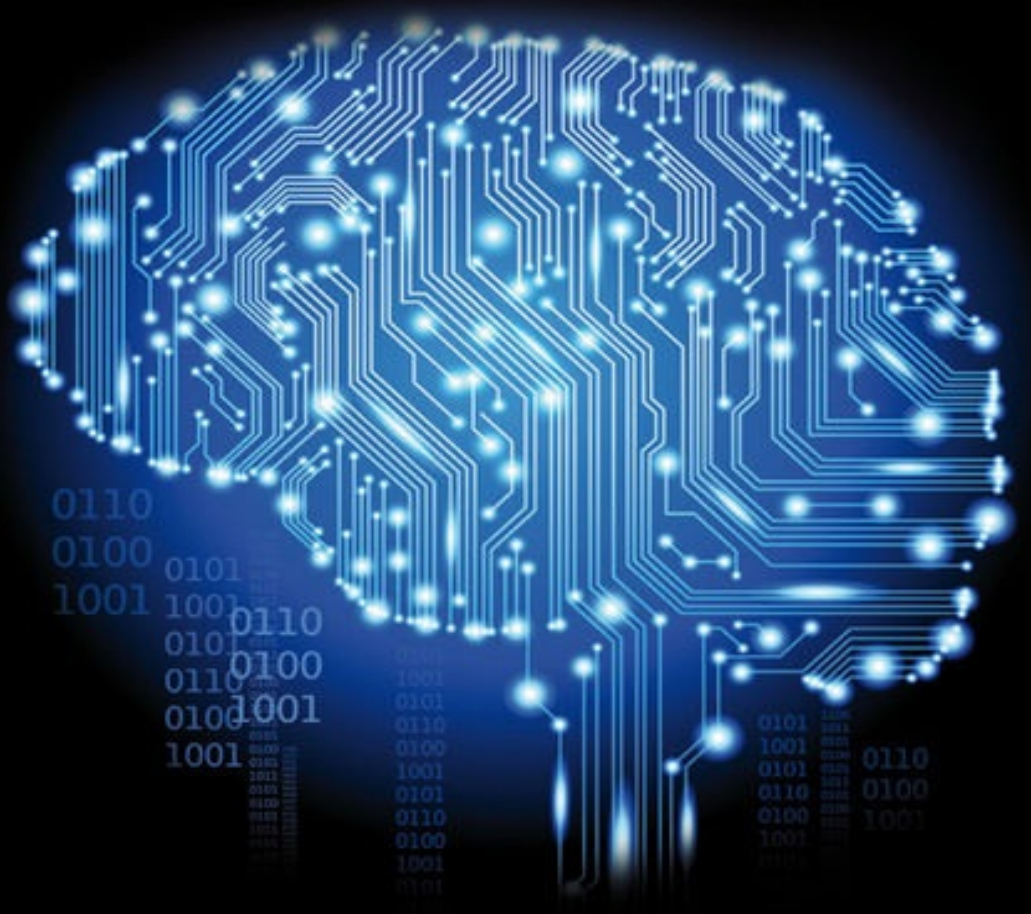
JOHN MACCORMICK

9 ALGORITMI
CHE HANNO CAMBIATO IL FUTURO



APOGEO

9 ALGORITMI CHE HANNO CAMBIATO IL FUTURO

John MacCormick

ISBN: 978.88.38788.91.8

www.apogeoeducation.com
www.maggiolieditore.it

e-mail: clienti.editore@maggioli.it

SOMMARIO

PREFAZIONE

> Per il mondo è arrivata un'epoca di dispositivi complessi, poco costosi ma di grande affidabilità; e qualcosa sicuramente ne verrà fuori.
>
> Vannevar Bush, "As We May Think", 1945

L'informatica sta trasformando la nostra società in modi profondi quanto i cambiamenti determinati dalla fisica e dalla chimica nei due secoli precedenti. In effetti, non c'è quasi aspetto della nostra vita che non sia già stato influenzato, o addirittura rivoluzionato, dalla tecnologia digitale. Data l'importanza dell'informatica per la società moderna, è un po' paradossale che ci sia una così scarsa conoscenza dei concetti fondamentali che hanno reso possibile tutto questo. Lo studio di questi concetti è il cuore della *computer science*, e questo nuovo libro di MacCormick è uno dei pochi che li presentino a un pubblico generale.

Questa relativa mancanza di considerazione dell'informatica come disciplina ha molte cause, ma una è che raramente viene insegnata nelle scuole superiori. Un'introduzione ad argomenti come quelli della fisica e della chimica di solito è considerata obbligatoria, ma spesso solo a livello universitario si studia l'informatica in sé. Poi, quello che spesso si insegna a scuola come "informatica" o "ICT" (information and communication technology) in genere è poco più che un addestramento pratico all'uso di pacchetti software. Non meraviglia che i ragazzi lo trovino noioso, e il loro entusiasmo naturale per l'uso della tecnologia informatica nel tempo libero e nelle comunicazioni è smorzato dall'impressione che la creazione di queste tecnologie manchi di profondità intellettuale. Questi sono i problemi che si pensa siano alla base della diminuzione del 50 per cento del numero degli studenti di informatica all'università [negli Stati Uniti] nell'ultimo decennio. Vista l'importanza cruciale della tecnologia digitale per la società moderna, non c'è mai stato un momento in cui fosse più importante riavvicinare tutti al fascino dell'informatica.

Nel 2008 ho avuto la fortuna di essere scelto per presentare la 180-esima serie di "conferenze natalizie" della Royal Institution, una tradizione iniziata da Michael Faraday nel 1826: era il primo anno che venivano dedicate al tema dell'informatica. Nel prepararle, ho passato molto tempo a pensare come spiegare l'informatica a un pubblico generico e mi sono reso conto che c'erano poche risorse, e praticamente nessun libro divulgativo, che potessero rispondere a quell'esigenza. Questo nuovo libro di MacCormick perciò è particolarmente benvenuto.

MacCormick ha compiuto un lavoro superbo, nel trasferire idee complesse dell'informatica a un pubblico generale. Molte di queste idee hanno una bellezza straordinaria e un'eleganza che basterebbero a renderle degne di attenzione. Un solo esempio: la crescita esplosiva delle attività commerciali sul Web è possibile solo grazie alla possibilità di trasmettere informazioni confidenziali (come i numeri di carta di credito) in modo sicuro via Internet. Quello di realizzare comunicazioni sicure su canali

“aperti” è stato considerato per decenni un problema intrattabile. Quando è stata trovata una soluzione, si è rivelata notevolmente elegante, e Mac-Cormick la spiega utilizzando analogie precise che non richiedono una precedente conoscenza informatica. Gemme come queste fanno di questo libro un contributo prezioso allo scaffale della letteratura sulla scienza, e lo raccomando caldamente.

Chris Bishop

Distinguished Scientist, Microsoft Research Cambridge
Vice President, The Royal Institution of Great Britain
Professor of Computer Science, University of Edinburgh

CAPITOLO 1

Introduzione: quali sono le idee straordinarie che i computer usano ogni giorno?

> Questo è un dono che ho… uno spirito follemente stravagante, pieno di forme, figure, fogge, oggetti, idee, percezioni, movimenti, rivoluzioni.
>
> William Shakespeare, *Pene d'amor perdute*

Come sono nate le grandi idee dell'informatica? Eccone un elenco (parziale):

- Negli anni Trenta del Novecento, prima ancora che venissero costruiti i primi computer digitali, un genio inglese fonda il campo dell'informatica, poi dimostra che certi problemi non potranno essere risolti da qualsiasi computer venga costruito in futuro, per quanto veloce, potente o ben progettato.
- Nel 1948, uno scienziato impiegato presso una compagnia telefonica pubblica un saggio che fonda il campo della teoria dell'informazione. Il suo lavoro consentirà ai computer di trasmettere un messaggio con precisione perfetta, anche quando la maggior parte dei dati fosse corrotta dalle interferenze.
- Nel 1956, un gruppo di accademici si riunisce a convegno a Dartmouth con l'obiettivo, esplicito e coraggioso, di fondare il campo dell'intelligenza artificiale. Dopo molti successi spettacolari e molte grandi delusioni, stiamo ancora aspettando che emerga un programma per computer davvero intelligente.
- Nel 1969, un ricercatore della IBM scopre un modo nuovo ed elegante di organizzare le informazioni in un database. La sua tecnica oggi è usata per immagazzinare e recuperare le informazioni per la maggior parte delle transazioni online.
- Nel 1974, ricercatori del laboratorio governativo inglese per le comunicazioni segrete scoprono un modo in cui due computer possono comunicare con sicurezza anche quando un altro computer può osservare tutti i segnali che passano dall'uno all'altro. I ricercatori sono vincolati dal segreto, ma per fortuna tre professori americani, in modo indipendente, scoprono ed estendono questa invenzione stupefacente che è alla base di tutte le comunicazioni sicure in Internet.
- Nel 1996, due studenti di Ph. D. della Stanford University decidono di lavorare insieme alla costruzione di un motore di ricerca per il Web. Qualche anno dopo avranno creato Google, il primo gigante digitale dell'era Internet.

Mentre godiamo della stupefacente crescita della tecnologia nel Ventunesimo secolo, è diventato impossibile usare un apparato di calcolo (che sia un cluster delle macchine più potenti sul pianeta o il dispositivo tascabile più alla moda) senza fare affidamento sulle

idee fondamentali dell'informatica, tutte nate nel Ventesimo secolo. Pensateci un momento: avete fatto qualcosa di particolarmente notevole, oggi? La risposta dipende dal vostro punto di vista. Forse avete effettuato una ricerca su un corpus di miliardi di documenti, scegliendo i due o tre più significativi per le vostre esigenze? Avete memorizzato o trasmesso molte migliaia di frammenti di informazione, senza nemmeno commettere un errore, nonostante le interferenze elettromagnetiche che disturbano tutti i dispositivi elettronici? Siete riusciti a concludere una transazione online, anche se molte migliaia di altri clienti stavano bombardando simultaneamente lo stesso server? Avete comunicato qualche informazione confidenziale (per esempio, il numero della vostra carta di credito) in modo sicuro su cavi che possono essere spiati da decine di altri computer? Avete sfruttato la magia della compressione per ridurre una fotografia di molti megabyte a una dimensione più compatta, adatta per l'invio in un messaggio di posta elettronica? Oppure, senza nemmeno pensarci, avete sfruttato l'intelligenza artificiale di un dispositivo mobile che corregge automaticamente quello che scrivete sulla sua minuscola tastiera?

Sono tutte imprese notevoli, che si basano sulle scoperte profonde elencate prima: la maggior parte degli utenti di computer sfruttano queste idee ingegnose molte volte ogni giorno, spesso senza nemmeno rendersene conto! Obiettivo di questo libro è spiegare questi concetti, le grandi idee dell'informatica che usiamo quotidianamente, al pubblico più ampio possibile. Ciascun concetto è spiegato senza dare per scontata alcuna conoscenza tecnica di informatica.

Algoritmi: i mattoni da costruzione del genio sulla punta delle dita

Fin qui, ho parlato di grandi "idee" dell'informatica, ma gli informatici parlano spesso di "algoritmi". Qual è la differenza fra un'idea e un algoritmo? Tanto per cominciare, che cosa *è* un algoritmo? La risposta più semplice è che un algoritmo è una ricetta precisa che specifica la sequenza esatta dei passi da compiere per risolvere un problema. Un ottimo esempio è un algoritmo che studiamo tutti da bambini a scuola: quello per sommare fra loro due numeri di molte cifre. Ne vedete un esempio qui sopra. L'algoritmo è costituito da una successione di passi che iniziano più o meno in questo modo: "Primo, somma l'ultima cifra di ciascuno dei due numeri, scrivi la cifra finale del risultato e riporta l'eventuale altra cifra alla colonna successiva a sinistra; secondo, somma le cifre della colonna successiva e la cifra eventualmente riportata dalla colonna precedente…" e via di questo passo.

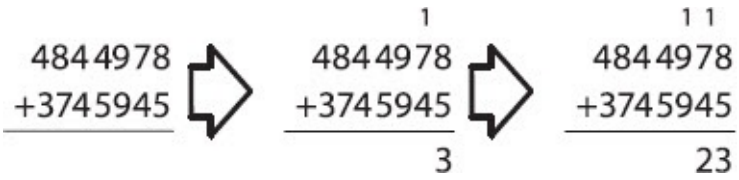


Figura 1.1 I primi due passi dell'algoritmo per l'addizione di due numeri.

Notate l'aria quasi meccanica di questi passi. È proprio una delle caratteristiche fondamentali di un algoritmo: ciascun passo deve essere assolutamente preciso, non deve richiedere intuizioni o congetture. Così, ciascuno di questi passi puramente meccanici può essere programmato in un computer. Un'altra caratteristica importante di un algoritmo è che funziona sempre, quali che siano gli *input*. L'algoritmo dell'addizione che abbiamo imparato a scuola ha effettivamente questa proprietà: non importa quali siano i due numeri

che si vogliono sommare, l'algoritmo alla fine darà la risposta corretta. Per esempio, ci vorrebbe di sicuro un po' di tempo, ma potreste tranquillamente usare questo algoritmo per sommare due numeri di mille cifre.

Forse vi sarà venuto qualche dubbio su questa definizione dell'algoritmo come una ricetta precisa, meccanica. Quanto precisa, esattamente, deve essere la ricetta? Quali operazioni fondamentali sono consentite? Per esempio, nell'algoritmo dell'addizione visto sopra, va bene dire semplicemente "somma le cifre", oppure dobbiamo in qualche modo specificare tutto l'insieme delle tabelline della somma per i numeri a una sola cifra? Questi particolari possono sembrare innocui o addirittura pedanti, ma nulla è più lontano dal vero: le risposte a queste domande sono proprio il nocciolo dell'informatica e hanno collegamenti anche con la filosofia, la fisica, le neuroscienze e la genetica. Le domande profonde su quello che davvero è un algoritmo si riducono tutte alla fine a una proposizione, nota con il nome di *tesi di Church-Turing*. Riprenderemo questo argomento nel Capitolo 10, dove discuteremo dei limiti teorici del calcolo e alcuni aspetti della tesi di Church-Turing. Nel frattempo, la nozione informale di algoritmo come ricetta molto precisa basterà perfettamente per i nostri scopi.

Ora sappiamo che cos'è un algoritmo, ma che cosa ha a che vedere con i computer? I computer debbono essere programmati con istruzioni molto precise, perciò, perché un computer possa risolvere per noi un problema particolare, dobbiamo sviluppare un algoritmo per quel problema. In altre discipline scientifiche, per esempio in matematica e in fisica, i risultati importanti spesso sono racchiusi in un'unica formula (esempi famosi sono il teorema di Pitagora, $a^2 + b^2 = c^2$, o la formula di Einstein $E = mc^2$). Le grandi idee dell'informatica invece in genere descrivono *come* risolvere un problema – utilizzando un algoritmo, ovviamente. Perciò l'obiettivo principale di questo libro è spiegare che cosa faccia del vostro computer il vostro genio personale: i grandi algoritmi che il vostro computer usa tutti i giorni.

Che cosa fa di un algoritmo un grande algoritmo?

Questo ci porta alla domanda insidiosa: quali algoritmi sono davvero "grandi"? L'elenco dei candidati potenziali è molto lungo, ma ho utilizzato alcuni criteri essenziali per restringere l'elenco per questo libro. Il primo criterio, e il più importante, è che l'algoritmo sia utilizzato dai normali utenti di computer ogni giorno. Il secondo criterio importante è che l'algoritmo deve risolvere un problema concreto, del mondo reale – un problema come comprimere un file o trasmetterlo in modo accurato su un canale rumoroso. Per i lettori che sanno già qualcosa di informatica, il riquadro nella pagina successiva spiega qualche conseguenza di questi due primi criteri.

Il primo criterio (l'uso quotidiano da parte di comuni utenti di computer) elimina gli algoritmi usati principalmente dai professionisti, come i compilatori e le tecniche di verifica dei programmi. Il secondo criterio (applicazione concreta a un problema specifico) elimina molti dei grandi algoritmi che sono fondamentali per i corsi universitari di informatica, fra cui gli algoritmi di ordinamento come quicksort, gli algoritmi dei graficome quello del cammino più breve di Dijkstra e strutture di dati come le tabelle *hash*. Questi algoritmi sono senza alcun dubbio grandi e soddisfano tranquillamente il primo criterio, dato che la maggior parte delle applicazioni eseguite dagli utenti comuni li usa costantemente. Ma questi sono algoritmi generici: possono

essere applicati a un gran numero di problemi diversi. In questo libro, ho scelto di concentrarmi su algoritmi per problemi specifici, perché per gli utenti comuni hanno una motivazione più evidente.

Qualche dettaglio in più sulla scelta degli algoritmi per questo libro. Non si dà per scontato che il lettore sappia qualcosa di informatica, ma se avete una formazione in questo campo questo riquadro spiega perché molti dei vostri algoritmi preferiti non sono trattati in queste pagine.

Il terzo criterio è che l'algoritmo sia relativo in primo luogo alla *teoria* informatica. Questo elimina le tecniche che riguardano l'hardware, come le CPU, i monitor e le reti, e fa porre meno l'accento sulla progettazione di infrastrutture come Internet. Perché scelgo di concentrarmi sulla teoria informatica? In parte perché la percezione generale dell'informatica è sbilanciata: in genere si pensa che riguardi soprattutto la programmazione (cioè il "software") e la progettazione di apparecchiature (cioè l'"hardware"), mentre molte delle idee più belle dell'informatica sono completamente astratte e non ricadono in nessuna di queste categorie. Mettendo l'accento su queste idee teoriche, la mia speranza è che ci siano più persone che cominciano a comprendere la natura dell'informatica come disciplina intellettuale.

Forse avrete notato che ho elencato criteri per eliminare potenziali "grandi" algoritmi, evitando la questione, molto più difficile, di definire direttamente la "grandezza". Per questo, mi sono basato sulla mia intuizione. Al centro di ogni algoritmo spiegato nel libro sta un "trucco" ingegnoso che fa funzionare il tutto. La presenza di un momento "aha", quando il trucco viene svelato, è quello che rende la spiegazione di questi algoritmi un'esperienza entusiasmante per me e, spero, anche per voi. Poiché userò la parola "trucco" molto spesso, devo precisare che non si tratta di trucchi fatti per ingannare; il senso è piuttosto quello dei "trucchi del mestiere" o anche dei trucchi dei prestigiatori: tecniche astute per raggiungere risultati che altrimenti sarebbero difficili o impossibili.

Perciò ho fatto affidamento sulla mia intuizione per scegliere quelli che credo siano i trucchi più ingegnosi del mondo informatico. Il matematico inglese G.H. Hardy, in una frase molto citata della sua *Apologia di un matematico*, in cui cercava di spiegare a tutti perché i matematici fanno quel che fanno, diceva: "La bellezza è il primo test: non c'è un posto permanente al mondo per la brutta matematica". Questo stesso test della bellezza vale per le idee teoriche alla base dell'informatica. Perciò il criterio definitivo per gli algoritmi presentati è quello che potrei definire il test della bellezza di Hardy: spero di essere riuscito a comunicare al lettore almeno parte della bellezza che sento presente in ciascuno di questi algoritmi.

Passiamo agli algoritmi specifici che ho scelto di presentare. L'impatto profondo dei motori di ricerca è forse l'esempio più ovvio di una tecnologia che influenza tutti gli utenti di computer, perciò nessuno si meraviglierà che abbia incluso alcuni degli algoritmi centrali della ricerca sul Web. Il Capitolo 2 descrive come i motori di ricerca utilizzino l'*indicizzazione* per trovare i documenti che soddisfano un'interrogazione, e il Capitolo 3 spiega *PageRank*, la versione originale dell'algoritmo utilizzato da Google per fare in modo che i documenti più pertinenti compaiano in testa all'elenco dei risultati.

Anche se non ci fermiamo a pensarci molto spesso, quasi tutti siamo almeno

consapevoli che i motori di ricerca usano alcune idee profonde dell'informatica per dare risultati di incredibile potenza. Invece, alcuni fra gli altri grandi algoritmi spesso vengono chiamati in causa senza nemmeno che l'utente se ne renda conto. La crittografia a chiave pubblica, descritta nel Capitolo 4, è uno di questi algoritmi. Ogni volta che visitate un sito sicuro (quando all'inizio dell'indirizzo si legge *https* anziché *http*) usate quell'aspetto della crittografia a chiave pubblica che è definito *scambio delle chiavi* per instaurare una sessione sicura. Il Capitolo 4 spiega come avvenga questo scambio di chiavi.

L'argomento del Capitolo 5, i codici a correzione d'errore, è un'altra classe di algoritmi che usiamo continuamente senza rendercene conto. In effetti, sono probabilmente la grande idea di uso più frequente di tutti i tempi. Questi codici consentono a un calcolare di individuare *e di correggere* gli errori nei dati memorizzati o trasmessi, senza dover ricorrere a una copia di riserva o alla ritrasmissione. Questi codici sono ovunque: sono utilizzati in tutte le unità a disco, in molte trasmissioni in rete, su Cd e Dvd, addirittura in alcune memorie di computer, ma svolgono il loro compito così bene che non ne siamo mai consapevoli.

Il Capitolo 6 costituisce un po' un'eccezione, perché è sgattaiolato nell'elenco delle grandi idee informatiche nonostante violi il primo criterio, cioè che gli i normali utenti lo usino ogni giorno. Il riconoscimento di forme (*pattern recognition*) è la classe di tecniche grazie alle quali i computer riconoscono informazioni estremamente variabili, come la scrittura a mano, il parlato e i volti. In effetti, nel primo decennio del ventunesimo secolo la maggior parte delle macchine di uso quotidiano non utilizzava queste tecniche ma nel 2011, mentre scrivo queste parole, l'importanza del riconoscimento di forme aumenta rapidamente: i dispositivi mobili con piccole tastiere virtuali richiedono la correzione automatica, i tablet debbono riconoscere gli input scritti a mano e tutti questi dispositivi (in particolare gli smartphone) sono sempre più spesso ad attivazione vocale. Alcuni siti web addirittura usano il riconoscimento di forme per stabilire che tipo di pubblicità mostrare ai loro utenti. Inoltre, sono un po' di parte, perché il riconoscimento di forme è il mio campo di ricerca. Così il Capitolo 6 descrive tre fra le tecniche di *pattern recognition* più interessanti ed efficaci: i classificatori del vicino più prossimo, gli alberi di decisione e le reti neurali.

Gli algoritmi di compressione, di cui si parla nel Capitolo 7, costituiscono un altro insieme di grandi idee che contribuiscono a fare di un computer un genio sulla punta delle nostre dita. Gli utenti a volte applicano la compressione direttamente, magari per risparmiare spazio su disco o per ridurre le dimensioni di una fotografia prima di spedirla per posta elettronica, ma la compressione viene usata ancora più spesso "in incognito": senza che lo sappiamo, quel che scarichiamo o carichiamo in rete può essere compresso per risparmiare occupazione di banda e i centri dati spesso comprimono i dati dei clienti per ridurre i costi. Quei 5 GB di spazio che il vostro provider di posta elettronica vi mette a disposizione probabilmente occupano molto meno di 5 GB sulle sue memorie di massa!

Il Capitolo 8 tratta alcuni fra gli algoritmi fondamentali dei database. L'accento cade sulle tecniche astute per garantire la *coerenza* (*consistency*), cioè per fare in modo che le relazioni in un database non si contraddicano mai a vicenda. Senza queste tecniche ingegnose, la maggior parte della nostra vita online (fra l'altro, gli acquisti online e l'interazione con reti sociali come Facebook) collasserebbe in un groviglio di errori.

Questo capitolo spiega quale sia realmente il problema della coerenza e come gli informatici lo risolvano senza sacrificare la formidabile efficienza che ci aspettiamo dai sistemi online.

Nel Capitolo 9 vedremo una delle indiscutibili gemme dell'informatica teorica: le firme digitali. La possibilità di "firmare" digitalmente un documento elettronico sembra a prima vista impossibile. Certo, penserete, una firma del genere deve essere costituita da informazioni digitali, che possono essere copiate senza fatica da chiunque voglia produrre una firma falsa. La risoluzione di questo paradosso è uno dei risultati più degni di nota dell'informatica.

Seguiremo una prospettiva del tutto diversa nel Capitolo 10: anziché descrivere un grande algoritmo che esiste già, parleremo di un algoritmo che *sarebbe* grande, se esistesse. Sorprendentemente, scopriremo che questo particolare grande algoritmo è impossibile. Il che fissa dei limiti assoluti alla capacità dei computer di risolvere problemi, e discuteremo brevemente le conseguenze di questo risultato per la filosofia e la biologia.

Nella conclusione, riuniremo alcuni fili comuni all'insieme dei grandi algoritmi e passeremo un po' di tempo a immaginare che cosa possa riservarci il futuro. Ci sono altri grandi algoritmi che ci aspettano, là fuori, o li abbiamo già trovati tutti?

