

# L' INFORMATICA: DUE CONCETTI DI BASE

Fabrizio Luccio, Pisa 2020

## La decidibilità

l'algoritmo

non tutti i problemi sono risolubili

## La complessità

la crescita esponenziale

problemi trattabili e intrattabili

# 1. La decidibilità

legata al concetto di numerabilità che nasce alla fine del 1800 nella teoria degli insiemi infiniti.

Consideriamo le sequenze finite di caratteri di un alfabeto finito, e.g. {0,1}

0	1										
1	2										
00	01	10	11								
3	4	5	6								
000	001	010	011	.	.	.	111				
7	8	9	10	.	.	.	14				
0000	0001	.	.	.	.	.	.	.	.	.	.
15	16	.	.	.	.	.	.	.	.	.	.

Le sequenze finite sono numerabili

	0	1	2	3	4	5	6	.	.
F0	1	1	0	1	0	1	1	.	.
F1	0	1	0	0	0	1	0	.	.
F2	0	0	0	1	1	0	1	.	.
F3	1	0	1	1	1	0	1	.	.
..	.	.	.	.	.	.	.	.	.

Tabella delle funzioni  $F_i: \mathbb{N} \rightarrow \{0,1\}$

	0	1	2	3	4	5	6	.	.
F0	1	1	0	1	0	1	1	.	.
F1	0	1	0	0	0	1	0	.	.
F2	0	0	0	1	1	0	1	.	.
F3	1	0	1	1	1	0	1	.	.
..	.	.	.	.	.	.	.	.	.

$$F3(4) = 1$$

	0	1	2	3	4	5	6	.	.
F0	1	1	0	1	0	1	1	.	.
F1	0	1	0	0	0	1	0	.	.
F2	0	0	0	1	1	0	1	.	.
F3	1	0	1	1	1	0	1	.	.
..	.	.	.	.	.	.	.	.	.

I valori  $F_i(i)$  appaiono sulla diagonale principale della tabella

	0	1	2	3	4	5	6	.	.	
F0	1	1	0	1	0	1	1	.	.	
F1	0	1	0	0	0	1	0	.	.	
F2	0	0	0	1	1	0	1	.	.	
F3	1	0	1	1	1	0	1	.	.	
Fj	0	0	1	0	.	.	.	.	.	?

La funzione  $F_j$ :  $F_j(i)=0$  se e solo se  $F_i(i)=1$

$F_j$  non appartiene all'elenco

Le funzioni non sono numerabili

La numerabilità degli algoritmi (sequenze finite) e la non numerabilità delle funzioni (sequenze infinite) implica l'esistenza di funzioni non calcolabili

