mathcal{C}) = \text{DATI}_{\perp}^{\text{DATI}}$  allora l’informatica può fare tutto.

Se ci trovassimo nella prima situazione dovremmo individuare una sorta di “recinto” per dividere le funzioni calcolabili da quelle che non lo sono.

Calcolare una funzione vuole dire *risolvere un problema*: ad ognuno di questi associo una **funzione soluzione**, che mi permette di risolvere in modo automatico il problema.

Grazie a questa definizione, calcolare le funzioni equivale a risolvere problemi.

In che modo possiamo risolvere l’inclusione?

Un primo approccio è quello della **cardinalità**: viene definita come una funzione che associa ad ogni insieme il numero di elementi che contiene.

Sembra un ottimo approccio, ma presenta alcuni problemi: infatti, funziona solo quando l’insieme è finito, mentre è molto fragile quando si parla di insiemi infiniti.

Ad esempio, gli insiemi

- $\mathbb{N}$  dei numeri naturali e
- $\mathbb{P}$  dei numeri naturali pari

sono tali che  $\mathbb{P} \subsetneq \mathbb{N}$  ma hanno cardinalità  $|\mathbb{N}| = |\mathbb{P}| = \infty$ .

Dobbiamo dare una definizione diversa di cardinalità, visto che possono esistere “*infiniti più fitti/densi di altri*”, come abbiamo visto nell’esempio precedente.