È un buon momento per una parola di avvertimento sullo stile di questo libro. È essenziale, per ogni scritto scientifico, indicare chiaramente le fonti, ma le citazioni interrompono il flusso del testo e gli danno un sapore accademico. La leggibilità e l'accessibilità sono le mie priorità, in questo caso, perciò non ci sono citazioni nel corpo del testo. Tutte le fonti sono però chiaramente identificate nella sezione "Letture consigliate" alla fine del libro. Questa sezione indica anche ulteriori materiali che i lettori interessati possono utilizzare per scoprire di più sui grandi algoritmi dell'informatica.

Dato che abbiamo iniziato con gli avvertimenti, debbo dire anche che mi sono preso una piccola "licenza poetica" con il titolo. I nostri "nove algoritmi che hanno cambiato il futuro" sono, senza alcun dubbio, rivoluzionari, ma sono esattamente nove? Se ne può discutere, e tutto dipende da che cosa si considera essere un "singolo" algoritmo. Vediamo dunque da dove arriva il numero "nove". Escludendo introduzione e conclusione, il libro ha nove capitoli, ciascuno dei quali parla di algoritmi che hanno rivoluzionato un diverso tipo di attività, come la crittografia, la compressione o il riconoscimento di forme. Quindi i "nove algoritmi" in realtà si riferiscono a nove classi di algoritmi per svolgere queste nove attività di calcolo.

Perché ci dovrebbero interessare i grandi algoritmi?

Spero che questa rapida rassegna delle idee affascinanti che vedremo nelle prossime pagine vi abbia fatto venire voglia di tuffarvi e scoprire come funzionano realmente, ma forse vi starete ancora chiedendo: qual è l'obiettivo ultimo? Permettetemi dunque qualche breve osservazione sulla vera finalità di questo libro. Non è un manuale pratico: dopo averlo letto non sarete esperti di sicurezza informatica, né di intelligenza artificiale, né di altro ancora. È vero che magari ne ricaverete qualche conoscenza utile. Per esempio: avrete capito probabilmente meglio come verificare le credenziali dei siti web "sicuri" e dei pacchetti software "firmati"; saprete scegliere accortamente fra compressione con e senza perdita per fini diversi; e magari saprete usare con maggiore efficacia i motori di

ricerca, avendo capito alcuni aspetti delle loro tecniche di indicizzazione e ordinamento.

Questi però sono vantaggi relativamente secondari rispetto al vero obiettivo del libro. Dopo averlo letto *non* sarete utenti enormemente più competenti, ma *avrete* una comprensione più profonda della bellezza delle idee che utilizzate costantemente, giorno dopo giorno, su tutti i vostri dispositivi informatici.

Perché questa è una buona cosa? Consentitemi un'analogia. Decisamente non sono esperto di astronomia, anzi, sono piuttosto ignorante e vorrei saperne di più. Ogni volta che guardo il cielo di notte, però, quel poco che so di astronomia intensifica il piacere di quell'esperienza. In qualche modo, sapere che cosa sto guardando mi dà un senso di soddisfazione e di meraviglia. Spero ardentemente che, dopo aver letto questo libro, proverete qualche volta lo stesso senso di soddisfazione e di meraviglia mentre userete un computer. Riuscirete veramente ad apprezzare la più diffusa e la più imperscrutabile delle scatole nere del nostro tempo: il vostro personal computer, il genio sulla punta delle vostre dita.

CAPITOLO 2

L’indicizzazione nei motori di ricerca: trovare aghi nel pagliaio più grande del mondo

> Huck, stando qui potresti toccare quella fessura, da cui sono uscito, con una canna da pesca. Prova a vedere se ci riesci.
>
> Mark Twain, *Le avventure di Tom Sawyer*[1]

I motori di ricerca hanno cambiato profondamente la nostra vita. La maggior parte di noi effettua interrogazioni molte volte al giorno, ma di rado ci fermiamo a pensare come possano funzionare questi meravigliosi strumenti. La grande quantità di informazioni disponibili e la velocità e la qualità dei risultati ci sembrano ormai così normali che proviamo un senso di frustrazione se una domanda non trova risposta nel giro di qualche secondo. Tendiamo a dimenticare che ogni ricerca estrae un ago dal più grande pagliaio del mondo: il World Wide Web.

Il servizio superbo che ci forniscono i motori di ricerca non è solo il risultato dell’impiego di una grande quantità di tecnologia speciale per affrontare il problema. Certo, ognuna delle grandi aziende che gestisce uno dei motori di ricerca principali utilizza una rete internazionale di enormi centri dati, con migliaia di server e apparecchiature di rete all’avanguardia, ma tutto questo hardware sarebbe inutile senza i brillanti algoritmi necessari per organizzare e recuperare le informazioni che chiediamo. In questo capitolo e nel successivo, quindi, andremo a scoprire alcune delle gemme algoritmiche che lavorano per noi ogni volta che effettuiamo una ricerca sul Web. Come vedremo, due dei compiti principali di un capitolo sono l’identificazione di corrispondenze (*matching*) e l’ordinamento (*ranking*). Questo capitolo si occupa di una bella tecnica di identificazione di corrispondenze, il trucco delle metaparole. Nel prossimo capitolo, ci dedicheremo al problema dell’ordinamento ed esamineremo il famoso algoritmo PageRank di Google.

Corrispondenza e ordinamento

Sarà utile iniziare con uno sguardo generale su quel che succede quando inviamo una richiesta di ricerca sul Web, una interrogazione. Ci saranno, come abbiamo già detto, due fasi principali: l’identificazione di corrispondenze e l’ordinamento. Nella pratica, i motori di ricerca combinano le due attività in un processo unico, per ragioni di efficienza, ma le due fasi sono concettualmente distinte, perciò ipotizzeremo che l’identificazione delle corrispondenze sia completata prima che inizi l’ordinamento. La Figura 2.1 illustra un esempio, dove l’interrogazione è “London bus timetable” o “Orari autobus Londra”. La fase di identificazione delle corrispondenze risponde alla domanda “quali pagine web

soddisfano l’interrogazione?” – in questo caso, tutte le pagine che citano gli orari degli autobus londinesi.

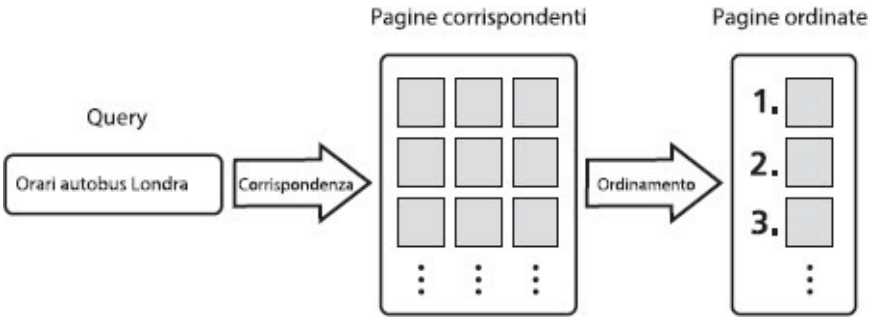


Figura 2.1 Le due fasi della ricerca nel Web: identificazione di corrispondenze e ordinamento. È possibile che ci siano migliaia o milioni di corrispondenze dopo la prima fase, che poi vanno ordinate per pertinenza decrescente nella seconda fase.

Ma per molte interrogazioni dei motori di ricerca, nella realtà, possono esistere centinaia, migliaia o addirittura milioni di *hit*, di pagine che corrispondono al criterio di ricerca, e gli utenti dei motori in genere preferiscono esaminare solo una manciata di risultati, magari cinque, dieci al massimo. Perciò un motore di ricerca deve essere in grado di scegliere, fra un numero molto grande di risultati, i migliori. Un buon motore di ricerca non solo estrarrà i risultati migliori, ma li presenterà anche nell’ordine più utile, cioè con la pagina più adatta in prima posizione, seguita dalla successiva in ordine di pertinenza e così via.

Il compito di selezionare i pochi risultati migliori nell’ordine giusto è il compito dell’ordinamento o *ranking*, seconda e cruciale fase che segue quella iniziale di ricerca di corrispondenze. In un campo in cui la concorrenza è micidiale, i motori di ricerca vivono o muoiono a seconda della qualità dei loro sistemi di ordinamento. Nel 2002, la quota di mercato dei tre maggiori motori di ricerca negli USA era all’incirca uguale: Google, Yahoo e MSN ricevevano ciascuno poco meno del 30% di tutte le interrogazioni di ricerca del paese (MSN poi ha cambiato nome ed è diventato Live Search e infine Bing). Nell’arco di pochi anni, Google è riuscito ad aumentare drasticamente la sua quota di mercato, retrocedendo Yahoo e MSN a quote inferiori al 20%. Quasi tutti pensano che l’ascesa di Google alla cima del settore sia stata merito dei suoi algoritmi di ordinamento, perciò non si esagera dicendo che i motori di ricerca vivono o muoiono a seconda della qualità dei loro sistemi di ordinamento. Però, lo abbiamo già detto, degli algoritmi di ordinamento ci occuperemo nel prossimo capitolo; per ora, ci concentriamo sulla fase dell’identificazione di corrispondenze.

AltaVista: il primo algoritmo di matching alla scala del Web

Dove inizia la nostra storia? Verrebbe spontaneo dire con Google, il maggiore successo tecnologico degli inizi del ventunesimo secolo, ma sarebbe sbagliato. La storia degli inizi di Google, come progetto di ricerca per il Ph. D. di due studenti della Stanford University è tanto appassionante quanto sorprendente. Larry Page e Sergey Brin sono riusciti nel 1998 a mettere insieme un assortimento eterogeneo di macchine per realizzare un nuovo tipo di motore di ricerca: meno di dieci anni dopo la loro azienda era diventata il più grande gigante digitale dell’era di Internet.

Ma l’idea delle ricerche sul Web circolava già da anni. Fra le prime offerte

commerciali ci furono Infoseek e Lycos (lanciati entrambi nel 1994) e AltaVista, lanciato nel 1995. Per qualche anno, alla metà degli anni Novanta, AltaVista fu il re dei motori di ricerca. A quel tempo ero specializzando in informatica e ricordo bene di essere rimasto colpito dall'ampiezza dei risultati di AltaVista: per la prima volta un motore di ricerca aveva indicizzato completamente tutto il testo di tutte le pagine del Web, non solo, ma restituiva i risultati in un batter d'occhi. Il nostro viaggio alla comprensione di questo sensazionale risultato tecnologico inizia con un concetto (letteralmente) antico: l'indicizzazione.

Buona, vecchia indicizzazione

Il concetto di *indice* è l'idea fondamentale alla base di ogni motore di ricerca, ma i motori di ricerca non hanno inventato gli indici: si tratta, in effetti, di un'idea vecchia quanto la scrittura stessa. Gli archeologi hanno scoperto una biblioteca di un tempio babilonese di 5000 anni fa che catalogava per argomento le sue tavolette in cuneiforme, perciò l'idea dell'indicizzazione può ambire a buon diritto al titolo di più vecchia fra le idee utili per l'informatica.

Oggi la parola "indice" di solito indica l'indice analitico, una sezione alla fine di un testo di saggistica o di consultazione: tutti i concetti che si potrebbe voler trovare in quel libro sono elencati in un ordine ben preciso (solitamente in ordine alfabetico) e a ciascun concetto segue un elenco delle posizioni (di solito numeri di pagina) in cui si parla di quel concetto. Un libro di zoologia può avere una voce di indice del tipo "ghepardo 124, 156", che significa che la parola "ghepardo" si trova alle pagine 124 e 156. (Come piccola curiosità, potreste cerca la parola "indice" nell'indice di questo libro. Se non ci sono errori, vi rimanderà a questa pagina.)

L'indice di un motore di ricerca funziona nello stesso modo dell'indice di un libro: le "pagine" sono pagine del World Wide Web e i motori assegnano un numero di pagina diverso a ogni singola pagina web. (Sì, ci sono tantissime pagine, parecchi miliardi, ma i computer sono bravissimi nel trattare numeri molto grandi.) La Figura 2.2 presenta un esempio che renderà tutto più concreto. Supponiamo che il World Wide Web sia fatto solo di tre pagine web, a cui sono assegnati i numeri 1, 2, 3.

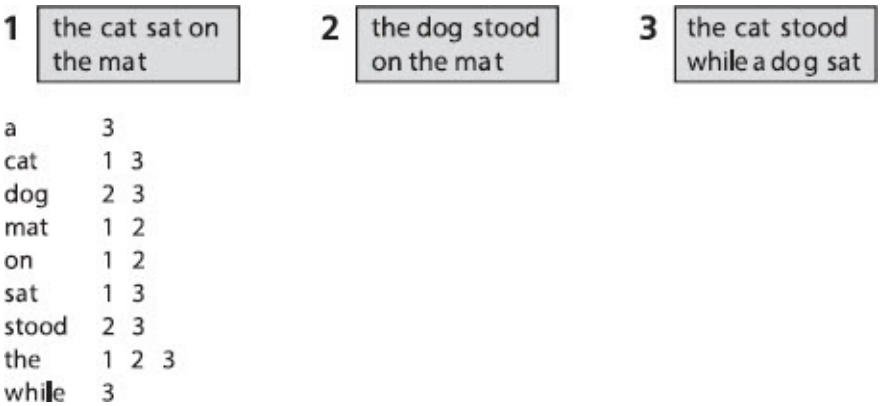


Figura 2.2 In alto: un World Wide Web immaginario costituito da tre pagine solamente, identificate dai numeri 1, 2, 3. In basso: un indice molto semplice, con relativi numeri di pagina.

Un computer può costruire un indice di queste tre pagine innanzitutto stendendo un elenco di tutte le parole che compaiono nelle pagine e poi ordinando alfabeticamente l'elenco. L'*elenco delle parole* (*word list*) in questo caso particolare sarà "a, cat, dog, mat,

on, sat, stood, the, while”. Poi il computer esplorerà le pagine parola per parola e per ogni parola prenderà nota del numero della pagina in cui compare. Dal risultato finale, in figura, si può vedere subito che la parola “cat” è presente nelle pagine 1 e 3, ma non nella 2 e che la parola “while” appare solo nella pagina 3.

Con questo metodo molto semplice, un motore di ricerca può già dare una risposta a molte domande semplici. Per esempio, supponiamo che inseriate l’interrogazione cat. Il motore di ricerca può rapidamente saltare al lemma cat nel suo elenco di parole. (Dato che l’elenco delle parole è in ordine alfabetico, un computer può trovare rapidamente un lemma, come un essere umano può trovare rapidamente un lemma in un dizionario.) Appena trova il lemma cat, il motore di ricerca può subito fornirvi l’elenco delle pagine in cui compare, in questo caso 1 e 3. I motori di ricerca moderni arricchiscono la presentazione dei risultati con un frammento di ciascuna pagina, ma in genere trascureremo questi particolari e ci concentreremo su come fa il motore a sapere quali siano i numeri di pagina che sono *hit* per l’interrogazione avanzata.

Un altro esempio semplicissimo: verifichiamo la procedura per l’interrogazione dog. In questo caso, il motore di ricerca trova rapidamente la voce dog e restituisce i risultati 2 e 3. Ma che cosa succede se l’interrogazione contiene più parole, per esempio cat dog? Questo significa che volete trovare pagine che contengano sia la parola “cat” sia la parola “dog”. Ancora una volta, è una cosa facile per il motore di ricerca, grazie all’indice esistente. Prima cerca le due parole singolarmente, per sapere in quali pagine siano presenti e ottiene le risposte 1, 3 per “cat” e 2, 3 per “dog”. Poi può rapidamente esaminare i due elenchi di risultati, per vedere se ci siano numeri di pagina uguali. In questo caso le pagine 1 e 2 non soddisfano questo requisito, ma la pagina 3 è presente in entrambi gli elenchi, perciò la risposta finale è costituita da un solo risultato, la pagina 3. Una strategia molto simile si applica per interrogazioni con più di due parole. Per esempio, l’interrogazione cat the sat restituisce come risultati le pagine 1 e 3, che sono gli elementi comuni agli elenchi per “cat” (1, 3), “the” (1, 2, 3) e “sat” (1, 3).

Fin qui, sembrerebbe che costruire un motore di ricerca sia molto facile: la più semplice possibile delle tecnologie di indicizzazione sembra funzionare bene anche per interrogazioni con più parole. Purtroppo, però, questa impostazione semplice è del tutto inadeguata per i motori di ricerca moderni: le ragioni sono diverse, ma per ora ci concentreremo su un solo problema, quello delle *interrogazioni a sintagma*. Sono le interrogazioni che cercano un sintagma preciso, un’espressione complessa e non semplicemente le occorrenze di varie parole in qualsiasi punto di una pagina. Nella maggior parte dei motori di ricerca, queste interrogazioni si indicano con l’uso delle virgolette: così, per esempio, l’interrogazione “cat sat” ha un significato molto diverso dall’interrogazione cat sat. L’interrogazione cat sat cerca pagine che contengano le parole “cat” e “sat” in qualsiasi punto e in qualsiasi ordine, mentre “cat sat” cerca pagine che contengano la parola “cat” seguita immediatamente dalla parola “sat”. Nel nostro banale esempio con tre sole pagine, cat sat ha come risultati le pagine 1 e 3, ma “cat sat” restituisce un solo risultato, la pagina 1.

In che modo un motore di ricerca può eseguire efficacemente una interrogazione a sintagma? Restiamo all’esempio di “cat sat”. Sembrerebbe che il primo passo sia fare la stessa cosa che si fa per una normale interrogazione a più parole cat sat: recuperare

dall’elenco delle parole l’elenco delle pagine in cui ciascuna parola occorre, in questo caso 1, 3 per “cat” e 1, 3 anche per “sat”. Ma a questo punto il motore è bloccato. Sa per certo che entrambe le parole sono presenti nelle pagine 1 e 3, ma non ha modo di dire se vi compaiano una subito dopo l’altra e nell’ordine giusto. Si potrebbe pensare che il motore possa tornare a consultare le pagine web originali per stabilire se vi compaia o meno quell’espressione precisa. In effetti sarebbe una soluzione, ma molto, molto inefficiente, perché comporta che si vadano a rileggere tutti i contenuti di tutte le pagine che *potrebbero* contenere quell’espressione, e il numero di quelle pagine potrebbe essere enorme. Non dimenticate che stiamo ragionando su un esempio estremamente piccolo, con tre sole pagine, mentre un motore di ricerca reale deve dare risultati corretti per decine di miliardi di pagine web.

Il trucco della posizione della parola

La soluzione a questo problema è la prima idea davvero ingegnosa che ha contribuito al buon funzionamento dei motori di ricerca moderni: l’indice non deve conservare solo i *numeri* di pagina, ma anche le *posizioni* all’interno delle pagine. Queste posizioni non sono nulla di misterioso: indicano semplicemente la posizione di una parola all’interno della pagina, per cui la terza parola ha la posizione 3, la ventinovesima la posizione 29 e via elencando. Il nostro intero insieme di dati costituito dalle tre pagine è mostrato nella Figura 2.3, dove sono state aggiunte le posizioni delle parole. Sotto si vede l’indice costruito tenendo conto anche della posizione delle parole. Vediamo un paio di esempi, per essere sicuri di aver capito bene. La prima riga dell’indice è “a 3-5” e significa che la parola “a” si presenta una sola volta nell’insieme dei dati, cioè nella pagina 3, e che è la quinta parola di quella pagina. La riga più lunga dell’indice è “the 1-1 1-5 2-1 2-5 3-1”, che dà le posizioni esatte di tutte le occorrenze della parola “the” nell’insieme dei dati: due volte nella pagina 1 (alle posizioni 1 e 5), due volte nella pagina 2 (alle posizioni 1 e 5) e una volta nella pagina 3 (alla posizione 1).

Ricordiamo perché abbiamo introdotto le posizioni delle parole nelle pagine: per risolvere il problema di come rispondere in modo efficiente a una interrogazione a sintagma. Vediamo allora dove ci porta il nuovo indice. Riprendiamo la stessa interrogazione di prima, “cat sat”. I primi passi sono gli stessi del vecchio indice: estraiamo le posizioni delle singole parole dall’indice, cosicché per “cat” otteniamo 1-2, 3-2 e per “sat” otteniamo 1-3, 3-7. Fin qui tutto bene: sappiamo che gli unici risultati possibili per l’interrogazione “cat sat” sono le pagine 1 e 3. Ma, come prima, ancora non siamo sicuri che in quelle pagine si presenti proprio quella espressione precisa – è possibile che le due parole siano nella pagina, ma non una vicina all’altra oppure non nell’ordine giusto. Per fortuna, è facile verificarlo adesso con le informazioni sulla posizione. Concentriamoci sulla pagina 1, per cominciare. Dalle informazioni dell’indice, sappiamo che “cat” compare nella posizione 2 della pagina 1 (è questo il significato di 1-2), e sappiamo che “sat” compare nella posizione 3 della pagina 1 (è questo il significato di 1-3). Ma se “cat” è nella posizione 2 e “sat” è nella posizione 3, allora sappiamo che “sat” compare subito dopo “cat” (perché 3 è il successore immediato di 2) e perciò l’esatta espressione che stiamo cercando, “cat sat”, deve essere presente in questa pagina, a partire dalla posizione 2!

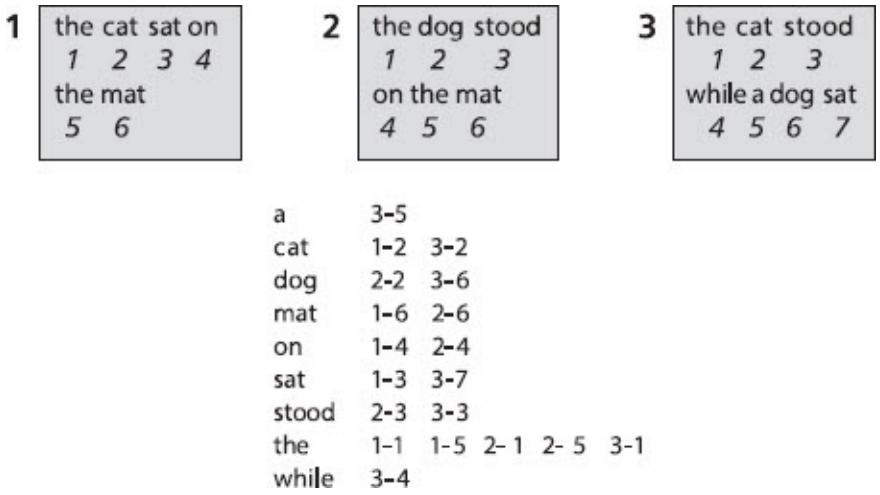


Figura 2.3 In alto: le nostre tre pagine web a cui è stata aggiunta l'indicazione delle posizioni delle parole. In basso: un nuovo indice che include sia i numeri delle pagine, sia le posizioni delle parole nella pagina.

So che sono un po' noioso, ma il motivo per esaminare questo esempio in tutti i suoi particolari è capire *esattamente* quali informazioni si usano per arrivare alla risposta. Notate che abbiamo trovato un risultato per il sintagma "cat sat" esaminando solo le informazioni dell'indice (1-2, 3-2 per "cat" e 1-3, 3-7 per "sat") e non le pagine web originali. Questo è un aspetto fondamentale, perché abbiamo dovuto esaminare solo due voci dell'indice, invece di rileggere tutte le pagine che potevano essere risultati validi, e potrebbero esserci letteralmente milioni di pagine simili nel caso di un motore di ricerca reale che cerca di rispondere a una interrogazione a sintagma reale. Per riassumere: l'inclusione nell'indice della posizione delle parole nelle pagine ci ha permesso di rispondere a un'interrogazione a sintagma esaminando solo un paio di righe nell'indice, invece di leggere per esteso un gran numero di pagine web. Questo semplice trucco della posizione delle parole è uno degli elementi chiave per il funzionamento dei motori di ricerca!

In realtà, non abbiamo ancora finito di elaborare l'esempio "cat sat". Abbiamo elaborato le informazioni per la pagina 1, ma non quelle per la pagina 3. Il ragionamento però è simile: vediamo che "cat" compare nella posizione 2, e "sat" nella posizione 7, perciò non possono essere vicine, perché 7 non è il successore immediato di 2. Perciò sappiamo che la pagina 3 *non* è un risultato valido per l'interrogazione a sintagma "cat sat", anche se *è* un risultato valido per la ricerca a più parole cat sat.

Incidentalmente, il trucco della posizione delle parole è importante anche per altri motivi. Per esempio, consideriamo il problema di trovare parole che siano vicine l'una all'altra. In certi motori di ricerca, lo si può fare inserendo la parola chiave NEAR nell'interrogazione; AltaVista offriva questo strumento sin dagli inizi e lo fa ancora nel momento in cui scrivo. Come esempio, supponiamo che in un certo specifico motore di ricerca l'interrogazione cat NEAR dog trovi le pagine in cui la parola "cat" occorre a distanza di non più di cinque parole dalla parola "dog". Come possiamo rispondere in modo efficiente a questa interrogazione per il nostro insieme di dati? Utilizzando le posizioni delle parole è facile. La voce di indice per "cat" è 1-2, 3-2 e la voce di indice per "dog" è 2-2, 3-6, perciò vediamo subito che la pagina 3 è l'unico risultato possibile. Sulla pagina 3, poi, "cat" occorre nella posizione 2, mentre "dog" è nella posizione 6, perciò la distanza fra le due parole è 6 – 2, cioè 4. Quindi "cat" *occorre* a una distanza non superiore alle cinque parole da "dog" e la pagina 3 è un risultato valido per

l’interrogazione cat NEAR dog. Anche in questo caso, notate l’efficienza con cui è stato possibile rispondere: non c’è stato bisogno di leggere i contenuti effettivi di alcuna pagina web, ma sono stati consultate solo due voci dell’indice.

