

Informatica Teorica



Anno Accademico 2022/2023
Fabio Zanasi
<https://www.unibo.it/sitoweb/fabio.zanasi>
Undicesima lezione

A che punto siamo

Abbiamo visto:

introduzione alla teoria della computabilità

quali problemi sono calcolabili?

In questa parte del corso:

introduzione alla teoria della complessità
computazionale

quali problemi sono calcolabili in maniera efficiente?

Riferimenti bibliografici

M. Sipser

Introduction to the theory of computation.

S. Arora, B. Barak

Computational Complexity: A Modern Approach.

Preludio

Una lettera di Gödel (1956)

Princeton, 20.III.1956

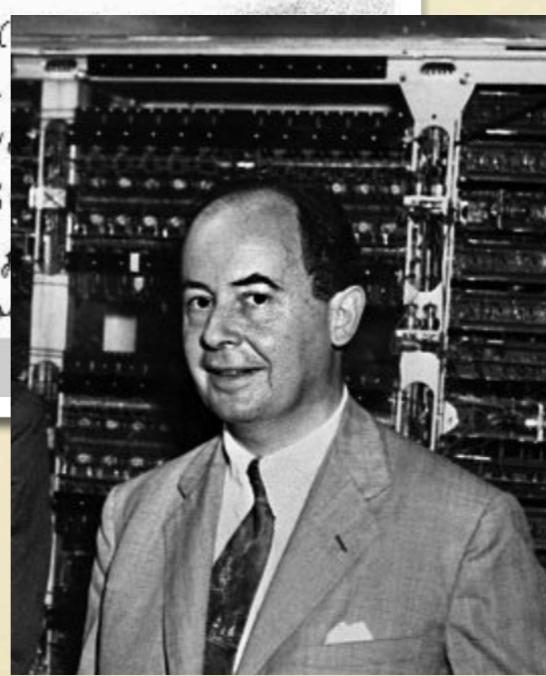
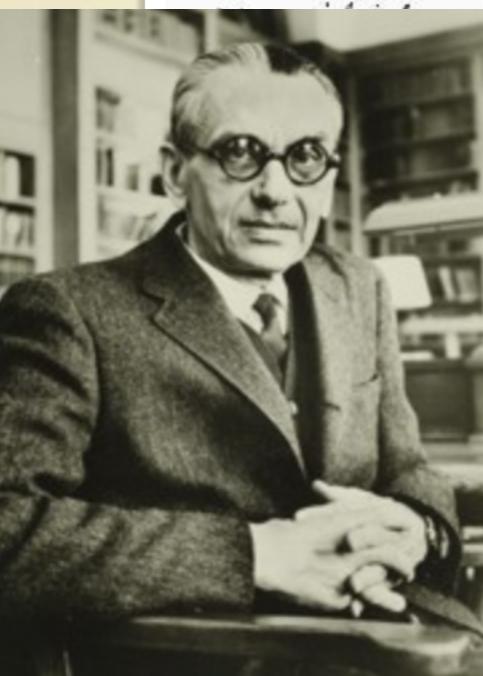
Lieber Herr v. Neumann!

Ich habe mit größtem Bedauern von Ihrer Erkrankung gehört. Die Nachricht kam mir ganz unerwartet. Morgenstem hatte mir zwar schon im Sommer von einem Schwächeanfall erzählt, den Sie einmal hatten, aber er meinte damals, dass dem keine größere Bedeutung beizumessen sei.

Wie ich höre, haben Sie sich in den letzten Monaten einer radikalen Behandlung unterzogen. Ich frage mich, ob diese den gewünschten Erfolg hatte und ob

ich höre, jetzt kräftiger fühlen, kann, Ihnen über ein matter zu schreiben, über das mich

The Ansicht ist interessant: Man kann offenbar leicht eine Turingmaschine konstruieren, welche von jeder Formel F des angegeben Funktionen-Kalküls n. jeder natürl. Zahl n zu entscheiden gestattet, ob F einen Beweis der Länge n hat [Länge = Anzahl der Symbole]. Sei $\Psi(F, n)$ die Anzahl da Schritte, die die Maschine dazu benötigt u. sei $\varphi(n) = \max_F \Psi(F, n)$. Die Frage ist, wie rasch $\varphi(n)$ für eine optimale Maschine wächst. Man kann zeigen $\varphi(n) \geq K n$. Wenn es wirklich eine Maschine mit ~~effektiv~~ $\varphi(n) \sim K n$ gäbe, hätte das Folgerungen. Es würde nämlich offenbar bedeuten, Unlösbarkeit der Entscheidungsarbeit der Mathematik bei vollständig durch Maschinen er-



Una lettera di Gödel (1956)

Dai risultati di Turing e Church sappiamo che l'*Entscheidungsproblem* è indecidibile.

Dato un sistema deduttivo P e una formula F , esiste una dimostrazione di F in P ?

Tuttavia, sappiamo anche che, se poniamo una restrizione di lunghezza, il problema diventa decidibile.

Dato un sistema deduttivo P e una formula F , esiste una dimostrazione di F in P di lunghezza al più n ?

Una lettera di Gödel (1956)

Since, as I hear, you are feeling stronger now, I would like to take the liberty to write to you about a mathematical problem; your view on it would be of great interest to me: Obviously, it is easy to construct a Turing machine that allows us to decide, for each formula F of the restricted functional calculus and every natural number n , whether F has a proof of length n [length = number of symbols]. Let $\psi(F, n)$ be the number of steps required for the machine to do that, and let $\varphi(n) = \max_F \psi(F, n)$. The question

is, how rapidly does $\varphi(n)$ grow for an optimal machine? It is possible to show that $\varphi(n) \geq Kn$. If there really were a machine with $\varphi(n) \sim Kn$ (or even just $\sim Kn^2$) then that would have consequences of the greatest significance. Namely, this would clearly mean that the thinking of a mathematician in the case of yes-or-no questions could be completely¹ replaced by machines, in spite of the unsolvability of the Entscheidungsproblem. n would merely have to be chosen so large that, when the machine does not provide a result, it also does not make any sense to think about the problem [17, p. 375].

Una lettera di Gödel (1956)

Gödel considera una funzione:

$$\varphi : \begin{array}{l} \text{lunghezza } n \text{ massima} \\ \text{della dimostrazione} \\ \text{da trovare} \end{array} \longrightarrow \begin{array}{l} \text{numero di passi} \\ \text{di computazione} \\ \text{di una TM} \end{array}$$

qual è la curva di crescita di φ ?