ma richiede di porre una **definizione formale** al concetto di **algoritmo**

Nel 1936 Alan Turing definisce l'algoritmo attraverso una famosa **macchina**



La Macchina di Turing:  $MT = \{S, A, M, T\}$

$S = \{\alpha, \beta, \gamma, \dots, \omega\}$

$\alpha$  stato iniziale

$\omega$  stato finale

$A = \{a, b, c, \dots, z\}$

$z$  blank

$M = \{<, -, >\}$

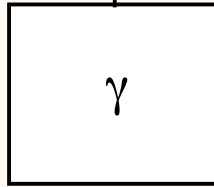
$<$  sinistra

$-$  fermo

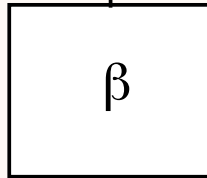
$>$  destra

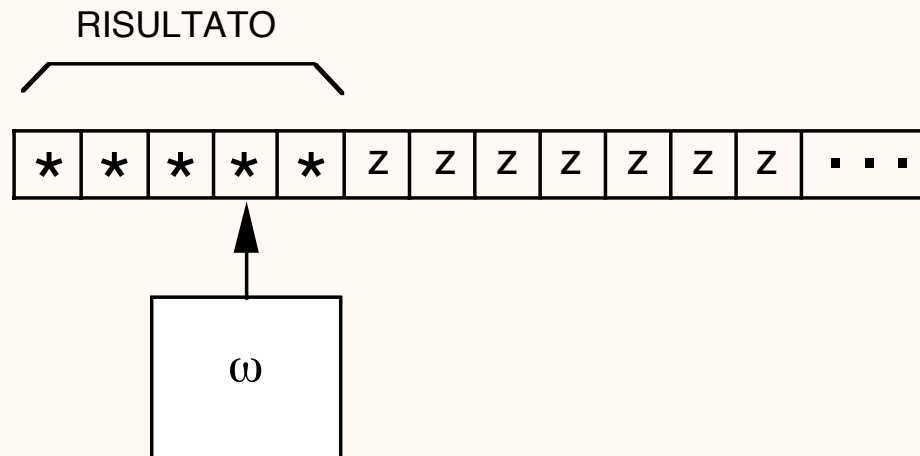
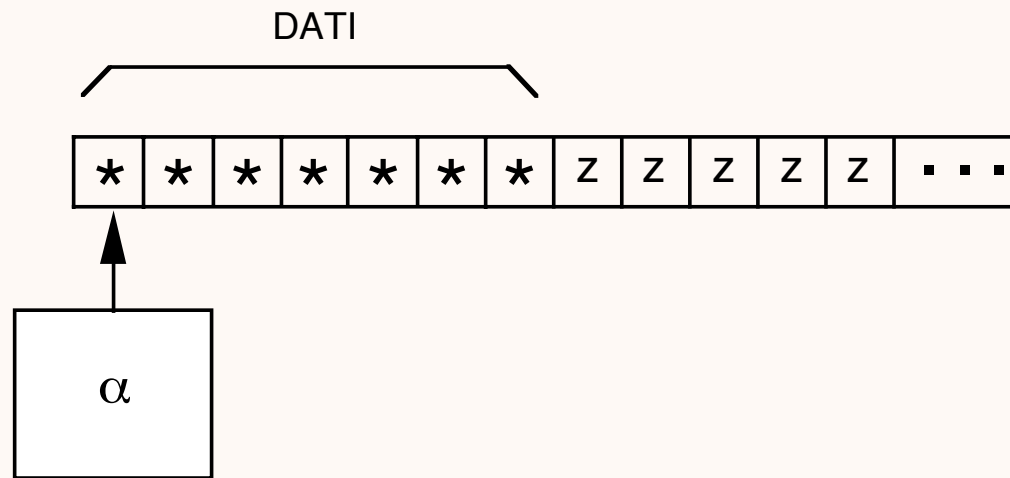
$T: S \times A \rightarrow S \times A \times M$

per esempio:  $T(\gamma, r) \rightarrow (\beta, m, <)$



$$T(\gamma, r) = (\beta, m, <)$$

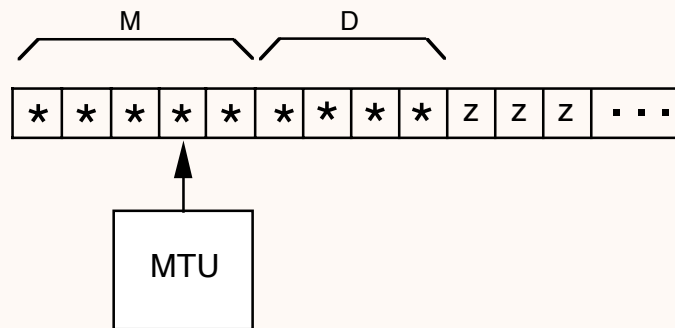




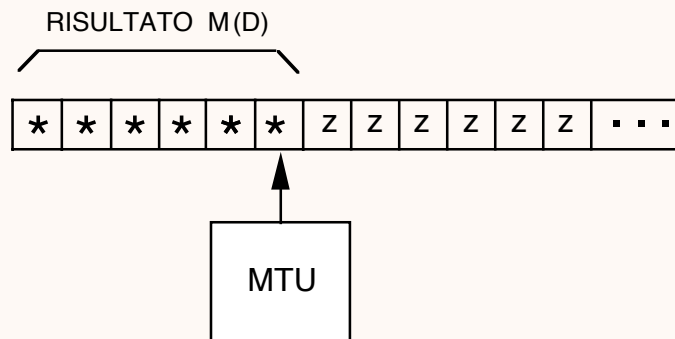
Una Macchina di Turing  $MT$  che termina il suo calcolo in tempo finito per qualsiasi dato d'ingresso è un algoritmo. Algoritmi diversi corrispondono a  $MT$  diverse.

Poiché una  $MT$  si descrive con una sequenza finita di simboli, essa ha la stessa natura della sequenza  $D$  dei suoi dati. Dunque si può fornire la  $M$  come sequenza di dati per a un'altra  $MT$ .

