Teoria della Calcolabilità

- Si occupa delle questioni fondamentali circa la **potenza** e le **limitazioni** dei sistemi di calcolo.
- L'origine risale alla prima metà del ventesimo secolo, quando i logici matematici iniziarono ad esplorare i concetti di
  - computazione
  - algoritmo
  - problema risolvibile per via algoritmica

  e dimostrarono l'esistenza di *problemi non risolvibili*, ossia problemi che non ammettono un algoritmo di risoluzione.

  $\Rightarrow$ *Problemi non decidibili*

Problemi computazionali

Problemi formulati matematicamente di cui cerchiamo una soluzione algoritmica.

Classificazione:

- problemi non decidibili
- problemi decidibili
  - problemi **trattabili** (costo polinomiale)
  - problemi **intrattabili** (costo esponenziale)

Teoria della Complessità Computazionale

- **Classificare i problemi decidibili**, dividendoli in classi definite sulla base della quantità di risorse a disposizione:
  - spazio di memoria
  - tempo di calcolo.

Calcolabilità e complessità

- **Calcolabilità:** nozioni di algoritmo e di problema non decidibile.
- **Complessità:** nozione di algoritmo efficiente e di problema intrattabile.
- La calcolabilità ha lo scopo di classificare i problemi in *risolvibili* e *non risolvibili*, mentre la complessità in *"facili"* e *"difficili"*.

ESISTENZA DI PROBLEMI INDECIDIBILI

Insiemi numerabili

- Due insiemi A e B hanno lo stesso numero di elementi

  $\Leftrightarrow$

  si può stabilire una **corrispondenza biunivoca** tra i loro elementi.
- Un insieme è **numerabile** (possiede una infinità numerabile di elementi)

  $\Leftrightarrow$

  i suoi elementi possono essere messi in **corrispondenza biunivoca con i numeri naturali**.

Insiemi numerabili

- Un insieme numerabile è un insieme i cui elementi possono essere enumerati, ossia descritti da una sequenza del tipo

$$a_1, a_2, \ldots, a_n, \ldots$$

Insiemi numerabili: esempi

- Insieme dei numeri naturali N
- Insieme dei numeri interi Z:
  - $n \leftrightarrow 2n + 1 \qquad n \geq 0$
  - $n \leftrightarrow 2|n| \qquad n < 0$
- Insieme dei numeri naturali pari:
  - $n \leftrightarrow 2n$
- Insieme dei numeri razionali Q
- Insieme delle stringhe su un alfabeto finito.

Insiemi equivalenti

- Due insiemi A e B sono **equivalenti**, o **hanno la stessa potenza**, se esiste una corrispondenza biunivoca tra gli elementi di A e gli elementi di B.
- Due insiemi finiti sono equivalenti ⇔ hanno lo stesso numero di elementi.
- Gli ***insiemi numerabili*** sono tutti e soli gli insiemi equivalenti a N.

Insiemi non numerabili

Sono tutti gli insiemi non equivalenti a N.

Esempi:

- insieme dei numeri reali compresi nell'intervallo chiuso [0,1]
- insieme dei numeri reali compresi nell'intervallo aperto (0,1)
- insieme dei numeri reali
- insieme di tutte le linee nel piano
- insieme delle funzioni in una o più variabili.

Problemi computazionali

L'insieme dei problemi computazionali **NON** è numerabile.

Problemi e funzioni

- Un problema computazionale può essere visto come una funzione matematica che associa ad ogni insieme di dati, espressi da k numeri interi, il corrispondente risultato, espresso da *j* numeri interi

$$f: N^k \rightarrow N^j$$

- L'insieme delle funzioni $f: N^k \rightarrow N^j$ *NON* è numerabile.

Diagonalizzazione

F = { funzioni f | $f: N \rightarrow \{0,1\}$ }

ogni $f \in F$ può essere rappresentata da una sequenza infinita:

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

o, se possibile, da una regola finita di costruzione:

$$f(x) = \begin{cases} 0 & x \ pari \\ 1 & x \ dispari \end{cases}$$

Diagonalizzazione

Teorema

L'insieme F non è numerabile.

Dim.