La ‘congettura’ di Gödel: $\varphi(n)$ è un polinomio Kn^2 , per un qualche coefficiente K .

In termini moderni, diremmo che il problema appartiene alla classe di complessità P, cioè ha complessità polinomiale.

Misurare la complessità

Un esempio

$$A = \{0^k 1^k \mid k \geq 0\}$$

Programma della TM che decide A:

1. Leggi l'input e rigetta se trovi uno 0 a destra di un 1.
2. Fino a che ci sono sia valori 0 che valori 1 sul nastro: scansiona il nastro eliminando un singolo 0 e un singolo 1.
3. Se rimangono ancora valori 0 o 1, rigetta. Se invece il nastro è vuoto, accetta.

n passi

al più
n/2 x n
passi

al più n passi

Supponi che l'input abbia lunghezza n.

Quanti passi di computazione richiede l'algoritmo?

Complessità di tempo

Sia M una TM che si ferma su tutti gli input. La sua **complessità di tempo** è definita come la funzione $f : \mathbb{N} \rightarrow \mathbb{N}$, dove $f(n)$ è il massimo numero di passi che M impiega a fermarsi su arbitrari input di lunghezza n .

Ad esempio, la TM appena vista ha complessità:

$$n + (n/2 \times n) + n$$

Una nozione fragile?

Quanto é 'robusto' questo metodo di misurare la complessità di un algortimo?

- Dipende dal modello di calcolo
- Dipende da cosa intendiamo per 'passi' di computazione
- È sensibile a variazioni minime - ad esempio una subroutine che richiede un tempo costante, indipendente dall'input.
- Dipende dalla codifica dell'input e come misuriamo la sua lunghezza.

notazione Big-O

Un modo più efficace di esprimere la complessità di un algoritmo è usando una notazione che **astrae** dettagli implementativi poco rilevanti, e fornisce solo una **stima** significativa del suo tempo di esecuzione.

Date funzioni $f, g : \mathbb{N} \rightarrow \mathbb{N}$, scriviamo $f(n) = O(g(n))$ e diciamo che $g(n)$ è un **bound superiore asintotico** (*asymptotic upper bound*) se esistono $c \in \mathbb{N}$, e $m \in \mathbb{N}$ tali che per ogni $n \geq m$

$$f(n) \leq c \times g(n)$$

In altre parole, $g(n)$ è sempre grande almeno quanto $f(n)$ per n sufficientemente grandi e modulo un fattore costante c .

notazione Big-O: esempi

$$A = \{0^k 1^k \mid k \geq 0\}$$

Programma della TM che decide A:

1. Leggi l'input e rigetta se trovi uno 0 a destra di un 1.
2. Fino a che ci sono sia valori 0 che valori 1 sul nastro: scansiona il nastro eliminando un singolo 0 e un singolo 1.
3. Se rimangono ancora valori 0 o 1, rigetta. Se invece il nastro è vuoto, accetta.

n passi

al più
n/2 x n
passi

al più n passi

Qual è la complessità dell'algoritmo in notazione Big-O?

notazione Big-O: esempi

$$A = \{0^k 1^k \mid k \geq 0\}$$

Programma della TM che decide A:

1. Leggi l'input e rigetta se trovi uno 0 a destra di un 1.
2. Fino a che ci sono sia valori 0 che valori 1 sul nastro: scansiona il nastro eliminando un singolo 0 e un singolo 1.
3. Se rimangono ancora valori 0 o 1, rigetta. Se invece il nastro è vuoto, accetta.

O(n) passi

O(n²)
passi

O(n) passi

La complessità dell'algoritmo nel suo complesso è perciò

$$O(n) + O(n^2) + O(n) = O(n^2)$$

notazione Big-O: esempi

Data la funzione $f(n) = 5n^3 + 2n^2 + 22n + 6$, abbiamo $f(n) = O(n^3)$.

Perché? Calcoliamo seguendo la definizione. $5n^3 + 2n^2 + 22n + 6 \leq 6n^3$ per $n \geq 10$.

Esercizio: data $f(n) = 2n^3 + 4n^4$, qual è x tale che $f(n) = O(x)$?

Data la funzione $f(n) = 2^n + 81n^8$, abbiamo $f(n) = O(2^n)$.

notazione Big-O: esempi

Il caso dei logaritmi. Ricorda: $x = \log_2 n$ se $2^x = n$.

Cambiare 2 con una differente base $b \in \mathbb{N}$ cambia il valore del logaritmo solo di un fattore costante:

$$\log_b n = \log_2 n \times (1 / \log_2 b).$$

Perciò nell'uso della notazione Big-O, non serve specificare la base dei logaritmi:

$$f(n) = O(\log n)$$

notazione Big-O: esempi

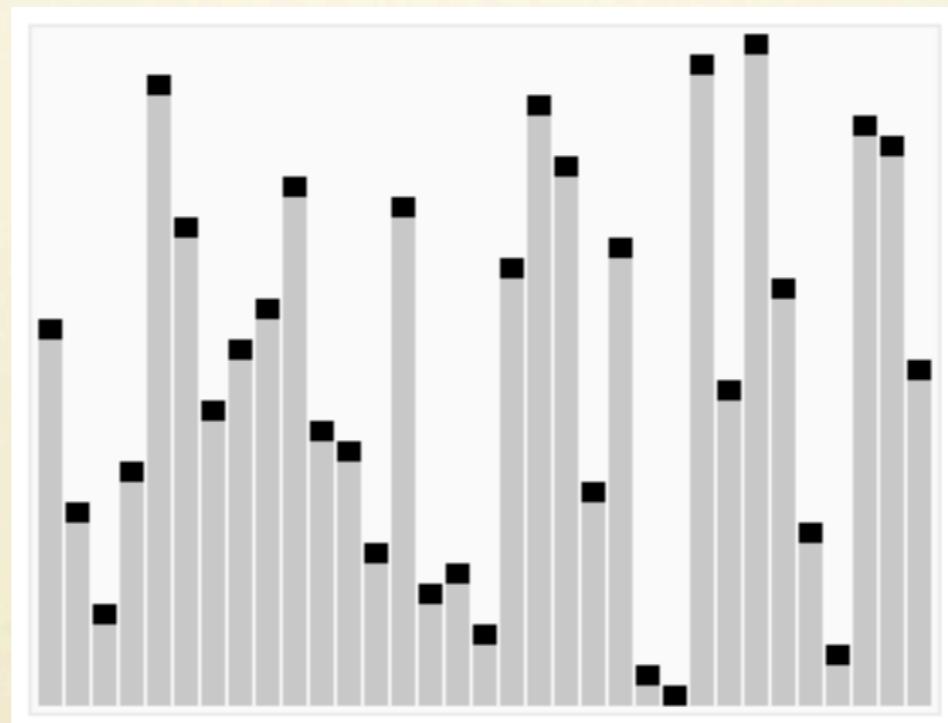
Binary search



$O(\log n)$

notazione Big-O: esempi

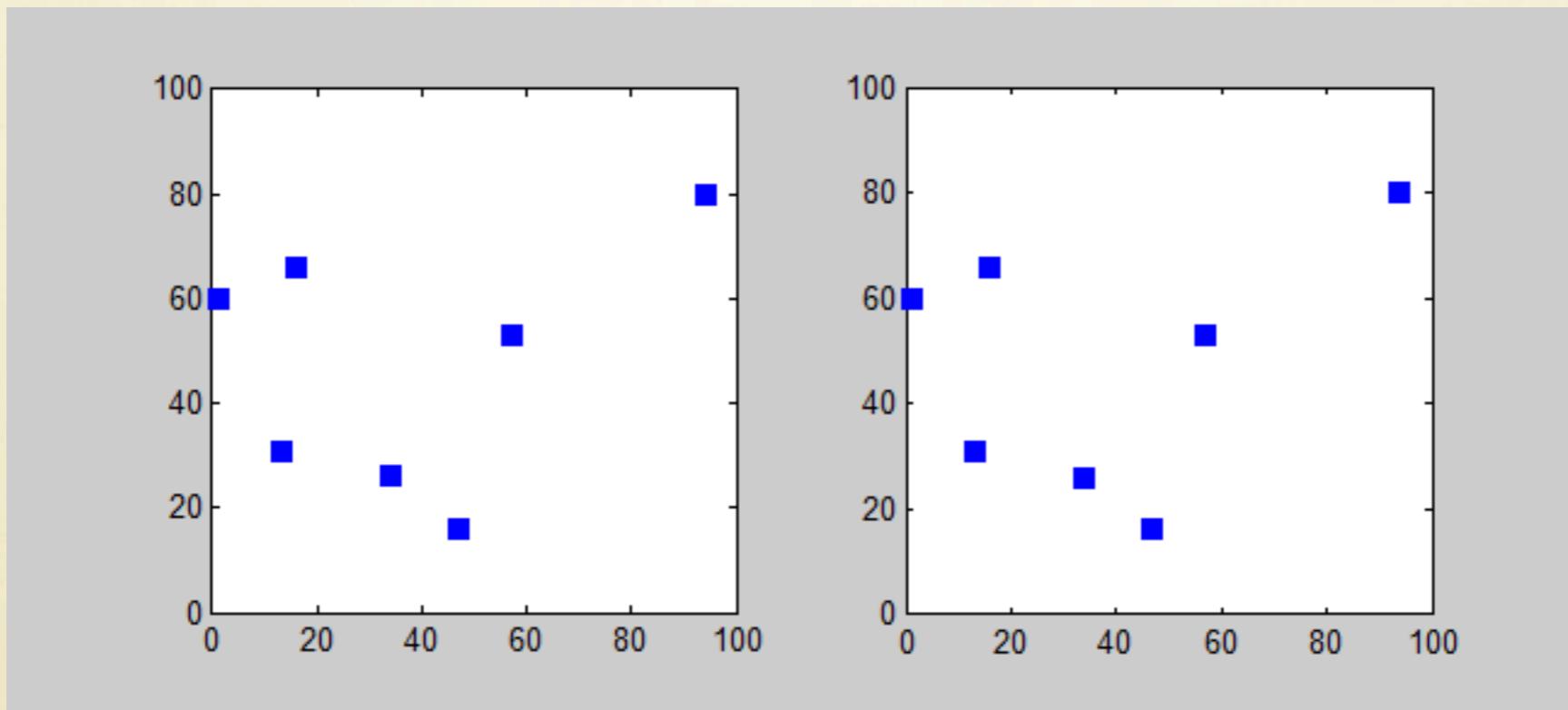
Quicksort



$O(n^2)$

notazione Big-O: esempi

Problema del commesso viaggiatore



$O(n!)$

Classi di Complessità

Data una funzione $f : \mathbb{N} \rightarrow \mathbb{N}$, definiamo **la classe di complessità** (di tempo) **TIME($t(n)$)** come la collezione di tutti i linguaggi decidibili da una TM (deterministica, a un nastro) in tempo $O(t(n))$.

Esempio: il linguaggio $A = \{0^k 1^k \mid k \geq 0\}$ è nella classe TIME(n^2).

Alcune considerazioni

- L'analisi asintotica tramite la notazione Big-O risolve alcune fragilità della nostra misurazione della complessità, ma non tutte.
- Ad esempio, sullo sfondo rimane il problema della codifica: se l'algoritmo lavora su grafi, possiamo misurare la complessità come una funzione sulla lunghezza di una sua specifica codifica, ma ci sono altre possibili scelte 'giuste' (misurare la complessità rispetto ad una diversa codifica, o al numero di nodi), e non sempre equivalenti.
- La classe di complessità dipende anche dal modello di calcolo.
- Un'altra scelta 'arbitraria' è considerare esclusivamente l'analisi del caso peggiore: la classe di complessità dell'algoritmo sarà determinata dall'input di lunghezza n su cui è più inefficiente. In *average-case complexity*, si studia invece la media dei tempi d'esecuzione su input di lunghezza n .

