



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 che non ammettono un algoritmo di risoluzione**
 - ⇒ **Problemi non decidibili**

1



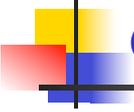
Problemi computazionali

Problemi formulati matematicamente di cui cerchiamo una soluzione algoritmica

Classificazione

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

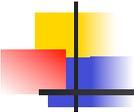
2



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*"

3



4

ESISTENZA DI PROBLEMI INDECIDIBILI

5

Insiemi numerabili

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



si può stabilire una **corrispondenza biunivoca** tra i loro elementi

- Un insieme è **numerabile** (possiede una infinità numerabile di elementi)



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

6

Insiemi numerabili

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

$$a_1, a_2, \dots, a_n, \dots$$

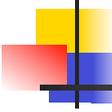
7

Insiemi numerabili: esempi

- Insieme dei numeri naturali \mathbb{N}
- Insieme dei numeri interi \mathbb{Z}
 - $n \leftrightarrow 2n + 1 \quad n \geq 0$
 - $n \leftrightarrow 2|n| \quad n < 0$

0, -1, 1, -2, 2, -3, 3, -4, 4, ...
- Insieme dei numeri naturali pari
 - $2n \leftrightarrow n \quad 0, 2, 4, 6, 8, \dots$
- Insieme delle stringhe finite di simboli presi da un alfabeto finito

8



Insiemi numerabili: esempi

- Insieme delle stringhe finite di simboli di un alfabeto finito

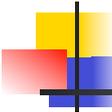
$a, b, \dots, z,$

$aa, ab, \dots, az, ba, bb, \dots, bz, \dots, za, \dots, zz,$

$aaa, \dots, aaz, aba, \dots, abz, \dots, azz, baa, \dots, zzz,$

$aaaa, \dots$

9



Insiemi non numerabili

Sono tutti gli insiemi non equivalenti a \mathbb{N}

Esempi:

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

10



Problemi computazionali

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

11



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

12

Diagonalizzazione

$F = \{ \text{funzioni } f \mid f: \mathbb{N} \rightarrow \{0,1\} \}$

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

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

13

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

14

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	...
...			

15

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:
differisce da tutte nei valori posti sulla diagonale principale

16

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	.	.	.		

17

Diagonalizzazione

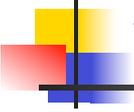
- Per assurdo: $\exists j$ t.c. $g(x) = f_j(x)$
- allora $g(j) = f_j(j)$, ma per definizione

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

- cioè $g(j) \neq f_j(j)$

\Rightarrow contraddizione!!!

18



Diagonalizzazione

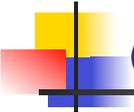
Per qualunque numerazione scelta, esiste sempre almeno una funzione esclusa

\Rightarrow *F non è numerabile*

- Si possono considerare linee arbitrarie che attraversano la tabella toccando tutte le righe e tutte le colonne esattamente una volta,
- e definire funzioni che assumono in ogni punto un valore opposto a quello incontrato sulla linea

Esistono infinite funzioni di F escluse da qualsiasi numerazione

19



Conclusione

- $F = \{ f: \mathbb{N} \rightarrow \{0,1\} \}$ non è numerabile
- A maggior ragione, non sono numerabili gli insiemi delle funzioni
 - $f: \mathbb{N} \rightarrow \mathbb{N}$
 - $f: \mathbb{N} \rightarrow \mathbb{R}$
 - $f: \mathbb{R} \rightarrow \mathbb{R}$
 - $f: \mathbb{N}^k \rightarrow \mathbb{N}^j$

\Rightarrow

L'insieme dei problemi computazionali non è numerabile

20



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})

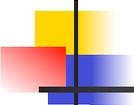
21



Il concetto di algoritmo

Algoritmo
sequenza finita di operazioni, completamente e univocamente determinate

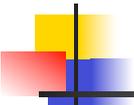
22



Algoritmi

La formulazione di un algoritmo dipende dal modello di calcolo utilizzato

- “programma” per un modello matematico astratto, come una Macchina di Turing
- algoritmo per in pseudocodice per RAM
- programma in linguaggio C per un PC

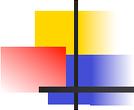


Algoritmi

Qualsiasi modello si scelga, gli algoritmi devono esservi descritti, ossia rappresentati da sequenze finite di caratteri di un alfabeto finito

⇒ gli algoritmi sono possibilmente infiniti, ma numerabili

possono essere ‘elencati’ (messi in corrispondenza biunivoca con l’insieme dei numeri naturali)



Problemi computazionali

I problemi computazionali
 (funzioni matematiche che associano ad
 ogni insieme di dati il corrispondente
 risultato)

non sono numerabili

25



Il problema della rappresentazione

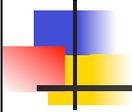
Drastica perdita di potenza
gli algoritmi sono numerabili
e sono meno dei problemi computazionali, che hanno
la potenza del continuo

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

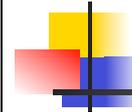
⇒

**Esistono problemi privi di un
 corrispondente algoritmo di calcolo**

UN PROBLEMA INDECIDIBILE: il problema dell'arresto



27



Il problema dell'arresto

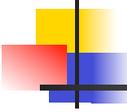
Esistono dunque problemi non calcolabili

