Algoritmi, Complessità, Calcolabilità: l'essenza dell'informatica

Gabriele Vanoni*

12 Aprile 2017

Abstract

Siamo circondati dalle tecnologie informatiche, ma che cosa le fa funzionare davvero? Sentiamo ogni giorno parlare di algoritmi, ma che cosa rappresentano? I computer possono risolvere qualsiasi problema? Con quale velocità? La risposta a queste domande sta nella teoria, ed è sorprendentemente indipendente dalla tecnologia. Cercherò nella maniera più chiara possibile di dare delle risposte raccontando alcuni degli aspetti più interessanti di una scienza che spesso non è considerata tale, nonostante tra i suoi fondatori vanti nomi del calibro di Gödel e Turing, che la idearono ben prima della nascita dei computer.

Indice

*Politecnico di Milano,
gabriele.vanoni@mail.polimi.it

1 Introduzione

Disclaimer

1. Il contenuto di questa lezione copre normalmente un corso universitario di almeno **50 ore**. Non pretendo quindi di essere esauriente nè che capiate tutto. Lo scopo è quello di farvi vedere alcune delle principali **idee** dell'informatica teorica, in modo che possiate capire di cosa tratta questa materia.

2. Non sempre presenterò gli argomenti in maniera rigorosa e formale perché ciò richiederebbe tecnicismi matematici pesanti. Cercherò di fare affidamento sulla vostra **intuizione**. Per una trattazione completa ma comunque accessibile si vedano i testi in bibliografia.

Informatica, alcune definizioni

> "Computer science is no more about computers than astronomy is about telescopes"
>
> Edsger Dijkstra

> "I shall be sorry if computer science ever flies apart into two disciplines, one logical and one technological"
>
> Robin Milner

> "Il padre dell'informatica è l'ingegneria, ma la madre è la logica"
>
> Maria Emilia Maietti

Un problema anche linguistico

- In italiano il termine informatica è omnicomprensivo. In inglese esistono almeno due differenti locuzioni: **computer science** e **information technology**.

- Storicamente l'informatica nasce nei dipartimenti di **matematica** delle università, venti anni prima della creazione del primo calcolatore elettronico.

- Quando si comincia la costruzione dei calcolatori allora entrano in campo gli **ingegneri**.

- Oggi anche chi fa un sito web è considerato un informatico.

- Ci occuperemo di quella che viene chiamata **informatica teorica**, cioè dei fondamenti matematici alla base della disciplina.

2 Preliminari matematici

Insiemi

Definizione 1 (assioma di comprensione di Cantor e Frege)**.** Un'insieme è una collezione di oggetti, caratterizzati da una proprietà.

Esempio 2. $\Omega = \{x \in \mathbb{N} | \, x > 2\}$ i.e. $\Omega = \{3, 4, 5, ...\}$

Questa definizione NON è corretta, e porta al **paradosso** di Russell.

Paradosso di Russel
Consideriamo l'insieme degli insiemi che non appartengono a se stessi $A = \{x \,|\, x \notin x\}$.
Es. L'insieme $\mathbb{P}$ dei numeri pari appartiene ad A, infatti $\mathbb{P} \notin \mathbb{P}$.
$x \in A \longleftrightarrow x \notin x$. Chiediamoci se $A \in A$.
$A \in A \longleftrightarrow A \notin A$. Assurdo!!!

La moderna teoria degli insiemi **ZF** vieta la costruzione di questo tipo di insiemi "patologici", utilizzando assiomi più restrittivi.

Dimostrazioni per induzione

Il principio di induzione è una caratteristica intrinseca dei **numeri naturali**, infatti fa parte degli **assiomi di Peano**, le regole che li definiscono.

Principio di induzione
Sia $P(n)$ una proprietà dei numeri naturali. Se P è verificata per un $c \in \mathbb{N}$ e supponendola vera per $c \leq n \leq k$ posso dimostrarla per $n = k + 1$ allora $P(n)$ è valida per ogni $n \geq c$.

Se $P(n)$ è vera per c allora è vera per $c + 1$, e quindi per $c + 2$...