La classe P

La classe P

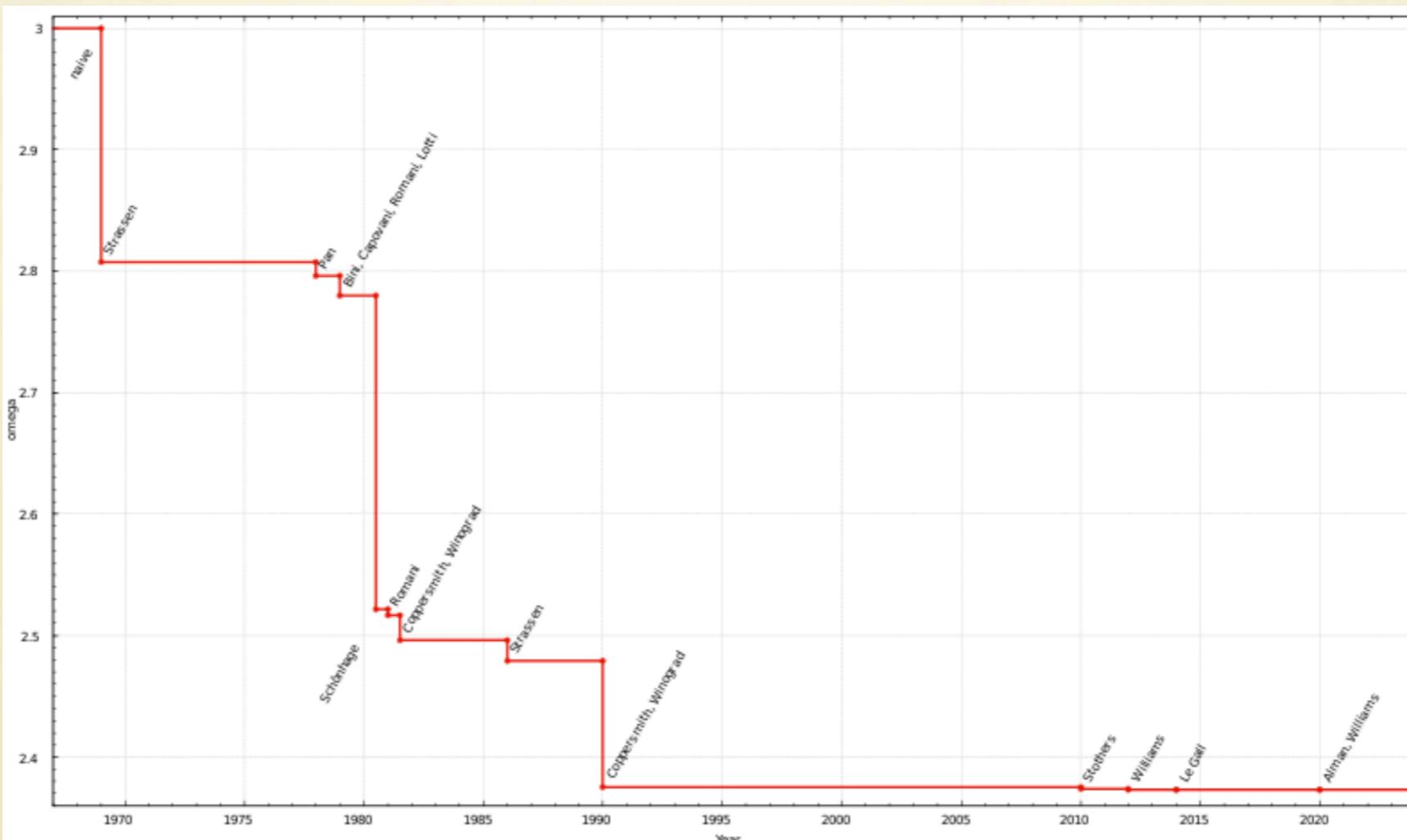
P è la classe dei linguaggi decidibili da una TM (deterministica, a un nastro) in tempo **polinomiale**. Ovvero:

$$P = \bigcup_k TIME(n^k)$$

Perché P?

- Nella pratica dello sviluppo del software, un algoritmo che lavora in tempo polinomiale viene solitamente considerato ‘ragionevole’.
- Il bound polinomiale può essere in realtà molto alto (esempio: $O(n^{10})$), ma l’esperienza ci ha rivelato che, qualora un algoritmo polinomiale è conosciuto, è solitamente possibile renderlo più efficiente.

Perché P?



Miglioramenti del tempo di calcolo
nell'algoritmo che moltiplica due matrici.

Perché P

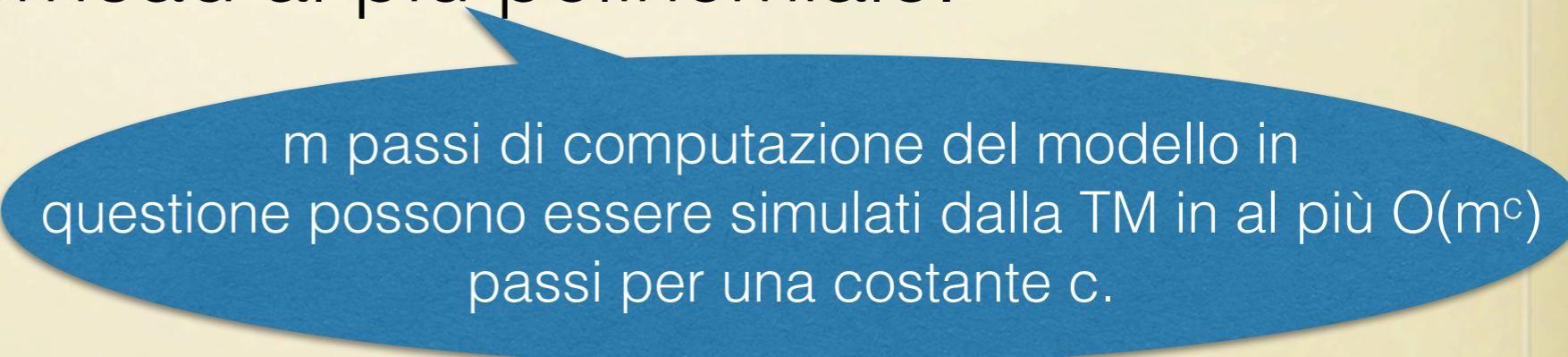
Non tutte: un'eccezione è la codifica
unaria di un numero naturale
 $5 = 1111$

- Un altro aspetto riguarda la codifica. La maggioranza delle codifiche per strutture come grafi, alberi, matrici, automi, etc. come stringhe richiede tempo polinomiale, e produce una stringa di output di lunghezza polinomiale rispetto all'input.
- Dunque un algoritmo della classe P che lavora, ad esempio, su grafi, non `cambia di classe' a seconda della codifica scelta.

Perché P?

Tesi di Church-Turing rafforzata

Ogni modello di calcolo deterministico fisicamente realizzabile può essere simulato da una TM (deterministica, su nastro singolo) con overhead al più polinomiale.

