Il più grande problema irrisolto dell'informatica

Un breve percorso intorno al concetto di complessità computazionale

Marco Liverani
liverani@mat.uniroma3.it

5 aprile 2005

Liceo Scientifico Statale "G. Peano" - Roma

Il Novecento un periodo di consolidamento ed “esplosione” della Matematica (e delle altre discipline scientifiche)

Il Novecento è stato un secolo cruciale per tutti gli aspetti fondamentali della matematica:

- il calcolo e l’analisi matematica, con il consolidamento del calcolo integrale e differenziale
- l’algebra e la teoria dei numeri
- la geometria, la geometria differenziale, la geometria algebrica, ecc.
- la logica matematica e i fondamenti stessi della matematica

Nascita di nuove applicazioni e anche di nuove discipline

Abbiamo assistito alla **diffusione sempre più ampia della matematica** e delle **applicazioni della matematica** (la crittografia, la finanza, l'ottimizzazione di processi produttivi, l'astronomia, la meteorologia, ecc.)

Abbiamo assistito alla soluzione di problemi che erano stati posti alla Matematica da altre discipline (non a caso diversi premi Nobel per l'Economia sono stati assegnati a matematici)

La Matematica, insieme con la scoperta di nuove tecnologie, **ha permesso la nascita di nuove discipline**, come l'**Informatica**

Uno stretto rapporto tra Matematica ed Informatica

L'Informatica aiuta le altre scienze a risolvere problemi (oltre alla Matematica anche la Fisica, la Biologia, la Chimica, ...), ad esempio utilizzando i computer per svolgere calcoli difficili o molto lunghi

Per la Matematica però è vero anche il viceversa:

- la Matematica fornisce all'Informatica il **linguaggio** ed il **formalismo** necessari per definire sinteticamente problemi, proprietà ed algoritmi
- la Matematica offre all'Informatica la possibilità di **dimostrare** la correttezza di algoritmi e modelli di calcolo
- l'Informatica pone alla Matematica alcuni **problemi fondamentali**, alcuni dei quali ancora non hanno trovato risposta

Il Novecento: conferma di alcune ipotesi...

Il ’900 per la Matematica (e non solo) è stato sconvolgente: partendo con slancio da una grande sicurezza e partendo dall’idea di essere ad un passo dalla soluzione dei grandi “misteri” e dei grandi problemi irrisolti, si è giunti ad un punto in cui molte certezze sono crollate

All’inizio del ’900 il grandissimo matematico **David Hilbert** ha posto alcuni problemi alla comunità scientifica per delineare lo sviluppo futuro della Matematica (es.: la dimostrazione della “coerenza dell’aritmetica”, la dimostrazione dell’Ultimo Teorema di Fermat, ed altri)

Alcuni di questi problemi sono stati risolti positivamente (es.: nel ’95 il matematico Wiles ha dimostrato l’Ultimo Teorema di Fermat), altri rimangono ancora aperti ed alcuni hanno prodotto l’esito opposto a quello immaginato da Hilbert

... e crollo di grandi certezze!

Nel 1931 il giovanissimo matematico viennese **Kurt Gödel** dimostrò **l'incompletezza dell'Aritmetica** e l'impossibilità di dimostrare tutte le proprietà vere all'interno di una certa teoria!

Ma non solo. Nell'ambito della Fisica, ad esempio, nel 1927 **Heisenberg**, uno dei fondatori della Meccanica Quantistica, formulò un **principio di indeterminazione** destinato a introdurre un terremoto nella Fisica moderna: esistono limiti teorici invalicabili alla possibilità di compiere misurazioni precise sulle "variabili coniugate" del moto di una particella (es.: posizione e quantità di moto) e questo non dipende dalla precisione degli strumenti utilizzati!

Arrivano i nostri: l'Informatica e la Macchina di Turing!

Le **necessità di calcolo** introdotte dalle evoluzioni scientifiche del '900 e l'idea di poter **automatizzare non soltanto i calcoli**, ma anche **l'elaborazione di nuove proprietà**, una volta definiti correttamente i punti di partenza (gli assiomi) di una teoria, portarono allo sviluppo di nuovi modelli di "calcolo automatico"

Nel 1937 il giovane (26 anni) matematico inglese **Alan Turing** propose un modello di calcolo tra i più importanti per la nascita delle teorie informatiche: la **macchina di Turing**

Non è un vero calcolatore, ma soltanto un **modello astratto** di macchina di calcolo, **equivalente ad un computer**

Di nuovo speranze infrante!

Tesi di Church-Turing: se una funzione è calcolabile secondo un qualsiasi formalismo esistente e non, allora lo è anche con una macchina di Turing

Bene, allora **la Macchina di Turing è molto potente**: è proprio vero che è equivalente ad un moderno computer!