Sfruttando tale principio è possibile costruire **dimostrazioni** di teoremi che riguardino proprietà di numeri naturali.

Esempio 3 (Somma dei primi n naturali). $\sum_{h=1}^{n} h = \frac{n(n+1)}{2}$. Dimostrazione per induzione su n.
Caso base: $n = 1$. $\frac{1 \cdot 2}{2} = 1$.
Ipotesi di induzione: suppongo vera la proprietà per $1 \leq n \leq k$.
Passo induttivo: dimostro la proprità per $n = k + 1$. $\sum_{h=1}^{k+1} h = \sum_{h=1}^{k} h + (k + 1) = \frac{k(k+1)}{2} + k + 1 = \frac{k(k+1)+2k+2}{2} = \frac{k^2+3k+2}{2} = \frac{(k+2)(k+1)}{2} = P(k + 1)$.

Insieme delle parti $\wp(\Omega)$

Definizione 4 (cardinalità)**.** La cardinalità di un insieme Ω finito è n se Ω contiene n elementi e viene indicata con $|\Omega|$.

Definizione 5 (insieme delle parti)**.** L'insieme delle parti di un insieme Ω è l'insieme $\wp(\Omega)$ che contiene tutti i suoi sottoinsiemi.

Teorema 6. *La cardinalità dell'insieme delle parti di un insieme finito* Ω *è* $2^{|\Omega|}$.

Dimostrazione. Per induzione sulla cardinalità di Ω, $|\Omega| = n$.
Caso base: $n = 0$. Ω è l'insieme vuoto, l'unico suo sottoinsieme è Ω stesso, per cui $1 = 2^0$.
Ipotesi d'induzione: supponiamo la proprietà vera per $n \leq k$.
Passo induttivo: dimostriamo la proprietà per $n = k + 1$. Consideriamo Ω' ottenuto da Ω togliendo un elemento x. Per ipotesi di induzione ha 2^k sottoinsiemi. Possiamo ottenere tutti i sottoinsiemi considerando che ogni elemento di $\wp(\Omega')$ dovrà essere considerato due volte: una volta così com'è e una volta aggiungendo x. Perciò $|\wp(\Omega)| = 2\wp(\Omega') = 2 \cdot 2^k = 2^{k+1}$. □

Teorema 7 (di Cantor)**.** *Per ogni insieme* Ω, $|\wp(\Omega)| > |\Omega|$.

Dimostrazione. Certamente $|\wp(\Omega)| \geq |\Omega|$. Possiamo infatti trovare una funzione iniettiva $f : \Omega \to \wp(\Omega)$ cioè che mappi ogni elemento di Ω su un diverso elemento di $\wp(\Omega)$. Per esempio $f(x) = \{x\}$.
Dobbiamo ora far vedere che non esiste una $g : \Omega \to \wp(\Omega)$ suriettiva. Supponiamo che esista per assurdo. Consideriamo l'insieme $T = \{x \in \Omega \,|\, x \notin g(x)\}$. Sia x un elemento di Ω per cui $T = g(x)$, esiste per la suriettività di g. Allora $x \in T \longleftrightarrow x \in g(x) \longleftrightarrow x \notin g(x)$. Contraddizione. □

Il teorema si applica anche agli insiemi infiniti, definendo una **gerarchia**.

Funzioni

Definizione 8. Una funzione f è una corrispondenza tra due insiemi A e B che associa ad ogni elemento di A uno e un solo elemento di B.

Ci occuperemo in particolare di funzioni da $\mathbb{N}$ ad $\mathbb{N}$.

Esempio 9. $f = f(n) = n + 1$ i.e.
$f(0) = 1$, $f(1) = 2$, $f(2) = 3$, ...

Possiamo immaginarla come una tabella infinita:

n	0	1	2	3	...
$f(n)$	1	2	3	4	...

3 Algoritmi

3.1 L'idea

Che cos'è un algoritmo?

Dare una definizione precisa e rigorosa è difficile (ma si può, anche in diversi termini) e ci ritorniamo dopo.
Il termine è sulla bocca di tutti: solo un mese fa si parlava dell'algoritmo sbagliato per il calcolo delle tariffe degli abbonamenti del treno.