In realtà, le interrogazioni NEAR non sono molto importanti per gli utenti dei motori di ricerca, nella pratica: quasi nessuno le usa e la maggior parte dei motori di ricerca non le consente nemmeno. Nonostante questo, la possibilità di eseguire interrogazioni NEAR è davvero fondamentale per i motori di ricerca reali, perché i motori eseguono costantemente, dietro le quinte, interrogazioni di questo tipo. Per capire il perché, dobbiamo prima dare uno sguardo a uno degli altri grandi problemi dei motori di ricerca moderni: il problema dell’ordinamento.

Ordinamento e prossimità

Fin qui ci siamo concentrati sulla fase della corrispondenza: il problema di trovare in modo efficiente tutti i risultati validi, gli *hit*, per una interrogazione data. Ma, come abbiamo detto, la seconda fase, l’ordinamento, è essenziale per un motore di ricerca di qualità: è la fase in cui si selezionano pochi risultati ottimali per presentarli all’utente,

Analizziamo un po’ meglio il concetto di ordinamento o *ranking*. Da che cosa dipende realmente la posizione di una pagina in questa classifica? La domanda qui non è “Questa pagina *soddisfa* l’interrogazione?” bensì “Questa pagina è *rilevante* per l’interrogazione?”. Gli informatici usano la parola “rilevanza” o “pertinenza” per parlare di quanto una data pagina sia adeguata o utile come risposta a una interrogazione particolare.

Concretamente, supponiamo siate interessati a sapere che cosa causa la malaria, e che inseriate in un motore di ricerca l’interrogazione malaria causa. Per semplificare le cose, supponiamo che ci siano due soli risultati per l’interrogazione, le due pagine mostrate nella figura della pagina seguente. Date un’occhiata a queste pagine. Dovrebbe esservi subito chiaro, in quanto esseri umani, che la pagina 1 parla effettivamente della causa della malaria, mentre la pagina 2 sembra la descrizione di qualche campagna militare in cui semplicemente capita, per puro caso, che vengano usate entrambe le parole “causa” e “malaria”. Perciò la pagina 1 è senza alcun dubbio più “pertinente” per la ricerca *malaria causa* rispetto alla pagina 2. Ma i computer non sono esseri umani e non esiste un modo facile per far “capire” a un computer gli argomenti delle due pagine, perciò potrebbe sembrare impossibile che un motore di ricerca riesca a ordinare correttamente i due risultati.

alla	1-19	
...		
causa	1-6	2-2
...		
malaria	1-8	2-19
...		
zanzara	2-15	

Figura 2.4 In alto: due esempi di pagine web che citano la malaria. In basso: parte dell'indice costruito a partire dalle due pagine in alto.

In questo caso, però, esiste un modo semplice per ottenere l'ordinamento giusto. Statisticamente, le pagine in cui le parole dell'interrogazione occorrono *vicino* l'una all'altra è più probabile siano pertinenti, rispetto alle pagine in cui le parole dell'interrogazione sono molto distanti. Nell'esempio della malaria, vediamo che le parole "malaria" e "causa" sono a distanza di due sole parole nella pagina 1, ma sono separate da 14 parole nella pagina 2. (Ricordate che il motore di ricerca può stabilirlo rapidamente esaminando solo le voci dell'indice, senza dover tornare a leggere le pagine web stesse.) Perciò, anche se il computer non "capisce" realmente l'argomento dell'interrogazione, può *congetturare* che la pagina 1 sia più pertinente della pagina 2, perché le parole dell'interrogazione sono molto più vicine fra loro nella pagina 1 che non nella pagina 2.

Per riassumere: gli essere umani non usano molto le interrogazioni NEAR, ma i motori di ricerca usano costantemente le informazioni sulla prossimità per migliorare le loro classifiche, e il motivo per cui possono farlo in modo efficiente è l'uso del trucco della posizione delle parole.

Sappiamo che i Babilonesi utilizzavano l'indicizzazione 5000 anni prima che nascessero i motori di ricerca. Questi ultimi non hanno inventato neanche il trucco della posizione delle parole: si tratta di una tecnica ben nota, utilizzata in altri tipi di recupero delle informazioni prima che Internet facesse il suo ingresso sulla scena.

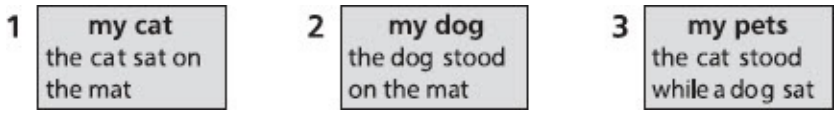


Figura 2.5 Un insieme di pagine web che hanno un titolo e un corpo.

Nel prossimo paragrafo invece parleremo di un nuovo trucco che a quanto pare è stato inventato proprio dai progettisti di motori di ricerca, il *trucco della metaparola*. Un uso molto abile di questo trucco e di varie altre idee collegate ha contribuito a catapultare AltaVista sulla vetta del mondo dei motori di ricerca verso la fine degli anni Novanta.

Il trucco della metaparola

Fin qui abbiamo usato esempi di pagine web molto semplici ma, come probabilmente saprete, le pagine web in genere sono molto strutturate, con titoli, intestazioni,

collegamenti e immagini, mentre fin qui le abbiamo trattate come semplici elenchi di parole. Ora andiamo a scoprire come i motori di ricerca tengono conto anche della struttura delle pagine web. Per mantenere il massimo livello di semplicità possibile, introdurremo solo un elemento di strutturazione: consentiremo alle nostre pagine di avere un *titolo* (*title*) in testa, seguito dal *corpo* (*body*) della pagina. La Figura 2.5 mostra il nostro esempio di tre pagine che ora hanno anche un titolo.

In realtà, per analizzare la struttura delle pagine nello stesso modo dei motori di ricerca, dobbiamo sapere qualcosa di più su come le pagine sono scritte. Le pagine web sono composte in un linguaggio particolare, che consente ai browser di visualizzarle con una presentazione elegante. (Il linguaggio più usato si chiama HTML, ma come sia fatto nei particolari HTML non è importante per noi qui.) Le istruzioni di formattazione per intestazioni, titoli, collegamenti, immagini e così via vengono scritte utilizzando parole speciali definite *metaparole* (*metaword*). Per esempio, la metaparola utilizzata per indicare l'inizio del titolo di una pagina web può essere <titleStart> e la metaparola per indicare la fine del titolo può essere <titleEnd>. Analogamente, si può indicare l'inizio del corpo della pagina con <bodyStart> e concluderlo con <bodyEnd>. Non fatevi confondere dai simboli "<" e ">": sono presenti su quasi tutte le tastiere e spesso vengono indicati con il loro significato matematico di "minore" e "maggiore". Qui però non hanno proprio nulla a che vedere con la matematica, sono semplicemente simboli che servono a contrassegnare le metaparole come qualcosa di diverso dalle normali parole di una pagina web.

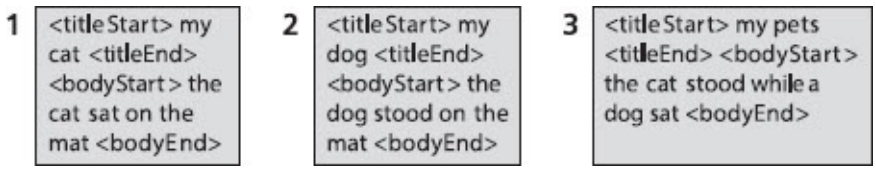


Figura 2.6 Lo stesso insieme di pagine web della figura precedente, ora presentate come potrebbero essere scritte con l'uso di metaparole, anziché come verrebbero visualizzate in un browser.

Date un'occhiata alla Figura 2.6, che mostra esattamente lo stesso contenuto della figura precedente, ma ora come sarebbe effettivamente scritto nelle pagine web, anziché come verrebbe visualizzato in un browser. La maggior parte dei browser consente di esaminare i contenuti grezzi di una pagina web con un comando da menu come "Visualizza sorgente" o "Origine": vi consiglio di fare una prova alla prima occasione. (Notate che le metaparole usate qui, come <titleStart> e <titleEnd> sono di mia invenzione, esempi facilmente riconoscibili per facilitare la comprensione. Nel vero HTML, le metaparole sono denominate *tag* o *marcatori*. I tag che indicano inizio e fine di un titoli in HTML sono <title> e </title>: cercateli dopo aver usato il comando "Visualizza sorgente".)

Quando si costruisce un indice, è semplice includere tutte le metaparole. Non servono nuovi trucchi: si memorizzano le posizioni delle metaparole esattamente come si fa per le parole normali. Le Figure 2.7 e 2.8 mostrano l'indice costruito a partire dalle tre pagine web con le loro metaparole. Date un'occhiata alle figure e sincerateVi che non sta succedendo niente di misterioso. Per esempio, la voce "mat" indica 1-11, 2-11, il che significa che "mat" è l'undicesima parola della pagina 1 e anche l'undicesima parola della pagina 2. Le metaparole sono trattate nello stesso modo, perciò la voce di indice per "<titleEnd>", con 1-4, 2-4, 3-4, significa che "<titleEnd>" è la quarta parola di pagina 1, pagina 2 e pagina 3.

Questo trucco, indicizzare le metaparole come le parole normali, è il “trucco della metaparola”. Può sembrare di una semplicità disarmante, ma questo trucco è fondamentale per consentire ai motori di ricerca di eseguire ricerche accurate ed effettuare ordinamenti di buona qualità. Vediamo un esempio semplice. Supponiamo che un motore di ricerca ammetta un tipo particolare di interrogazione con la parola chiave IN, così che un’interrogazione boat IN TITLE restituisca come risultati solo pagine che hanno la parola “boat” nel titolo e giraffe IN BODY trovi solo pagine che contengono “giraffe” nel loro corpo. Badate che la maggior parte dei motori reali non consente interrogazioni IN esattamente in questo modo, ma alcuni permettono di avere l’equivalente se si fa clic su una opzione “ricerca avanzata”, dove si può specificare che le parole cercate debbono essere nel titolo o in qualche altra parte specifica di un documento. Facciamo finta che esista la parola chiave IN solo per semplificare la spiegazione. In realtà, nel momento in cui scrivo, Google permette di effettuare una ricerca nei titoli con la parola chiave intitle:, perciò l’interrogazione intitle:boat restituisce pagine con “boat” nel titolo. Provate!

a	3-10
cat	1-3 1-7 3-7
dog	2-3 2-7 3-11
mat	1-11 2-11
my	1-2 2-2 3-2
on	1-9 2-9
pets	3-3
sat	1-8 3-12
stood	2-8 3-8
the	1-6 1-10 2-6 2-10 3-6
while	3-9
<bodyEnd>	1-12 2-12 3-13
<bodyStart>	1-5 2-5 3-5
<titleEnd>	1-4 2-4 3-4
<titleStart>	1-1 2-1 3-1

Figura 2.7 L’indice per le pagine web mostrate nella figura precedente, metaparole comprese.

Vediamo come fa un motore di ricerca a rispondere in modo efficiente alla interrogazione dog IN TITLE per le tre pagine dell’esempio nelle ultime figure. Innanzitutto, estrae la voce di indice per “dog”, che è 2-3, 2-7, 3-11. Poi (e forse non ve l’aspettavate, ma seguitemi un attimo) estrae le voci di indice *sia* per <titleStart> *sia* per <titleEnd>. Le informazioni estratte fin qui sono riepilogate nella Figura 2.8 (per il momento potete ignorare circonferenze e rettangoli).

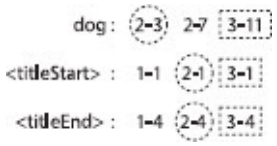


Figura 2.8 Ed ecco come un motore effettua la ricerca di dog IN TITLE.

Il motore di ricerca poi comincia a esplorare la voce “dog”, esamina ciascuna delle sue occorrenze e verifica se sia o meno all’interno di un titolo. Il primo risultato per “dog” è l’elemento 2-3, racchiuso in una circonferenza, che corrisponde alla terza parola della pagina numero 2. Esaminando gli elementi di <titleStart>, il motore può scoprire dove inizia il titolo della pagina 2 (deve essere il primo numero che inizia con “2-”). In questo caso arriva all’elemento circolettato 2-1, il che significa che il titolo della pagina 2 inizia con la parola numero 1. Nello stesso modo, può scoprire dove termina il titolo della pagina 2. Non fa altro che esaminare gli elementi della voce <titleEnd>, alla ricerca di un numero

che inizi con “2-”, e si ferma quando incontra l’elemento circolettato 2-4. Dunque, il titolo della pagina 2 finisce con la parola 4.

Tutto quello che sappiamo fin qui è riassunto dagli elementi circolettati nella figura, che ci dicono che il titolo della pagina 2 inizia con la parola 1 e finisce con la parola 4, e che la parola “dog” è la parola 3. Il passo finale è facile: poiché 3 è maggiore di 1 e minore di 4, siamo sicuri che questa occorrenza della parola “dog” è effettivamente all’interno di un titolo; di conseguenza la pagina 2 sarà un risultato valido per l’interrogazione dog IN TITLE.

Il motore di ricerca può passare adesso all’occorrenza successiva di “dog”, che è 2-7 (la settima parola della pagina 2), ma, poiché sappiamo già che la pagina 2 è un risultato valido, possiamo lasciar perdere questo caso e passare all’occorrenza successiva, 3-11, che è evidenziata da un rettangolo. Questo ci dice che “dog” è la parola 11 della pagina 3. Perciò cominciamo a esaminare gli elementi successivi alle posizioni attualmente circolettate nelle righe di <title-Start> e <titleEnd>, alla ricerca di elementi che inizino con “3-”. (È importante notare che non dobbiamo tornare all’inizio di ogni riga: possiamo riprendere da dove eravamo arrivati nell’esplorazione per l’occorrenza precedente.) In questo semplice esempio, “3-” è casualmente proprio il numero successivo in entrambi i casi: 3-1 per <titleStart> e 3-4 per <titleEnd>, evidenziati dai rettangoli per comodità. Ancora una volta dobbiamo stabilire se l’occorrenza di “dog” che stiamo esaminando, a 3-11, si trova all’interno di un titolo o no. Le informazioni nei rettangoli ci dicono che nella pagina 3 “dog” è la parola 11, mentre il titolo inizia con la parola 1 e finisce con la 4. Poiché 11 è maggiore di 4, sappiamo che questa occorrenza di “dog” è dopo la fine del titolo e perciò *non* è nel titolo, e di conseguenza la pagina 3 non è un risultato valido per l’interrogazione dog IN TITLE.

Quindi il trucco delle metaparole consente a un motore di ricerca di fornire risposta a interrogazioni sulla struttura di un documento in modo estremamente efficiente. L’esempio riguardava solo la ricerca all’interno dei titoli delle pagine, ma tecniche molto simili consentono di cercare parole nei collegamenti ipertestuali, nelle descrizioni delle immagini e in varie altre parti utili delle pagine web. E tutte queste interrogazioni possono trovare risposta con la stessa efficienza dell’esempio precedente: proprio come per le interrogazioni viste prima, il motore di ricerca non deve tornare a esaminare le pagine web originali, ma può rispondere consultando solo un piccolo numero di voci d’indice. E, cosa altrettanto importante, deve esaminare ciascun elemento di ogni voce *una volta sola*. Ricordate quello che è successo quando abbiamo completato l’elaborazione per la prima occorrenza della pagina 2 e siamo passati al possibile risultato di pagina 3: anziché tornare all’inizio delle voci <titleStart> e <titleEnd> il motore può continuare la sua esplorazione dal punto in cui era arrivato. Questo è un aspetto fondamentale per l’efficienza delle interrogazioni IN.

Le interrogazioni sui titoli e altre “interrogazioni strutturali” che dipendono dalla *struttura* di una pagina web sono simili alle interrogazioni NEAR di cui abbiamo parlato prima, nel senso che gli esseri umani le utilizzano raramente, mentre i motori di ricerca le applicano internamente di continuo. Il motivo è lo stesso: i motori di ricerca vivono o muovono in base alla qualità delle loro capacità di ordinamento, e queste possono essere migliorate significativamente sfruttando la struttura delle pagine web. Per esempio, le

pagine che hanno la parola “dog” nel titolo è molto più probabile che contengano informazioni sui cani delle pagine che contengono la parola “dog” solo nel corpo. Perciò, quando un utente invia la semplice interrogazione dog, un motore di ricerca può effettuare internamente una ricerca dog IN TITLE (anche se l’utente non l’ha richiesto esplicitamente) per trovare pagine che è più probabile *parlino di* cani, anziché semplicemente citare la parola incidentalmente.

I trucchi dell’indicizzazione e della ricerca di corrispondenza non sono tutto

Costruire un motore di ricerca non è impresa facile. Il prodotto finale è come una macchina di enorme complessità con molti ingranaggi e molte leve, che devono essere tutti montati correttamente perché il sistema sia utile. Perciò è importante rendersi conto che i due trucchi presentati in questo capitolo da soli non risolvono il problema di costruire un indice efficace per un motore di ricerca. Il trucco della posizione delle parole e quello delle metaparole però danno sicuramente il *gusto* di come i veri motori di ricerca costruiscono e usano gli indici.

Il trucco delle metaparole ha contribuito al successo di AltaVista là dove altri avevano fallito, cioè nel trovare corrispondenze in modo efficiente in tutto il Web. Lo sappiamo perché il trucco delle metaparole è descritto in una richiesta di brevetto del 1999 per gli Usa presentata da AltaVista, con il titolo “Constrained Searching of an Index”. Nonostante la sua raffinata costruzione, l’algoritmo per le corrispondenze di AltaVista non è stato sufficiente per rimanere a galla nella turbolenza dei primi tempi del settore delle ricerche. Come già sappiamo, l’efficacia nel trovare corrispondenze è solo una faccia della medaglia: l’altra grande sfida è *ordinare* i risultati. E, come vedremo nel prossimo capitolo, la comparsa di un nuovo tipo di algoritmo di ordinamento è stata sufficiente a eclissare Alta-Vista e a portare Google in vetta.

---

1 Tr. it. di Marisa De Maria, Editoriale del Drago, Milano, 1989.

CAPITOLO 3

PageRank: la tecnologia che ha lanciato Google

> Il computer di *Star Trek* non sembra tanto interessante. Gli chiedono delle cose a caso, lui ci pensa per un po'. Penso che possiamo fare di meglio.
>
> Larry Page (cofondatore di Google)

Dal punto di vista architettonico, il garage è una struttura normalmente modesta, ma nella Silicon Valley i garage hanno un significato imprenditoriale speciale: molte fra le grandi aziende di tecnologia della "valle del silicio" sono nate, o almeno hanno avuto la loro fase di incubazione, in un garage. E non è una tendenza iniziata con il boom delle dot.com negli anni novanta: oltre 50 anni prima, nel 1939, quando l'economia mondiale ancora cercava di riprendersi dalla Grande depressione, la Hewlett-Packard ha mosso i suoi primi passi nel garage di Dave Hewlett a Palo Alto in California. Parecchi decenni più tardi, nel 1976, Steve Jobs e Steve Wozniak si sono messi a lavorare nel garage di Jobs a Los Altos, sempre in California, dopo aver fondato la loro azienda, quella Apple che oggi è diventata leggendaria. (Anche se le leggende popolari vorrebbero che la Apple sia stata fondata nel garage, Jobs e Wozniak in realtà hanno cominciato a lavorare in una camera da letto, ma sono rimasti in fretta a corto di spazio e hanno dovuto trasferirsi nel garage.) Forse ancor più degno di nota dei primi passi di HP e Apple è il lancio di un motore di ricerca, chiamato Google, che era in funzione in un garage di Menlo Park in California, quando l'azienda è stata ufficialmente fondata nel 1998.

In quel momento Google aveva già attivato il suo servizio di ricerca per il Web da oltre un anno, inizialmente dai server della Stanford University, dove entrambi i soci fondatori erano studenti di dottorato. Solo quando l'ampiezza di banda necessaria per la crescente popolarità del servizio divenne eccessiva per Stanford i due studenti, Larry Page e Sergey Brin, spostarono la loro sede operativa nel garage, oggi famoso, di Menlo Park. Devono aver fatto proprio le cose per bene, perché solo tre mesi dopo la costituzione legale della società Google è stato indicato da *PC Magazine* come uno dei 100 migliori siti del 1998.

E qui comincia davvero la nostra storia: secondo le parole di *PC Magazine*, Google si era meritato quel posto in classifica per la sua "infallibile capacità di restituire risultati estremamente pertinenti". Ricorderete dal capitolo precedente che i primi motori di ricerca commerciali erano stati lanciati quattro anni prima, nel 1994: come ha fatto Google dal suo garage a compensare questo ritardo di quattro anni, saltando a piè pari davanti ai già diffusi Lycos e AltaVista per quel che riguarda la qualità delle ricerche? La domanda non ammette una risposta semplice, ma uno dei fattori più importanti, in particolare in quei primi tempi, è stato l'algoritmo innovativo utilizzato da Google per ordinare i risultati delle ricerche: un algoritmo noto con il nome di *PageRank*.

È un po' un gioco di parole: è un algoritmo che ordina pagine (*page*, in inglese), ma è anche l'algoritmo di ordinamento di Larry Page, il suo principale inventore. Page e Brin hanno reso pubblico l'algoritmo nel 1998, in un saggio di un convegno accademico, "The Anatomy of a Large-scale Hypertextual Web Search Engine". Come fa pensare già il titolo, il saggio fa molto di più che descrivere PageRank: è a tutti gli effetti una descrizione completa del sistema Google, allo stato in cui era nel 1998. Sepolta fra i particolari tecnici del sistema c'è una descrizione di quella che può ben essere considerata la prima gemma algoritmica destinata a emergere nel ventunesimo secolo: l'algoritmo PageRank. In questo capitolo esploreremo come e perché questo algoritmo è in grado di trovare aghi nei pagliai, fornendo coerentemente i risultati più pertinenti come *top hit* di una interrogazione di ricerca.

Il trucco del collegamento ipertestuale

Probabilmente sapete già che cos'è un collegamento ipertestuale o *hyperlink*: è un'espressione contenuta in una pagina web che porta a un'altra pagina web quando viene attivata (con un clic del mouse o qualche altro metodo) La maggior parte dei browser mostra i collegamenti ipertestuali in caratteri blu sottolineati, in modo che risultino facilmente visibili.

Quella dei collegamenti ipertestuali è un'idea sorprendentemente vecchia. Nel 1945, più o meno contemporaneamente ai primi sviluppi dei computer elettronici stessi, l'ingegnere americano Vannevar Bush pubblicava un saggio visionario, "As We May Think". In quel saggio di ampia portata, Bush descriveva una serie di potenziali nuove tecnologie, fra cui una macchina che chiamava *memex*. Un memex avrebbe memorizzato documenti e li avrebbe automaticamente indicizzati, ma avrebbe fatto anche molto di più. Avrebbe consentito "l'indicizzazione associativa … per cui qualsiasi elemento si può fare in modo che a comando ne selezioni immediatamente e automaticamente un altro", in altre parole, una forma rudimentale di collegamento ipertestuale!

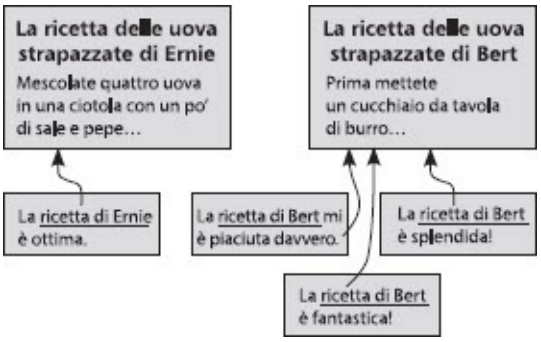


Figura 3.1 Le basi del trucco del collegamento ipertestuale. Sono mostrate sei pagine web, ciascuna rappresentata da un rettangolo. Due pagine contengono ricette per le uova strapazzate, mentre le altre quattro sono pagine che hanno collegamenti ipertestuali a quelle ricette. Il trucco del collegamento ipertestuale classifica la pagina di Bert al di sopra di quella di Ernie, perché Bert ha tre collegamenti in ingresso mentre Ernie ne ha solo uno.

I collegamenti ipertestuali hanno fatto molta strada dal 1945. Sono uno degli strumenti più importanti utilizzati dai motori di ricerca per effettuare l'ordinamento e sono fondamentali per la tecnologia PageRank di Google, che ora cominceremo a esplorare seriamente.