Però, leggendola al contrario, la Tesi di Church dice anche che se una funzione non è calcolabile con una macchina di Turing non è calcolabile con nessun altro mezzo di calcolo ...

... e siccome si può dimostrare che **esistono funzioni che non sono Turing-calcolabili** allora ...

... **esistono funzioni e problemi che non sono calcolabili affatto**: neanche con il più potente dei calcolatori possiamo risolverli!

Siamo arrivati di fronte a limiti invalicabili della scienza

Per la **matematica**, Gödel prova che è **impossibile dimostrare** tutte le proprietà vere all'interno di una teoria matematica

Per la **fisica**, Heisenberg afferma che è **impossibile misurare** tutte le quantità di un sistema di particelle

Per l'**informatica**, Church afferma che è **impossibile risolvere per via algoritmica** ogni problema

Un programma per il futuro

Allora è inutile studiare la Matematica, la Fisica e l'Informatica?
No, tutt'altro!

Se è vero che esistono aspetti del mondo fisico che non possono essere misurati direttamente o proprietà matematiche vere che però non possono essere dimostrate nell'ambito di una certa teoria o problemi che non possono essere calcolati, ossia non possono essere risolti in modo esatto mediante un algoritmo ed un programma per un computer ...

... allora la sfida è quella di trovare altri metodi per stimare o misurare indirettamente certe grandezze fisiche, o nuove teorie più potenti, per provare determinate proprietà matematiche, o algoritmi approssimanti che ci permettano di stimare con precisione ciò che non è possibile calcolare in modo esatto con un computer

Si deve solo fare attenzione nell'usare gli strumenti che queste discipline ci offrono!

Oggi parleremo di...

Parleremo di uno dei problemi fondamentali dell'Informatica che oggi **ancora non è stato risolto**: il cosiddetto problema "**P = NP**"

È un problema di quelli determinanti: sulle diverse possibili risposte a questo problema si basano molte delle teorie e delle applicazioni più avanzate dell'informatica

Gli americani, si sa, sono bizzarri, sembra che per loro tutto sia un grande far west: un'importante fondazione americana per la ricerca matematica ha fissato addirittura una "taglia" di 1.000.000$ su questo problema

Di cosa si tratta?

Utilizzare al meglio gli strumenti dell'Informatica significa anche conoscere e studiare il comportamento degli algoritmi nella soluzione di problemi

Parleremo quindi di soluzioni di problemi con l'ausilio del calcolatore, parleremo di **algoritmi** e di come si possa stabilire la **"bontà" di un algoritmo**, di come si possano confrontare fra loro algoritmi differenti per **stimarne l'efficienza**

Oggi esistono problemi che, anche con il più potente dei computer, richiederebbero centinaia di anni per essere risolti

Ci chiediamo se *prima o poi* si riuscirà a trovare per ogni problema un **algoritmo efficiente** che possa **risolverlo in tempi ragionevoli**, o se questo obiettivo è **impossibile**

Algoritmi

Sono “ricette” che descrivono **in passi elementari** un procedimento che ci permette di risolvere un problema, svolgere un calcolo o portare a termine un’operazione articolata

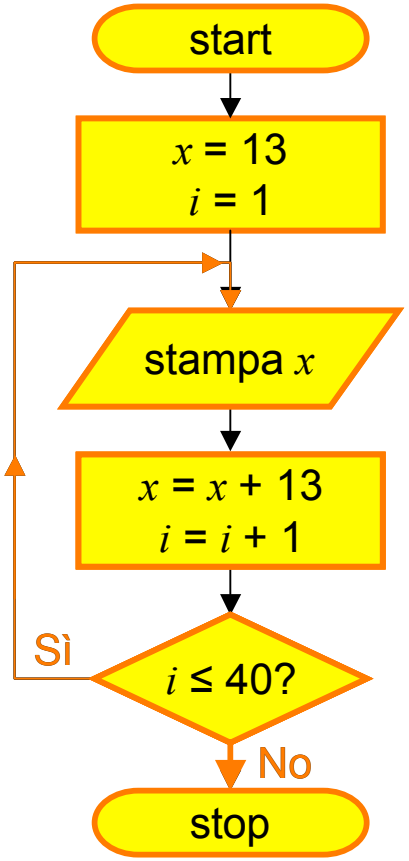
Gli algoritmi devono essere procedure costituite da un **numero finito di passi**

Non solo: il procedimento definito da un algoritmo deve **terminare dopo un numero finito di operazioni**

Un esempio di algoritmo

Problema: stampare i primi 40 multipli di 13

1. assegna il valore 13 alla variabile x
2. assegna il valore 1 alla variabile i
3. stampa il valore della variabile x
4. calcola x + 13 ed assegna il risultato ad x
5. incrementa di 1 la variabile i
6. se $i \leq 40$ allora torna al passo 3
7. fermati



È un procedimento composto da **7 passi**. Ogni passo del procedimento presenta **un'istruzione elementare**.
È facile convincersi che risolve il problema in un **numero finito di operazioni** (quante?).

