I limiti dell'elaborazione computerizzata e la questione P=NP

Circa venticinque anni fa venne riportata alla luce una lettera risalente al 1956 in cui Kurt Gödel descriveva a John von Neumann un problema di natura matematica che può essere visto come la prima formulazione della "**questione P=NP**", uno dei problemi più rilevanti nell'ambito dell'informatica teorica e dagli innumerevoli risvolti pratici, ad esempio sulla sicurezza dei metodi crittografici.

Figura 1: Kurt Gödel (1906-1978), di origini austriache e naturalizzato statunitense, è stato uno dei più grandi logici del novecento, noto soprattutto per i suoi lavori sull'incompletezza delle teorie matematiche. Dopo il ritrovamento della lettera indirizzata a von Neumann divenne un punto di riferimento anche per la comunità dell'informatica teorica.

Il problema venne posto all'attenzione della comunità scientifica internazionale nel 1971 da Stephen Cook e poco dopo da Dick Karp, e fu subito evidente che si trattava, in qualche misura, di un problema dalle caratteristiche eccezionali la cui soluzione richiedeva lo sviluppo di conoscenze matematiche ed informatiche ben superiori a quelle dell'epoca. Le previsioni dei più pessimisti colsero nel segno. Infatti, ancora oggi, nonostante gli innumerevoli sforzi compiuti per tentare di dare una soluzione all'enigmatico problema, la **questione P=NP** non solo è rimasta aperta, ma è finanche stata inclusa nella lista dei cosiddetti *problemi del millennio* (Millennium Prize Problems). Questa lista, comprendente i più complessi problemi aperti della matematica e dell'informatica, è stata posta all'attenzione della comunità scientifica dall'Istituto Matematico Clay, che nel 2000 ha messo in palio, per ciascuno di essi, un premio di un milione di dollari al primo ricercatore in grado di esibire una corretta soluzione.

Figura 2: L'icona che rappresenta l'Istituto Matematico Clay è inspirata all'opera "Figureight Knot Complement vii/ CMI" dello scultore Helaman Ferguson. La lista dei problemi compilata dall'istituto comprende sette distinti problemi di natura matematica, uno dei quali, la "Congettura di Poincaré", è stato risolto nel 2002 dal matematico russo Grigorij Jakovlevič Perel'man. Gli altri sei problemi, tra cui la questione P=NP, sono ad oggi aperti.

Ma cosa ha dunque di così affascinante e rilevante la **questione P=NP**? E, tanto per iniziare, cosa si nasconde dietro le sigle **P** ed **NP** che compongono i termini di questa strana equazione?

Informatica, problemi ed algoritmi

Per dare una risposta esauriente alle domande che ci siamo posti e data la profondità della questione che intendiamo affrontare, è necessario iniziare a riflettere sulla natura stessa dell'informatica e su quali sono gli obiettivi che si prefigge di raggiungere questa disciplina. Il termine informatica è oggi infatti diventato di uso comune, con contorni spesso non ben definiti in cui sono mescolati aspetti tecnologici con altri di natura più metodologica. L'utilizzo del termine nell'accezione comune non è perfettamente allineato con ciò che si intende nell'ambito della ricerca scientifica, ma tuttavia coglie bene una sfaccettatura cruciale della cosiddetta "scienza dell'informatica" (meglio nota con la forma inglese di *computer science*), cioè l'idea di poter studiare e sviluppare sistemi basati su computer e calcolatori elettronici in grado di risolvere in modo automatico problemi della vita quotidiana.