Tutto questo ha permesso a Turing di definire la **Macchina Universale MTU** che accetta come dati  $M$  e  $D$  e simula il calcolo di  $M$  su  $D$



MTU simula il  
calcolo di M su D



Dunque la Macchina Universale  
è un calcolatore

Cerchiamo ora una **funzione non calcolabile**, cioè una funzione che non ammette algoritmo di calcolo . . . .

. . . . o equivalentemente un problema per cui non esiste algoritmo di soluzione.

Il primo individuato da Turing fu il **Problema della terminazione**: Per una MT arbitraria  $M$  e i suoi dati arbitrari  $D$  decidere (in tempo finito) se il calcolo di  $M$  su  $D$  termina in tempo finito.

In termini programmatici vogliamo dimostrare che **non esiste un algoritmo HALT** per decidere (in tempo finito) se un altro algoritmo arbitrario  $A$ , operando su dati arbitrari  $D$ , termina o no:

$\text{HALT}(A, D) = \text{true}$ , se  $A(D)$  termina

$\text{HALT}(A, D) = \text{false}$ , se  $A(D)$  non termina

La dimostrazione è basata sull' **antinomia del mentitore**, combinazione di autoriferimento e negazione



Definiamo un nuovo algoritmo  $P$  che lavora su una sequenza  $A$  che rappresenta un algoritmo:

$P(A)$

```
while (HALT(A,A) == true) A = A;  
print "A(A) non termina";
```

$P(A)$  termina se e solo se  $A(A)$  non termina

$P(P)$  termina se e solo se  $P(P)$  non termina !!!

È un'antinomia: il punto debole è  
l'ammissione che HALT esista

Ma vediamo una conseguenza forse inaspettata del problema della terminazione

La **congettura di Goldbach** afferma che ogni numero pari  $n > 2$  si può esprimere come somma di **due numeri primi**

$$8 = 4 + 4 \text{ (non primi), ma anche } 8 = 3 + 5 \text{ (primi)}$$

La congettura non è stata mai dimostrata, ma è stata verificata fino a valori altissimi di  $n$ .

## Consideriamo allora l'algoritmo:

GOLD

```
per ogni  $n$  pari a partire da 4
  costruisci tutte le coppie  $a, b$  con  $a+b=n$ ;
  if esiste una coppia  $a$  e  $b$  di primi
    procedi con il prossimo valore di  $n$ 
  else stampa  $n$  e arresta il calcolo.
```

Se la congettura fosse falsa, GOLD si arresterebbe sul primo valore di  $n$  che non la verifica, altrimenti continuerebbe a girare all'infinito. Quindi GOLD termina se e solo se la congettura è falsa per un certo valore di  $n$ .

**Se esistesse l'algoritmo HALT:**

HALT(GOLD) = true (cioè GOLD si arresta)

⇒ la congettura è falsa

HALT(Gold) = false (cioè GOLD non si arresta)

⇒ la congettura è vera

Se esistesse (e io conoscessi) l'algoritmo HALT,  
avrei un modo banale di dimostrare quasi gratis  
moltissimi teoremi e congetture sui numeri interi !

Successivamente è stato stabilito che molti problemi sono indecidibili per trasformazione con il problema della terminazione.

## 2. La complessità

Una legge di base è la crescita esponenziale

Impiegando  $k$  simboli il numero  $p$  di "parole" di lunghezza  $n$  cresce esponenzialmente con  $n$ :

$$p = k^n$$

Alfabeto inglese:  $k = 26$ ,  $n = 3$ ,  $p = 26^3 = 17.576$

Parole: aaa aab aac . . . . zzz

Cifre decimali:  $k = 10$ ,  $n = 3$ ,  $p = 10^3 = 1000$

Numeri: 000 001 002 . . . . 999



La complessità (di calcolo)

è il "tempo" (cioè il numero di operazioni elementari) necessario per risolvere un problema.

Sono **intrattabili** i problemi che richiedono tempo esponenziale nella dimensione dell'input

## Complessità polinomiale e esponenziale

Un algoritmo di complessità polinomiale risolve un certo problema su  $n$  dati in tempo  $t = cn^s$  su un computer A.