Allora **è un algoritmo**!

Un esempio di algoritmo

$x = 13$, $i = 1$

stampo x = **13**

$x = x + 13 = 26$, $i = i + 1 = 2$, $i \leq 40$ allora proseguo

stampo x = **26**

$x = x + 13 = 39$, $i = i + 1 = 3$, $i \leq 40$ allora proseguo

stampo x = **39**

$x = x + 13 = 52$, $i = i + 1 = 4$, $i \leq 40$ allora proseguo

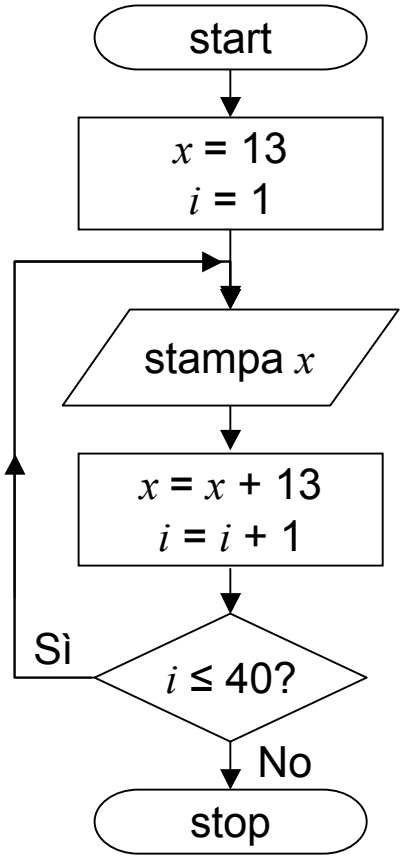
stampo x = **52**

. . .

$x = x + 13 = 520$, $i = i + 1 = 40$, $i \leq 40$ allora proseguo

stampo x = **520**

$x = x + 13 = 533$, $i = i + 1 = 41$, $i \leq 40$ allora mi fermo



Tutte le “procedure” sono algoritmi?

Ogni volta che definiamo qualcosa è bene assicurarsi che non si stia definendo nulla di banale...

Consideriamo il seguente problema: *Stampare tutti i numeri interi maggiori di zero*

Soluzione:

1. assegna il valore 1 alla variabile i
2. stampa il valore della variabile i
3. incrementa di 1 il valore della variabile i
4. vai al passo 2

I passi sono solo 4, sono composti da istruzioni elementari... è vero, ma il procedimento che descrive **non terminerà mai**! Dunque **non è un algoritmo**

Un dubbio...

Forse esistono problemi che non sono risolubili mediante un algoritmo?

È possibile che esistano problemi talmente difficili che non si possa progettare un algoritmo in grado di risolverli in un tempo finito, anche mettendo insieme tutta l'abilità dei migliori progettisti e dei migliori programmatori del mondo?

Dipende forse dalla potenza del computer che utilizzo per eseguire l'algoritmo?

Il problema precedente chiedeva di stampare tutti i naturali, ossia un numero infinito di elementi... forse era proprio la natura del problema che mal si prestava ad essere risolta con un algoritmo

Esistono problemi che pur ammettendo soluzioni di lunghezza finita, richiedono necessariamente un tempo infinito per essere risolti?

Misurare l’efficienza

A fronte di un certo problema possono essere proposti algoritmi differenti per risolverlo: come faccio a stabilire quale è il migliore?

L’algoritmo migliore è **il più efficiente**: quello che riesce a risolvere il problema con il **minor utilizzo di risorse** (della macchina) e nel **minor tempo possibile**

Efficienza = Velocità?

Uno stesso algoritmo, eseguito su macchine diverse, impiega tempi diversi!

Allora l'efficienza di un algoritmo è legata alla potenza della macchina su cui lo eseguo?

No: **una buona misura dell'efficienza deve prescindere dal calcolatore** su cui eseguirò l'algoritmo, altrimenti invece di misurare l'efficienza dell'algoritmo misurerei l'efficienza della macchina!

Efficienza = Difficoltà del problema?

Un'**istanza di un problema** è data dal problema insieme ai dati del problema stesso

Ad esempio un problema "astratto" è quello di calcolare la derivata di un polinomio generico

$$p(n) = x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1} x + a_n$$

mentre un'istanza di questo problema è la richiesta di calcolare la derivata di un polinomio specifico

$$p(x) = x^3 - 12x^2 + 5$$

Lo stesso algoritmo, eseguito su due istanze diverse dello stesso problema, richiede tempi diversi per giungere alla soluzione. Allora l'efficienza di un algoritmo **è legata anche alla particolare istanza del problema** che sto considerando?

