



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**.
⇒ 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 resolvibili*, mentre la complessità in "*facili*" e "*difficili*".

3

ESISTENZA DI PROBLEMI INDECIDIBILI



4



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}).

5



Il concetto di algoritmo

- 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.

6



Algoritmi

- *La formulazione di un algoritmo dipende dal modello di calcolo utilizzato:*
 - *“programma” per un modello matematico astratto, come una Macchina di Turing*
 - *programma in linguaggio C per un PC*
 - *etc.*



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.



Problemi computazionali

I problemi computazionali

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

non sono numerabili.

9



Il problema della rappresentazione

Drastica perdita di potenza:

le rappresentazioni che possiamo costruire con alfabeti finiti sono numerabili,

e sono meno dei problemi computazionali, che hanno la potenza del continuo.



Il problema della rappresentazione

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



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

11



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.**

12



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.

13



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)**.

14



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.*

15



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).

16



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`).

17



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.

18

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;
```

19

Conggettura di Goldbach.

XVIII secolo

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

Conggettura falsa → Goldbach() si arresta

Conggettura vera → Goldbach() NON si arresta

20



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.

21



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

22



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.

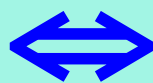
23



DIMOSTRAZIONE

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

$ARRESTO(A,A) = 1$



$A(A)$ termina

24

DIMOSTRAZIONE

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

PARADOSSO (A)

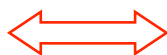
```
while (ARRESTO(A,A)) {  
    ;  
}
```

25

DIMOSTRAZIONE

L' ispezione dell' algoritmo PARADOSSO
mostra che:

PARADOSSO(A) termina



x = ARRESTO(A,A) = 0



A(A) non termina

26

DIMOSTRAZIONE

Cosa succede calcolando PARADOSSO(PARADOSSO)?

PARADOSSO(PARADOSSO) termina



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



PARADOSSO(PARADOSSO) non termina

contraddizione!

27

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!*

28



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).

29



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.

30



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**.*

31



Equazioni diofantee

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

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

dove p è un polinomio a coefficienti interi.

32



Il decimo problema di Hilbert

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

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

stabilire se p ammette soluzioni intere.

33



Teorema

Il decimo problema di Hilbert non è calcolabile.

34



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.

35



MODELLI DI CALCOLO E CALCOLABILITÀ

36



Modelli di calcolo

La teoria della calcolabilità dipende dal modello di calcolo?

oppure ...

la decidibilità è una proprietà del problema?

37



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 teoria della calcolabilità dipende dal modello di calcolo?

38



Modelli di calcolo

Esistono diversi modelli di calcolo, e dunque diverse caratterizzazioni formali del concetto di algoritmo e di problema decidibile

Tutte le caratterizzazioni sono risultate equivalenti: un problema decidibile in un certo modello di calcolo è decidibile anche negli altri, purché opportunamente codificato in accordo ai diversi formalismi

39



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

40



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.

41



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*

42