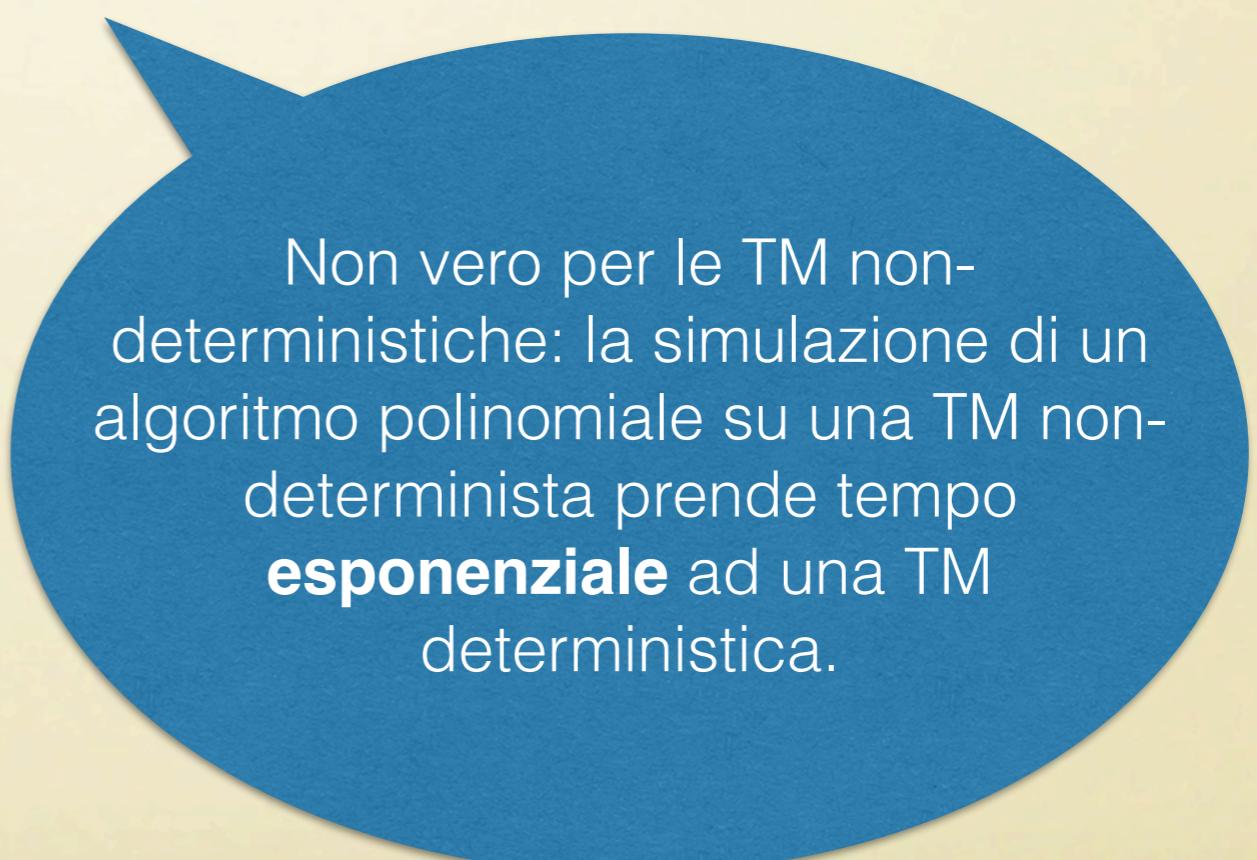


m passi di computazione del modello in questione possono essere simulati dalla TM in al più $O(m^c)$ passi per una costante c .

Se vera, la tesi asserisce che la classe P è robusta, nel senso di essere invariante rispetto al modello di computazione deterministico scelto.

Perché P?

Per esempio, ogni TM con un nastro può simularne in tempo $O(t^2(n))$ una con k nastri che lavora in $t(n)$.



Non vero per le TM non-deterministiche: la simulazione di un algoritmo polinomiale su una TM non-determinista prende tempo **esponenziale** ad una TM deterministica.

La classe EXP

P é la classe dei linguaggi decidibili in un tempo ‘ragionevole’.

$$EXP = \bigcup_k TIME(2^{n^k})$$

EXP é la classe dei linguaggi decidibili in un tempo ‘irragionevole’.

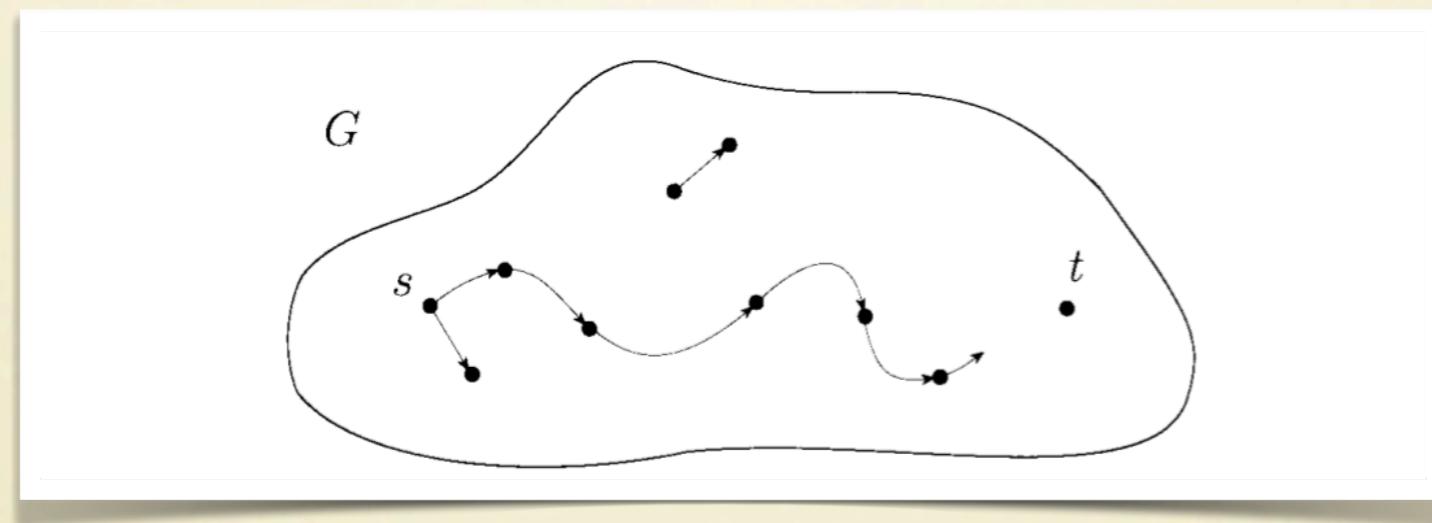
P vs EXP

- La classe EXP intuitivamente è propria degli algoritmi che eseguono un'analisi brute-force, esplorando lo spazio di tutte le possibili soluzioni a un problema.
- P e EXP sono radicalmente differenti nella pratica. Dato un input di lunghezza $n = 100$, un algoritmo che lavora in n^3 impiegherà 1 milione di passi (ragionevole, se assumiamo per esempio che ogni passo impieghi un millisecondo). Uno che lavora in 2^n ne impiegherà .