Supponiamo, ad esempio, di avere necessità di esibire un certificato di nascita e di recarci presso l'Ufficio Anagrafe del proprio Comune di residenza. Allo sportello ci sarà chiesto un documento identificativo e, sulla base del nostro nome e delle informazioni personali che ci identificano univocamente, l'addetto cercherà negli archivi la documentazione di interesse al fine di poter rilasciarcene una copia. Ma come avviene la ricerca in questi archivi? Fino all'avvento dei computer e dell'informatica, l'addetto si trovava costretto a ricercare la documentazione consultando in prima persona gli innumerevoli faldoni risposti più o meno ordinatamente sugli scaffali dell'ufficio. I faldoni riportavano l'anno solare di riferimento e internamente gli atti erano registrati in ordine cronologico. Per cui l'addetto procedeva individuando, per prima cosa, i faldoni di interesse, per poi scorrerne le pagine fino ad arrivare al mese ed al giorno di nascita dell'interessato. Alla fine, con un po' di pazienza, l'addetto riusciva nel suo scopo. Ma i tempi di questo processo erano decisamente imparagonabili a quei pochi millesimi di secondi che oggi intercorrono, in un ufficio informatizzato, tra la richiesta formulata dall'addetto ad un terminale e la risposta che gli viene fornita dal computer consultando una versione digitalizzata dei faldoni di cui sopra.

In questo esempio, il *problema* che stiamo affrontando è quello della ricerca di una specifica informazione in un archivio. La soluzione informatica (più ovvia) consiste semplicemente nel rendere disponibile in formato digitale tutti gli atti d'ufficio, e nel programmare un computer in modo che analizzi uno per uno questi atti al fine di segnalare all'operatore quelli registrati proprio nella data di interesse per l'utente che si è presentato

allo sportello. Questa soluzione, che in termini informatici va sotto il nome di *algoritmo,* diventa più efficiente della soluzione affidata alla consultazione "manuale" dei faldoni, grazie alle potenzialità di calcolo di un moderno elaboratore che, in pochi istanti, è in grado di analizzare decine di migliaia di atti al fine di individuare quelli di interesse. Dunque, in questo esempio, è proprio la potenza di calcolo a fare la differenza. Ma non sempre affidarsi ciecamente alla potenza di calcolo ci conduce a realizzare sistemi informatici che siano poi funzionanti nella pratica. A volte, infatti, è importante scendere nel merito di *come* le soluzioni informatiche sono sviluppate, cioè di come l'algoritmo risolutivo sia stato congegnato.

Figura 3: Nell'esempio del rilascio di un certificato di nascita la soluzione informatica rende efficiente l'accesso e la consultazione di tutta la documentazione degli atti d'ufficio opportunamente digitalizzati. Più in generale, si può pensare che qualunque tipologia di informazione testuale, come ad esempio un libro, possa essere digitalizzata e resa disponibile elettronicamente al fine di ottimizzare le operazioni di ricerca (e agevolare la divulgazione). In quest'ambito si inquadra, ad esempio, il famoso progetto Gutenberg, iniziativa avviata dall'informatico Michael Hart nel 1971, proprio con l'obiettivo di costituire una biblioteca di versioni elettroniche liberamente riproducibili e consultabili di libri. Il progetto Gutenberg è la più antica iniziativa del settore (delle biblioteche digitali).

Per chiarire meglio questo ultimo concetto, riprendiamo l'esempio dell'Ufficio Anagrafe ipotizzando che il Sindaco del Comune abbia deciso, a fini promozionali, di celebrare con una festa pubblica il residente più longevo, magari ultracentenario. In termini astratti, abbiamo dunque a che fare con un insieme $\{r_1,\ldots,r_n\}$ di n residenti le cui informazioni anagrafiche sono tutte digitalizzate negli archivi informatici degli uffici. Il nostro problema è questa volta quello di individuare, all'interno di questo insieme, il residente più anziano. Il Sindaco emana dunque un bando pubblico per la definizione di una soluzione informatica, ed al bando rispondono due ditte presentando due algoritmi differenti:

1. La prima ditta propone un sistema informatico basato su un algoritmo che prende il primo residente r_1 e ne confronta l'anzianità con quella di r_2, poi con quella di r_3 e così via, fino a confrontarla con quella di r_n. Dunque, per r_1 effettua $n-1$ confronti, e se in tutti questi confronti r_1 è sempre risultato il più anziano, allora r_1 sarà il festeggiato! Se così non fosse, l'algoritmo passerebbe ad r_2 procedendo al confronto di r_2 con tutti gli altri $n-1$ residenti. Se neanche r_2 risultasse il più anziano, si passerebbe ad r_3, e così via. Si noti che dopo aver considerato l'i-esimo residente r_i, il numero complessivo di confronti effettuati è pari alla somma del termine $(n-1)$ ripetuta i volte, cioè $i \times (n-1)$. Dunque, nel caso peggiore, che si verifica se si scoprisse che proprio r_n è il più anziano, l'algoritmo richiede $n \times (n-1)$ confronti.
2. La seconda ditta propone invece un algoritmo basato sul concetto di "*più anziano corrente*". L'idea è di analizzare, uno ad uno, i residenti $r_1,\ldots,r_n$. Al primo passo, r_1 sarà considerato il più anziano corrente (proprio perché abbiamo analizzato solo lui, al momento). Al secondo passo, si confronterà r_2 con *il più anziano corrente* (cioè con r_1) e il più vecchio tra i due diventerà il nuovo *più anziano corrente*. Al terzo passo, si confronterà r_3 con il più anziano corrente (che sarà uno tra r_1 e r_2) e il più vecchio tra i due diventerà il nuovo *più anziano corrente.* Alla fine, quando tutti i residenti saranno stati analizzati, il *più anziano corrente* sarà veramente il più anziano in assoluto. Si noti che in questo caso, per ogni residente che viene considerato dopo il primo, un solo confronto (con il *più anziano corrente*) sarà effettuato. Pertanto, il numero complessivo di confronti è $n-1$.

I due algoritmi sopra descritti sono entrambi corretti, ma abbiamo visto che essenzialmente il primo richiede *n x (n-1)* operazioni, mentre il secondo ne richiede solo *n*. Ad esempio, su un Comune di *1000* abitanti (cioè *n=1000*), il primo approccio riesce a calcolare il residente più anziano con *1000 x 999* confronti (dunque circa 1 milione!), mentre al secondo ne bastano solo *1000*. Quale soluzione sceglieresti se foste voi il Sindaco?

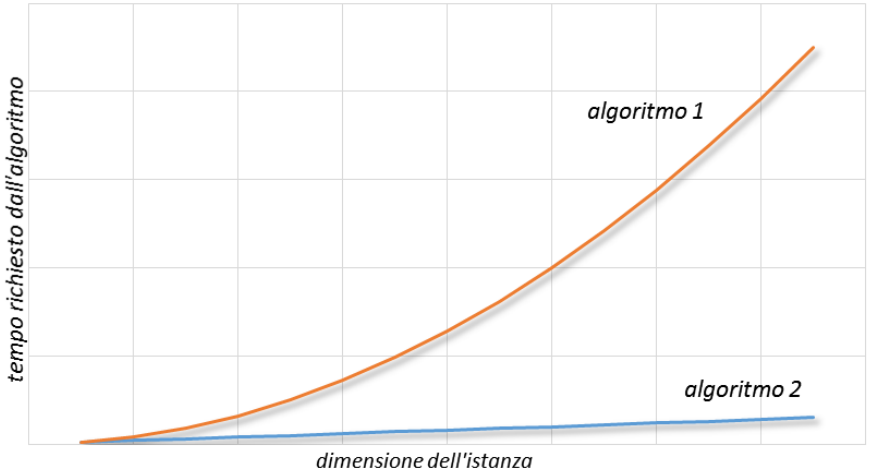

Figura 4: Confronto del tempo richiesto dai due algoritmi proposti per l'individuazione del residente più anziano in un Comune. L'algoritmo 1 richiede un numero di confronti pari a n x (n-1), mentre l'algoritmo 2 richiede solo n-1 confronti. Si dice anche che il primo algoritmo ha complessità "quadratica" (in quanto dipende essenzialmente da n moltiplicato per se stesso, e dunque da n elevato al quadrato), mentre il secondo ha complessità "lineare" (dipende essenzialmente da n).