- Per assurdo, F sia numerabile.
- Possiamo enumerare ogni funzione:

  assegnare ad ogni $f \in F$ un numero progressivo nella numerazione, e costruire una tabella (infinita) di tutte le funzioni

Diagonalizzazione

x	0	1	2	3	4	5	6	7	8	...
$f_0(x)$	1	0	1	0	1	0	0	0	1	...
$f_1(x)$	0	0	1	1	0	0	1	1	0	...
$f_2(x)$	1	1	0	1	0	1	0	0	1	...
$f_3(x)$	0	1	1	0	1	0	1	1	1	...
$f_4(x)$	1	1	0	0	1	0	0	0	1	...
...				...						...

Diagonalizzazione

Consideriamo la funzione $g \in F$

$$g(x) = \begin{cases} 0 & f_x(x) = 1 \\ 1 & f_x(x) = 0 \end{cases}$$

g non corrisponde ad alcuna delle f_i della tabella poiché differisce da tutte nei valori posti sulla diagonale principale.

Diagonalizzazione

x	0	1	2	3	4	5	6	7	8	...
$f_0(x)$	**1**	0	1	0	1	0	0	0	1	...
$f_1(x)$	0	**0**	1	1	0	0	1	1	0	...
$f_2(x)$	1	1	**0**	1	0	1	0	0	1	...
$f_3(x)$	0	1	1	**0**	1	0	1	1	1	...
$f_4(x)$	1	1	0	0	**1**	0	0	0	1	...
...										
g(x)	**0**	**1**	**1**	**1**	**0**	.	.	.		

Diagonalizzazione

- Per assurdo: $\exists\ j$ t.c. $g(x) = f_j(x)$
- allora $g(j) = f_j(j)$, ma

$$g(j) = f_j(j) = \begin{cases} 0 & f_j(j) = 1 \\ 1 & f_j(j) = 0 \end{cases}$$

- cioè $g(j) \neq f_j(j)$.

$\Rightarrow$ contraddizione!!!

Conclusione

- ***F* non è numerabile,** e a maggior ragione, non sono numerabili gli insiemi delle funzioni:

  $f: N \rightarrow N$

  $f: N \rightarrow R$

  $f: R \rightarrow R$

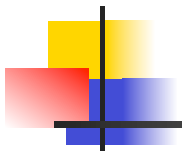
  $f: N^k \rightarrow N^j$

  L'insieme dei problemi computazionali non è numerabile.

Il problema della rappresentazione

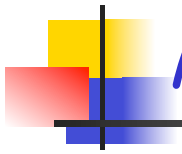
L'informatica rappresenta tutte le sue entità (quindi anche gli algoritmi) in forma digitale, come ***sequenze finite di simboli di alfabeti finiti*** (e.g., {0,1});

descrive dunque un ***mondo numerabile***.



Il concetto di algoritmo

- Il concetto di **algoritmo** è l'elemento centrale della teoria della calcolabilità.
- Un **algoritmo** è un procedimento di calcolo che consente di pervenire alla soluzione di un problema, numerico o simbolico, mediante una *sequenza finita di operazioni, completamente e univocamente determinate*.



Algoritmi, funzioni e problemi

Consideriamo algoritmi che ricevono in ingresso k valori interi, e restituiscono una soluzione data da j numeri interi, con $k,j \geq 1$.

Un algoritmo può essere visto come una *procedura per il calcolo della funzione (risoluzione del problema)*

$$f: N^k \rightarrow N^j$$

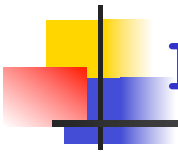
che associa ad ogni insieme di dati il risultato ad essi corrispondente.

Algoritmi

Possiamo descrivere gli algoritmi come programmi per un calcolatore (RAM). Questa rappresentazione è costituita da sequenze finite di simboli:

Gli algoritmi sono un'infinità numerabile!

Le funzioni matematiche (e quindi i problemi computazionali) non sono numerabili.

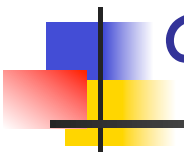


Il problema della rappresentazione

$$|\{\text{Problemi}\}| > > |\{\text{Algoritmi}\}|$$

$\Rightarrow$

Esistono funzioni (problemi) per cui non esiste un algoritmo di calcolo!



MODELLI DI CALCOLO E CALCOLABILITÀ

Modelli di calcolo

La teoria della calcolabilità dipende dal modello di calcolo?

oppure ...

la decidibilità è una proprietà del problema?