Il primo passo per capire PageRank è un'idea semplice, che chiameremo *trucco del collegamento ipertestuale*. Il modo migliore di spiegarlo è con un esempio. Supponiamo

siate interessati a scoprire come si preparano le uova strapazzate e facciate una ricerca nel Web su questo argomento. Qualsiasi ricerca nel Web reale produrrebbe milioni di risultati, ma, per semplicità, supponiamo che escano solo due pagine, una che si chiama “La ricetta delle uova strapazzate di Ernie”, mentre l’altra è “La ricetta delle uova strapazzate di Bert”. Sono rappresentate nella Figura 3.1, insieme con alcune altre pagine web che hanno collegamenti ipertestuali o alla ricetta di Bert o a quella di Ernie. Per semplicità (ancora), immaginiamo che le quattro pagine mostrate siano *le uniche* pagine in tutto il Web che hanno un collegamento all’una o all’altra ricetta. I collegamenti ipertestuali sono indicati dal testo sottolineato, le frecce mostrano la destinazione a cui porta il collegamento.

La domanda è: quale dei due risultati deve occupare il primo posto in classifica, la ricetta di Bert o la ricetta di Ernie? Da esseri umani, non faremmo molta fatica a leggere le pagine che si collegano alle due ricette e a farci una nostra valutazione. Sembra che entrambe le ricette siano buone, ma le persone mostrano molto più entusiasmo per la ricetta di Bert che per quella di Ernie. In mancanza di ulteriori informazioni, quindi, probabilmente ha più senso che Bert stia in classifica prima di Ernie.

Purtroppo i computer non sono molto bravi a capire il significato effettivo di una pagina web, perciò un motore di ricerca non può esaminare le quattro pagine che si collegano ai due risultati e valutare con quanto entusiasmo ciascuna ricetta venga consigliata. I computer però eccellono nel contare, perciò un metodo semplice è quello di *contare* quante pagine si collegano a ciascuna delle ricette (in questo caso sono una per Ernie e tre per Bert) e ordinare le ricette in base al numero dei collegamenti in entrata. Ovviamente questo metodo non ha la stessa precisione che potrebbe avere un essere umano che leggesse tutte le pagine e determinasse l’ordinamento manualmente, ma è comunque una tecnica utile. Statisticamente, in assenza di altre informazioni, il numero dei collegamenti in ingresso può essere un buon indicatore di quanto è probabile che la pagina sia utile, ovvero “autorevole”. In questo caso il punteggio è Bert 3, Ernie 1, perciò la pagina di Bert finisce più in alto in classifica quando il motore di ricerca presenta i suoi risultati all’utente.

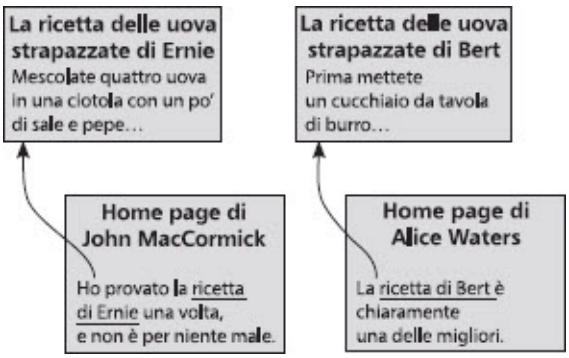


Figura 3.2 Le basi del trucco dell’autorevolezza. Ci sono quattro pagine web: due sono ricette per le uova strapazzate, due sono pagine che contengono un collegamento ipertestuale alle ricette. Uno dei collegamenti proviene dall’autore di questo libro (che non *è uno chef famoso), uno dalla home page della famosa chef Alice Waters. Il trucco dell’autorevolezza fa salire in classifica la pagina di Bert più di quella di Ernie, perché il collegamento in ingresso alla pagina di Bert ha una maggiore “autorevolezza” di quello in ingresso alla pagina di Ernie.*

Probabilmente avrete già visto qualche possibile problema per il “trucco del collegamento ipertestuale”. Una questione ovvia è che a volte i collegamenti sono utilizzati per giudizi *negativi* anziché positivi. Per esempio, immaginatevi una pagina web con un collegamento alla ricetta di Ernie e che dica “Ho provato la ricetta di Ernie, ed è

terribile”. Collegamenti come questo, che criticano una pagina anziché consigliarla, fanno effettivamente in modo che il trucco del collegamento ipertestuale attribuisca a certe pagine un posto migliore in classifica di quel che meriterebbero. Ma nella pratica si dà il caso che i collegamenti ipertestuali siano più spesso consigli anziché critiche, perciò il trucco rimane utile nonostante quest’ovvia debolezza.

Il trucco dell’autorevolezza

Forse vi sarete già chiesti perché i collegamenti in ingresso a una pagina debbano essere trattati tutti allo stesso modo. Una raccomandazione da parte di un esperto non vale di più di quella di chi è alle prime armi? Torniamo all’esempio precedente delle uova strapazzate, ma ora con un diverso insieme di collegamenti in ingresso. La Figura 3.2 mostra la nuova situazione: le pagine di Bert e Ernie ora hanno lo stesso numero di collegamenti in ingresso (solo uno), ma quello che va a Ernie parte dalla mia pagina, mentre quello di Bert arriva dalla famosa chef Alice Waters.

In assenza di altre informazioni, quale ricetta scegliereste? Ovviamente, è meglio scegliere quella consigliata da un famoso chef, anziché quella consigliata dall’autore di un libro di informatica. Questo principio di base è quello che chiameremo il “trucco dell’autorevolezza”: i collegamenti che arrivano da pagine con elevata “autorevolezza” contribuiranno a un avanzamento in classifica più di quelli che arrivano da pagine con minore autorevolezza.

Il principio è ottimo, ma in questa forma è del tutto inservibile per i motori di ricerca. Come fa un computer a stabilire automaticamente che l’autorevolezza di Alice Waters quando parla di uova strapazzate è molto maggiore della mia? Ecco un’idea che potrebbe aiutarci: combiniamo il trucco del collegamento ipertestuale con quello dell’autorevolezza. Tutte le pagine inizialmente hanno autorevolezza 1, ma se una pagina ha collegamenti ipertestuali in ingresso, la sua autorevolezza si calcola sommando l’autorevolezza di tutte le pagine che hanno un collegamento ad essa. In altre parole, se le pagine X e Y hanno collegamenti alla pagina Z, l’autorevolezza di Z è la somma dell’autorevolezza di X e di quella di Y.

La Figura 3.3 mostra un esempio particolareggiato per il calcolo dell’autorevolezza delle due ricette delle uova strapazzate. I punteggi finali sono racchiusi in un cerchietto. Ci sono due pagine che si collegano alla mia home page; queste pagine a loro volta non hanno collegamenti in ingresso, perciò la loro autorevolezza è 1. L’autorevolezza della mia pagina è la loro somma, quindi è 2. La pagina di Alice Waters ha 100 collegamenti in ingresso, ciascuno dei quali da pagine di autorevolezza 1, perciò ottiene un valore 100. La ricetta di Ernie ha un solo collegamento in ingresso, ma è da una pagina che vale 2, perciò sommando tutti i punteggi relativi ai collegamenti in ingresso (in questo caso ce n’è uno solo), Ernie si guadagna un’autorevolezza di 2. Anche la ricetta di Bert ha un solo collegamento in ingresso, ma vale 100 e così il suo punteggio finale è 100. Dato che 100 è maggiore di 2, la pagina di Bert avrà in classifica una posizione migliore di quella di Ernie.

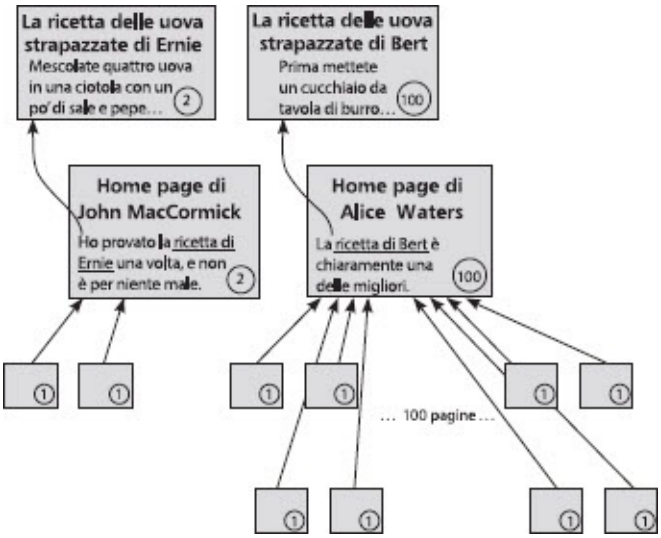


Figura 3.3 Un semplice calcolo dei "punteggi di autorevolezza" per le due ricette delle uova strapazzate. Il valore dell'autorevolezza è racchiuso in un cerchietto.

Il trucco del navigatore casuale

Sembrerebbe che abbiamo trovato una strategia per calcolare automaticamente i valori dell'autorevolezza e che funzioni bene, senza che il computer debba realmente comprendere i contenuti di una pagina. Purtroppo, questo metodo incappa in un problema grave: è possibile che i collegamenti ipertestuali formano quello che gli informatici chiamano un "ciclo" o un "anello". Si ha un ciclo se si può tornare al punto di partenza semplicemente seguendo i collegamenti ipertestuali.

La Figura 3.4 ce ne dà un esempio. Ci sono cinque pagine web: A, B, C, D, E. Se partiamo da A, possiamo seguire il collegamento da A a B e poi quello da B a E; da E poi possiamo seguire il collegamento che ci riporta ad A, da dove siamo partiti. Questo significa che A, B ed E formano un ciclo.

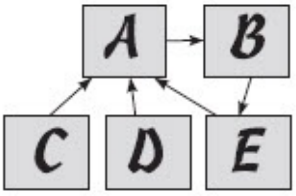


Figura 3.4 Un esempio di ciclo di collegamenti ipertestuali. Le pagine A, B, E formano un ciclo perché si può partire da A, seguire il collegamento verso B, poi da B verso E e infine tornare al punto di partenza in A.

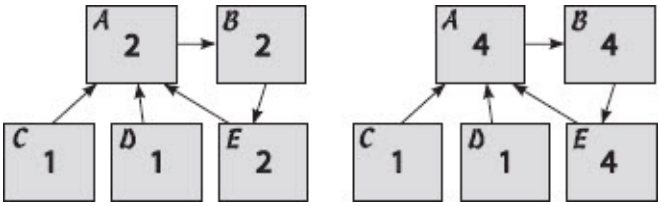


Figura 3.5 Il problema causato dai cicli. A, B ed E non sono mai aggiornati, e il loro punteggio continua a crescere indefinitamente.

La nostra definizione di "valore (o punteggio) di autorevolezza" (che combina il trucco del collegamento ipertestuale e quello dell'autorevolezza) finisce in guai seri ogni volta che c'è un ciclo. Vediamo che cosa succede in questo particolare esempio. Le pagine C e D non hanno collegamenti in ingresso, perciò la loro autorevolezza è 1. C e D hanno entrambi un collegamento ad A, perciò il punteggio di A è la somma di quelli di C e D,

ovvero 1 + 1 = 2. Poi B riceve il punteggio 2 da A, E riceve il punteggio 2 da B. (La situazione fin qui è riassunta nella parte sinistra della figura.) Ma a questo punto A non è più aggiornato: riceve ancora 1 sia da C sia da D, ma riceve anche un 2 da E, per un totale di 4. Ma ora non è più aggiornato B: riceve un 4 da A. Ma poi bisogna aggiornare anche E, che prende un 4 da B. (Ora siamo alla situazione della parte destra della figura.) E così via: ora A è 6, perciò B è 6 ed E è 6, ma allora A è 8, … Avete sicuramente afferrato l'idea. Continueremmo all'infinito, con i punteggi che continuano ad aumentare a ogni ciclo.

Se si calcola l'autorevolezza in questo modo, si finisce in un problema "uovo o gallina". Se si conosce l'autorevolezza di A, si può calcolare quella di B ed E; e se si conosce l'autorevolezza di B ed E, si può calcolare quella di A. Ma siccome ciascuna dipende dalle altre, il problema sembra insolubile.

Per fortuna il problema si può risolvere con una tecnica che chiameremo *trucco del navigatore casuale*. Fate attenzione: la descrizione iniziale di questo trucco non ha alcuna somiglianza con i trucchi del collegamento ipertestuale e dell'autorevolezza di cui abbiamo parlato fin qui. Una volta che avremo visto i meccanismi fondamentali del trucco del navigatore casuale, analizzeremo le sue notevoli proprietà: in effetti, combina i tratti positivi degli altri due trucchi, ma funziona anche quando sono presenti cicli di collegamenti ipertestuali.

Immaginiamo una persona che navighi a caso in Internet. Per essere più precisi, il nostro navigatore parte da una pagina scelta a caso da tutto il World Wide Web. Poi esamina tutti i collegamenti ipertestuali che partono da quella pagina, ne sceglie uno a caso e ci fa clic sopra. Poi esamina la nuova pagina e sceglie a caso uno dei suoi collegamenti ipertestuali. Il procedimento continua, e ogni nuova pagina viene scelta a caso facendo un clic su un collegamento ipertestuale nella pagina precedente. La Figura 3.6 mostra un esempio, in cui immaginiamo che tutto il World Wide Web sia costituito da 16 pagine soltanto. I rettangoli rappresentano pagine web, le frecce rappresentano collegamenti ipertestuali fra le pagine. Ci sono quattro pagine identificate come A, B, C, D, per semplificare i ragionamenti successivi. Le pagine visitate dal navigatore sono in grigio più scuro, i collegamenti che segue sono in nero e le frecce tratteggiate rappresentano ripartenze casuali, di cui diremo fra poco.

C'è una particolarità nel procedimento: ogni volta che viene visitata una pagina, c'è una *probabilità di ripartenza* costante (poniamo il 15%) che il navigatore *non* faccia clic su uno dei collegamenti ipertestuali disponibili ma riavvii il procedimento scegliendo un'altra pagina a caso da tutto il Web. Potete pensare che c'è un 15% di probabilità che il navigatore si annoi, data una qualsiasi pagina, e che decida di seguire una nuova catena di collegamenti. Per qualche esempio, esaminate meglio la figura. Questo particolare navigatore è partito dalla pagina A e ha seguito tre collegamenti ipertestuali a caso prima di arrivare alla pagina B, annoiarsi e ripartire dalla pagina C. Dopo aver seguito due altri collegamenti a caso, una nuova ripartenza. (Incidentalmente, tutti gli esempi di navigatori casuali in questo capitolo utilizzano una probabilità di ripartenza del 15%, che è la stessa utilizzata dai due fondatori di Google, Page e Brin, nell'articolo originale che descriveva il prototipo del loro motore di ricerca.)

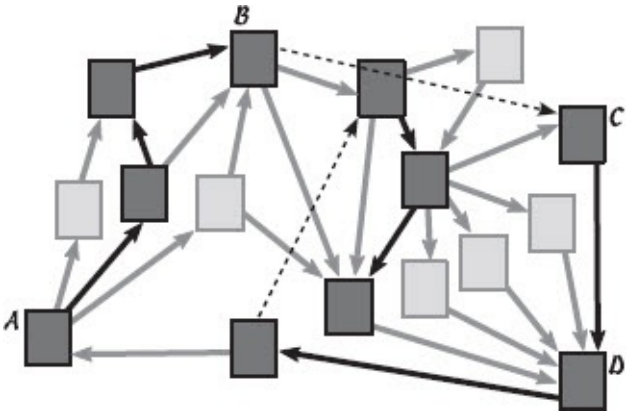


Figura 3.6 Il modello del navigatore casuale. Le pagine visitate dal navigatore sono in grigio più scuro e le frecce tratteggiate rappresentato ripartenze casuali. La prova inizia dalla pagina A e segue collegamenti ipertestuali scelti a caso, con l'interruzione di due ripartenze casuali.

È facile simulare questo procedimento con un computer. Ho scritto un programma che fa proprio questo e l'ho fatto girare fino a che il navigatore non aveva visitato 1000 pagine. (Ovviamente, questo non significa 1000 pagine diverse, valgono anche le visite alla stessa pagina e in questo piccolo esempio tutte le pagine sono state visitate molte volte.) I risultati delle 1000 visite simulate sono presentati nella parte superiore della Figura 3.7. Si può vedere che la pagina D è stata quella visitata più spesso, con 144 visite.

Proprio come nei sondaggi dell'opinione pubblica, si può migliorare l'accuratezza della simulazione aumentando il numero dei campioni casuali. Ho fatto girare nuovamente la simulazione, questa volta aspettando che il navigatore visitasse un milione di pagine. (Nel caso ve lo state chiedendo, c'è voluto meno di mezzo secondo, sul mio computer!) Con un numero così grande di visite, è meglio presentare i risultati in forma di percentuali, ed è questo che potete vedere nella parte inferiore della Figura 3.7. Anche questa volta la pagina D era quella visitata più spesso, con il 15% delle visite.

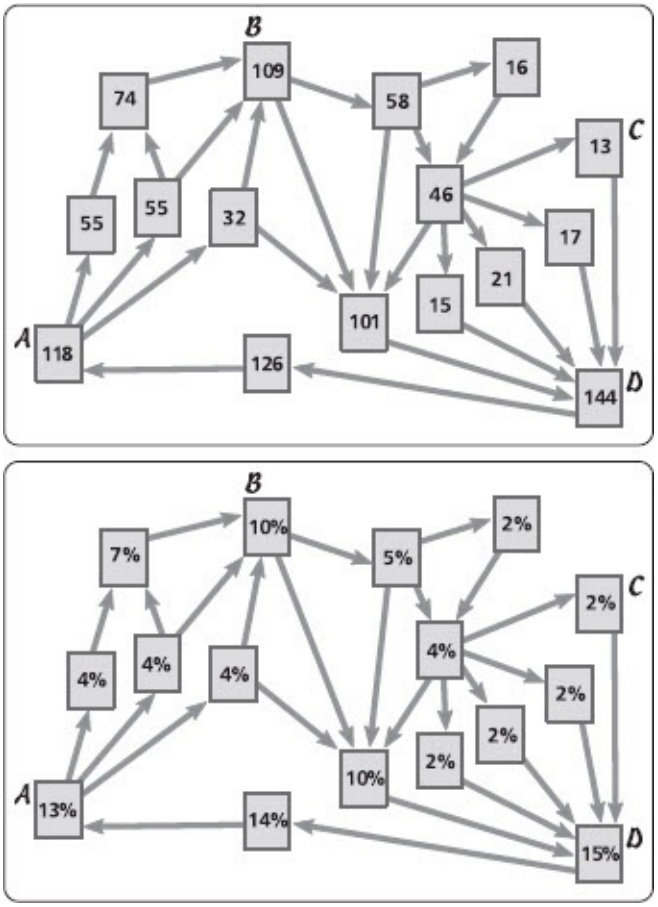


Figura 3.7 Simulazioni del navigatore casuale. In alto: numero di visite a ciascuna pagina in una simulazione di 1000 visite. Sotto: percentuale di visite a ciascuna pagina in una simulazione di un milione di visite.

Qual è il legame fra il nostro modello del visitatore casuale e il trucco dell'autorevolezza che vorremmo utilizzare per ordinare le pagine web? Le percentuali calcolate dalle simulazioni del navigatore casuale sono esattamente quello che ci serve per misurare l'autorevolezza di una pagina. Definiamo allora *punteggio di autorevolezza del navigatore* di una pagina web come la percentuale del tempo che un navigatore casuale dedicherebbe a visitare quella pagina. Cosa notevole, il punteggio di autorevolezza del navigatore incorpora entrambi i nostri trucchi precedenti per ordinare per importanza le pagine web. Li esamineremo uno alla volta.

Innanzitutto, avevamo il trucco del collegamento ipertestuale: l'idea di fondo qui è che una pagina con molti collegamenti in ingresso debba avere una posizione migliore. Questo è vero anche nel modello del navigatore casuale, perché una pagina con molti collegamenti in ingresso ha molte probabilità di essere visitata. La pagina D nella parte ineriore della figura nella pagina seguente è un buon esempio: ha cinque collegamenti in ingresso, più di ogni altra pagina nella simulazione, e finisce per avere il punteggio di autorevolezza del navigatore più alto (15%).

In secondo luogo, avevamo il trucco dell'autorevolezza. L'idea di fondo era che il collegamento proveniente da una pagina molto autorevole debba migliorare il posizionamento di una pagina più di un collegamento in arrivo da una pagina meno autorevole. E il modello del navigatore casuale tiene conto anche di questo. Perché? Perché un collegamento proveniente da una pagina molto popolare avrà maggiori possibilità di essere seguito rispetto a un collegamento da una pagina poco popolare. Per vedere un esempio nella nostra simulazione, confrontate le pagine A e C nel riquadro

inferiore della figura: ciascuna ha esattamente un collegamento in ingresso, ma la pagina A ha un punteggio di autorevolezza del navigatore più alto (13% rispetto a 2%) grazie alla qualità del suo collegamento in ingresso.

Notate che il modello del navigatore casuale incorpora simultaneamente sia il trucco del collegamento ipertestuale sia quello dell'autorevolezza. In altre parole, tiene conto sia della qualità sia della quantità dei collegamenti in ingresso. Lo dimostra la pagina B: riceve il suo punteggio relativamente elevato (10%) grazie a tre collegamenti in arrivo da pagine con punteggi moderati, che vanno dal 4 al 7%.La bellezza del trucco del navigatore casuale è che, a differenza di quello dell'autorevolezza, funziona perfettamente indipendentemente dall'esistenza di cicli di collegamenti ipertestuali. Tornando al nostro esempio precedente delle uova strapazzate (pagina 32), possiamo facilmente eseguire una simulazione del navigatore casuale. Dopo vari milioni di visite, la mia simulazione ha dato i punteggi di autorevolezza del navigatore indicati nella Figura 3.8. Notate che, come nel calcolo precedente con il trucco dell'autorevolezza, la pagina di Bert riceve un punteggio molto superiore a quello della pagina di Ernie (28% contro 1%), nonostante il fatto che entrambe abbiano esattamente un collegamento in ingresso. Perciò Bert sarebbe in una posizione migliore, fra i risultati di una ricerca sul Web per "uova strapazzate".

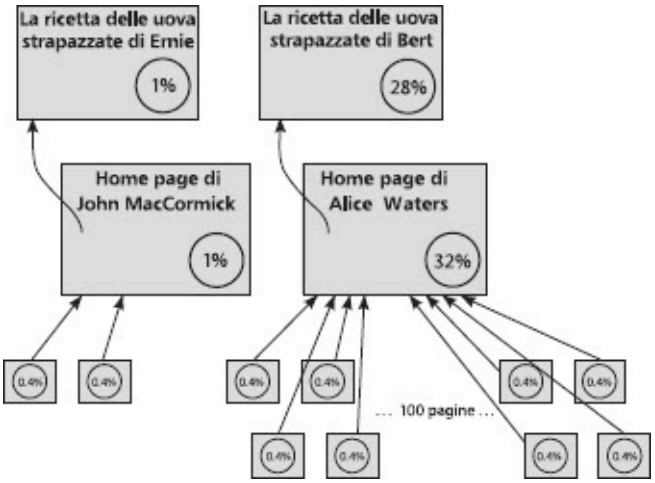


Figura 3.8 I punteggi di autorevolezza del navigatore per l'esempio delle uova strapazzate di pagina 29. Sia Bert sia Ernie hanno esattamente un collegamento in ingresso che contribuisce a determinare l'autorevolezza delle loro pagine, ma quella di Bert avrà un posizionamento migliore nei risultati di una ricerca nel Web per "uova strapazzate".

Passiamo ora all'esempio più difficile, quello delle Figure 3.4 e 3.5, che metteva in una difficoltà insormontabile il nostro trucco dell'autorevolezza a causa del ciclo di collegamenti. Anche in questo caso è facile eseguire una simulazione al computer dei navigatori casuali, e si ottengono i punteggi di autorevolezza del navigatore della Figura 3.9. I punteggi determinati con questa simulazione ci dicono quale sia l'ordinamento finale che utilizzerebbe un motore di ricerca al momento di presentare i suoi risultati: la pagina A per prima, seguita da B, poi da E, con C e D alla pari in ultima posizione.

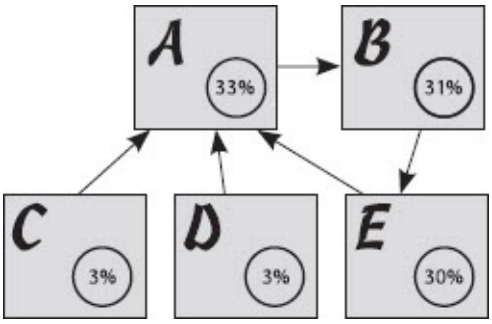


Figure 3.9 Punteggi di autorevolezza del navigatore casuale per l'esempio precedente con un ciclo di collegamenti ipertestuali (Figure 3.4 e 3.5). Il trucco del navigatore casuale non ha difficoltà a calcolare punteggi appropriati, nonostante la presenza di un ciclo (A→ B → E → A)

PageRank in pratica

Il trucco del navigatore casuale è stato descritto dai due fondatori di Google nel loro famoso saggio del 1998, "The Anatomy of a Large-Scale Hypertextual Web Search Engine" e, insieme con molte altre tecniche, varianti di questo trucco vengono utilizzate tuttora dai grandi motori di ricerca. Esistono però parecchi fattori che complicano le cose, il che significa che le tecniche effettivamente utilizzate dai motori di ricerca moderni sono un po' diverse da quella del navigatore casuale appena descritta.