E' evidente che la nostra preferenza ricadrebbe sul secondo approccio, che costituisce un algoritmo più *efficiente* del primo. Questo vale indipendentemente dalla potenza di calcolo del computer su cui la soluzione sarà implementata, suggerendoci dunque un aspetto della *computer science* che spesso sfugge alla visione dell'informatica nel senso comune e che è completamente scorrelato dagli aspetti tecnologici. L'informatica non studia solo algoritmi per risolvere i problemi, ma si pone come obiettivo di studiare il *modo migliore, cioè più efficiente,* con cui tali problemi possono essere risolti. La vera sfida è, in altri termini, quella di individuare l'algoritmo più efficiente possibile per risolvere un dato problema.

A questo punto la sfida potrebbe essere chiara, ma si potrebbe comunque sospettare che la sfida sia, in un certo senso, artificiale e che la ricerca di un algoritmo ottimale sia solo una interessante questione speculativa ma dal limitato impatto pratico. Infatti, il lettore malizioso potrebbe subito argomentare che, oggigiorno, un computer può eseguire in pochi istanti milioni di operazioni. Pertanto, poca differenza fa, nella pratica, se un algoritmo richiede 1 milione di operazioni o solo 1000. In effetti, nel nostro esempio l'osservazione maliziosa è tutto sommato ben fondata. Tra un algoritmo che richiede *n x (n-1)* operazioni ed uno che richiede *n-1* operazioni, difficilmente concrete vere differenze nelle prestazioni possono emergere (con gli attuali elaboratori) sui numeri tipici di *n* che corrispondono ai residenti di un Comune. Anzi, anche un algoritmo ancora meno efficiente che richiedesse *n x n x n* (cioè n^3) operazioni sarebbe comunque una soluzione che, nella pratica, poco si differenzierebbe dall'algoritmo ottimo che impiega solo *n-1* operazioni. Più in generale, calcolatrice alla mano, si potrebbe argomentare che questo continua a valere per un qualunque algoritmo che impiega n^k operazioni, ove *k* è una qualunque costante. Algoritmi di questo tipo, che richiedono n^k operazioni, sono detti *polinomiali*. E quel famoso simbolo **P**, da cui la nostra analisi è partita, nulla altro sta ad indicare che l'insieme di tutti i problemi per i quali sia possibile esibire un algoritmo polinomiale.

Individuare il residente più anziano in un dato comune è dunque un problema che appartiene alla classe **P**. In effetti, molti problemi che emergono nella vita di tutti i giorni sono (per nostra fortuna) anch'essi membri della classe **P**, e dunque efficientemente risolubili su un calcolatore. Ad esempio, un altro illustre e ben conosciuto problema in **P** è quello di calcolare il percorso più breve, diciamo in auto, per andare dalla propria

abitazione ad una certa località di villeggiatura. Infatti, qualunque *navigatore* presente di serie in molte delle vetture più moderne è in grado di calcolare questo percorso ottimale in pochissimi secondi.