Esempio: PATH

$\text{PATH} = \{\langle G, s, t \rangle \mid G \text{ é un grafo diretto che ha un percorso diretto da } s \text{ a } t. \}$



Teorema: PATH é in P.

PATH é in P

Dimostrazione:

Considera il seguente algoritmo:

Su input $\langle G, s, t \rangle$:

1. Contrassegna il nodo s .
2. Ripeti fino a che nessun nuovo nodo é contrassegnato: [scansiona tutti gli archi di G . Se trovi un arco da nodo a contrassegnato ad uno b non contrassegnato, contrassegna il nodo b .]
3. Se t é contrassegnato, accetta. Se no, rigetta.

Analisi della complessità: [1] e [3] sono chiaramente in P. La subroutine di [2] é in P ed é ripetuta (al massimo) per il numero di nodi di G , quindi [2] nel suo complesso é in P. Perciò l'algoritmo é in P.

Un algoritmo in EXP per PATH

Un algoritmo alternativo, più inefficiente:

(Brute-force) Esamina tutti i percorsi diretti in G, alla ricerca di uno da s a t.

Analisi della complessità: se il numero di nodi è n, il numero di potenziali percorsi da esaminare è n^n , perciò la complessità è esponenziale (classe EXP).

Una nota sulla codifica

Abbiamo dimostrato che PATH è in P assumendo che la complessità venga misurata rispetto al numero di *nodi* nel grafo.

Tuttavia, una TM non lavora direttamente sul grafo, bensì su una sua codifica come stringa di un alfabeto.

Per esempio, possiamo codificare un grafo come una matrice di adiacenza (dove la cella (i,j) è 1 se c'è un arco dal nodo i al nodo j , e 0 altrimenti), e una matrice di adiacenza come un numero naturale.

In che modo possiamo essere certi che questo non influenzi la nostra analisi della complessità di PATH? Perché ciascuna di queste codifiche impiega tempo al più polinomiale, e modifica la dimensione dell'input di un fattore al più polinomiale.

Altri problemi in P: panoramica

n è primo? Sì [AKS, 2002]

Il grafo G è connesso? Sì [Breadth-first search]

Data una espressione regolare r e la
stringa s , s è in $L(r)$? Sì

Date espressioni regolari r,s , $L(r) =$
 $L(s)$? Probabilmente no (PSPACE-completo)

Problema del commesso viaggiatore Probabilmente no (NP-completo)

Dalla posizione P in una partita di scacchi,
quale giocatore ha una strategia vincente.

Probabilmente no
(PSPACE-complete)

La classe NP

Misurare la complessità di una TM non-deterministica

Ripasso:

Sia M una TM che si ferma su tutti gli input. La sua **complessità di tempo** è definita come la funzione $f : \mathbb{N} \rightarrow \mathbb{N}$, dove $f(n)$ è il massimo numero di passi che M impiega a fermarsi su arbitrari input di lunghezza n .

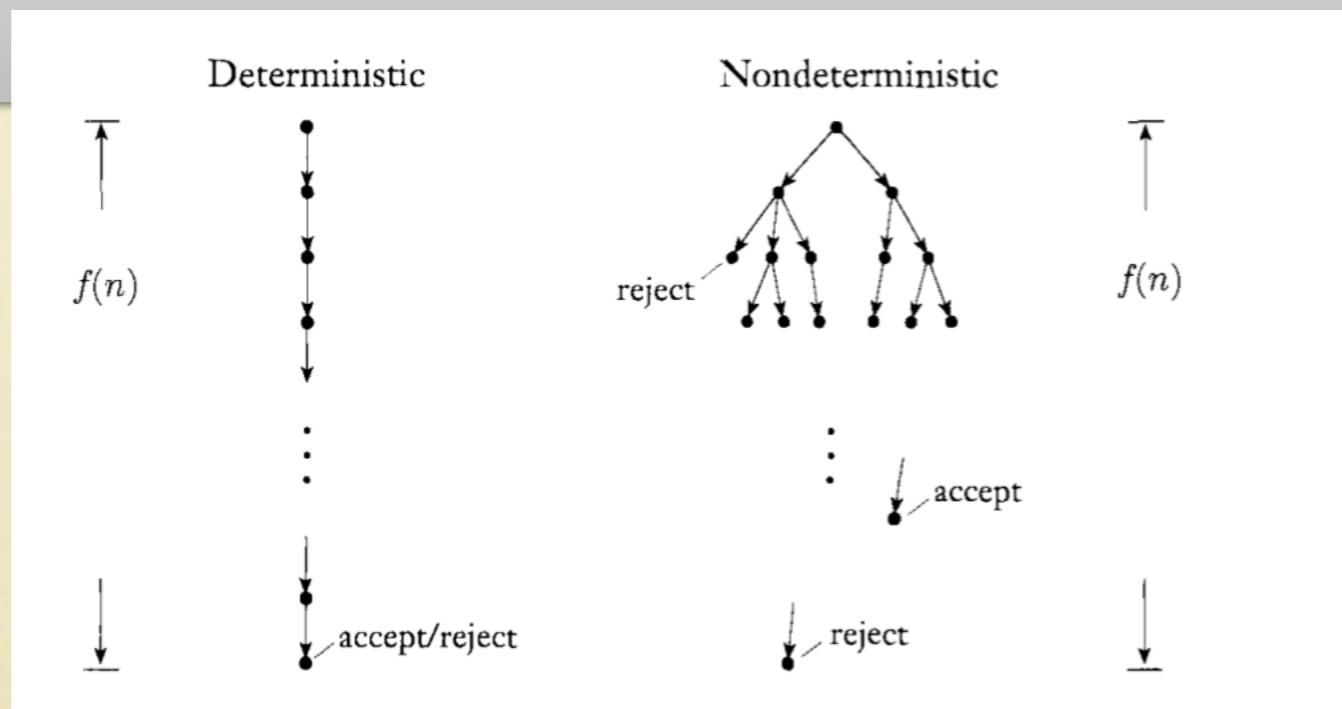


Immagine da Sisper

Determinismo vs Non-determinismo

Ripasso:

Qualsiasi TM non-deterministica M può essere simulata da una TM deterministica M' .