Esempio 10. Il metodo che utilizziamo solitamente per sommare due numeri interi è un algoritmo. Partendo dalle unità, sommiamo in colonna le cifre aggiungendo i riporti se necessario.
Possiamo dire informalmente che un **algoritmo** è una procedura che dato un **input** restituisce un **output**, utilizzando un **numero finito** di **regole**.
Considerando che possiamo **codificare** sotto forma di **numeri naturali** sia input sia output gli **algoritmi** sono un sottoinsieme delle **funzioni** da $\mathbb{N}$ ad $\mathbb{N}$.

3.2 Complessità

La complessità degli algoritmi

In genere si è interessati a trovari **algoritmi efficienti**, veloci per risolvere i problemi. La misura di efficienza di un algoritmo si chiama **complessità computazionale**. Normalmente viene espressa come una funzione della **dimensione** n dell'**input**. Per esempio $C = n$, $C = n^2$, $C = 2^n$. (Trascuro i dettagli della notazione O-grande).

Esempi

- Ricerca binaria. Se abbiamo una lista di n oggetti ordinata, ad esempio un dizionario, e vogliamo trovare uno specifico elemento un buon metodo è partire dal centro della lista. Se l'abbiamo trovato abbiamo finito altrimenti sappiamo se dobbiamo cercare nella prima metà o nella seconda. In ogni caso anche lì partiremo dalla metà iterando il procedimento. In questa maniera il massimo numero di tentativi sarà il massimo numero di volte per cui possiamo dividere la lista per due ovvero k tale che $\frac{n}{2^k} = 1$, cioè $k = \log_2 n$.
- La somma in colonna. Se consideriamo come dimensione dell'input n la lunghezza dei numeri da sommare, il tempo che impieghiamo per la somma è proporzionale ad n.
- Un algoritmo di ordinamento. Consideriamo una lista di n numeri. Volgiamo ordinarla in ordine crescente. Un modo semplice è quello di prendere il minimo e metterlo in testa, poi considerare i rimanenti, prendere il minimo e metterlo al secondo posto, e così via. Per cercare il minimo dobbiamo scandire tutta la lista facendo quindi n operazioni, poi per trovare il "secondo minimo" dovremo farne $n-1$ e così via. Il numero di operazioni è quindi $\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \simeq n^2$.
- Il problema del commesso viaggiatore. Consideriamo un insieme di città collegate da strade. Dobbiamo trovare il cammino di lunghezza minima che le colleghi tutte, che passi da ogni città una volta sola e che abbia la stessa città di partenza e arrivo. Vedremo che sostanzialmente non esiste un algoritmo che nel caso peggiore non ci costringa a provare tutti i possibili cammini che sono $n!$.

Classi di complessità P e NP

Definizione 11. La classe di problemi **P** è composta da tutti i problemi che sono risolvibili in tempo polinomiale (problemi trattabili).

Definizione 12. La classe di problemi **NP** è composta da tutti i problemi la cui soluzione è verificabile in tempo polinomiale (problemi intrattabili).

Teorema 13. $\mathbf{P} \subseteq \mathbf{NP}$.

Dimostrazione. Chiaramente se un problema è risolvibile in tempo polinomiale posso verificare una particolare soluzione in tempo polinomiale risolvendo il problema e confrontando la soluzione con quella che volevo verificare. □

Ci chiediamo se l'inclusione sia stretta, ovvero se ci siano dei problemi che stanno in **NP** ma non in **P**.

La risposta vale **1 milione** di dollari!

NP-completezza

Esistono alcuni problemi, chiamati **NP**-completi, che sono i più difficili della classe **NP**. Sono tutti della stessa difficoltà e le loro istanze possono essere trasformate in quelle di un altro problema **NP**-completo in tempo polinomiale.

Sono problemi di importante interesse pratico:

*Alcuni problemi **NP**-completi*

- Il problema del commesso viaggiatore
- Il problema dello zaino
- La soddisfacibilità delle formule della logica proposizionale (SAT)
- Il Sudoku ($n^2 \times n^2$)

Se trovassimo una soluzione polinomiale anche per uno solo dei seguenti problemi, vorrebbe dire che l'avremmo trovata per tutti e che $\mathbf{P} = \mathbf{NP}$.