Lo stesso algoritmo, su un computer  $k$  volte "più veloce" di A, risolve in tempo  $t$  lo stesso problema su  $m$  dati:

$$cn^s = t \quad cm^s = kt \quad \text{dunque} \quad m = k^{1/s} n$$

Ripetiamo il ragionamento con un algoritmo di complessità  $c2^n$ :

$$c2^n = t \quad c2^m = kt \quad \text{dunque} \quad m = n + \log_2 k$$

# Le due principali classi di complessità

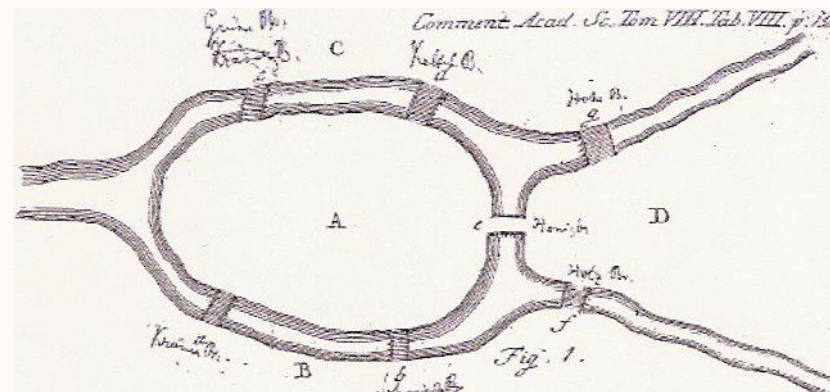
**P** è la classe dei problemi che si **risolvono** in tempo polinomiale.

**NP** è la classe dei problemi che si sanno risolvere solo in tempo esponenziale ma si **verificano** in tempo polinomiale.

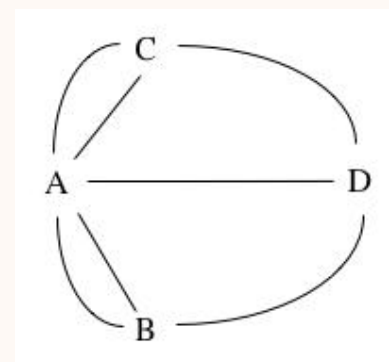
**P = NP ?**

Per la risoluzione di questo problema è in palio un premio di 1M\$

La città di  
Königsberg, 1735



e il suo grafo



Un percorso che traversa tutti gli archi esattamente una volta e torna nel nodo di partenza si chiama **ciclo Euleriano**.

Condizione di Eulero: **esiste** un ciclo Euleriano **se e solo se** tutti i nodi hanno grado pari. La condizione si verifica in **tempo polinomiale** nelle dimensioni del grafo.

Un percorso che traversa tutti i nodi esattamente una volta e torna nel nodo di partenza si chiama **ciclo Hamiltoniano**.

Per quanto è noto, la verifica di esistenza di un ciclo Hamiltoniano richiede **tempo esponenziale** nelle dimensioni del grafo.

## Tre problemi tra i "più difficili" in NP

**Biologia molecolare:** dato un insieme arbitrario di frammenti di sequenze di DNA stabilire se esiste una sequenza incognita ma di lunghezza assegnata che contenga al suo interno ognuno dei frammenti dati.

**Trasporti:** stabilire se un insieme di casse di pesi diversi possono essere caricate su un assegnato numero di camion senza superare il peso massimo sopportabile da ciascun camion.

**Algebra:** stabilire se un'equazione algebrica di secondo grado in due variabili  $x, y$ , a coefficienti interi, ammette una soluzione in cui  $x$  e  $y$  hanno valori interi (se l'equazione è di primo grado il problema è in P).

La distinzione tra problemi polinomiali e esponenziali è alla base della **crittografia** che utilizza funzioni "one way", cioè "facili da calcolare" e "difficili" da invertire :

Vediamo come due partner possano costruire una loro chiave segreta con un metodo noto a tutti, anche se tutti ne intercettano la comunicazione