Efficienza di un algoritmo

L'efficienza di un algoritmo è data dal **numero di operazioni elementari** compiute dall'algoritmo, **calcolate in funzione della "dimensione dell'input"**, ossia in funzione del numero di dati che dovranno essere elaborati

L'efficienza di un algoritmo non è dunque un numero, ma una funzione

Calcolo dell'efficienza: un esempio

Trovare l'elemento minimo in un insieme di n numeri interi $\{x_1, x_2, \ldots, x_n\}$

Soluzione:

1. considero inizialmente un elemento a caso (ad esempio x_1, il primo) e lo considero come il "candidato" ad essere il minimo elemento dell'insieme
2. confronto il candidato con tutti gli altri elementi e ogni volta che trovo un elemento più piccolo del mio "candidato" lo scambio con il candidato stesso
3. al termine dei confronti il candidato è sicuramente il minimo

Complessivamente **ho eseguito n confronti**: l'efficienza dell'algoritmo è direttamente proporzionale alla "dimensione dell'input"

Calcolo dell'efficienza: un altro esempio

Ordinare in ordine crescente un insieme di n numeri $\{x_1, x_2, \ldots, x_n\}$

Soluzione:

1. per i = 1, 2, . . . , n ripeti le seguenti operazioni:
2. trova l'elemento minimo nel sottoinsieme $\{x_i, x_{i+1}, \ldots, x_n\}$ e scambialo con x_i

Per n volte devo trovare il minimo su insiemi sempre più piccoli: dunque eseguo $n + (n - 1) + (n - 2) + \ldots + 2 + 1 \approx n^2$ **operazioni**

Confronto dell'efficienza di algoritmi

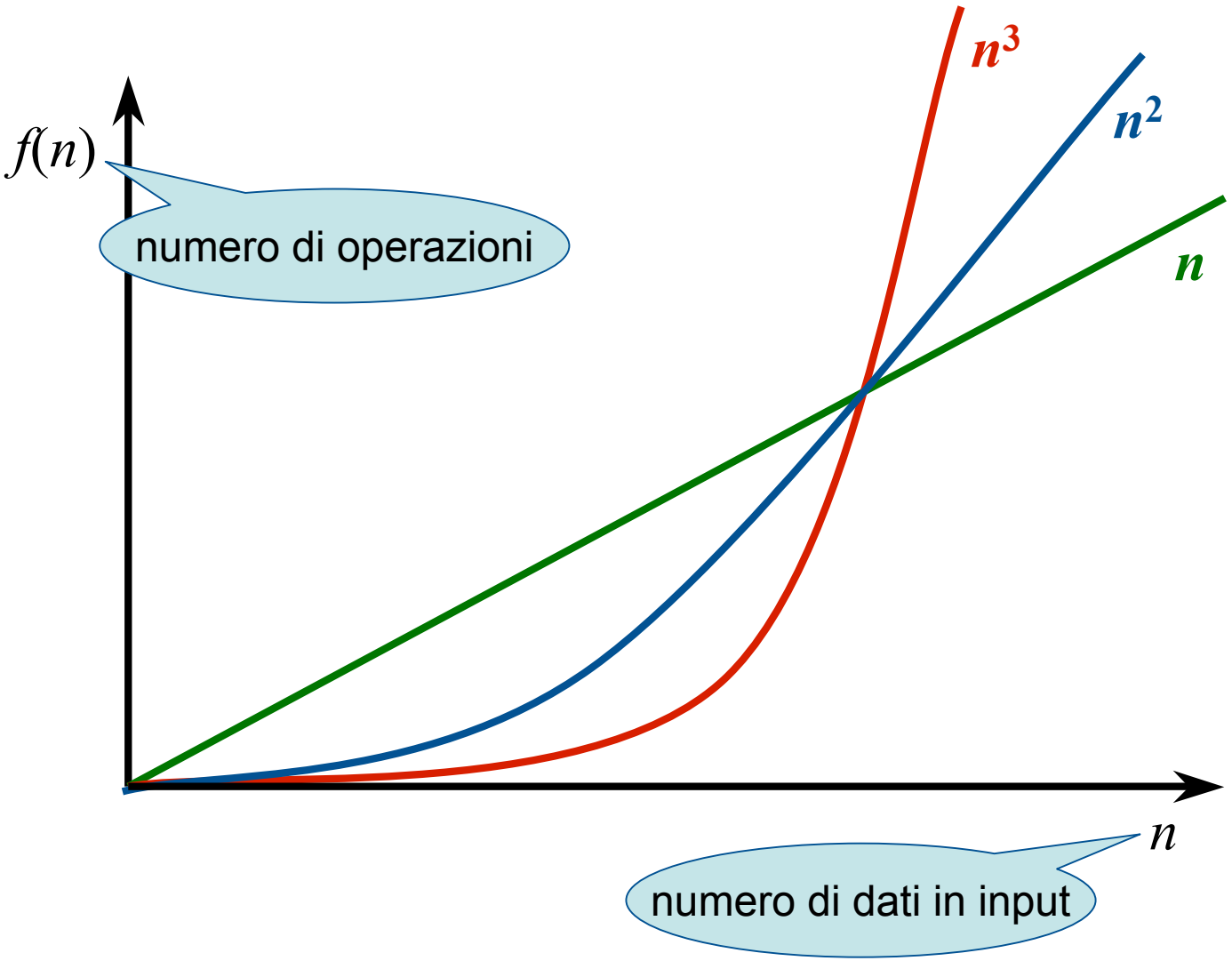
Possiamo esprimere l'efficienza di un algoritmo mediante una funzione $f(n)$ della variabile n: esprime **il numero di operazioni compiute dall'algoritmo a fronte di un'istanza del problema di dimensione n**