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**

28

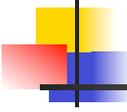


Il problema dell'arresto

È un problema posto in forma decisionale:

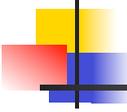
Arresto: $\{Istanze\} \rightarrow \{0,1\}$

*Per i problemi decisionali, la calcolabilità è in genere chiamata **decidibilità***



Il problema dell'arresto

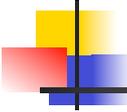
*Presi ad arbitrio un **algoritmo A** e i suoi **dati di input D**, decidere in tempo finito se la computazione di **A** su **D** termina o no*



Il problema dell'arresto

- Algoritmo che indaga sulle proprietà di un altro algoritmo, trattato come dato di input
 - È legittimo: possiamo usare lo stesso alfabeto per codificare algoritmi e i loro dati di ingresso (sequenze di simboli dell'alfabeto)
 - Una stessa sequenza di simboli può essere quindi interpretata sia come un programma, sia come un dato di ingresso di un altro programma

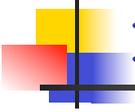
31



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)**

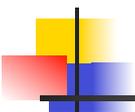
32



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)

33



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 $\text{fattore} = p$)

34

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;

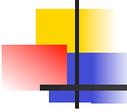
```

35

Algoritmo GOLDBACH

- Scandisci in ordine crescente i numeri naturali pari maggiori di 2, fino a trovare un numero che **NON** sia esprimibile come la **somma di due numeri primi**
- Se e quando questo accade, stampa il numero e termina

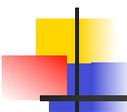
36



Algoritmo GOLDBACH

- Trova il più piccolo numero intero pari (maggiore o uguale a 4) che **NON** sia la somma di due numeri primi
- Si **arresta** quando trova $n \geq 4$ che **NON** è la somma di due primi

37



Congettura di Goldbach.

XVIII secolo

“ogni numero intero pari $n \geq 4$ è la somma di due numeri primi”

Congettura **falsa** → Goldbach() si **arresta**

Congettura **vera** → Goldbach() **NON** si **arresta**

38



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

39



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

40



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

41



DIMOSTRAZIONE

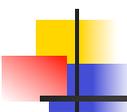
In particolare possiamo scegliere $D = A$,
cioè considerare la computazione $A(A)$

$ARRESTO(A,A) = 1$

\longleftrightarrow

$A(A)$ termina

42



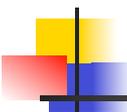
DIMOSTRAZIONE

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

PARADOSSO(A)

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

43



DIMOSTRAZIONE

L'ispezione dell'algoritmo PARADOSSO
mostra che

PARADOSSO(A) termina

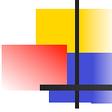


x = ARRESTO(A,A) = 0



A(A) non termina

44



DIMOSTRAZIONE

Cosa succede calcolando PARADOSSO(PARADOSSO)?

PARADOSSO(PARADOSSO) termina

↔

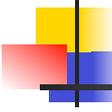
$x = \text{ARRESTO}(\text{PARADOSSO}, \text{PARADOSSO}) = 0$

↔

PARADOSSO(PARADOSSO) non termina

contraddizione!

45



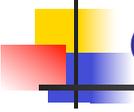
DIMOSTRAZIONE

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

Dunque non può esistere nemmeno l'algoritmo ARRESTO

Il problema dell'arresto è indecidibile

46

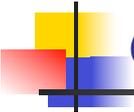


Osservazione

Non può esistere un algoritmo che decida in tempo finito se un algoritmo arbitrario A termina la sua computazione su dati arbitrari D

ciò non significa che non si possa decidere in tempo finito la terminazione di algoritmi particolari

il problema è indecidibile su una coppia $\langle A, D \rangle$ scelta arbitrariamente

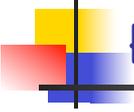


Osservazione

L' algoritmo ARRESTO costituirebbe uno strumento estremamente potente

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

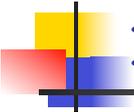
48



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*

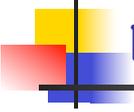
49



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**

50



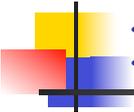
Equazioni diofantee

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

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

dove p è un polinomio a coefficienti interi

51



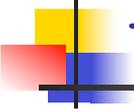
Il decimo problema di Hilbert

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

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

stabilire se p ammette soluzioni intere

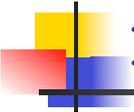
52



Teorema

*Il decimo problema di Hilbert è
algoritmicamente INDECIDIBILE*

53



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

54

MODELLI DI CALCOLO E CALCOLABILITÀ



55

Modelli di calcolo



La teoria della calcolabilità dipende dal
modello di calcolo?

oppure ...

la decidibilità è una proprietà del
problema?

56



Modelli di calcolo

I linguaggi di programmazione esistenti sono tutti equivalenti?

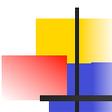
Ce ne sono alcuni più potenti e/o più semplici di altri?

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?

57



La tesi di Church-Turing

Tutti i (ragionevoli) modelli di calcolo

- *risolvono esattamente la stessa classe di problemi*
- dunque si equivalgono nella possibilità di risolvere problemi, pur operando con diversa efficienza

58



La TESI di Church-Turing

- La decidibilità è una proprietà del problema
- 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

59