Uno di questi fattori di complicazione va diritto al cuore di PageRank: l'assunto che i collegamenti ipertestuali conferiscano legittimamente autorevolezza è discutibile. Abbiamo già visto che, anche se i collegamenti possono rappresentare critiche anziché apprezzamenti positivi non si tratta in pratica di un problema significativo. Un problema molto più grave è che è possibile abusare del trucco del collegamento ipertestuale per migliorare artificialmente la posizione delle proprie pagine web. Supponiamo che gestiate un sito LibriLibriLibri.com che vende (ma pensa un po') libri. Con una tecnologia automatizzata, è relativamente facile creare un gran numero, diciamo 10.000, di pagine web diverse, ma tutte con un collegamento a LibriLibriLibri.com. Quindi, se i motori di ricerca calcolassero l'autorevolezza di PageRank esattamente come abbiamo descritto qui, LibriLibriLibri.com potrebbe ottenere senza meritarselo un punteggio migliaia di volte superiore a quello di altre librerie online, con una posizione migliore e quindi con maggiori vendite.

I motori di ricerca definiscono *web spam* questo tipo di abuso. (Il termine deriva da un'analogia con lo spam nella posta elettronica: i messaggi indesiderati nella vostra casella di posta elettronica sono simili a pagine web indesiderate che ingombrano i risultati di una ricerca nel Web.) Identificare ed eliminare i vari tipi di web spam è importante per tutti i motori di ricerca. Nel 2004, per esempio, alcuni ricercatori alla Microsoft hanno trovato oltre 300.000 siti che ricevevano collegamenti da *esattamente* 1001 pagine – cosa assai sospetta. Andando a esaminare manualmente questi siti, i ricercatori hanno trovato che la stragrande maggioranza dei collegamenti ipertestuali in ingresso erano web spam.

I motori di ricerca sono impegnati in una corsa agli armamenti contro gli spammer e cercano continuamente di migliorare i loro algoritmi per classificare in modo sensato i risultati. Questa spinta costante a migliorare PageRank ha favorito molta ricerca accademica e industriale su altri algoritmi che usano la struttura a collegamenti ipertestuali del Web per ordinare le pagine. Algoritmi di questo tipo spesso sono indicati come

algoritmi di ordinamento basati sui collegamenti.

Un altro fattore di complicazione è legato all'efficienza dei calcoli di PageRank. I nostri punteggi di autorevolezza del navigatore sono stati calcolati eseguendo simulazioni casuali, ma simulazioni simili su tutto il Web richiederebbero troppo tempo per essere utili. I motori di ricerca non calcolano i valori di PageRank simulando navigatori casuali; usano invece tecniche matematiche che danno le stesse risposte ma con un costo computazionale minore. Abbiamo studiato la tecnica della simulazione del navigatore per il suo fascino intuitivo, e perché descrive *che cosa* calcolano i motori di ricerca, non *come* lo calcolano.

Val la pena anche di notare che i motori di ricerca determinano i loro ordinamenti utilizzando molto di iù di un semplice algoritmo di ordinamento basato sui collegamenti come PageRank. Già nella loro descrizione di Google nel 1998, i suoi due fondatori citavano parecchie altre caratteristiche che contribuivano all'ordinamento dei risultati delle ricerche. Come potete immaginare, la tecnologia è andata avanti: nel momento in cui scrivo, il sito web di Google dichiara che, per valutare l'importanza di una pagina, vengono utilizzati "oltre 200 segnali".

Nonostante le molte complessità dei motori di ricerca moderni, la bella idea al cuore di PageRank, cioè che le pagine autorevoli possano conferire autorevolezza ad altre pagine attraverso i collegamenti ipertestuali, rimane valida. Questa idea ha aiutato Google a detronizzare AltaVista, trasformandola da piccola startup a regina delle ricerche in poch i. Senza l'idea centrale di PageRank, la maggior parte delle interrogazioni annegherebbero in un mare di pagine web irrilevanti. PageRank è in effetti una gemma algoritmica grazie alla quale un ago può salire senza fatica in cima al suo pagliaio.

CAPITOLO 4

Crittografia a chiave pubblica: spedire segreti su una cartolina

> Chi conosce quelle cose più segrete su di me che sono nascoste al mondo?
>
> Bob Dylan, *Covenant Woman*

Gli esseri umani amano il pettegolezzo, e amano i segreti. E, dato che l'obiettivo della crittografia è comunicare segreti, siamo tutti crittografiper natura. Ma gli esseri umani possono comunicare con segretezza più facilmente dei computer. Se volete raccontare un segreto a un amico, potete semplicemente sussurrarglielo all'orecchio, ma per i computer non è altrettanto facile, non hanno la possibilità di "sussurrare" un numero di carta di credito all'orecchio di un altro computer. Se i computer sono collegati attraverso Internet, non hanno alcun controllo su dove vada quel numero di carta di credito e quali altri computer possano riceverlo. In questo capitolo vedremo come si possa aggirare questo problema, utilizzando una delle idee informatiche più ingegnose e più ricche di conseguenze di tutti i tempi: la crittografia a chiave pubblica.

A questo punto, vi chiederete magari perché il titolo di questo capitolo parli di "spedire segreti su una cartolina". La Figura 4.1 dà una risposta: comunicare con cartoline è una buona analogia per dimostrare la potenza della crittografia a chiave pubblica. Se voleste inviare un documento confidenziale a qualcuno, ovviamente chiudereste il documento in una busta sigillata in modo sicuro, prima di spedirlo. Questo non garantisce ancora la confidenzialità, ma è un passo sensato nella direzione giusta. Se invece sceglieste di scrivere il vostro messaggio confidenziale sul retro di una cartolina prima di spedirla, la confidenzialità sarebbe ovviamente violata: chiunque si trovi in mano la cartolina (gli impiegati dell'ufficio postale, per esempio) basta che la guardi e potrà leggere il messaggio.

Questo è esattamente il problema che debbono affrontare i computer quando cercano di comunicare confidenzialmente fra loro via Internet. Poiché qualsiasi messaggi in Internet normalmente passa attraverso numerosi computer, i cosiddetti router, i contenuti del messaggio possono essere visti da chiunque abbia accesso ai router, e fra questi possono esserci dei potenziali malintenzionati. Quindi, ogni singolo dato che lascia il vostro computer ed entra in Internet è come se fosse scritto su una cartolina.

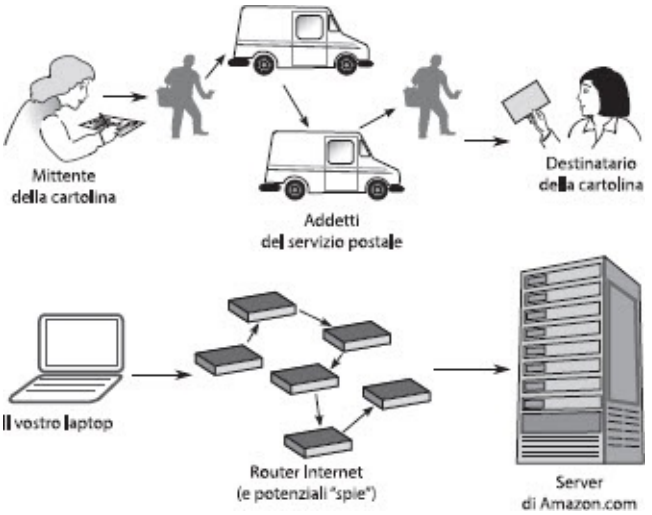


Figura 4.1 L'analogia della cartolina: è ovvio che spedire una cartolina attraverso il sistema postale non mantenga segreti i contenuti della cartolina. Per lo stesso motivo, un numero di carta di credito spedito dal vostro laptop ad Amazon.com può essere facilmente letto da chiunque voglia origliare, se non è opportunamente cifrato.

Probabilmente avrete già pensato a una soluzione rapida per il problema della cartolina. Perché non usare semplicemente un codice segreto per cifrare ciascun messaggio prima di scriverlo sulla cartolina? In effetti, la cosa può funzionare, se conoscete già la persona a cui spedite la cartolina e vi siete accordati, in passato, sul codice segreto da usare. Il problema vero sorge quando inviate una cartolina a qualcuno che non conoscete. Se usate un codice segreto per scrivere sulla cartolina, gli addetti del servizio postale non riusciranno a leggere il vostro messaggio, ma non ci riuscirà neanche il legittimo destinatario! La vera forza della crittografia a chiave pubblica è che vi consente di utilizzare un codice segreto che solo il destinatario può decifrare, nonostante non abbiate mai avuto occasione di mettervi segretamente d'accordo su quale codice utilizzare.

Notate che i computer hanno lo stesso problema di comunicare con destinatari che non "conoscono". Per esempio, la prima volta che acquistate qualcosa da Amazon.com con la vostra carta di credito, il vostro computer deve trasmettere il numero della carta a un server di Amazon; il vostro computer però non ha mai comunicato in precedenza con il server di Amazon, perciò le due macchine non hanno avuto occasione in passato di concordare un codice segreto e qualsiasi accordo cerchino di stringere può essere osservato da tutti i router che le loro comunicazioni attraversano lungo la strada.

Torniamo all'analogia della cartolina. Certo, la situazione sembra un po' paradossale: il destinatario vedrà esattamente le stesse informazioni che vede il postino, ma in qualche modo sa come decifrare il messaggio, mentre il postino non lo sa. La crittografia a chiave pubblica è una soluzione del paradosso.

Cifrare con un segreto condiviso

Cominciamo con un semplice esperimento mentale. Abbandoniamo l'analogia della cartolina per qualcosa di ancora più semplice: la comunicazione verbale in una stanza. In particolare, siete in una stanza con l'amico Arnold e con Eve, che invece vi è avversa. Volete comunicare segretamente un messaggio ad Arnold, senza che Eve possa capirlo. Il messaggio è magari un numero di carta di credito, ma rendiamoci le cose ancora più semplici, e supponiamo che sia un numero di carta di credito brevissimo – solo una cifra fra 1 e 9. Inoltre, dobbiamo dire che l'unico modo in cui potete comunicare con Arnold è

parlando ad alta voce, perciò Eve potrà sentirvi. Non sono permessi sotterfugi come sussurrargli nell'orecchio o passargli un bigliettino.

Immaginiamo che il numero che volete comunicare sia 7, ed ecco come potreste procedere. Innanzitutto, provate a pensare un numero che Arnold conosce ma Eve no. Diciamo, per esempio, che voi e Arnold siete amici di vecchia data e da bambini vivevate nella stessa via. Entrambi, supponiamo, giocavate nel giardino di casa al 322 di Pleasant Street. Supponiamo anche che Eve non vi conoscesse quando eravate bambini e, in particolare, che non conosca il numero civico della casa in cui voi e Arnold giocavate sempre. Allora potete dire ad Arnold: "Ehi, Arnold, ricordi il numero civico della casa di Pleasant Street dove giocavamo sempre da bambini? Bene, allora se prendi quel numero e ci sommi quel numero di carta di credito a una cifra a cui sto pensando, il risultato è 329".

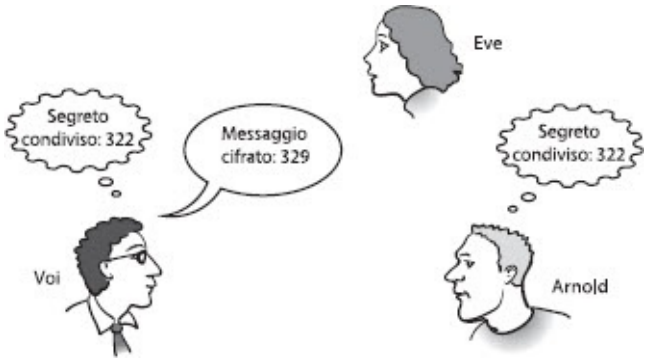


Figura 4.2 Il trucco dell'addizione: il messaggio 7 viene cifrato sommandolo al numero segreto condiviso, 322. Arnold può decifrarlo sottraendo il segreto condiviso, ma Eve no.

Ora, basta che Arnold ricordi correttamente il numero civico di quella casa e potrà ricavare il numero della carta di credito sottraendo il numero civico dal totale che gli avete comunicato, 329. Calcola 329 meno 322 e ottiene 7, che è il numero di carta di credito che volevate comunicargli. Eve invece non ha idea di quale sia il numero della carta di credito, nonostante abbia sentito benissimo tutto quello che avete detto ad Arnold. La Figura 4.2 visualizza tutto il procedimento.

Perché questo metodo funziona? Voi e Arnold avete quello che gli informatici definiscono un *segreto condiviso*: il numero 322. Entrambi conoscete questo numero, mentre Eve non lo conosce, perciò potete utilizzare il segreto condiviso per comunicare segretamente qualsiasi altro numero vogliate, semplicemente sommando e dichiarando il totale, mentre l'altra parte sottrarrà il segreto condiviso. Sapere quale sia il totale non serve a nulla a Eve, perché non sa quale sia il numero che deve sottrare.

Che ci crediate o meno, se avete capito questo semplice "trucco dell'addizione" di un segreto condiviso a un messaggio privato come un numero di carta di credito, allora sapete già quasi tutto su come funziona la crittografia su Internet! I computer usano continuamente questo trucco, ma perché sia davvero sicuro bisogna tener conto ancora di qualche particolare.

In primo luogo, i segreti condivisi di cui hanno bisogno i computer debbono essere molto più lunghi del numero civico 322. Se il segreto è troppo breve, un malintenzionato può semplicemente provare tutte le possibilità. Supponiamo per esempio di usare un numero civico a 3 cifre per cifrare un numero di carta *reale* a 16 cifre con il trucco dell'addizione. Ricordate che ci sono solo 999 numeri a tre cifre, perciò un avversario come Eve, che ha sentito la vostra conversazione, potrebbe preparare un elenco dei 999

numeri possibili, uno dei quali deve essere il numero della carta di credito. Un computer impiegherebbe un tempo brevissimo a provare tutti i 999 numeri di carta di credito, perciò dovremo usare un numero di cifre molto più grande perché il segreto condiviso possa essere utile.

In effetti, se sentite dire che un certo tipo di cifratura è “a un certo numero di bit”, come in “cifratura a 128 bit”, questa è effettivamente una descrizione della lunghezza del segreto condiviso. Si dice spesso che questo segreto condiviso è una “chiave”, poiché può essere usata per aprire, cioè per “decifrare” il messaggio. Se calcolate il 30% del numero dei bit della chiave, ottenete approssimativamente il numero di cifre della chiave. Poiché il 30% di 128 è circa 38, sappiamo che la crittografia a 128 bit usa una chiave che è un numero a 38 cifre[1]. Un numero a 38 cifre è più grande di un miliardo di miliardi di miliardi di miliardi e qualsiasi computer noto avrebbe bisogno di miliardi di anni per provare tutte le possibilità, perciò un segreto condiviso a 38 cifre è considerato molto sicuro.

C’è ancora un piccolo dettaglio che impedisce il funzionamento della versione semplice del trucco dell’addizione nella vita reale: l’addizione produce risultati che possono essere analizzati statisticamente, il che significa che qualcuno potrebbe immaginare quale sia la vostra chiave analizzando un gran numero dei vostri messaggi cifrati. Le tecniche crittografiche moderne, dette “cifrari a blocchi”, usano invece una variante del trucco dell’addizione.

In primo luogo, i messaggi lunghi vengono suddivisi in piccoli “blocchi” di dimensioni costanti, normalmente fra i 10 e i 15 caratteri. In secondo luogo, anziché semplicemente sommare un blocco del messaggio e la chiave, ciascun blocco viene trasformato varie volte secondo un insieme prestabilito di regole simili all’addizione ma in grado di “mescolare” meglio messaggio e chiave. Per esempio, una regola potrebbe dire qualcosa come “somma la prima metà della chiave alla seconda metà del blocco, inverti il risultato e somma la seconda metà della chiave alla prima metà del blocco”. In realtà le regole sono molto più complicate: i cifrari a blocco moderni in genere utilizzano 10 o più “round” di queste operazioni, il che significa che la successione delle operazioni indicate viene applicata più volte. Dopo un numero sufficiente di tornate, il messaggio originale è stato ben “rimescolato” e resisterà ad attacchi di tipo statistico, ma chiunque conosca la chiave potrà eseguire tutte le operazioni alla rovescia per ottenere il messaggio originale decifrato.

Nel momento in cui scrivo, il cifrari a blocchi più diffuso è l’Advanced Encryption Standard o AES. AES può essere utilizzato con varie impostazioni, ma un’applicazione normale può usare blocchi di 16 caratteri, con chiavi di 128 bit, e 10 tornate di operazioni di “rimescolamento”.

Fissare pubblicamente un segreto condiviso

Fin qui tutto bene. Abbiamo già visto in gran parte come funzioni effettivamente la crittografia in Internet: si spezza il messaggio in blocchi e si usa una variante del trucco dell’addizione per cifrare ciascun blocco. Ma questa è in realtà la parte facile: quella difficile è *fissare* un segreto condiviso. Nell’esempio di prima, quando eravate in una stanza con Arnold ed Eve, abbiamo barato un po’: abbiamo sfruttato il fatto che da

bambini voi e Arnold foste compagni di giochi e aveste già un segreto condiviso (il numero civico della casa in cui giocavate), che Eve non poteva conoscere. Ma se voi, Arnold ed Eve foste tutti estranei, e tentassimo di ripetere lo stesso esperimento? Esiste un modo in cui voi e Arnold potreste accordarvi su un segreto condiviso senza che Eve lo sappia? (Ricordate, niente sotterfugi, non potete sussurrare qualcosa nell'orecchio di Arnold o passargli un bigliettino che Eve non può vedere. Tutte le comunicazioni devono essere pubbliche.)

A prima vista sembrerebbe impossibile, invece esiste un modo ingegnoso per risolvere il problema. Gli informatici lo chiamano *scambio di chiavi Diffie-Hellman*, ma noi lo chiameremo *trucco delle vernici mescolate*.

Il trucco delle vernici mescolate

Per capire questo trucco, dobbiamo dimenticarci per un po' dei numeri di carta di credito e immaginare invece che il segreto che vogliate condividere sia un particolare colore di vernice. (Sì, è un po' strano, ma, come vedremo presto, è un buon modo di ragionare sul problema.) Supponiamo che vi troviate in una stanza con Arnold ed Eve e che ciascuno di voi abbia una grande collezione di barattoli di vernice. Avete a disposizione le stesse scelte di colori: ci sono molti colori diversi e ciascuno di voi ha molti barattoli di ciascun colore, perciò non c'è il rischio che qualcuno possa esaurire le sue vernici. Ogni barattolo ha un'etichetta chiara che riporta il colore, perciò è facile dare ad altri istruzioni specifiche su come mescolare fra loro vari colori: potete dire qualcosa come "mescola un barattolo di 'azzurro cielo' con sei barattoli di 'guscio d'uovo' e cinque di 'acquamarina'". Ma ci sono centinaia o migliaia di tinte diverse, perciò è impossibile stabilire quali colori siano stati impiegati in una miscela semplicemente guardandola o procedendo per tentativi, semplicemente perché ci sono troppi colori possibili.

Ora le regole del gioco cambieranno un poco. Ciascuno di voi avrà un suo angolo della stanza separato, per ragioni di riservatezza: un luogo in cui potete conservare la vostra raccolta di vernici e in cui potrete segretamente mescolare le vernici senza che altri vi vedano. Le regole di comunicazione sono però le stesse di prima: qualsiasi comunicazione fra voi, Arnold ed Eve deve essere pubblica; non potete invitare Arnold nel vostro angolino privato! Un'altra regola stabilisce come potete condividere le miscele di vernici. Potete dare un lotto di vernice a una delle altre persone nella stanza, ma solo mettendo il contenitore per terra in mezzo alla stanza; poi dovete aspettare che qualcun altro lo raccolga. Questo significa che non potete mai essere sicuri di chi raccoglierà il vostro lotto di vernice. Il modo migliore è prepararne abbastanza per tutti, e lasciare vari contenitori in mezzo alla stanza, in modo che chiunque voglia averne un po' potrà prenderselo. Questa regola è semplicemente un'estensione del fatto che tutte le comunicazioni debbono essere pubbliche: se date una certa miscela ad Arnold senza darla anche a Eve, avreste una sorta di comunicazione "privata" con Arnold, il che è contro le regole.

Ricordate che questo gioco del mescolamento di vernici serve a spiegare come fissare un segreto condiviso. A questo punto di sicuro vi starete chiedendo che cosa diavolo abbiano a che fare le vernici mescolate con la crittografia, ma abbiate pazienza. State per scoprire un trucco stupefacente che è effettivamente utilizzato dai computer per fissare segreti condivisi in un luogo pubblico come Internet!

In primo luogo, dobbiamo conoscere l'obiettivo del gioco. L'obiettivo è che voi e Arnold produciate la *stessa* miscela di vernici, senza dire a Eve come produrla. Se ci riuscite, diremo che voi e Arnold avete fissato una "miscela segreta condivisa". Potete avere tutte le conversazioni pubbliche che volete, e potete anche portare barattoli di vernice avanti e indietro dal centro della stanza alla vostra area privata in cui preparare le vostre miscele.

Ora cominciamo il viaggio nelle idee ingegnose alla base della crittografia a chiave pubblica. Il nostro tucco delle vernici mescolate sarà suddiviso in quattro passi.

Passo 1. Voi e Arnold scegliete un "colore privato".

Il vostro colore privato non è la stessa cosa della miscela segreta condivisa che alla fine produrrete, ma sarà uno degli ingredienti della miscela. Potete scegliere come colore privato qualsiasi colore vogliate, ma dovete ricordarvelo. Ovviamente, il vostro colore privato sarà quasi sicuramente diverso da quello di Arnold, dato che ci sono così tanti colori fra cui scegliere. Per esempio, supponiamo che il vostro colore privato sia lavanda e quello di Arnold cremisi.

Passo 2. Uno di voi dichiara pubblicamente gli ingredienti di un nuovo, diverso colore, che chiameremo il "colore pubblico".

Anche in questo caso, potete scegliere quello che volete. Supponiamo dichiariate che il colore pubblico è giallo margherita. Notate che esiste solo un colore pubblico (non due distinti, uno per voi e uno per Arnold) e, ovviamente, Eve sa quale sia il colore pubblico, perché l'avete dichiarato pubblicamente.

Passo 3. Sia voi sia Arnold create una miscela combinando un barattolo del colore pubblico con un barattolo del vostro colore privato. Questo produce la vostra "miscela pubblica-privata".

Ovviamente, la miscela pubblica-privata di Arnold sarà diversa dalla vostra, perché il suo colore privato è diverso dal vostro. Se restiamo all'esempio precedente, la vostra miscela pubblica-privata conterrà un barattolo di color lavanda e uno di giallo margherita, mentre quella di Arnold sarà una miscela di cremisi e giallo margherita.

A questo punto, voi e Arnold vorrete scambiarvi dei campioni delle vostre miscele pubbliche-private, ma ricordate che è proibito dare direttamente una miscela a una delle altre persone nella stanza: l'unico modo è prepararne vari lotti e lasciarli in mezzo alla stanza, in modo che chiunque voglia possa prenderne un campione. Questo è esattamente quello che fate voi e Arnold: ciascuno di voi prepara vari lotti della propria miscela pubblica-privata e li lascia in mezzo alla stanza. Eve può prendersi un lotto o due, se vuole ma, come scoprirà presto, non le servirà a nulla. La Figura 4.3 illustra la situazione dopo questo terzo passo del trucco delle vernici mescolate.

Se ci pensate un po', a questo punto potete immaginare il trucco finale che consentirà a voi e ad Arnold di creare una miscela segreta condivisa identica, senza far trapelare il segreto a Eve. Ecco la risposta:

Passo 4. Prendete un lotto della miscela pubblica-privata di Arnold e la portate nel vostro angolo. Ora aggiungete un barattolo del vostro colore privato. Intanto Arnold

prende un lotto della *vostra* miscela pubblica-privata e se la porta nel suo angolino, dove vi aggiunge un barattolo del *suo* colore privato.

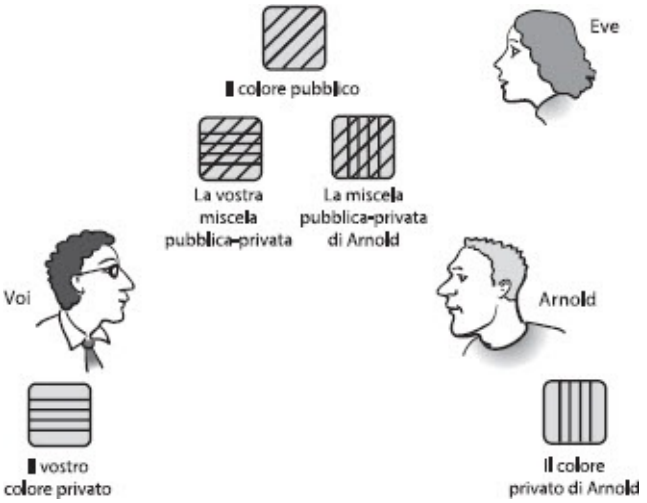


Figura 4.3 Il trucco delle vernici mescolate, passo 3: le miscele pubblicheprivate sono disponibili a chiunque le voglia.