D'ora in avanti chiameremo questa misura dell'efficienza **complessità computazionale** dell'algoritmo

A parità di correttezza della soluzione prodotta a fronte di una stessa istanza dello stesso problema, considereremo migliore l'algoritmo con la complessità computazionale più bassa

Se *A* e *B* sono due algoritmi che risolvono lo stesso problema e se $f(n)$ e $g(n)$ sono le funzioni che esprimono la complessità dei due algoritmi, allora ***A* è migliore di *B* se, al crescere di n, risulta $f(n) \leq g(n)$**

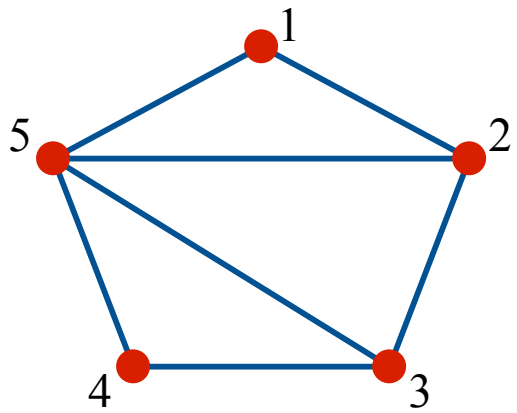
Confronto dell'efficienza di algoritmi



Un esempio più sofisticato

Un **grafo** è un oggetto matematico su cui possono essere definite rigorosamente molte proprietà e che, in particolare in Informatica, è molto utile per formulare modelli di problemi reali (es.: reti di computer, relazioni fra individui, ...)

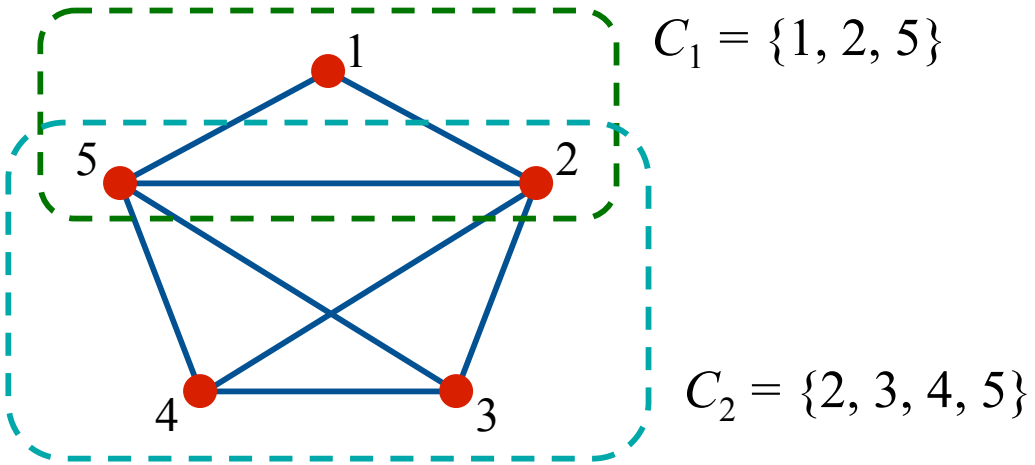
Un grafo $G = (V,E)$ è costituito da un insieme di **vertici** $V = \{v_1, v_2, \ldots, v_n\}$ e da un insieme di **spigoli** E che collegano alcune coppie di vertici



Il problema delle “clique” su un grafo

Una **clique** in un grafo G è un **sottografo C completo, massimale**

Un sottografo è completo quando ogni coppia di suoi vertici è collegata da uno spigolo



Problema “clique”: *dato un grafo trovare tutte le sue clique*

Il problema delle “clique” su un grafo

Problema “clique”: *dato un grafo trovare tutte le sue clique*

Soluzione:

1. sia $G = (V, E)$ il grafo ricevuto in input
2. per ogni sottoinsieme C di V ripeti le seguenti operazioni:
   a. verifica se C forma un sottografo completo di G
   b. verifica se C è anche massimale
   c. se entrambe le verifiche danno esito positivo allora C è una clique

Facciamo un **esperimento** con un programma vero e proprio...