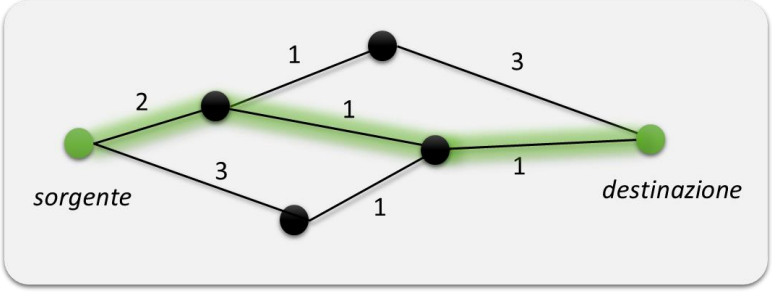

Figura 5: In termini informatici, il problema di calcolare il percorso più breve per andare da un punto "sorgente" ad un punto "destinazione" viene modellato come un problema su grafo. Gli archi del grafo rappresentano le vie di comunicazioni, e i nodi rappresentano gli "incroci" che consentono di muoversi da una via ad un'altra. Ogni arco viene etichettato con un numero che rappresenta la lunghezza (ad esempio in metri o chilometri) della via associata. Il problema è dunque quello di trovare il cammino che connette la sorgente e la destinazione e la cui somma dei pesi sugli archi sia la minima possibile. Ad esempio, il cammino minimo nel grafo riportato nella figura ha lunghezza complessiva pari a 4 (diciamo chilometri). Il problema può essere risolto con il noto algoritmo di Dijkstra, proposto nel 1956 dall'informatico olandese Edsger Dijkstra che lo pubblicò successivamente nel 1959.

Oltre P e la zona grigia

Abbiamo ora acquisito il primo concetto chiave per comprendere appieno il significato della **questione P=NP**. Abbiamo infatti compreso cosa sia **P**, e resta dunque da definire cosa sia **NP**. A tal fine ci servirà riflettere su un ulteriore concetto, che introduciamo anche in questo caso con un problema esemplificativo.

Il problema ci riporta indietro nel tempo, al VI secolo A.C., quando la scuola che faceva capo a Pitagora iniziò ad investigare le proprietà delle forme geometriche definendo il concetto di "proporzione media ed estrema", ribattezzato successivamente in epoca rinascimentale come "proporzione divina", e noto ai giorni nostri come "sezione aurea". Consideriamo due quantità *L* ed *R*, tali che *L* sia maggiore di *R*. Queste due lunghezze sono dette in *proporzione divina* tra di esse se il rapporto *L/R* coincide con il rapporto *(R+L)/L*. Un caso particolare è quando *R* è proprio l'unità di riferimento. Ad esempio, se *L* ed *R* sono delle lunghezze, assumiamo allora che *R* sia *1 metro*. In tal caso, la proporzione divina si riduce a verificare che il rapporto *L/1*, cioè *L* stesso, coincida con il rapporto *(1+L)/L*. Dunque, una lunghezza di *L* metri soddisfa tale proprietà se vale che *L=(1+L)/L*, ossia se *L x L = 1 + L*. Poniamoci ora proprio il problema che si posero i pitagorici: esiste una lunghezza L che soddisfa questa equazione?

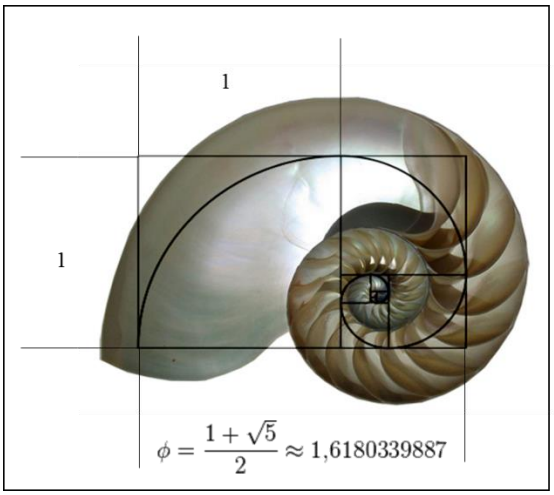

Figura 6: Un valore di L che soddisfa il rapporto della sezione aurea in effetti esiste, ed è il numero irrazionale riportato in figura (circa 1,618). Tale valore è detto anche "golden ratio" (il rapporto d'oro) in quanto riveste un ruolo estremamente rilevante in numerosi ambiti, dalle arti figurative alle scienze naturali. Molte delle proporzioni presenti in natura e riprese dagli artisti, in particolare rinascimentali e pre-rinascimentali, sono infatti riconducibili alla sezione aurea.