Incredibile, avete creato due miscele identiche! Verifichiamolo: avete aggiunto il vostro colore privato (lavanda) alla miscela pubblica-privata di Arnold (cremisi e giallo margherita), così che la miscela finale è 1 lavanda, 1 cremisi, 1 giallo margherita. E la miscela finale di Arnold? Ha aggiunto il suo colore privato (cremisi) alla vostra miscela pubblica-privata (lavanda e giallo margherita), quindi la sua miscela finale è 1 cremisi, 1 lavanda, 1 giallo margherita. Esattamente uguale alla vostra. Davvero è una miscela segreta condivisa. La Figura 4.4 illustra la situazione dopo questo passo finale del trucco delle vernici mescolate.

Ed Eve? Perché non può creare un lotto di questa miscela segreta condivisa? Il motivo è che non sa quale sia il vostro colore privato né quello privato di Arnold e ha bisogno di almeno uno dei due per riprodurre la miscela segreta condivisa. Voi e Arnold l'avete tagliata fuori, perché non avete mai esposto da soli i vostri colori privati, in mezzo alla stanza. Ciascuno di voi invece ha combinato il proprio colore privato con il colore pubblico prima di esporlo, ed Eve non ha modo di "separare nelle loro componenti" le miscele pubbliche-private, così da ottenere un campione puro di uno dei colori privati.

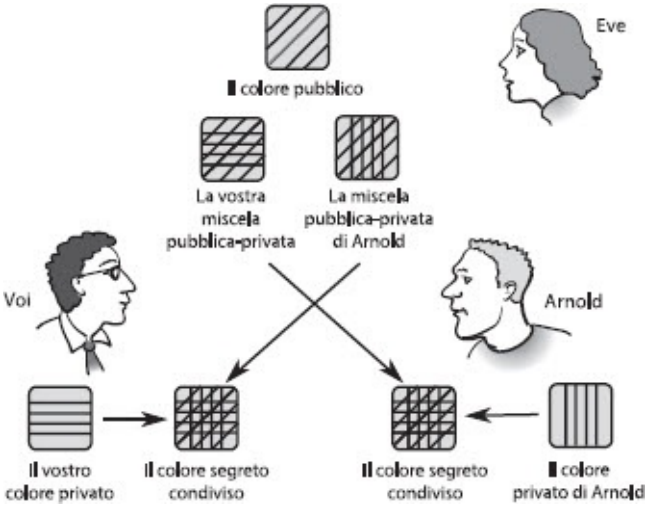


Figura 4.4 Il trucco delle vernici mescolate, passo 4: solo voi e Arnold potete produrre il colore segreto condiviso, combinando le miscele indicate dalle frecce.

Eve, quindi, ha accesso *solo* alle due miscele pubbliche-private. Se mescola un lotto

della vostra miscela pubblica-privata con un lotto della miscela pubblica-privata di Arnold, il risultato conterrà 1 cremisi, 1 lavanda e 2 giallo-margherita. In altre parole, rispetto alla miscela segreta condivisa, la miscela di Eve ha una parte di giallo margherita in più. La sua miscela ha troppo giallo e, poiché non c'è modo di "separare" le vernici, non può eliminare quel giallo di troppo. Potreste pensare che Eve aggiri il problema aggiungendo più cremisi e più lavanda, ma ricordate che non sa quali siano i vostri colori privati, perciò non saprà che i colori da aggiungere sono proprio quelli. Può solo mescolare la *combinazione* di cremisi più giallo margherita o di lavanda più giallo margherita, e in ogni caso la sua miscela conterrà troppo giallo.

Mescolare vernici con i numeri

Se vi è chiaro il trucco delle vernici mescolate, avete capito l'essenziale di come i computer fissano segreti condivisi in Internet. Qui, ovviamente, non si usano vernici: i computer usano numeri, e per mescolare i numeri usano la matematica. La matematica che usano effettivamente non è troppo complicata, ma lo è abbastanza da lasciare un po' confusi, a prima vista. Perciò, per il nostro prossimo passo verso la comprensione di come si fissano segreti condivisi in Internet, useremo un po' di "finta" matematica. Il punto è che, per tradurre in numeri il trucco delle vernici mescolate, ci serve una *azione a senso unico*. Qualcosa che possa essere *fatto*, ma non *disfatto*. Nel trucco delle vernici l'azione a senso unico era "mescolare le vernici". È facile mescolare delle vernici e ottenere un nuovo colore, ma è impossibile "scomporre" la miscela e riottenere i colori originali. Per questo miscelare vernici è un'azione a senso unico.

Abbiamo detto che useremo della "finta" matematica. Ecco di che cosa faremo finta: che *moltiplicare due numeri è un'azione a senso unico*. Vi rendete conto di sicuro che dobbiamo fare finta: l'inverso della moltiplicazione è la divisione, ed è facile "disfare" una moltiplicazione semplicemente eseguendo una divisione. Per esempio, se partiamo con il numero 5 e poi lo moltiplichiamo per 7, otteniamo 35. È facile "disfare" questa moltiplicazione partendo da 35 e dividendo per 7, il che ci ridà il 5 da cui eravamo partiti.

Comunque, continueremo a fare finta e proveremo a fare un altro gioco con voi, Arnold e Eve. Questa volta, assumiamo che tutti conosciate la moltiplicazione, ma nessuno sappia invece come dividere un numero per un altro. L'obiettivo è simile a quello di prima: voi e Arnold cercate di fissare un segreto condiviso, ma questa volta sarà un numero anziché un colore. Valgono le solite regole: tutte le comunicazioni debbono essere pubbliche, perciò Eve potrà sentire tutte le conversazioni fra voi e Arnold.

Bene, tutto quello che dobbiamo fare adesso è tradurre il trucco delle vernici mescolate nel mondo dei numeri:

Passo 1. Anziché scegliere un "colore privato", voi e Arnold scegliete un "numero privato".

Diciamo che voi scegliete il 4 e Arnold sceglie il 6. Ora ripensate agli altri passaggi del trucco delle vernici mescolate: dichiarare il colore pubblico, preparare una miscela pubblica-privata, scambiare la miscela pubblica-privata con quella di Arnold e infine aggiungere il colore privato alla miscela pubblica-privata di Arnold per ottenere il colore segreto condiviso. Non dovrebbe essere difficile vedere come tradurre tutto in numeri, utilizzando la moltiplicazione come azione a senso unico al posto della miscela di vernici.

Prendetevi un paio di minuti per vedere se riuscite a ricostruire l’esempio da soli, prima di continuare a leggere.

La soluzione non è difficile da seguire; avete entrambi scelto i vostri numeri privati (4 e 6), perciò il passo successivo è

Passo 2. Uno di voi annuncia un “numero pubblico” (invece del colore pubblico nel trucco delle vernici mescolate).

Diciamo che scegliate come numero pubblico il 7.

Il passo seguente nel trucco delle vernici era creare una miscela pubblica-privata. Ma abbiamo deciso che invece di miscelare vernici avreste moltiplicato numeri. Perciò tutto quello che dovete fare è

Passo 3. Moltiplicate il vostro numero privato (4) e il numero pubblico (7) per ottenere il vostro “numero pubblico-privato”, 28.

Potete dichiararlo pubblicamente, in modo che Arnold ed Eve sappiano entrambi che il vostro numero pubblico-privato è 28 (non c’è più bisogno di portare in giro barattoli di vernice). Arnold fa la stessa cosa con il suo numero privato: lo moltiplica per il numero pubblico e dichiara il suo numero pubblico-privato, che è 6 × 7, ovvero 42. La Figura 4.5 illustra la situazione a questo punto del procedimento.

Ricordate l’ultimo passo del trucco delle vernici? Prendevate la miscela pubblica-privata di Arnold e ci aggiungevate un barattolo del vostro colore privato per produrre il colore segreto condiviso. Qui succede esattamente la stessa cosa, con la moltiplicazione al posto della miscelazione di vernici:

Passo 4. Prendete il numero pubblico-privato di Arnold, che è 42, è lo moltiplicate per il vostro numero privato 4, e il risultato è il *numero segreto condiviso*, 168.

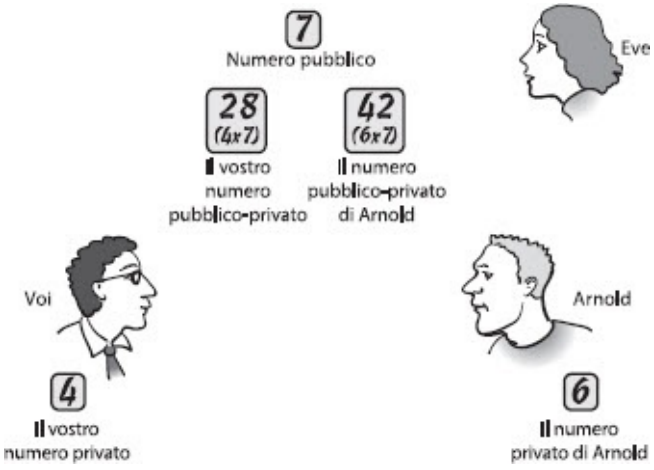


Figura 4.5 Il trucco della miscelazione dei numeri, passo 3: i numeri pubbliciprivati sono disponibili per chiunque li voglia.

Intanto Arnold prende il *vostro* numero pubblico-privato, 28, e lo moltiplica per il *suo* numero privato e, meraviglia!, ottiene lo stesso numero segreto condiviso, poiché 28 × 6 = 168. Il risultato finale è illustrato nella Figura 4.6.

In realtà, se ci pensate bene, non c’è nulla di cui meravigliarsi. Quando Arnold e voi siete riusciti a produrre lo stesso colore segreto condiviso, è stato perché avete mescolato fra loro gli stessi tre colori originali, ma in ordine diverso: ciascuno di voi ha mantenuto

privato uno dei colori, combinandolo con una miscela, pubblicamente disponibile, degli altri due. Lo stesso è successo qui con i numeri. Siete arrivati entrambi allo stesso segreto condiviso moltiplicando gli stessi tre numeri: 4, 6, 7. (Sì, come potete verificare, 4 × 6 × 7 = 168.) Ma *voi* ci siete arrivati mantenendo privato il 4 e "mescolandolo" (cioè, moltiplicandolo) con la miscela pubblicamente disponibile di 6 e 7 (cioè 42) che era stata dichiarata da Arnold. Invece, *Arnold* è arrivato al segreto condiviso mantenendo privato il 6 e mescolandolo con la miscela disponibile pubblicamente di 4 e 7 (cioè 28) che avevate dichiarato voi.

Come abbiamo fatto nel trucco della miscela di vernici, verifichiamo che Eve non abbia modo di stabilire il segreto condiviso. Eve sente il valore di ciascun numero pubblico-privato quando viene dichiarato, perciò sente voi dire "28" e Arnold dire "42". Sa anche qual è il numero pubblico, che è 7. Perciò se *Eve* sapesse fare le divisioni, potrebbe ricavare immediatamente i vostri segreti, semplicemente osservando che 28 : 7 = 4 e 42 : 7 = 6. E poi potrebbe ricavare il segreto condiviso calcolando 4 × 6 × 7 = 168. Per fortuna, però, abbiamo deciso di usare una matematica di fantasia in questo gioco: abbiamo assunto che la moltiplicazione sia un'azione a senso unico e pertanto Eve *non* sa come dividere e dei numeri 28, 42 e 7 non sa che farsene. Può moltiplicarli fra loro, ma questo non le dice nulla del segreto condiviso. Per esempio, se prende 28 × 42 = 1176, è fuori strada. Come nel gioco delle vernici il suo risultato aveva un eccesso di giallo, qui il suo risultato ha troppi 7. Il segreto condiviso ha un solo fattore 7, poiché 168 = 4 × 6 × 7, ma il tentativo di Eve di determinare il segreto ha due fattori 7, poiché 1176 = 4 × 6 × 7 × 7. E non ha modo di liberarsi di quel 7 di troppo, perché non sa come fare le divisioni.

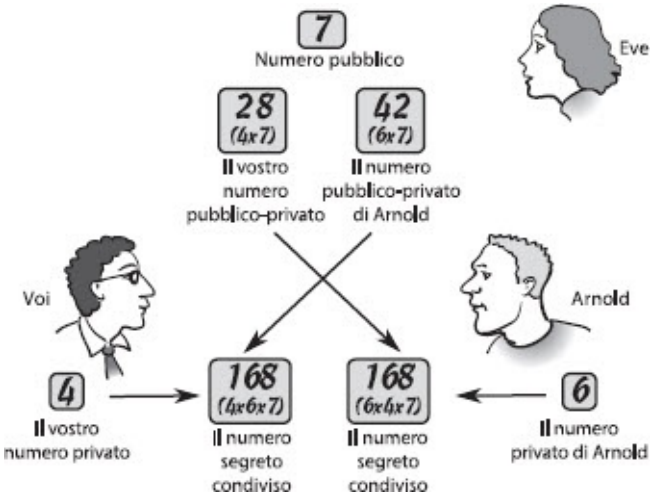


Figura 4.6 Il trucco della miscelazione dei numeri, passo 4: solo voi e Arnold potete costruire il numero segreto condiviso, moltiplicando i numeri indicati dalle frecce.

Miscele di vernici nella vita reale

Abbiamo visto tutti i concetti fondamentali necessari ai computer per fissare segreti condivisi in Internet. L'unico punto debole del procedimento è che usa una "finta" matematica, in cui abbiamo fatto finta che nessuna delle persone coinvolte sappia fare le divisioni. Per completare la ricetta, ci serve un'operazione matematica reale che sia facile da eseguire (come miscelare le vernici) ma praticamente impossibile da "disfare" (come il separare le vernici componenti). Nella realtà dei computer, l'operazione di miscelazione è il cosiddetto *elevamento a potenza discreta* e l'operazione inversa è il *logaritmo discreto*. Poiché non esiste alcun metodo noto grazie al quale un computer possa calcolare in modo

efficiente i logaritmi discreti, l'elevamento a potenza discreta è proprio il tipo di azione a senso unico che ci serve. Per spiegare bene l'elevamento a potenza discreta, ci servono due semplici idee matematiche, e dovremo anche scrivere qualche formula. Se non vi piacciono le formule, potete saltare il resto del paragrafo: sapete già quasi tutto sull'argomento. Se invece volete sapere veramente come fanno i computer a compiere questa magia, continuate a leggere.

La prima idea matematica importante che ci serve è l'*aritmetica dell'orologio*, una cosa che in realtà conosciamo tutti bene: ci sono solo 12 numeri in un orologio, perciò ogni volta che la lancetta delle ore supera il 12, ricomincia a contare da 1. Un'attività che inizia alle 10 e dura 4 ore finisce alle 2, perciò possiamo dire che nel sistema di un orologio a 12 ore 10 + 4 = 2. In matematica, l'aritmetica dell'orologio funziona nello stesso modo, tranne per due particolari: (i) la dimensione dell'orologio può essere qualsiasi numero (e non soltanto il 12 dei nostri comuni orologi) e (ii) si comincia a contare da 0 anziché da 1.

La Figura 4.7 presenta un esempio con un orologio di dimensione 7. Notate che i numeri sull'orologio sono 0, 1, 2, 3, 4, 5 e 6. Per l'aritmetica di un orologio di dimensione 7, si sommano e si moltiplicano i numeri come al solito, ma ogni volta che si genera una risposta si conserva solo il *resto* dopo aver diviso per 7. Perciò, per calcolare 12 + 6, prima sommiamo come al solito, ottenendo 18. Poi notiamo che il 7 in 18 sta due volte (il che fa 14) con il resto di 4. Perciò la risposta finale è

$$12 + 6 = 4 \text{ (orologio di dimensione 7)}$$

Nell'esempio seguente, usiamo 11 come dimensione dell'orologio. (Come vedremo poi, le dimensioni dell'orologio in una implementazione reale sarebbero molto, molto più grandi. Usiamo numeri piccoli per semplicità.) Prendere il resto dopo aver diviso per 11 non è difficile, poiché tutti i multipli di 11 hanno cifre ripetute, come 66 e 88. Ecco qualche esempio di calcoli con un orologio di dimensione 11:

$$7 + 9 + 8 = 24 = 2 \text{ (orologio di dimensione 11)}$$

$$8 \times 7 = 56 = 1 \text{ (orologio di dimensione 11)}$$

La seconda idea matematica che ci serve è la *notazione delle potenze*. Nulla di strano: è solo un modo rapido per indicare il prodotto di molti fattori uguali. Invece di scrivere 6 × 6 × 6× 6, che è il prodotto di quattro fattori tutti uguali a 6, possiamo scrivere 6^4. E si possono combinare le potenze con l'aritmetica dell'orologio. Per esempio:

$$3^4 = 3 \times 3 \times 3 \times 3 = 81 = 4 \text{ (orologio di dimensione 11)}$$

$$7^2 = 7 \times 7 = 49 = 5 \text{ (orologio di dimensione 11)}$$

Figura 4.7 A sinistra: quando si usa un orologio di dimensione 7, il numero 12 diventa 5; basta partire da zero e contare 12 unità in senso orario, come indica la freccia. A destra: usando ancora un orologio di dimensione 7, vediamo che 12 + 6 = 4; partendo da 5, dove siamo arrivati nell'illustrazione di sinistra, aggiungiamo ancora 6 unità in senso orario.

La Tabella 4.1 elenca le prime dieci potenze di 2, 3 e 6 rispetto a un orologio di dimensione 11. Saranno utili per l'esempio che faremo fra poco. Prima di procedere, dunque, verificate che vi sia ben chiaro come è stata generata la tabella. Diamo un'occhiata all'ultima colonna. Il primo elemento della colonna è 6, che è quanto dire 6^1. L'elemento successivo è 6^2, cioè 36; ma, siccome usiamo un orologio di dimensione 11 e 36 è 33 + 3, l'elemento riportato in tabella è un 3. Per calcolare il terzo elemento della colonna, pensate che si debba calcolare $6^3 = 6 \times 6 \times 6$, ma c'è un metodo più semplice. Abbiamo già calcolato 6^2 per le dimensioni di orologio che ci interessano, e abbiamo visto che è 3. Per ottenere 6^3, basta moltiplicare il risultato precedente per 6. Questo ci dà $3 \times 6 = 18 = 7$ (orologio di dimensione 11). E l'elemento successivo è $7 \times 6 = 42 = 9$ (orologio di dimensione 11) e così via fino alla fine della colonna.

Tabella 4.1 La tabella elenca le prime dieci potenze di 2, 3 e 6 utilizzando un orologio di dimensione 11. Come spiegato nel testo, ciascun elemento può essere calcolato a partire da quello della riga precedente nella stessa colonna mediante semplicissime operazioni aritmetiche.

n	2^n	3^n	6^n
1	2	3	6
2	4	9	3
3	8	5	7
4	5	4	9
5	10	1	10
6	9	3	5
7	7	9	8
8	3	5	4
9	6	4	2
10	1	1	1

Bene, siamo pronti finalmente a fissare un segreto condiviso, come fanno i computer nella vita reale. Come al solito, voi e Arnold cercherete di condividere un segreto, mentre Eve vi spia e cerca di scoprire che cos'è il segreto.

Passo 1. Voi e Arnold scegliete indipendentemente un *numero privato*.

Per avere calcoli semplici, useremo numeri molto piccoli in questo esempio. Supponiamo dunque che voi scegliete come numero privato 8, e che Arnold scelga 9. I due numeri 8 e 9 non sono in sé segreti condivisi, ma sono come i colori privati nel trucco delle vernici: verranno usati come *ingredienti* per "miscelare" un segreto condiviso.

Passo 2. Voi e Arnold vi accordate pubblicamente su due *numeri pubblici*: una dimensione di orologio (useremo 11 in questo esempio) e un altro numero, detto la *base* (useremo la base 2).

Questi numeri pubblici, 11 e 2, sono analoghi al colore pubblico che voi e Arnold avete concordato all'inizio del trucco delle vernici. Notate che qui l'analogia cede un po':

mentre là ci bastava un solo colore pubblico, qui sono necessari due numeri pubblici.

Passo 3. Voi e Arnold create separatamente un *numero pubblico-privato* (PPN), mescolando il vostro numero privato con i numeri pubblici, usando le potenze e l'aritmetica dell'orologio.

Specificamente, il "mescolamento" avviene secondo la formula

$$\text{PPN} = \text{base}^{\text{numero privato}} \text{ (dimensioni dell'orologio)}$$

Scritta a parole la formula può sembrare strana, ma nella pratica è molto semplice. Nel nostro esempio, possiamo calcolare le risposte consultando la Tabella 4.1:

$$\text{voi PPN} = 2^8 = 3 \text{ (orologio di dimensione 11)}$$

$$\text{Arnold PPN} = 2^9 = 6 \text{ (orologio di dimensione 11)}$$

Potete vedere la situazione dopo questo passo nella Figura 4.8. Questi numeri pubblici-privati sono esattamente analoghi alle "miscele pubbliche-private" create nel terzo passo del trucco delle vernici. Là avete mescolato un barattolo di colore pubblico una parte del vostro colore privato per creare la vostra miscela pubblica-privata. Qui avete mescolato il vostro numero privato con i numeri pubblici utilizzando le potenze e l'aritmetica dell'orologio.

Passo 4. Voi e Arnold separatamente prendete il numero pubblico-privato dell'altro e lo mescolate ciascuno con il proprio numero privato.

Questo viene fatto in base alla formula

Segreto condiviso =

$$= \text{PPN dell'altro}^{\text{numero privato}} \text{ (dimensione dell'orologio)}$$

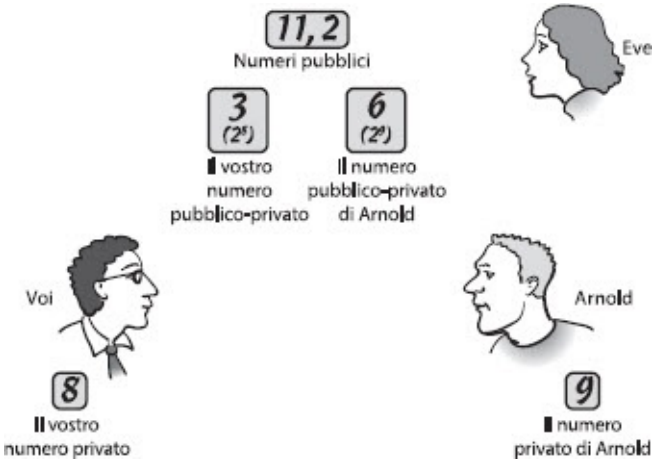


Figura 4.8 Miscelazione dei numeri nella vita reale, passo 3: i numeri pubblici-privati (3 e 6), calcolati con le potenze e l'aritmetica dell'orologio, sono disponibili a chiunque voglia usarli. Il "2^8" mostrato sotto il 3 ci ricorda come è stato calcolato il 3, ma il fatto che 3 = 2^8 in un orologio di dimensione 11 non è reso pubblico. Analogamente, il "2^9" sotto il 6 rimane privato.

Ancora una volta, la formula sembra un po' arcana scritta a parole, ma consultando la tabella i numeri sono in realtà molto semplici:

$$\text{Il vostro segreto condiviso} = 6^8 =$$

$$= 4 \text{ (orologio di dimensione 11)}$$

Il segreto condiviso di Arnold $= 3^9 =$

$= 4$ (orologio di dimensione 11)

La situazione finale è rappresentata nella Figura 4.9.

Naturalmente, il vostro segreto condiviso e il segreto condiviso di Arnold finiscono per essere lo stesso numero (in questo caso 4). Dipende da qualche complessità matematica in più, in questo caso, ma l'idea di fondo è la stessa di prima: anche se avete mescolato i vostri ingredienti in ordine diverso, sia voi sia Arnold avete usato gli stessi ingredienti e perciò avete ottenuto lo stesso segreto condiviso.