Cosa è successo durante l'esperimento?

Abbiamo cercato la soluzione del problema su un grafo con $n = 6$ vertici e il programma ha visualizzato le 3 clique in un batter d'occhio

Abbiamo cercato la soluzione su un grafo con $n = 8$ vertici e il programma ha visualizzato le 5 clique contenute in G in pochi secondi

Abbiamo provato a cercare la soluzione su un grafo con $n = 9$ vertici e ... dopo diversi minuti il problema non era ancora stato risolto!

Come è possibile? Come è possibile che modificando così poco il grafo l'algoritmo risolutore impieghi un tempo tanto maggiore per arrivare alla soluzione?

Analizziamo l'algoritmo

Ai passi "a", "b" e "c" si chiede di compiere delle verifiche piuttosto semplici: si tratta per lo più di compiere un numero di operazioni proporzionale al numero di vertici di G

Al passo "2" però chiediamo di ripetere le operazioni "a,b,c" **per ogni sottoinsieme** di V

Quanti sono i sottoinsiemi di V? Ad esempio se $V = \{1, 2, 3, 4\}$ **l'insieme dei sottoinsiemi** di V (o **insieme delle parti** di V) è il seguente:

$$\wp(V) = \{\varnothing, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\},$$
$$\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\} \Rightarrow 16 \text{ sottoinsiemi!}$$

Se V ha n elementi allora ha 2^n sottoinsiemi!

Come cresce la complessità...!

Nel primo problema (minimo su un insieme) aumentando di un elemento l'insieme in input, si aumenta di un'operazione il lavoro dell'algoritmo: $f(n) = n$

Nel secondo problema (ordinamento di un insieme) aumentando la cardinalità dell'insieme in input si aumenta in modo quadratico il lavoro dell'algoritmo: $f(n) = n^2$

Nel terzo problema (clique su un grafo) aumentando di un elemento l'insieme in input si raddoppia il lavoro dell'algoritmo: **la crescita è esponenziale**: $f(n) = 2^n$

$$f(n) = 2^n \Rightarrow f(3) = 8, f(4) = 16, \ldots, f(20) = 1.048.576, \ldots,$$
$$f(100) = 1.267.650.600.228.229.401.496.703.205.376$$

Facciamo qualche conto

Se usiamo un computer che impiega 0,000001 secondi (un milionesimo di secondo) per compiere un'operazione, allora...

		Dimensione dell'input (n)			
		n=10	n=20	n=40	n=60
Complessità dell'algoritmo ($f(n)$)	$f(n) = n$	0,00001 s	0,00002 s	0,00004 s	0,00006 s
	$f(n) = n^2$	0,0001 s	0,0004 s	0,0016 s	0,0025 s
	$f(n) = n^5$	0,1 s	3,2 s	1,7 min	13,0 min
	$f(n) = 2^n$	0,001 s	1,0 s	12,7 giorni	**366 secoli!**

C’è soluzione e soluzione

Alcuni algoritmi sono **accettabili**, al crescere della dimensione dell’input anche il tempo di attesa per ottenere la soluzione **cresce in modo ragionevole**

Altri sono **inaccettabili**: il tempo di attesa **cresce troppo rapidamente!**

Gli algoritmi accettabili sono quelli con complessità polinomiale, quelli per cui la funzione complessità è espressa da un polinomio (es.: $f(n) = n^k$)

Gli algoritmi inaccettabili sono quelli con complessità superpolinomiale o addirittura esponenziale (es.: $f(n) = k^n$)

C’è problema e problema

Possiamo passare dal considerare la **complessità di un algoritmo** alla **complessità di un problema**, se adottiamo come misura della complessità del problema la **complessità dell’algoritmo più efficiente** che risolve tale problema

- Il problema della ricerca del minimo è un problema di complessità lineare, $f(n) = n$
- Il problema dell’ordinamento è di complessità quadratica, $f(n) = n^2$ (in realtà la complessità del problema è $f(n) = n \log_2 n$)
- Il problema clique è di complessità esponenziale: non si conosce nessun algoritmo risolutivo più efficiente di quello di ricerca esaustiva che ha complessità esponenziale, $f(n) = 2^n$

Raggruppiamo i problemi

Una delle grandi passioni dei matematici è quella di “classificare” gli oggetti in base alle loro proprietà

Definiamo la **classe P dei problemi polinomiali** come l’insieme dei problemi che possono essere **risolti** da un algoritmo di **complessità polinomiale**

Non è una definizione banale, infatti il problema del minimo, il problema dell’ordinamento e moltissimi altri problemi rientrano in questa categoria...