In breve, M' esplora breadth-first tutti i possibili cammini di computazione di M .

Se M richiede tempo polinomiale, M' richiederà tempo esponenziale rispetto alla lunghezza dell'input.

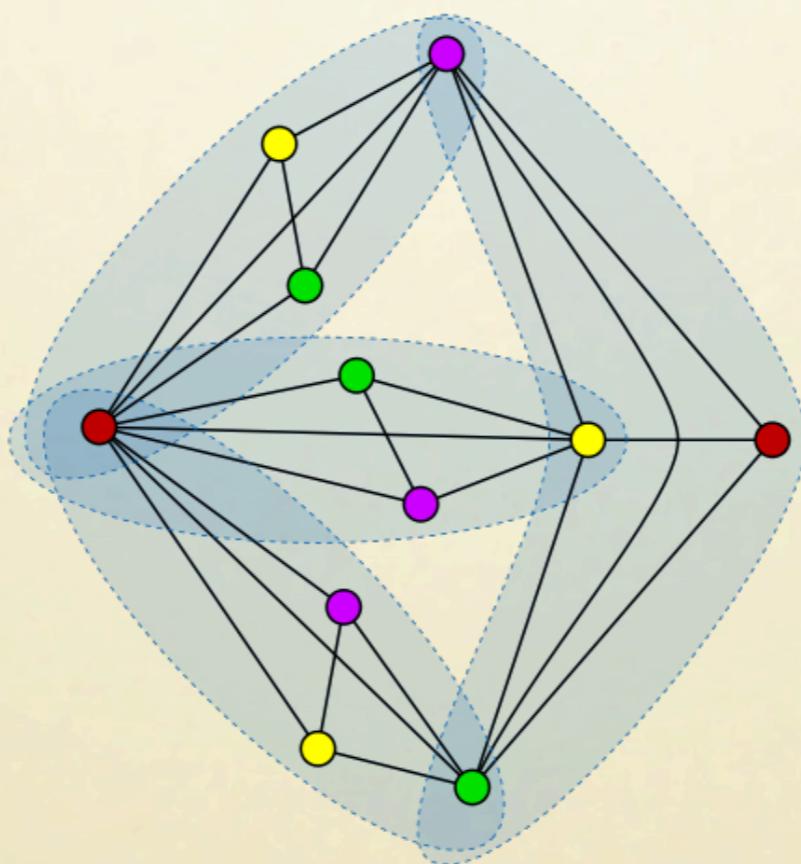
La classe NP

Data una funzione $f : \mathbb{N} \rightarrow \mathbb{N}$, definiamo **la classe di complessità** (di tempo) **NTIME($t(n)$)** come la collezione di tutti i linguaggi decidibili da una TM (non-deterministica, a un nastro) in tempo $O(t(n))$.

$$NP = \bigcup_k NTIME(n^k)$$

Esempio: CLIQUE

Dato un grafo indiretto, un clique è un sotto-grafo dove tutti i nodi sono collegati tra loro da un arco.



Esempio: CLIQUE

$\text{CLIQUE} = \{\langle G, k \rangle \mid \text{Il grafo } G \text{ contiene un clique di } k \text{ nodi.}\}$

CLIQUE è in NP. Ecco un algoritmo non-deterministico polinomiale che lo calcola.

Su input $\langle G, k \rangle$:

1. Seleziona non-deterministicamente un sottoinsieme C di k nodi di G .
2. Verifica che G colleghi tutti i nodi di C tramite archi. Se sì, accetta. Se no, rigetta.

Una caratterizzazione alternativa di NP

Un linguaggio A é **verificabile** se esiste una TM M (che termina sempre, ed accetta o rigetta) con la proprietà

$w \in A$

se e solo se

esiste una string c
tale che M
accetta $\langle w, c \rangle$

Se M lavora in tempo polinomiale, diciamo che A é **verificabile polinomialmente**.

Intuitivamente, c é un **certificato** del fatto che w sia in A.
Nota che, se M lavora in tempo polinomiale, può accedere un certificato di lunghezza al più polinomiale in w.

Una caratterizzazione alternativa di NP

Teorema Un linguaggio é in NP se e solo se é verificabile polinomialmente.

Una caratterizzazione alternativa di NP

Esiste c tale che V accetta $\langle w, c \rangle$

se e solo se

M accetta w

Idea della dimostrazione

In una direzione, sia M una TM non-deterministica che decide un linguaggio L in tempo polinomiale. Costruiamo un verificatore V polinomiale per L nel seguente modo:

Su input $\langle w, c \rangle$:

1. Simula M su w , trattiamo c come descrizione codificata delle scelte non-deterministiche da prendere nell'albero di computazione di M (analogamente a quanto fatto nella dimostrazione che una TM deterministica ne può simulare una deterministica).
2. Se il ramo di computazione esplorato è accettante, accetta.

Una caratterizzazione alternativa di NP

Esiste c tale che V accetta $\langle w, c \rangle$

se e solo se

M accetta w

Idea della dimostrazione

Nella direzione opposta, sia V un verificatore polinomiale per L . Costruiamo una TM M non-deterministica che decide L nel seguente modo:

Su input w di lunghezza n :

1. Seleziona non-deterministicamente una stringa w di lunghezza al più n^k .
2. Simula V su input $\langle w, c \rangle$
3. Se V accetta, accetta. Se no, rigetta.

Esempio: CLIQUE (di nuovo)

$\text{CLIQUE} = \{\langle G, k \rangle \mid \text{Il grafo } G \text{ contiene un clique di } k \text{ nodi.}\}$

Ecco un verificatore polinomiale per CLIQUE. L'idea è che il certificato c è (una codifica del) clique stesso.

Su input $\langle\langle G, k \rangle, c \rangle$:

1. Verifica che c sia un sottografo di k nodi in G .
2. Verifica se ogni nodo in c sia collegato a qualsiasi altro nodo in c da un arco.
3. Se entrambi i test sono positivi, accetta. Se no, rigetta.

P vs NP

P vs NP

P é la classe dei linguaggi A per cui la domanda " $w \in A?$ " può essere **risposta** in maniera efficiente.

NP é la classe dei linguaggi A per cui la correttezza di una risposta alla domanda " $w \in A?$ " può essere **verificata** in maniera efficiente.