In termini astratti, il problema dell'esistenza di questa lunghezza *L* è un problema di esistenza di una soluzione ad una equazione in cui abbiamo utilizzato solo le operazioni di somma e prodotto. Più in generale, potremmo considerare più lunghezze incognite nel nostro problema, ad esempio L_1, L_2,…L_n, e porci domande del tipo: è vero che *esiste* un certo valore per L_1 tale che, *per tutti* i valori possibili di L_2, *esiste* un valore di L_3 (e via dicendo…) tale che una certa equazione, o anche un certo insieme di equazioni, abbia soluzione? Nell'ambito della matematica e dell'informatica teorica queste tipologie di domande sono state accuratamente analizzate, e si è dimostrato che sorprendentemente è possibile fornire una risposta automatizzata ad esse. In altri termini, è possibile definire un algoritmo che ci dica se la risposta ad una qualunque domanda del tipo sopra illustrato sia affermativa o negativa. Ma c'è di più. I ricercatori sono infatti riusciti a dimostrare un limite *invalicabile* nella complessità di un qualunque algoritmo risolutivo. Infatti, se denotiamo con *n* il numero di (lunghezze) incognite sommato al numero complessivo di operatori aritmetici coinvolti, allora dobbiamo aspettarci che qualunque algoritmo risolutivo richieda almeno 2^n operazioni, dunque un numero *esponenziale*. Questo vuol dire che il problema non è in **P**!

E questa volta c'è una brutta sorpresa in agguato del lettore malizioso che continui a pensare che in fondo, anche in questo caso, la potenza dei computer potrebbe comunque la differenza. Proviamo, infatti, a vedere cosa accade al crescere di *n*. Calcolatrice alla mano, vediamo che 2^{10} corrisponde a circa 1000 operazioni, che 2^{20} ci porta ad un milione di operazioni, e che con 2^{30} le operazioni diventano un miliardo. Tutto sommato già un numero consistente, eppure *n* è solo uguale a 30 in questo caso. Infatti, la velocità con cui 2^n cresce al crescere di *n* è sorprendente, e per $n=100$ già il valore ha superato una ragionevole stima del numero complessivo di atomi di cui è composta la nostra Terra. In altri termini, 2^{100} è un numero così grande da superare finanche la nostra stessa capacità di percezione ed immaginazione, ed un computer per calcolare una soluzione eseguendo questo mostruoso numero di operazioni impiegherebbe miliardi, miliardi, miliardi,…, e miliardi di anni. Eppure $n=100$ è ancora un numero molto piccolo, quasi un'inezia in termini informatici. Abbiamo quindi toccato con mano che se un problema non appartiene alla classe **P** e se per questo siamo in grado di dimostrare che il migliore algoritmo possibile è *esponenziale*, allora nessuna speranza concreta vi è di poterlo risolvere con un elaboratore. Dunque, la soluzione è teoricamente possibile, ma in pratica occorrerebbero svariati miliardi di anni di tempo per ottenere una risposta da un qualsivoglia moderno calcolatore. Diremo che problemi di questo tipo sono *intrinsecamente esponenziali*.

Conosciamo ora problemi polinomiali e problemi intrinsecamente esponenziali. Ma non tutto è bianco (polinomiale) o nero (esponenziale). Esiste infatti una zona grigia tra questi due estremi, ed è proprio qui che

si nasconde il secondo concetto base nella definizione della **questione P=NP**. Consideriamo il seguente semplice problema. Vogliamo organizzare una partita di calcio tra *n* giocatori, ed il nostro obiettivo è di definire le squadre nel modo più equilibrato possibile. Ad ogni giocatore, associamo dunque una valutazione delle sue abilità e cerchiamo di dividere gli *n* giocatori in modo che la somma delle abilità dei giocatori nella due squadre sia identica. Nonostante la sua apparente semplicità, non è facile trovare una soluzione esatta che sia efficiente. La soluzione naïve potrebbe essere semplicemente di considerare tutte le possibili coppie di squadre che si possono definire con *n* giocatori, per poi selezionare la coppia che darà luogo alla partita più equilibrata possibile. Tuttavia il numero di possibilità che il nostro algoritmo dovrebbe considerare è dell'ordine di 2^n, e sappiamo che questo è un tipo di algoritmo da cui tenerci accuratamente alla larga.