Modelli di calcolo

I linguaggi di programmazione esistenti sono tutti equivalenti?

Ce ne sono alcuni più potenti e/o più semplici di altri?

Ci sono algoritmi descrivibili in un linguaggio, ma non in un altro?

È possibile che problemi oggi irrisolvibili possano essere risolti in futuro con altri linguaggi o con altri calcolatori?

La teorie della calcolabilità e della complessità dipendono dal modello di calcolo?

La tesi di Church-Turing

Tutti i (ragionevoli) calcolatori possono risolvere gli stessi problemi, e possono simularsi a vicenda.

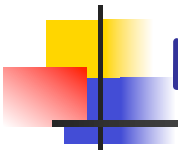
Tutti i diversi linguaggi di programmazione e i modelli di calcolo definiti si sono dimostrati equivalenti tra loro:

risolvono esattamente la stessa classe di problemi, ovvero calcolano la stessa classe di funzioni.

La tesi di Church-Turing

Incrementi qualitativi alla struttura di una macchina, o alle istruzioni di un linguaggio di programmazione, servono ***solo*** a:

- abbassare il tempo di esecuzione
- rendere più agevole la programmazione.



La tesi di Church-Turing

La tesi di Church-Turing non è dimostrabile.

Può essere solo accettata o rifiutata:

- *non si può escludere a priori la possibilità che un giorno venga individuato uno strumento in grado di calcolare una funzione ritenuta al momento non calcolabile.*
- *Vi sono ragioni convincenti per credere che ciò non potrà accadere.*

Il problema dell'arresto

Abbiamo dimostrato l'esistenza di funzioni/ problemi non calcolabili.

I problemi che si presentano spontaneamente sono tutti calcolabili.

Non è stato facile individuare un problema che non lo fosse.

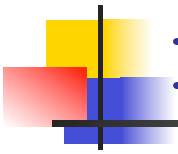
Turing (1930): **Problema dell'arresto**.

Il problema dell'arresto

- Considera *algoritmi che indagano sulle proprietà di altri algoritmi*, che sono trattati come dati.
- È legittimo: gli algoritmi sono rappresentabili con sequenze di simboli, che possono essere presi dallo stesso alfabeto usato per codificare i dati di input.
- Una stessa sequenza di simboli può essere quindi interpretata sia come un programma, sia come un dato di ingresso di un altro programma.

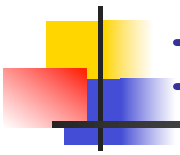
Il problema dell'arresto

- Un algoritmo A, comunque formulato, può operare sulla rappresentazione di un altro algoritmo B.
- Possiamo calcolare A(B).
- In particolare può avere senso calcolare A(A).



Il problema dell'arresto

*Presi ad arbitrio un algoritmo A e i suoi dati di input D, decidere **in tempo finito** se la computazione di A su D termina o no.*



Il problema dell'arresto

Consiste nel chiedersi se un generico programma termina la sua esecuzione,

oppure "va in ciclo", ovvero continua a ripetere la stessa sequenza di istruzioni all'infinito (supponendo di non avere limiti di tempo e memoria).

ESEMPIO: Stabilire se un intero p > 1 è primo.

```
Primo(p)
fattore = 2;
while (p % fattore != 0)
      fattore++;
return (fattore == p);
```

Termina sicuramente (la guardia del **while** diventa falsa quando fattore = p).

ESEMPIO

- Programma che trova il più piccolo numero intero pari (maggiore di 4) che NON sia la somma di due numeri primi.
- Il programma si arresta quando trova $n \geq 4$ che NON è la somma di due primi.

ESEMPIO

```
Goldbach()
n = 2;
do {
      n = n + 2;
      controesempio = true;
      for (p = 2; p ≤ n -2; p++) {
            q = n – p;
            if (Primo(p) && Primo(q))
                  controesempio = false;
      }
} while (!controesempio);
return n;
```

Congettura di Goldbach.

XVIII secolo

"ogni numero intero pari n ≥ 4 è la somma di due numeri primi"

Congettura **falsa** ➔ `Goldbach()` si arresta

Congettura **vera** ➔ `Goldbach()` **NON** si arresta

TEOREMA

Turing ha dimostrato che riuscire a dimostrare se un programma arbitrario si arresta e termina la sua esecuzione non è solo un'impresa ardua, ma in generale è IMPOSSIBILE!