... e abbiamo visto che esistono problemi, come quello delle clique di un grafo, che non appartengono alla classe **P**

Verificare una soluzione è più facile che trovarla

Fino ad ora ci siamo occupati di **algoritmi risolutori**, algoritmi in grado di **trovare la soluzione** di un'istanza di un problema

Possiamo progettare anche algoritmi che non sono in grado di trovare una soluzione, ma che, ricevuta in input un'istanza di un problema ed una ipotetica soluzione, possono **verificare** se quella è effettivamente una soluzione

Ad esempio un conto è trovare gli zeri di un'equazione di settimo grado, un conto è verificare se $x = 3$ è uno zero dell'equazione assegnata: verificare una soluzione è sicuramente più facile che trovarla!

Chiamiamo questi algoritmi **algoritmi di verifica**

Una nuova (più ampia) classe di problemi

Definiamo la **classe dei problemi NP** come la classe di quei problemi che possono essere **verificati** con un hanno un **algoritmo di verifica di complessità polinomiale**

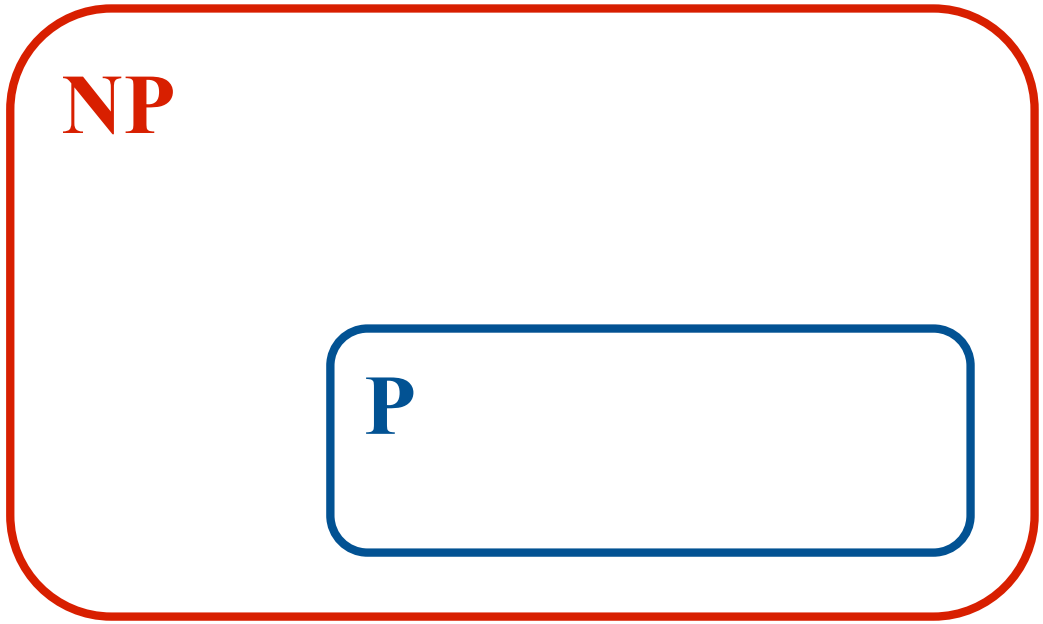
Sicuramente tutti i problemi che hanno un algoritmo risolutore polinomiale hanno anche un algoritmo di verifica polinomiale

Ma anche alcuni problemi di cui non si conosce un algoritmo risolutivo polinomiale, hanno invece un algoritmo di verifica polinomiale, ad esempio il problema clique!

Allora vale sicuramente la relazione

$$P \subset NP$$

La “classificazione” prende forma



P: problemi **risolubili** in tempo polinomiale

NP: problemi **verificabili** in tempo polinomiale

Il più grande problema dell'informatica

La domanda posta da questo problema a questo punto è semplice:

$$P = NP?$$

In altri termini: esiste *veramente* qualche problema **verificabile in tempo polinomiale** che **non possa essere risolto in tempo polinomiale**?

La domanda non è banale e neanche assurda: per i problemi come il problema clique, di cui sappiamo verificare la soluzione in tempo polinomiale, **fino ad oggi** nessuno è stato in grado di formulare un algoritmo polinomiale in grado di risolvere il problema

Ed inoltre nessuno fino ad oggi è stato in grado di dimostrare che un algoritmo risolutore polinomiale per quei problemi non esiste

Il più grande problema dell'informatica

Quindi la domanda "**P** = **NP**?" è legittima: esiste un algoritmo risolutore polinomiale per i problemi che oggi sono in **NP** ma non sono in **P**?