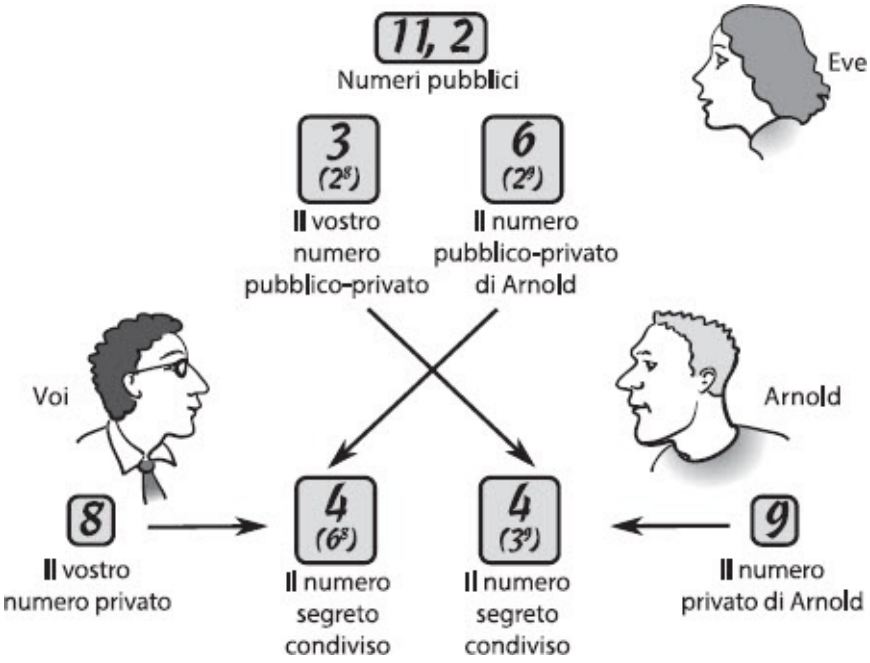


Figura 4.9 Miscelazione dei numeri nella vita reale, passo 4: solo voi e Arnold potete costruire il numero segreto condiviso, combinando gli elementi indicati dalle frecce, con le potenze e l'aritmetica dell'orologio.

E, come nelle versioni precedenti di questo trucco, Eve rimane scornata. Sa i due numeri pubblici (2 e 11) e sa anche i due numeri pubblici-privati (3 e 6), ma non può usare quello che sa per calcolare il numero segreto condiviso, perché non ha accesso a nessuno dei due ingredienti segreti (i numeri privati) che conservate voi e Arnold.

La crittografia pubblica in pratica

La versione finale del trucco delle vernici mescolate, in cui si mescolano numeri sfruttando le potenze e l'aritmetica dell'orologio, è uno dei modi in cui i computer stabiliscono realmente segreti condivisi in Internet. Il metodo particolare descritto qui è detto protocollo di scambio di chiavi Diffie-Hellman, dal nome di Whitfield Diffie e Martin Hellman, i primi a pubblicare l'algoritmo nel 1976. Ogni volta che entrate in un sito web sicuro (il cui indirizzo inizia con "https:" invece che con "http:"), il vostro computer e il server web con cui comunica creano un segreto condiviso, utilizzando il protocollo di Diffie-Hellman o una delle tante alternative simili. Una volta fissato il segreto condiviso, i due computer possono cifrare tutte le loro comunicazioni mediante una variante del trucco dell'addizione visto prima.

È importante precisare che, quando il protocollo di Diffie-Hellman si usa nella pratica, i numeri in gioco sono di gran lunga maggiori rispetto a quelli dei nostri esempi. Abbiamo usato un orologio di dimensione molto piccola (11), perché i calcoli fossero facili. Ma se

sceglieste un orologio pubblico di piccola dimensione anche il numero dei numeri privati possibili sarebbe piccolo (poiché si possono usare solo numeri privati minori della dimensione dell'orologio) e questo significa che qualcuno potrebbe usare un computer per provare tutti i numeri privati possibili fino a scoprire quello che produce il vostro numero pubblico-privato. Nel nostro esempio, ci sono solo 11 numeri privati possibili, perciò sarebbe un gioco da ragazzi penetrarlo. Invece le implementazioni reali del protocollo di Diffie-Hellman usano normalmente dimensioni di orologio che sono lunghe qualche centinaio di cifre, il che rende possibile un numero davvero enorme di numeri privati (molto più di un miliardo di miliardi di miliardi). Anche in quel caso, però, bisogna scegliere i numeri pubblici con una certa attenzione, perché abbiano le proprietà matematiche giuste: leggete il riquadro qui sotto, se siete interessati ad approfondire.

La proprietà più importante per i numeri pubblici di Diffie-Hellman è che la dimensione dell'orologio deve essere un numero primo, un numero cioè che non abbia altri divisori oltre a se stesso e a 1. Un altro requisito particolare è che la base deve essere una *radice primitiva* delle dimensioni dell'orologio. Questo significa che le potenze della base percorreranno tutti i possibili valori dell'orologio. Se osservate la Tabella 4.1, potete notare che 2 e 6 sono radici primitive di 11, ma 3 non lo è: le potenze di 3 percorrono i valori 3, 9, 5, 4, 1, ma non sono mai 2, 6, 7, 8 e 10.

Quando si scelgono una dimensione di orologio e una base per il protocollo di Diffie-Hellman, debbono essere soddisfatte alcune particolari proprietà matematiche.

Il metodo di Diffie-Hellman che abbiamo descritto è solo una delle molte tecniche brillanti per comunicare attraverso cartoline (elettroniche). Gli informatici definiscono quello di Diffie-Hellman un *algoritmo a scambio di chiavi*. Altri algoritmi a chiave pubblica funzionano in modo diverso e consentono di cifrare direttamente un messaggio per il destinatario, sfruttando informazioni pubbliche dichiarate da quel destinatario. Invece, un algoritmo a scambio di chiavi consente di fissare un segreto condiviso utilizzando le informazioni pubbliche fornite dal destinatario, ma la cifratura in sé viene effettuata mediante il trucco dell'addizione. Per la maggior parte delle comunicazioni in Internet, quest'ultima strada (quella che abbiamo visto in questo capitolo) è preferibile, perché richiede molta meno potenza di calcolo.

Ci sono però alcune applicazioni in cui è necessaria la crittografia a chiave pubblica nella sua completezza. Forse la più interessante fra queste applicazioni è la firma digitale, che spiegheremo nel Capitolo 9. Come scoprirete leggendo quel capitolo, il gusto delle idee nella crittografia a chiave pubblica di tipo completo è simile a quello che abbiamo già assaporato: l'informazione segreta viene "mescolata" con informazioni pubbliche in un modo matematicamente irreversibile, come le vernici colorate si mescolano in modo irreversibile. Il più famoso dei sistemi crittografici a chiave pubblica è quello noto come RSA, dal nome dei tre ricercatori che l'hanno pubblicato per primi: Ronald Rivest, Adi Shamir e Leonard Adleman. Nel capitolo 9 useremo l'RSA come esempio di funzionamento delle firme digitali.

La storia dell'invenzione di questi algoritmi a chiave pubblica è complessa e affascinante. Diffie e Hellman sono stati effettivamente i primi a pubblicare l'algoritmo di Diffie-Hellman, nel 1976, e Rivest, Shamir e Adleman sono stati effettivamente i primi a pubblica l'RSA nel 1978. Ma non finisce qui! Si è poi scoperto che il governo inglese

conosceva già sistemi simili da parecchi anni. Purtroppo gli inventori di quei precursori di Diffie-Hellman e RSA erano matematici che lavoravano al GCHQ, il laboratorio di comunicazioni del governo inglese. Perciò i risultati del loro lavoro sono rimasti sepolti in documenti coperti dal segreto e sono stati resi pubblici solo nel 1997.

RSA, Diffie-Hellman e altri sistemi crittografici a chiave pubblica non sono solo idee ingegnose. Si sono evoluti in tecnologie commerciali e in standard di Internet di grande importanza per le aziende così come per i singoli. La stragrande maggioranza delle transazioni che compiamo online ogni giorno non si potrebbero eseguire in sicurezza senza la crittografia a chiave pubblica. Gli inventori dell'RSA hanno brevettato il loro sistema negli anni Settanta, e il loro brevetto è scaduto solo verso la fine del 2000. La notte della scadenza del brevetto si è tenuto a San Francisco un party di celebrazione nella Great American Music Hall, una celebrazione forse del fatto che la crittografia a chiave pubblica resterà a lungo con noi.

---

1 Per chi conosce i sistemi numerici dell'informatica, qui sto parlando di cifre decimali, non di cifre binarie (bit). Per chi conosce i logaritmi, il fattore di conversione 30% per trasformare i bit in cifre decimali deriva dal fatto che $\log_{10}2 = 0{,}3$.

CAPITOLO 5

Codici a correzione di errore: sbagli che si aggiustano da soli

> Ma una cosa è dimostrare a un uomo che è in errore, e altra cosa metterlo in possesso della verità.
>
> John Locke, *Essay Concerning Human Understanding* (1690)[1]

Siamo ormai abituati ad avere a disposizione un computer ogni volta che ci serve, ma Richard Hamming, ricercatore ai laboratori della Bell Telephone Company negli anni Quaranta non era altrettanto fortunato: il calcolatore dell'azienda che gli serviva era utilizzato da altri reparti ed era a sua disposizione solo nei fine settimana. Potete immaginare la sua frustrazione, perciò, quando si continuavano a verificare crolli del sistema dovuti a errori nella lettura dei suoi dati. Ecco quello che ne diceva lo stesso Hamming:

> Per due fine settimana di fila sono venuto e ho trovato che tutto il mio materiale era stato scaricato e non era stato fatto nulla. Ero davvero irritato e seccato perché volevo quelle risposte e avevo perso due fine settimana. E così mi sono detto, "Dannazione, se la macchina può capire che c'è un errore, perché non può identificare la posizione dell'errore e correggerlo?"

È difficile trovare un caso altrettanto palese di difficoltà che aguzza l'ingegno, Hamming rapidamente creò il primo *codice a correzione d'errore* in assoluto: un algoritmo apparentemente magico che identifica e corregge gli errori nei dati del calcolatore. Senza questi codici, i nostri sistemi di elaborazione e di comunicazione sarebbero drasticamente più lenti, meno potenti e meno affidabili di quel che sono in realtà.

La necessità di rilevare e correggere gli errori

I computer hanno tre compiti fondamentali. Il più importante è eseguire elaborazioni, cioè, forniti certi dati di ingresso, devono trasformarli in qualche modo per produrre una risposta utile. Ma la capacità di calcolare risposte sarebbe essenzialmente inutile senza le altre due attività fondamentali che i computer svolgono: memorizzare e trasmettere i dati. (Oggi i computer per lo più conservano i dati nella loro memoria e su unità a disco e normalmente trasmettono i dati via Internet.) Provate a immaginare un computer che non possa né conservare né trasmettere informazioni: sarebbe, ovviamente, pressoché inutile. Sì, potrebbe svolgere qualche calcolo complesso (per esempio preparare un foglio di calcolo finanziario in cui è definito il budget di un'azienda), ma poi non sareste in grado di spedire i risultati a un collega o addirittura nemmeno di salvarli per poterli rivedere in seguito. In breve, trasmissione e memorizzazione dei dati sono veramente essenziali per i calcolatori moderni.

Ma a queste funzioni è associata una sfida enorme: i dati debbono essere *assolutamente esatti*, perché in molti casi anche un minuscolo errore può rendere inutili i dati. Anche noi esseri umani conosciamo bene la necessità di conservare e trasmettere informazioni senza errori. Se scrivete il numero di telefono di qualcuno, è essenziale che ogni cifra sia scritta correttamente, e che le cifre siano nell'ordine giusto. Se ci fosse anche un solo errore in una delle cifre, il numero di telefono sarebbe con tutta probabilità del tutto inutile a voi e a chiunque altro. In qualche caso, gli errori nei dati possono essere addirittura *peggio* che inutili. Per esempio, un errore in un file che contiene un programma può far sì che il programma vada in tilt o faccia cose diverse da quelle per cui era stato pensato. (Potrebbe addirittura cancellare documenti importanti o andare in tilt prima che abbiate la possibilità di salvare il lavoro svolto.) E un errore in qualche registrazione finanziaria può generare una effettiva perdita di denaro (se, poniamo, pensavate di acquistare delle azioni al prezzo di 5,34 euro l'una, mentre il valore reale era di 8,34 euro).

La quantità di informazioni prive di errori che noi esseri umani abbiamo bisogno di memorizzare è relativamente piccola, e non è difficile evitare gli errori semplicemente controllando con attenzione quando si sa che certe informazioni sono importanti, per esempio numeri di conto bancario, password, indirizzi di posta elettronica e simili. Invece, la quantità di informazioni che i calcolatori debbono memorizzare e trasmettere senza compiere errori è assolutamente immensa. Per avere un'idea delle dimensioni, supponiamo che abbiate qualche tipo di dispositivo informatico con una capacità di memorizzazione di 100 gigabyte (nel momento in cui scrivo, è la capacità normale di un laptop di basso costo). Questi 100 gigabyte equivalgono a circa 15 milioni di pagine di testo, perciò se il sistema di memoria della macchina facesse anche solo un errore ogni milione di pagine, ci sarebbero comunque (in media) 15 errori nel dispositivo, una volta che fosse completamente pieno. E lo stesso vale per la trasmissione di dati: se scaricate un programma che occupa 20 megabyte e il vostro calcolatore interpretasse erroneamente anche un solo carattere ogni milione di caratteri ricevuti, ci sarebbero probabilmente ancora più di 20 errori nel programma scaricato, ciascuno dei quali potrebbe provocare un "crash" potenzialmente costoso quando meno ve lo aspettate.

La morale è che, per un calcolatore, essere accurato per il 99,9999% del tempo è tutt'altro che soddisfacente. I calcolatori debbono essere in grado di memorizzare e trasmettere letteralmente miliardi di pezzi di informazione senza compiere un solo errore. Però debbono fare i conti con i problemi di comunicazione, come qualsiasi altro dispositivo. I telefoni sono un buon esempio: è ovvio che non trasmettono le informazioni alla perfezione, perché le conversazioni telefoniche spesso sono disturbate da scariche o altri tipi di rumore. Ma i telefoni non sono gli unici apparecchi ad avere problemi del genere: i fili elettrici sono soggetti a ogni genere di fluttuazione; le comunicazioni *wireless* sono costantemente soggette a interferenze; supporti fisici come dischi magnetici, CD e DVD possono essere rigati, danneggiati o semplicemente letti male a causa della polvere o di altre interferenze fisiche. Come si può sperare di raggiungere un tasso di errore inferiore a uno su molti miliardi, a dispetto di simili ovvi errori di comunicazione? Questo capitolo esplora le ingegnose idee informatiche che permettono la magia. Se si usano i trucchi giusti, anche canali di comunicazione estremamente inaffidabili possono essere utilizzati per trasmettere dati con tassi di errore incredibilmente bassi – così bassi che, in pratica, gli errori possono essere completamente eliminati.

Il trucco della ripetizione

Il trucco fondamentale per comunicare in modo affidabile su un canale inaffidabile è ben noto a tutti: per essere sicuri che le informazioni siano state comunicate correttamente, bisogna ripeterle un po' di volte. Se qualcuno vi detta un numero di telefono o un numero di conto bancario in una telefonata disturbata, probabilmente gli chiederete di ripetere almeno una volta, per essere sicuri che non ci siano errori.

I computer possono fare esattamente la stessa cosa. Supponiamo che un calcolatore della banca stia cercando di trasmettervi il saldo del vostro conto via Internet. Il saldo è 5213,75 euro, ma purtroppo la rete non è molto affidabile e c'è un 20% di probabilità che ogni singola cifra venga trasformata in qualche altra cosa. Così, la prima volta che viene trasmesso può darsi che il saldo risulti 5293,75 euro. Ovviamente, non sapete se la cifra sia corretta o meno: può anche darsi che tutte le cifre siano giuste, ma può anche essere che una o più siano errate e non avete modo di stabilirlo. Utilizzando il trucco della ripetizione, però, potete congetturare abbastanza bene quale sia il saldo vero. Immaginate di chiedere che il saldo venga ritrasmesso cinque volte, e che riceviate queste risposte:

trasmissione 1:	€	5	2	9	3	,	7	5
trasmissione 2:	€	5	2	1	3	,	7	5
trasmissione 3:	€	5	2	1	3	,	1	1
trasmissione 4:	€	5	4	4	3	,	7	5
trasmissione 5:	€	7	2	1	8	,	7	5

Notate che in alcuni casi ci sono più cifre errate, mentre in un caso (numero 2) i dati sono stati trasmessi senza alcun errore. Il punto cruciale è che non avete modo di sapere dove siano gli errori, perciò non siete in grado neanche di scegliere la seconda trasmissione come quella giusta. Quello che potete fare, però, è esaminare separatamente ciascuna cifra, considerando tutte le volte che è stata trasmessa e scegliendo il valore che si presenta più di frequente. Ecco di nuovo i risultati, con le cifre più comuni elencate nell'ultima riga:

trasmissione 1:	€	5	2	9	3	,	7	5
trasmissione 2:	€	5	2	1	3	,	7	5
trasmissione 3:	€	5	2	1	3	,	1	1
trasmissione 4:	€	5	4	4	3	,	7	5
trasmissione 5:	€	7	2	1	8	,	7	5
cifra più frequente:	€	5	2	1	3	,	7	5

Vediamo qualche altro esempio per chiarire bene l'idea. Esaminando la prima cifra trasmessa, vediamo che nei primi quattro casi la prima cifra era un 5, mentre nella trasmissione 5 era un 7. In altre parole, quattro trasmissioni dicono "5" e una sola dice "7". Non potete esserne assolutamente sicuri, ma il valore *più probabile* per la prima cifra del vostro saldo è 5. Passando alla seconda cifra, si vede che il 2 si presenta quattro volte e il 4 una volta sola, perciò 2 è la seconda cifra più probabile. La terza cifra è un po' più interessante, perché ci sono tre possibilità: 1 si presenta tre volte, 9 una e 4 una. Vale però lo stesso principio, e 1 è il valore più probabile. Continuando così per tutte le cifre, potete arrivare a una ipotesi finale per il saldo bancario completo: € 5213,75, che in questo caso è effettivamente il valore corretto.

Beh, è stato facile. Abbiamo già risolto il problema? In un certo senso sì, ma potreste sentirvi un po' insoddisfatti, e per due motivi. Innanzitutto, il tasso di errore per questo canale di comunicazione era del 20% solamente, e in qualche caso i calcolatori potrebbero aver bisogno di comunicare su canali molto più disturbati. In secondo luogo, e questa è forse l'obiezione più seria, la risposta finale casualmente in questo caso era quella giusta, ma nulla garantisce che sia sempre così: è solo una congettura, basata su quello che pensiamo sia più probabile sia il saldo vero. Per fortuna, entrambe queste obiezioni possono essere respinte abbastanza facilmente: basta aumentare il numero delle ritrasmissioni fino a che si ottiene tutta l'affidabilità che si vuole.

Supponiamo, per esempio, che il tasso di errore sia del 50 invece che del 20%, nell'ultimo esempio. Potremmo chiedere alla banca di trasmettere il saldo 1000 volte invece di 5 soltanto. Concentriamoci solo sulla prima cifra, perché poi il discorso si applica allo stesso modo alle altre. Poiché il tasso di errore è 50%, circa metà delle volte la cifra sarà trasmessa correttamente (5), mentre nell'altra metà dei casi sarà trasformata in qualche altro valore a caso. Perciò ci saranno circa 500 occorrenze di 5 e solo circa 50 delle altre cifre (0-4 e 6-9). I matematici possono calcolare le probabilità che una delle altre cifre si presenti più spesso del 5: risulta che, anche se trasmettessimo un nuovo saldo bancario ogni secondo usando questo metodo, dovremmo aspettare parecchie migliaia di miliardi di anni prima che si verifichi un caso di congettura errata per un saldo. La morale è che, ripetendo un numero sufficiente di volte un messaggio inaffidabile, si può renderlo affidabile quanto si vuole. (In questi esempi, abbiamo presupposto che gli errori si verifichino *in modo casuale*. Se invece qualcuno o qualcosa interferisse deliberatamente con la trasmissione e scegliesse quali errori creare, il trucco della ripetizione sarebbe molto più vulnerabile. Alcuni dei codici di cui parleremo più avanti funzionano bene anche rispetto a questo tipo di attacco malevolo.)

Quindi il problema della comunicazione inaffidabile può essere risolto usando il trucco della ripetizione, sostanzialmente eliminando le possibilità di errore. Purtroppo, però, il trucco della ripetizione non è sufficiente per i sistemi di calcolo moderni. Quando si trasmettono pochi dati come un saldo bancario non è troppo costoso ritrasmetterli 1000 volte, ma sarebbe ovviamente del tutto impraticabile trasmettere 1000 copie di un software di grandi dimensioni (poniamo, di 200 megabyte). Chiaramente, i computer debbono usare qualcosa di più sofisticato del trucco della ripetizione.

Il trucco della ridondanza

Anche se i calcolatori non usano il trucco della ripetizione come l'abbiamo esposto prima, ne abbiamo parlato per poter vedere il principio fondamentale della comunicazione affidabile in azione. Il principio fondamentale è che non si può semplicemente spedire il messaggio originale; bisogna inviare anche qualche altra cosa per aumentarne l'affidabilità. Nel caso della ripetizione, spedivamo altre copie del messaggio originale, ma ci sono altri tipi di materiali aggiuntivi che si possono inviare per migliorare l'affidabilità. Gli informatici parlano a questo proposito di "ridondanza". Qualche volta la ridondanza viene aggiunta al messaggio originale: vedremo questa tecnica con il prossimo trucco (il trucco della somma di controllo). Prima però vediamo un altro modo di aggiungere ridondanza, che in effetti trasforma il messaggio originale in un messaggio più lungo, "ridondante": il messaggio originale viene sostituito da un messaggio diverso, iù

lungo. Quando si riceve la versione più lunga, la si può ritrasformare nell'originale, anche se è stata corrotta da un canale di comunicazione rumoroso. Lo chiameremo semplicemente il *trucco della ridondanza*.

Un esempio per chiarire. Ricordate che stavamo tentando di trasmettere il saldo del conto in banca (€ 5213,75) su un canale di comunicazione inaffidabile che modificava in modo casuale il 20% delle cifre. Invece di tentare di trasmettere "€ 5213,75", trasformiamolo in un messaggio più lungo (e quindi "ridondante") che contenga le stesse informazioni. In questo caso, semplicemente, scriveremo il saldo in parole italiane, per esempio così:

> cinque due uno tre virgola sette cinque

Supponiamo ancora che circa il 20% dei caratteri in questo messaggio venga trasformato in qualche altro carattere a causa del rumore sul canale di comunicazione. Il messaggio potrebbe diventare una cosa del genere:

> cintue dve uno sre vwrgola sivpn civqie

Anche se è un po' faticoso da leggere, penso sarete d'accordo che chiunque sappia l'italiano può ipotizzare che questo messaggio corrotto rappresenti il saldo vero di € 5213,75.

Il punto fondamentale è che, avendo utilizzato un messaggio *ridondante*, è possibile identificare e correggere in modo affidabile tutte le singole trasformazioni subite dal messaggio. Se vi dico che i caratteri "cintue" rappresentano un numero in italiano e che solo un carattere è stato alterato, potete essere assolutamente certi che il messaggio originale era "cinque", perché non c'è alcun altro numero in italiano che si possa ottenere da "cintue" modificando un solo carattere. Ben diverso dal dire che le cifre "367" rappresentano un numero, ma una delle cifre è stata alterata: non avete modo di sapere quale fosse il numero originale, perché in questo messaggio non c'è ridondanza.

Anche se non abbiamo ancora esplorato esattamente come funzioni la ridondanza, abbiamo già visto che ha qualcosa a che fare con il rendere *più lungo* il messaggio, e che ogni parte del messaggio deve essere conforme a qualche tipo di *schema* (*pattern*) ben noto. In questo modo, ogni singolo cambiamento può prima essere identificato (perché non coincide con uno schema noto) e poi corretto (modificando l'errore così che la sequenza coincida con lo schema). Gli informatici chiamano *code words* (parole in codice) questi schemi noti. Nel nostro esempio, le parole in codice sono semplicemente numeri scritti in italiano, come "uno", "due", "tre" e così via.

Ora è venuto il momento di spiegare esattamente come funzioni il trucco della ridondanza. I messaggi sono costituiti da quelli che gli informatici chiamano "simboli": nel nostro esempio, i simboli sono le cifre 0-9 (ignoriamo il segno dell'euro e la virgola decimale per semplicità). A ciascun simbolo è associata una parola in codice. Nel nostro esempio al simbolo 1 è assegnata la parola "uno", al 2 è assegnata "due" e via di questo passo.

Per trasmettere un messaggio, prima si prende ciascun simbolo e lo si traduce nella corrispondente parola in codice, poi si trasmette il messaggio così trasformato su un canale di comunicazione inaffidabile. Quando si riceve il messaggio, si esamina ciascuna sua parte e si controlla se si tratta di una parola in codice valida. Se è valida (per esempio,

“cinque”), semplicemente la si trasforma nel simbolo corrispondente (5, nell’esempio). Se non è una parola valida (per esempio, “cintue”), si cerca quale parola vi si avvicini di più (in questo caso, “cinque”) e si trasforma *questa* nel simbolo corrispondente (in questo caso, 5).

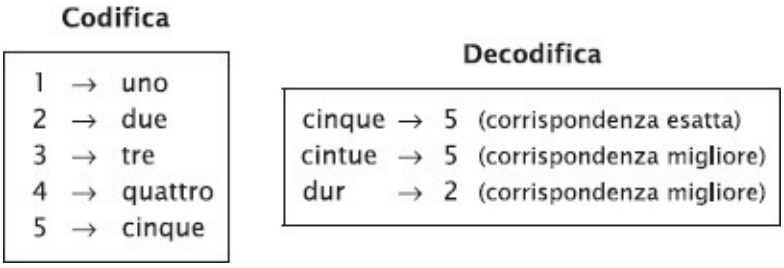


Figura 5.1 Un codice che usa parole in italiano al posto di cifre.