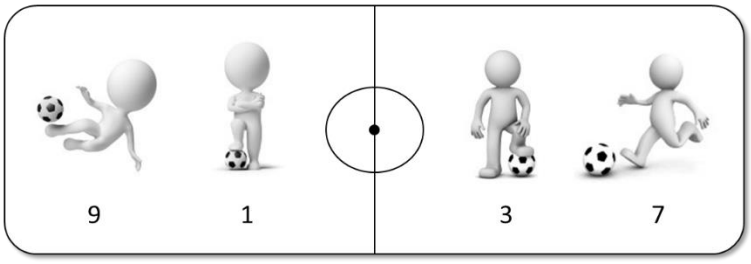

Figura 7: Un esempio del problema di costruzione di due squadre "equilibrate". Abbiamo quattro giocatori, di cui uno eccellente (che vale 9, in una scala da 1 a 10), uno buono (che vale 7), e due meno in gamba (che valgono 3 e 1, rispettivamente). In questo caso, è possibile formare due squadre perfettamente equilibrate, ciascuna delle quali ha un valore complessivo di 10.

Possiamo fare meglio di 2^n o questo è un limite intrinseco al problema? Per fare meglio avremmo bisogno di esibire un algoritmo più efficiente, cioè polinomiale. Per evidenziare che ciò non è possibile, avremmo bisogno di poter dimostrare che non può proprio esistere un algoritmo di questo tipo. Ma fino ad oggi, per questo semplice problema, noto come Number Partitioning, nessun ricercatore è mai riuscito né a trovare un algoritmo polinomiale, né a dimostrare che un algoritmo di questo tipo non possa proprio essere definito. Dunque, Number Partitioning vive in un limbo collocato tra i problemi polinomiali e quelli esponenziali, limbo che è chiamato nell'ambito della computer science come la classe dei problemi **NP**.

Questo limbo è, in effetti, abitato da moltissimi problemi che emergono nella vita quotidiana, spaziando dalla logistica, all'allocazione di risorse, alla definizione di protocolli criptografici, giusto per fare alcuni esempi. Dunque, Number Partitioning è in buona compagnia in **NP**. Per tutti questi problemi, allo stato attuale siamo solo in grado di esibire algoritmi la cui complessità è (almeno) 2^n, e dunque in sostanza non siamo in grado di risolverli in modo esatto per valori di *n* grandi o comunque "significativi". Su questa difficoltà si basano alcuni protocolli criptografici. Infatti, se per forzare un protocollo crittografico (che codifica i nostri acquisti via Internet) è necessario un algoritmo che impiega 2^n passi, ove *n* è ad esempio il numero di caratteri di una certa password, basta allora avere una password sufficientemente lunga al fine di rendere praticamente impossibile qualunque tentativo di violazione (che richiederebbe, come abbiamo visto, miliardi di anni per avere successo).

La questione P=NP

Siamo dunque arrivati al punto da poter rendere evidente il senso della **questione P=NP**. La questione è infatti di capire se tutti i problemi oggi collocati nel limbo di **NP** siano invece efficientemente risolubili.

Per maturare una propria visione della questione è utile ricordare un ulteriore comune denominatore dei problemi in **NP**: In primo luogo le soluzioni dei problemi in **NP** non sono mai oggetti "complicati" e possono, infatti, essere rappresentati efficientemente in un calcolatore. In secondo luogo, un qualunque problema in questa classe gode della proprietà che, se *magicamente* qualcuno ci proponesse una soluzione per una certa