3.3 Calcolabilità

Tutti i problemi sono risolvibili algoritmicamente?

L'**informatica** moderna è nata per rispondere a questa domanda.

Per prima cosa dobbiamo capire che cosa vuol dire algoritmicamente. **Church** e **Turing** hanno definito nel 1936 due **modelli astratti** di **algoritmo**, rispettivamente il **λ-calcolo** e la **macchina di Turing**. Noi ci accontentiamo di dire che un algoritmo è un **programma** scritto in un qualunque **linguaggio di programmazione** (C, Java, Python, etc) che termina.

Possiamo farlo sulla base di un importante risultato: tutti i formalismi di potenza massima conosciuti sono **equivalenti**. In altre parole:

Tesi di Church-Turing
Una funzione intuitivamente calcolabile è calcolabile da una macchina di Turing (o formalismo equivalente).

Quanti sono gli algoritmi?

Se pensiamo all'insieme di tutti gli **algoritmi** come l'insieme di tutti i **programmi** Il capiamo subito che il loro numero è **infinito**. Però capiamo anche che li possiamo **enumerare** cioè considerare in ordine i programmi π_1,π_2,π_3...
Possiamo cioè metterli in corrispondenza con i **numeri naturali**, costruendo una funzione $f : \mathbb{N} \to \Pi$ tale che $f(y) = \pi_y$.

L'enumerazione può essere fatta da un **algoritmo** e quindi siamo in grado di costruire un programma, chiamato **programma universale**, $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ che in

ingresso prenda il numero (di Gödel) y del programma π_y da eseguire, e l'input x del programma stesso calcolando così $g(y, x) = \pi_y(x)$.
Questo procedimento si chiama **aritmetizzazione** dei programmi o gödelizzazione.

Un problema indecidibile

Un classico problema **indecidibile** è il **problema dell'arresto**, ovvero trovare un algoritmo che decida, dati in ingresso il numero di Gödel y di un programma π_y e il suo input x se questo terminerà o meno, in formule:

$$h(y,x) = \begin{cases} 1 & \text{se } \pi_y(x) \text{ termina} \\ 0 & \text{altrimenti (cioè se } \pi_y(x) \text{ non termina)} \end{cases}$$

Teorema 14 (di Church)**.** *La funzione h non è algoritmicamente calcolabile.*

Dimostrazione. Sia per assurdo h calcolabile. Allora è calcolabile anche

$$k(x) = \begin{cases} 1 & \text{se } h(x,x) = 0 \text{ (cioè se } \pi_x(x) \text{ non termina)} \\ \text{non termina} & \text{altrimenti (cioè se } \pi_x(x) \text{ termina)} \end{cases}$$

quindi esiste un programma π_z che calcola k, cioè $\pi_z(x) = k(x)$.
Consideriamo allora $k(z)$, vediamo cioè cosa succede per un programma π_z che deve decidere la sua stessa terminazione.
Se $k(z) = 1$ (cioè termina), allora $h(z, z) = 0$, cioè $\pi_z(z) = k(z)$ non termina, quindi abbiamo una contraddizione.
Se $k(z)$ non termina allora $h(z, z) = 1$, cioè $\pi_z(z) = k(z)$ termina, altra contraddizione.
Quindi h non è calcolabile! □

Riferimenti bibliografici

[1] Scott Aaronson. *Quantum Computing Since Democritus.* Cambridge University Press, 2013.

[2] Dino Mandrioli and Paola Spoletini. *Informatica teorica.* CittàStudi, Torino, 2011.

[3] Giorgio Ausiello, Fabrizio D'Amore, Giorgio Gambosi, and Luigi Laura. *Linguaggi, modelli, complessità.* Franco Angeli, Milano, 2014.

[4] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Automi, linguaggi e calcolabilità.* Pearson, Milano, 2009.

[5] Thomas H. Cormen, Chrales E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduzione agli algoritmi e strutture dati.* McGraw-Hill Education, Milano, 2010.