- Se esistesse un algoritmo polinomiale per ogni problema che è in **NP** − **P** allora si potrebbe concludere che **P** = **NP**

- Viceversa basterebbe trovare anche un solo problema in **NP** per cui si dimostri l'impossibilità di costruire un algoritmo risolutore polinomiale per concludere che **P** ≠ **NP**

Trasformare un problema in un altro

Consideriamo ad esempio il problema $ax^3 + bx^2 + cx + d = 0$

Consideriamo poi il problema $\alpha x + \beta = 0$

Un'istanza del primo problema è ad esempio l'equazione $2x^3 - 3x + 1 = 0$
(ossia una assegnazione di valori ai coefficienti: $a = 2$, $b = 0$, $c = -3$, $d = 1$)

Un'istanza del secondo problema è una coppia di valori assegnati ai coefficienti α e β, ad esempio $\alpha = 3$, $\beta = -2$: $3x - 2 = 0$

È possibile trasformare rapidamente (in tempo polinomiale) ogni istanza del secondo problema in una istanza del primo, ponendo $a = 0$, $b = 0$, $c = \alpha$, $d = \beta$

Trasformare un problema in un altro

A questo punto posso utilizzare lo stesso procedimento risolutivo del primo problema per risolvere anche il secondo!

Questo significa che il primo problema (equazioni di terzo grado) è una generalizzazione del secondo (equazioni di primo grado) e quindi che il primo problema è **più difficile** del secondo

Il passaggio da un'istanza del secondo problema ad una istanza del primo problema è una trasformazione molto semplice: **è possibile costruire un algoritmo polinomiale per compiere tale trasformazione**

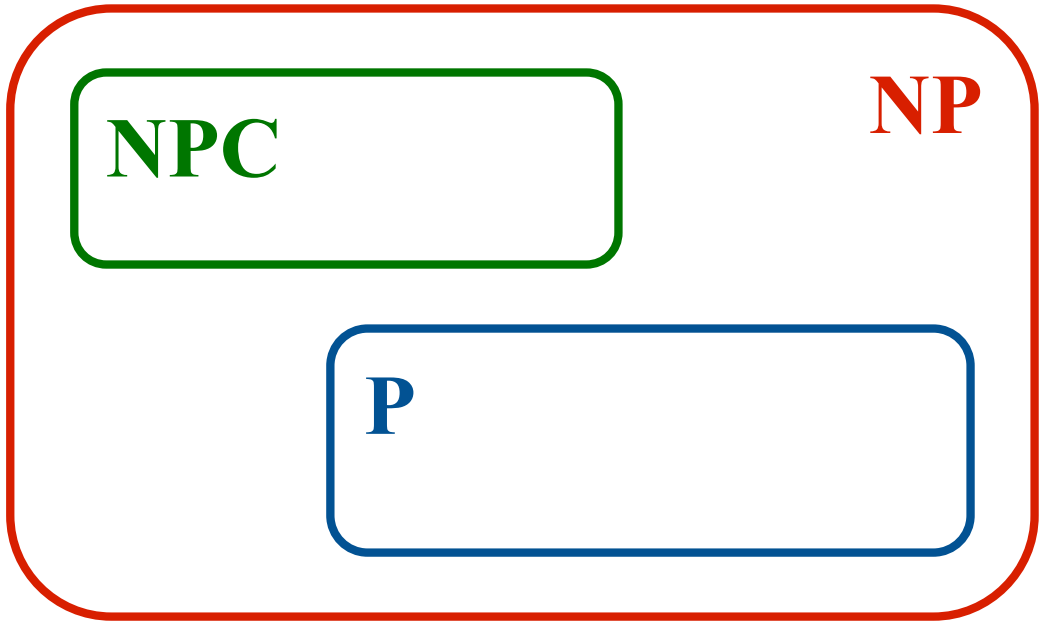
Una classe privilegiata di problemi difficili

Definiamo una classe speciale: la classe **NPC** dei **problemi NP-completi**, costituita da quei problemi Q che sono **NP** e tali che le istanze di tutti gli altri problemi **NP** possono essere trasformate in tempo polinomiale in istanze di Q

I problemi **NPC** sono allora il "club" dei **problemi più difficili** tra i problemi **NP**

Il problema clique fa parte della classe **NPC**

La “classificazione” si arricchisce



P: problemi **risolubili** in tempo polinomiale

NP: problemi **verificabili** in tempo polinomiale

NPC: problemi **NP** a cui ci si può **ricondurre** in tempo polinomiale

Il sugo della storia

Se trovo un algoritmo polinomiale per risolvere un problema NPC allora posso risolvere in tempo polinomiale tutti i problemi NP e quindi $P = NP$

Se dimostro che un solo problema NPC non può essere risolto in tempo polinomiale, allora questo è sufficiente per concludere che $P \neq NP$

La sfida è ancora aperta!