istanza, allora noi saremmo in grado di verificarne l'esattezza in tempo polinomiale. Per comprendere meglio, ritorniamo sull'esempio della formazione delle squadre di calcio, ed assumiamo che qualcuno ci presenti già le due formazioni ed il nostro obiettivo sia solo quello di verificare che le due squadre siano perfettamente equilibriate. A tal fine, sarà sufficiente calcolare la somma dei valori dei giocatori della prima squadra e verificare che essa coincida con la somma dei valori dei giocatori della seconda squadra. Dunque, mentre la generazione di una soluzione è un problema complesso da affrontare, la *verifica* di una soluzione è *sempre* efficientemente affrontabile. Questa è una caratteristica intrinseca di tutti i problemi in **NP**, e che li differenzia da tutti i problemi che abbiamo definito esponenziali (cioè per i quali è effettivamente noto che non possa esistere un algoritmo risolutivo polinomiale). Infatti, per i problemi esponenziali, la soluzione stessa potrebbe (al crescere di *n*) diventare talmente complessa che anche la sola idea di rappresentarla in termini informatici potrebbe non essere concretamente realizzabile.

Sotto questo punto di vista la **questione P=NP** ci chiede se sia vero che ogni qual volta ho un meccanismo per verificare in tempo polinomiale una soluzione che mi venga fornita dall'esterno (e per la quale è garantita l'esistenza di una efficiente rappresentazione), allora avrò sempre la garanzia di poter anche definire un algoritmo polinomiale per calcolare una soluzione senza alcun aiuto esterno. Il fatto che calcolare (da zero) una soluzione sia intuitivamente molto più complesso che verificare la correttezza di una soluzione fornitaci da altri è il motivo per cui i ricercatori sono propensi a credere che in realtà i problemi in **NP** non possano essere risolti in tempo polinomiale (e dunque che **P** sia diverso da **NP**). Quando però si entra nel merito e si prova a dare una concretezza formale a questa intuizione, ci si scontra con complicatissime questioni tecniche, tant'è che pochissimi progressi sono stati fino ad ora compiuti verso la formalizzazione di una prova. Anzi, autorevoli ricercatori iniziano anche ad argomentare sulla possibilità che la direzione giusta su cui investigare possa essere invece quella, decisamente meno intuitiva, di un mondo in cui **P** coincide con **NP**.

In effetti, se qualche ricercatore dimostrerà, prima o poi, che **P** è diverso da **NP** non molto cambierebbe nella nostra vita quotidiana. Ma un mondo in cui **P** fosse uguale ad **NP** sarebbe abbastanza diverso da quello di oggi. Da una parte, varie metodologie crittografiche crollerebbero e vari sistemi oggi ritenuti "sicuri" diventerebbero di colpo facilmente violabili. D'altra, molti problemi oggi ritenuti intrattabili (poiché il miglior noto algoritmo è del tipo 2^n) diventerebbero facilmente risolubili (non più difficili della ricerca del più anziano residente in un Comune), avanzando incredibilmente la nostra capacità di automazione con straordinarie ripercussioni sulla nostra vita quotidiana. Un mondo in cui **P** fosse uguale ad **NP** sarebbe un mondo in cui i computer sono più "intelligenti" di come li possiamo oggi immaginare, non solo in termini di potenza di calcolo ma anche in termini della *qualità* intrinseca di ciò che essi sono in grado di calcolare. Ecco perché rispondere a questa domanda vale ben oltre il milione di dollari offerto dall'istituto Clay.

Questo chiarisce dunque il senso della **questione P=NP**, ma una curiosità ancora potrebbe rimanere. Molti lettori avranno constatato che ripartire i giocatori in due squadre equilibrate è un problema ben noto ai bambini quando si apprestano ad organizzare una partitella tra gli amici. In effetti, nessuno ha spiegato loro che il problema che si accingono ad affrontare è intrattabile (**NP**). Dunque, senza perdersi d'animo essi lo risolvono semplicemente eleggendo due capisquadra che scelgono a turno i giocatori. Il risultato è spesso molto soddisfacente. Come è possibile? Avranno scoperto una via "pratica" per rifuggire alle difficoltà concettuali che il limbo dei problemi informatici sembra invece porci? Forse sì, ma questa è un'altra storia.