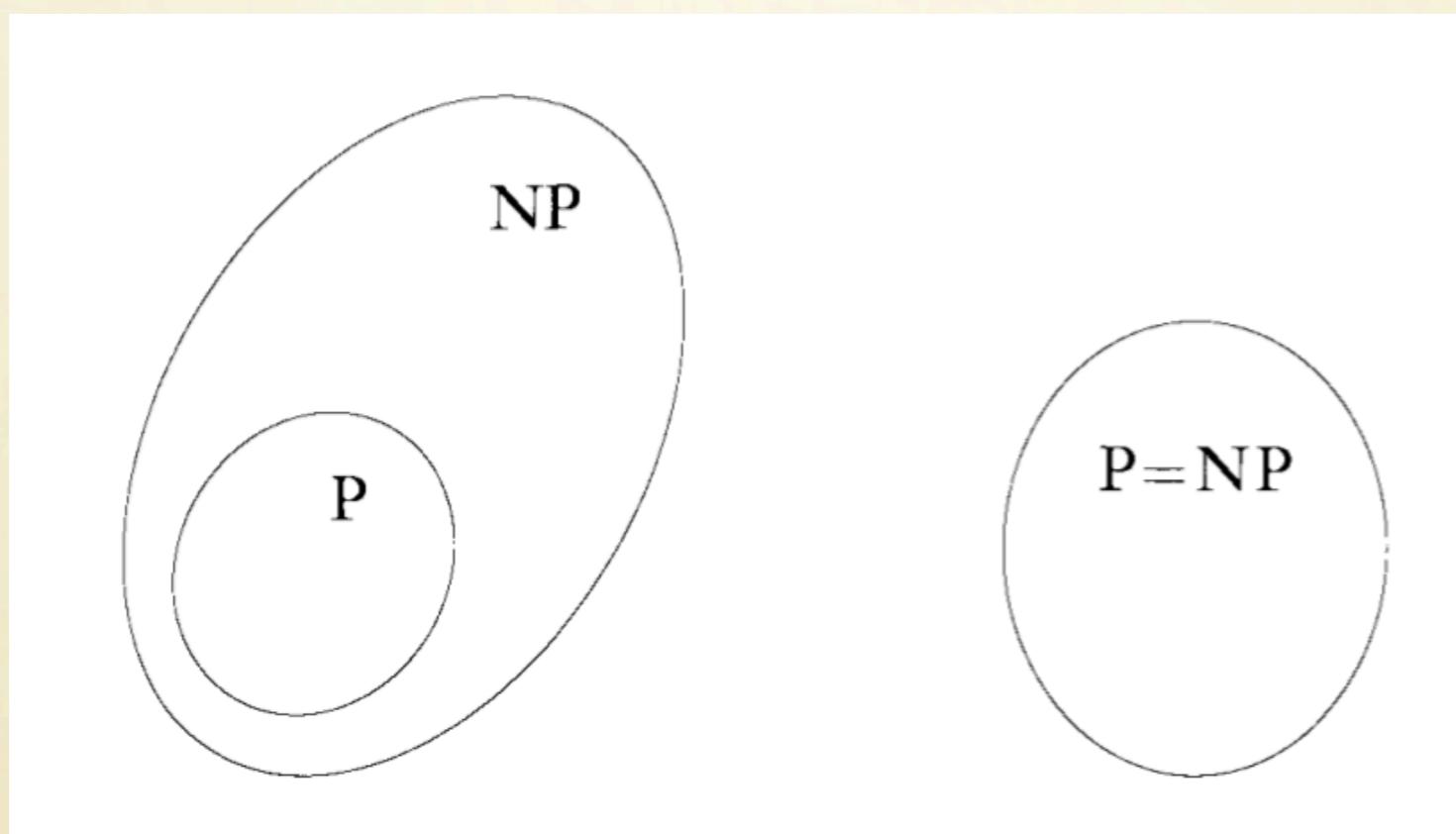
P vs NP

"P = NP?" è uno dei sette *millenium problems*. per i quali l'Istituto Matematico Clay mette in palio un milione di dollari.



P vs NP

Due situazioni possibili:



Ad oggi, sono stati fatti pochissimi progressi verso una soluzione.

P vs NP

“If **P** were equal to **NP**, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in “creative leaps”, no fundamental gap between solving a problem and recognising the solution once it’s found. Everyone who could appreciate a symphony would be Mozart; Everyone who could follow a step-by-step argument would be Gauss.”

– Prof. Scott Aaronson, 2006

P vs NP

Chi é così `pazzo' da credere che $P = NP$?

The New York Times

PROFILES IN SCIENCE

The Yoda of Silicon Valley

Donald Knuth, master of algorithms, reflects on 50 years of his opus-in-progress, "The Art of Computer Programming."

A portrait of Donald Knuth, an elderly man with glasses and a white shirt, sitting in a wicker chair. He is positioned in front of a large, tall stack of books on wooden shelves. The background is dark, making the books stand out.

P vs NP

Perché Knuth propende per $P = NP$?

1. Gli informatici hanno cercato di dimostrare che P è diverso da NP per lungo tempo, senza successo. Questo è un indizio che il contrario ($P = NP$) è vero.
2. D'altro canto, altrettante energie sono state impiegate per trovare algoritmi polinomiali per problemi NP , senza successo. È questo un indizio che P è diverso da NP ? Secondo Knuth no: questa strategia potrebbe essere sbagliata, un vicolo cieco.

P vs NP

Perché Knuth propende per $P = NP$?

3. Infatti, la dimostrazione che $P = NP$ potrebbe essere **non-costruttiva**. Dato un problema in NP, un algoritmo polinomiale potrebbe esistere, ma essere incredibilmente complicato, fuori dalla portata umana. Ad esempio, potrebbe essere di complessità $O(n^g)$, dove g è il numero di Graham.
4. C'è un precedente significativo per quanto sostiene Knuth: il teorema di Roberston-Seymour, il quale dimostra che un certo problema in teoria dei grafi è in P...ma non dà l'algoritmo polinomiale in sé (e non è chiaro come potremmo derivarlo in maniera esplicita).

P vs NP

Perché Knuth propende per $P = NP$?

5. In altre parole, secondo Knuth, anche qualora dimostrassimo che $P = NP$, la dimostrazione sarebbe quasi sicuramente non-costruttiva, e di nessuna utilità pratica.
4. Perciò, in conclusione, Knuth ritiene che P vs NP non sia un problema così interessante, o quantomeno è malposto.