TEOREMA

Il problema dell'arresto è INDECIDIBILE.

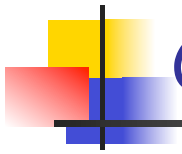
DIMOSTRAZIONE

Se il problema dell'arresto fosse decidibile, allora esisterebbe un algoritmo ARRESTO che:

- presi A e D come dati di input
- determina in **tempo finito** le risposte:

ARRESTO(A,D) = 1 se A(D) termina

ARRESTO(A,D) = 0 se A(D) non termina



Osservazione

L'algoritmo ARRESTO non può consistere in un algoritmo che simuli la computazione A(D):

se A non si arresta su D, ARRESTO non sarebbe in grado di rispondere NO (0) in tempo finito.

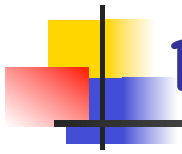
DIMOSTRAZIONE

In particolare possiamo scegliere D = A, cioè considerare la computazione A(A):

ARRESTO(A,A) = 1

⇔

A(A) termina



DIMOSTRAZIONE

Se esistesse l'algoritmo ARRESTO, esisterebbe anche il seguente algoritmo:

```
PARADOSSO(A)
    while (ARRESTO(A,A)) {
                ;
    }
```

DIMOSTRAZIONE

L'ispezione dell'algoritmo PARADOSSO mostra che:

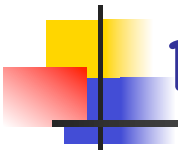
PARADOSSO(A) termina

$\Longleftrightarrow$

x = ARRESTO(A,A) = 0

$\Longleftrightarrow$

A(A) non termina



DIMOSTRAZIONE

Cosa succede calcolando PARADOSSO(PARADOSSO)?

PARADOSSO(PARADOSSO) **termina**

$\Longleftrightarrow$

x = ARRESTO(PARADOSSO, PARADOSSO) = 0

$\Longleftrightarrow$

PARADOSSO(PARADOSSO) **non termina**

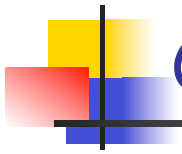
contraddizione!

DIMOSTRAZIONE

L'unico modo di risolvere la contraddizione è che l'algoritmo PARADOSSO non possa esistere.

Dunque non può esistere nemmeno l'algoritmo ARRESTO.

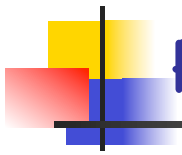
In conclusione, ***il problema dell'arresto è indecidibile!***



Osservazione

Come già osservato, l'algoritmo ARRESTO costituirebbe uno strumento estremamente potente:

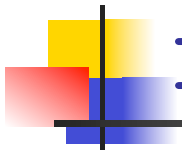
permetterebbe infatti di dimostrare congetture ancora aperte sugli interi (esempio: la congettura di Goldbach).



Problemi indecidibili

- Altri problemi lo sono:
  - Ad esempio, è indecidibile stabilire l'equivalenza tra due programmi (se per ogni possibile input, producono lo stesso output)
- "Lezione di Turing":

  non esistono algoritmi che decidono il comportamento di altri algoritmi esaminandoli dall'esterno, cioè senza passare dalla loro simulazione.



Il decimo problema di Hilbert

- *Esistono risultati di non calcolabilità relativi ad altre aree della matematica, tra cui la teoria dei numeri e l'algebra.*
- *Tra questi, occupa un posto di rilievo il ben noto decimo problema di Hilbert.*

Equazioni diofantee

Un'**equazione diofantea** è un'equazione della forma

$$p(x_1, x_2, \ldots, x_m) = 0$$

dove p è un polinomio a coefficienti interi.

Il decimo problema di Hilbert

Data un'arbitraria equazione diofantea, di grado arbitrario e con un numero arbitrario di incognite

$$p(x_1, x_2, \ldots, x_m) = 0$$

stabilire se p ammette soluzioni intere.

Teorema

Il decimo problema di Hilbert non è calcolabile.

Il decimo problema di Hilbert

La questione circa la calcolabilità di questo problema è rimasta aperta per moltissimi anni,

ha attratto l'attenzione di illustri matematici,

ed è stata risolta nel 1970 da un matematico russo allora poco più che ventenne, Yuri Matiyasevich.