Poniamo che la **moltiplicazione** sia facile e la **divisione** sia difficile

Utilizziamo un intero pubblico: per esempio **5**

Fabrizio

sceglie un intero  $f$

calcola  $F = 5 f = 35$

e invia  $F$  a Partner

calcola  $P f = 5 p f = 140$

Partner

sceglie un intero  $p$

calcola  $P = 5 p = 20$

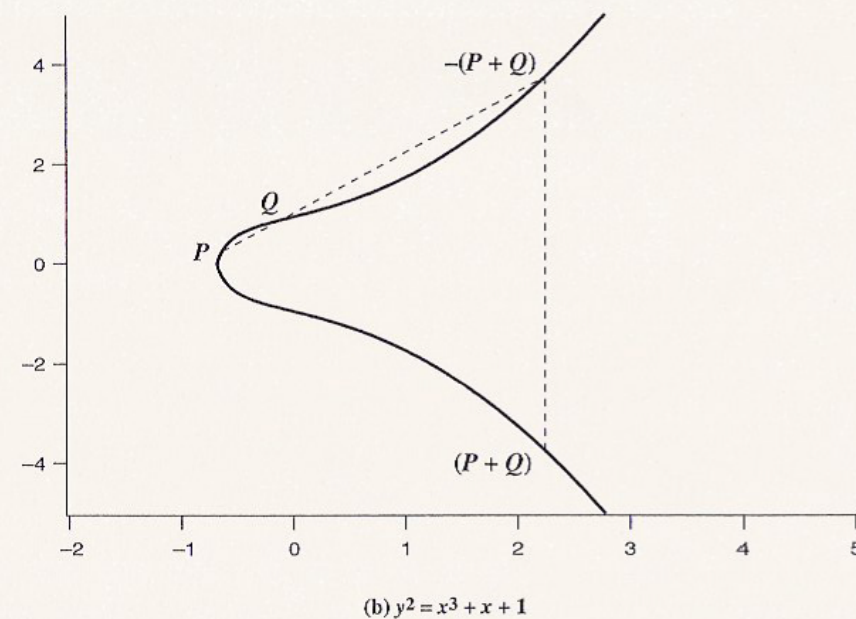
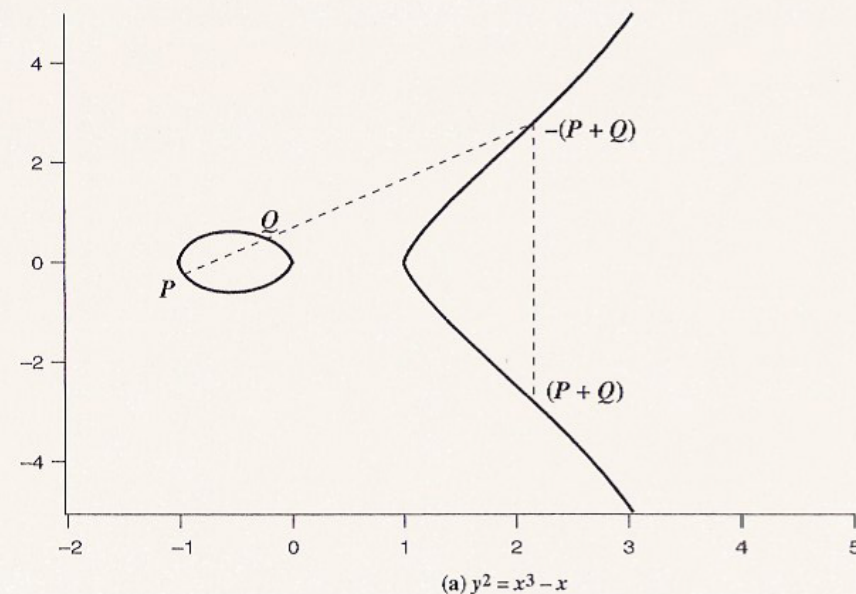
e invia  $P$  a Fabrizio

calcola  $F p = 5 f p = \dots$

## Le curve ellittiche

$$y^2 = x^3 + ax + b$$

Si considerano solo i  
punti a coordinate intere  
(e si opera "in modulo")



Un punto  $P$  può essere sommato a se stesso mediante la tangente in  $P$  alla curva

Moltiplicazione di  $P$  per un numero intero  $k$ :

$$P + P + \dots + P = kP$$

La moltiplicazione è eseguita con raddoppi e addizioni. Per esempio se  $k = 13$ :

$$13P = P + 4P + 8P = P + (2(2P)) + (2(4P))$$

Per  $R = k P$

dati  $k$  e  $P$ , si calcola  $R$  in tempo "breve"

dati  $R$  e  $P$ , si sa calcolare  $k$  solo in tempo esponenziale

## Costruzione di una chiave tra Alice e Bob

- I due concordano su una curva da usare e su un suo punto  $P$  (che possono essere noti a tutti)
- Alice sceglie a caso un intero segreto  $\alpha$ , calcola  $\alpha P$  e lo invia a Bob
- Bob sceglie a caso un intero segreto  $\beta$ , calcola  $\beta P$  e lo invia ad Alice
- Alice calcola la chiave comune  $k = \alpha\beta P$   
Bob calcola la chiave comune  $k = \beta\alpha P$

Arrivederci, e buona permanenza a casa