E questo è proprio tutto. I calcolatori usano effettivamente di continuo questo trucco della ridondanza per memorizzare e trasmettere informazioni. I matematici hanno inventato “parole in codice” più curiose di quelle italiane che abbiamo usato nell’esempio, ma per il resto il funzionamento delle comunicazioni informatiche affidabili è lo stesso. La Figura 5.2 offre un esempio reale, di un codice che i tecnici definiscono codice Hamming (7, 4), uno dei codici scoperti da Richard Hamming ai Bell Labs nel 1947, in risposta ai crash dei fine settimana di cui abbiamo parlato. (La Bell gli imponeva di brevettare i codici, e Hamming non li pubblicò se non tre anni dopo, nel 1950.) La differenza più ovvia rispetto al nostro codice precedente è che tutto è in termini di zero e uno. Ogni dato è memorizzato o trasmesso da un calcolatore come successione di zero e uno, qualsiasi codice usato nella vita reale deve essere limitato a queste due cifre.

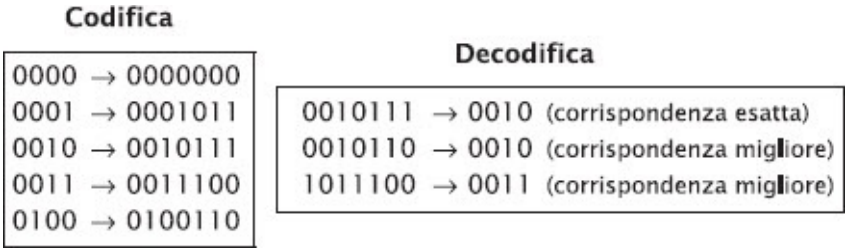


Figura 5.2 Un codice effettivamente usato dai calcolatori. Gli informatici lo chiamano codice Hamming (7, 4). Notate che il riquadro “codifica” elenca solo cinque dei 16 possibili input di quattro cifre; gli altri hanno parole in codice corrispondenti, ma qui sono stati omessi.

A parte questo, tutto funziona esattamente come prima. Nella codifica, a ciascun gruppo di quattro cifre viene aggiunta ridondanza e si genera una parola di sette cifre. Quando si decodifica, prima si cerca una corrispondenza esatta con le sette cifre ricevute e, se non si riesce, si prende la corrispondenza migliore. Potreste temere che, lavorando solo con uno e zero, ci possano essere più corrispondenze non esatte ma di “pari bontà” e che si possa finire con lo scegliere la decodifica errata. Questo particolare codice, però, è stato progettato molto abilmente e qualsiasi singolo errore nella parola a 7 cifre può essere corretto senza ambiguità. La matematica che sta alla base della progettazione di codici con questa caratteristica è molto elegante, ma non entreremo ulteriormente nei dettagli.

Val la pena di dire perché nella pratica si preferisca il trucco della ridondanza al trucco della ripetizione. Il motivo principale è il *costo* relativo dei due trucchi. Gli informatici misurano il costo dei sistemi a correzione di errore in funzione del “sovraccarico” (*overhead*). Il sovraccarico è semplicemente la quantità di informazione aggiuntiva che deve essere inviata per essere sicuri che un messaggio venga ricevuto correttamente. Il

sovraccarico del trucco della ripetizione è enorme, perché bisogna spedire molte copie complete del messaggio; il sovraccarico del trucco della ridondanza dipende dall'insieme specifico delle parole in codice che si usano. Nell'esempio precedente che usava parole italiane, il messaggio ridondante era lungo 39 caratteri (compresi gli spazi), mentre il messaggio originale era costituito da 6 cifre solamente, perciò anche il sovraccarico di questa particolare applicazione del trucco della ridondanza è piuttosto elevato, ma i matematici hanno costruito codici con una ridondanza molto minore, ma con prestazioni incredibilmente elevate, in termini di probabilità che un errore non venga rilevato. Il basso sovraccarico di questi codici è il motivo per cui i calcolatori usano il trucco della ridondanza invece di quello della ripetizione.

Fin qui abbiamo sempre fatto esempio di *trasmissione* delle informazioni mediante codici, ma tutto quello che abbiamo detto vale anche per la *memorizzazione* delle informazioni. CD, DVD e unità a disco si basano tutti sui codici a correzione d'errore per raggiungere quella straordinaria affidabilità che osserviamo nella pratica.

Il trucco della somma di controllo

Fin qui abbiamo considerato modi per *identificare* e *correggere* simultaneamente gli errori nei dati: il trucco della ripetizione e quello della ridondanza sono entrambi di questo genere. Esiste però un altro possibile modo di affrontare il problema nel suo complesso: possiamo non pensare a *correggere* gli errori e concentrarci invece solo sull'*identificarli*. (John Locke, filosofo del diciassettesimo secolo, era chiaramente cosciente della distinzione fra identificazione e correzione degli errori, come si può vedere dalla citazione in apertura di capitolo.) Per molte applicazioni è sufficiente rilevare l'esistenza di un errore, perché, se si rileva un errore non si fa altro che richiedere un'altra copia dei dati. E si può continuare a chiedere nuove copie, fino a che non se ne ottiene una senza errori. Questa è una strategia usata spesso: per esempio, quasi tutte le connessioni Internet usano questa tecnica. La chiameremo "trucco della somma di controllo", per motivi che saranno presto chiari.

Per capire il trucco della somma di controllo, sarà più comodo far finta che tutti i nostri messaggi siano formati solo da numeri. In effetti è una ipotesi molto realistica, poiché i calcolatori memorizzano tutte le informazioni sotto forma di numeri e le traducono in testo o immagini solo al momento di presentarle agli esseri umani. In ogni caso, è importante aver chiaro che qualsiasi scelta particolare di simboli per i messaggi non ha influenza sulle tecniche descritte in questo capitolo. Qualche volta è più comodo usare simboli numerici (le cifre 0-9), altre usare simboli alfabetici (i caratteri a-z). In ciascun caso, possiamo concordare su una traduzione fra questi insiemi di simboli. Per esempio, una traduzione ovvia da simboli alfabetici a simboli numeri è a → 01, b → 02, …, z → 26. Perciò in realtà non importa se studiamo una tecnica per trasmettere messaggi numerici o alfabetici; la tecnica poi può essere applicata a qualsiasi tipo di messaggio dopo aver effettuato una traduzione semplice e costante dei simboli.

A questo punto, dobbiamo capire che cosa sia effettivamente una somma di controllo. Esistono molti tipi diversi di somme di controllo, ma per ora fermiamoci alla "somma di controllo semplice". Il calcolo della somma di controllo semplice di un messaggio numerico è veramente semplicemente: si prendono le cifre del messaggio, le si somma, si tiene solo l'ultima cifra del risultato, e quella è la somma di controllo. Supponiamo che il

messaggio sia

4 6 7 5 6

La somma delle cifre allora è 4 + 6 + 7 + 5 + 6 = 28; conserviamo solo l'ultima cifra, perciò la somma di controllo semplice di questo messaggio è 8.

Ma come si usano le somme di controllo? Facile: si accoda la somma di controllo al messaggio originale prima di spedirlo. Poi, quando qualcun altro riceve il messaggio, può calcolare di nuovo la somma di controllo, confrontarla con quella che è stata spedita, e vedere se è corretta. Per questo si parla di "somma di controllo". Torniamo all'esempio precedente. La somma di controllo semplice del messaggio "46756" è 8, perciò trasmettiamo messaggio e somma di controllo:

4 6 7 5 6 8

Chi riceve il messaggio deve sapere che si sta usando il trucco della somma di controllo. Immaginando che lo sappia, può immediatamente sapere che l'ultima cifra, 8, non fa parte del messaggio originale, perciò può non considerarla e calcolare la somma di controllo di tutto il resto. Se non ci sono stati errori nella trasmissione del messaggio, calcolerà 4 + 6 + 7 + 5 + 6 = 28; conserverà solo l'ultima cifra (8), controllerà se è uguale alla somma di controllo che prima non ha preso in considerazione (sì, è uguale) e quindi ne concluderà che il messaggio è stato trasmesso correttamente. Che cosa succede invece se *c'è stato* un errore di trasmissione? Supponiamo che il 7 sia stato trasformato casualmente in 3. Il messaggio ricevuto allora sarà

4 6 3 5 6 8

Si mette da parte l'8 per il confronto successivo e si calcola la somma di controllo 4 + 6 + 3 + 5 + 6 = 24 e si tiene solo l'ultima cifra (4). Ma questa non è uguale all'8 che si era messo da parte, perciò si è sicuri che il messaggio è stato modificato durante la trasmissione. A questo punto, si richiede che il messaggio venga ritrasmesso, si attende di ricevere una nuova copia, poi si calcola di nuovo la somma di controllo e la si confronta con quella ricevuta. E si può continuare così fino a che non si ottiene un messaggio la cui somma di controllo sia corretta.

Sembra quasi troppo bello per essere vero. Ricordate che il "sovraccarico" di un sistema a correzione di errore è la quantità di informazioni aggiuntive che si devono spedire con il messaggio. Beh, qui sembra che abbiamo trovato il *non plus ultra* dei sistemi a basso sovraccarico, poiché, per quanto lungo sia il messagio, dobbiamo aggiungere solo una cifra in più (la somma di controllo) per rilevare un errore!

Purtroppo, questo sistema delle somme di controllo semplici *è* troppo bello per essere vero. Il problema sta in questo: la somma semplice che abbiamo descritto può rilevare al massimo *un* errore nel messaggio. Se gli errori sono due o più, può darsi che li rilevi ma può anche darsi di no. Vediamo qualche esempio:

		somma di controllo
messaggio originale	4 6 7 5 6	8

messaggio con un errore	1 6 7 5 6	5
messaggio con due errori	1 5 7 5 6	4
messaggio con due errori (diversi dai precedenti)	2 8 7 5 6	8

Il messaggio originale (46756) è lo stesso di prima, e uguale è la somma di controllo. Nella riga successiva c'è un messaggio con un errore (la prima cifra è 1 anziché 4) e la somma di controllo è 5. In effetti, potete probabilmente convincervi da soli che la modificazione di una *singola* cifra qualsiasi dà una somma di controllo che è diversa da 8, perciò è certo che rileverete qualsiasi errore singolo nel messaggio. Non è difficile dimostrare che questo è sempre vero: se c'è un solo errore, una somma di controllo semplice lo rileva con assoluta certezza.

Nella riga successiva della tabella, si vede un messaggio con due errori: entrambe le prime due cifre sono state modificate. In questo caso la somma di controllo è 4 e, poiché 4 è diverso dalla somma di controllo originale, 8, chi riceve il messaggio si accorge che è stato introdotto un errore. La difficoltà balza all'occhio nell'ultima riga della tabella: qui c'è un altro messaggio con due errori, sempre nelle prime due cifre. I valori però sono diversi e in questo caso succede che la somma di controllo sia comunque 8 – identica all'originale! Perciò chi ricevesse questo messaggio non riuscirebbe a rilevare l'esistenza di errori.

Per fortuna, si può aggirare il problema aggiungendo qualche ulteriore modifica al trucco della somma di controllo. Il primo passo è definire un nuovo tipo di somma: la chiameremo somma di controllo "a scaletta" perché pensare di salire una scala intanto che la si calcola è di aiuto. Immaginatevi di essere ai piedi una scala con i gradini numerati 1, 2, 3 e così via. Per calcolare una somma di controllo a scaletta si sommano le cifre come prima, ma ogni cifra deve essere moltiplicata per il numero dello scalino su cui ci si trova, e bisogna salire di un gradino a ogni cifra. Alla fine, si butta via tutto e si conserva solo l'ultima cifra, come nel caso della somma di controllo semplice. Perciò se il messaggio è

4 6 7 5 6

come prima, si calcola la somma a scaletta

$$(1 \times 4) + (2 \times 6) + (3 \times 7) + (4 \times 5) + (5 \times 6) =$$
$$= 4 + 12 + 21 + 20 + 30 =$$
$$= 87$$

Poi si butta via tutto tranne l'ultima cifra, che è 7. Perciò la somma di controllo a scaletta di "46756" è 7.

A che serve tutto questo? Si dà il caso che, se si includono *sia* la somma di controllo semplice *sia* quella a scaletta, si rilevano due errori qualunque in qualsiasi messaggio. Perciò il nostro nuovo trucco della somma di controllo consiste nel trasmettere il messaggio originale, poi due cifre in più: prima la somma di controllo semplice, poi quella a scaletta. Per esempio, il messaggio "46756" ora verrà trasmesso come

4 6 7 5 6 8 7

Quando si riceve il messaggio, bisogna sempre sapere, in base a un accordo precedente, quale trucco sia stato applicato. Immaginando che questo sia noto, è facile controllare se ci sono errori, come per la somma semplice. In questo caso si mettono prima da parte le ultime due cifre (8, che è la somma di controllo semplice, e 7, che è la somma a scaletta). Poi si calcola la somma di controllo semplice del resto del messaggio (56756, che dà 8) e si calcola la somma di controllo a scaletta (che risulta 7). Se *entrambe* le somme calcolate corrispondono a quelle inviate (e in questo caso è così) si ha la certezza che il messaggio o è corretto, oppure contiene almeno tre errori.

La tabella successiva mostra il procedimento in pratica. La tabella è identica alla precedente, solo che ora a ogni riga è stata aggiunta la somma di controllo a scala, ed è stata aggiunta una nuova riga con un esempio ulteriore. Quando c'è un errore, vediamo che entrambe le somme di controllo sono diverse da quelle del messaggio originale (5 al posto di 8, 4 al posto di 7). Quando ci sono due errori, è possibile che entrambe le somme di controllo siano diverse, come nella terza riga della tabella dove ci sono 4 al posto di 8 e 2 al posto di 7. Come abbiamo già visto, però, ci sono casi in cui la somma semplice non cambia in presenza di due errori. La quarta riga mostra un esempio, dove la somma di controllo semplice è ancora 8. Dato però che la somma di controllo a scaletta è diversa da quella originale (9 al posto di 7), sappiamo comunque che questo messaggio contiene degli errori. Nell'ultimo esempio, si vede che può succedere anche il contrario: questo è un esempio di due errori che danno una somma semplice diversa (9 al posto di 8), mentre la somma a scaletta è identica (7). Anche in questo caso però possiamo rilevare l'errore, perché almeno una delle due somme di controllo è diversa da quella originale. Ci vuole un po' più di perizia matematica per dimostrarlo, ma non è un caso: si dimostra che si è sempre in grado di rilevare gli errori, se non sono più di due.

		somma di controllo semplice e a scaletta
messaggio originale	4 6 7 5 6	8 7
messaggio con un errore	1 6 7 5 6	5 4
messaggio con due errori	1 5 7 5 6	4 2
messaggio con due errori (diversi dai precedenti)	2 8 7 5 6	8 9
messaggio con due errori (diversi dai precedenti)	6 5 7 5 6	9 7

Ora che abbiamo un'idea chiara del meccanismo fondamentale, dobbiamo aver chiaro che il trucco della somma di controllo appena descritto funziona sicuramente solo per messaggi relativamente brevi (meno di 10 cifre). Idee molto simili però possono essere applicate ai messaggi più lunghi. È possibile definire somme di controllo mediante certe sequenze di operazioni semplici, come sommare le cifre, moltiplicare le cifre secondo "scalette" di varia forma, e scambiare alcune cifre secondo schemi ben definiti. Può sembrare complicato, ma una macchina può calcolare queste somme di controllo a grandissima velocità e questo è un modo estremamente utile e pratico per rilevare la

presenza di errori nei messaggi.

Il trucco della somma di controllo descritto sopra produce solo due cifre di controllo (semplice e a scaletta) ma le somme di controllo reali di solito producono molte più cifre, a volte fino a 150. (Nel resto del capitolo parlo delle dieci cifre *decimali*, 0-9, non delle due cifre *binarie*, 0 e 1, che sono usate comunemente nelle comunicazioni informatiche.) La cosa importante è che il numero delle cifre nella somma di controllo (che siano 2, come nell'esempio precedente, o circa 150, come per alcune somme di controllo utilizzate in pratica) è *costante*. Anche se la lunghezza delle somme di controllo prodotte da qualsiasi algoritmo dato è costante, si possono calcolare somme di controllo di messaggi di qualsiasi lunghezza. Perciò, se un messaggio è molto lungo, anche una somma di controllo relativamente lunga come quelle a 150 cifre risulta minuscola in proporzione al messaggio stesso. Supponiamo per esempio di usare una somma di controllo di 100 cifre per verificare la correttezza di un pacchetto software di 20 megabyte scaricato dal Web. La somma di controllo è meno di un millesimo dell'1% delle dimensioni del software. Sono sicuro che sarete d'accordo che sia un livello di sovraccarico accettabile. E un matematico vi dirà che le probabilità di non rilevare un errore, quando si usa una somma di controllo tanto lunga, è così incredibilmente piccola da poter essere considerata nulla per tutti i fini pratici.

Come al solito, ci sono alcuni particolari tecnici importanti. Non è vero che qualsiasi sistema a somma di controllo da 100 cifre abbia questa resistenza incredibilmente elevata agli sbagli. Ci vuole un certo tipo di somma di controllo che gli informatici chiamano *funzione hash crittografica*, specialmente se le modifiche apportate al messaggio possono essere dovute a un intervento fraudolento, invece che alle fluttuazioni casuali di un canale di comunicazione rumoroso. Questo è un aspetto molto reale, perché è possibile che un hacker con cattive intenzioni cerchi di modificare il software da 20 megabyte in modo tale che abbia la stessa somma di controllo da 100 cifre, ma in realtà sia un software diverso, in grado di assumere il controllo del vostro computer! L'uso delle funzioni hash crittografiche elimina questa possibilità.

Il trucco dell'estrazione

Ora che conosciamo le somme di controllo, possiamo tornare al problema originale di rilevare *e* correggere gli errori di comunicazione. Sappiamo già come farlo, o in modo inefficiente con il trucco della ripetizione o in modo efficiente con il trucco della ridondanza; ritorniamo però a quest'ultimo, perché non abbiamo mai visto veramente come creare le parole in codice che costituiscono l'ingrediente chiave di questo trucco. Avevamo l'esempio dell'uso delle parole italiane per descrivere le cifre, ma questo particolare insieme di parole è meno efficiente di quelli che un calcolatore usa effettivamente. Abbiamo visto anche l'esempio concreto di un codice Hamming, ma senza spiegare come siano state generate le parole del codice.

Ora perciò vedremo un altro possibile insieme di parole in codice che si può usare per il trucco della ridondanza. Poiché questo è un caso molto speciale del trucco della ridondanza che consente non solo di rilevare ma anche di identificare o "estrarre" gli errori, lo chiameremo "trucco dell'estrazione".

Come abbiamo fatto per il trucco della somma di controllo, prenderemo in

considerazione solo messaggi numerici costituiti dalle cifre 0-9, ma ricordate sempre che lo facciamo solo per comodità. È molto semplice prendere un messaggio alfabetico e tradurlo in numeri, perciò la tecnica che descriveremo può essere applicata a qualsiasi tipo di messaggio.

Per semplicità, ipotizziamo che il messaggio sia lungo esattamente 16 cifre; ma anche questo non costituisce un limite per la tecnica nella pratica. Se si ha un messaggio più lungo e lo si suddivide in blocchi da 16 cifre e si opera su ciascun blocco separatamente. Se il messaggio è di meno di 16 cifre, lo si completa con tanti zeri quanti ne servono per raggiungere le 16 cifre.

Il primo passo nel trucco dell'estrazione è ridisporre le 16 cifre del messaggio in un quadrato, che si leggerà da sinistra verso destra e dall'alto verso il basso. Perciò se il messaggio effettivo è

4 8 3 7 5 4 3 6 2 2 5 6 3 9 9 7

viene riordinato nella forma

4	8	3	7
5	4	3	6
2	2	5	6
3	9	9	7

Poi si calcola la somma di controllo semplice di ogni riga e la si aggiunge alla destra della riga corrispondente:

4	8	3	7	2
5	4	3	6	8
2	2	5	6	5
3	9	9	7	8

Queste somme di controllo semplici si calcolano come abbiamo fatto in precedenza. Per esempio, per la seconda riga si calcola 5 + 4 + 3 + 6 = 18 e si prende solo l'ultima cifra, 8.

Il passo successivo consiste nel calcolare le somme di controllo semplici per ciascuna colonna e aggiungerle in una nuova riga in basso:

4	8	3	7	2
5	4	3	6	8
2	2	5	6	5
3	9	9	7	8
4	3	0	6	

Anche qui non c'è nulla di misterioso. Per esempio, per la terza colonna si calcola 3 + 3+ 5 + 9 = 20, cioè 0 tenendo solo l'ultima cifra.

Il passo successivo nel trucco dell'estrazione è riordinare tutto in modo che possa essere memorizzato o trasmesso una cifra alla volta. Lo si fa nel modo ovvio, leggendo le cifre da sinistra verso destra e dall'alto verso il basso. Perciò ci si ritrova con il seguente messaggio di 24 cifre:

4 8 3 7 2 5 4 3 6 8 2 2 5 6 5 3 9 9 7 8 4 3 0 6

Ora immaginate di aver ricevuto un messaggio che sia stato trasmesso utilizzando il trucco dell'estrazione. Quali passi dovete seguire per recuperare il messaggio originale e correggere eventuali errori di comunicazione? Proviamo con un esempio. Il messaggio originale sarà lo stesso di prima, ma, per rendere le cose più interessanti, supponiamo che ci sia stato un errore di comunicazione e che una delle cifre sia stata alterata. Non preoccupatevi ora di quale sia la cifra modificata: tra breve useremo il trucco dell'estrazione per stabilirlo.

Supponiamo dunque che il messaggio di 24 cifre che avete *ricevuto* sia

4 8 3 7 2 5 4 3 6 8 2 7 5 6 5 3 9 9 7 8 4 3 0 6

Il primo passo sarà disporre le cifre in un quadrato 5 per 5, dove si sa che l'ultima colonna e l'ultima riga corrispondono a somme di controllo che sono state inviate insieme con il messaggio originale:

4	8	3	7	2
5	4	3	6	8
2	7	5	6	5
3	9	9	7	8
4	3	0	6	

Poi si calcolano le somme di controllo semplici delle prime quattro cifre in ogni riga e colonna, scrivendo i risultati in un nuova riga e una nuova colonna, accanto ai valori ricevuti delle somme di controllo:

4	8	3	7	2	2
5	4	3	6	8	8
2	7	5	6	5	0
3	9	9	7	8	8
4	3	0	6		
4	8	0	6		

È fondamentale ricordare che qui ci sono due insiemi di somme di controllo: quelle che sono state *inviate* e quelle che sono state *calcolate*. Nella maggior parte dei casi, i due insiemi saranno uguali. Se sono identici, in effetti potete concludere che il messaggio è con tutta probabilità corretto. Ma se c'è stato un errore di comunicazione, qualcuna delle somme calcolate sarà diversa dalla corrispondente ricevuta.

Notate che nell'esempio ci sono due differenze: il 5 e lo 0 nella terza riga sono diversi, e lo stesso vale per il 3 e l'8 nella seconda colonna. Le somme che non coincidono sono evidenziate qui sotto:

4	8	3	7	2	2
5	4	3	6	8	8
2	7	5	6	5	[0]
3	9	9	7	8	8
4	3	0	6		
4	[8]	0	6		

Ed ecco l'idea chiave: la posizione di queste cifre diverse vi dice esattamente dove si sia verificato l'errore! *Deve* essere nella terza riga (perché tutte le altre righe hanno la somma corretta) e *deve* essere nella seconda colonna, perché tutte le altre colonne hanno la somma corretta). E, come potete vedere dal disegno seguente, queste due condizioni

identificano esattamente una possibilità: il 7 evidenziato a pagina seguente.

4	8	3	7	2	2
5	4	3	6	8	8
2	7	5	6	5	0
3	9	9	7	8	8
4	3	0	6		
4	8	0	6		

Ma non è ancora finita: abbiamo identificato l'errore, ma non l'abbiamo ancora corretto. Per fortuna è facile: dobbiamo sostituire il 7 sbagliato con un numero che renda corrette entrambe le somme di controllo. Si può vedere che la seconda colonna doveva avere come somma di controllo 3, mentre invece dà 8: in altre parole, la somma di controllo deve essere diminuita di 5. Perciò diminuiamo di 5 il 7 sbagliato, e otteniamo 2:

4	8	3	7	2	2
5	4	3	6	8	8
2	2	5	6	5	5
3	9	9	7	8	8
4	3	0	6		
4	3	0	6		

Potete addirittura verificare la validità del cambiamento, esaminando la terza riga: ora ha una somma di controllo 5, che coincide con quella ricevuta. L'errore è stato sia identificato sia corretto! L'ovvio passaggio finale consiste nell'estrarre il messaggio originale di 16 cifre corretto dalla matrice quadrata, leggendola da sinistra a destra e dall'alto verso il basso (ignorando riga e colonna finale, ovviamente). Il risultato è

4 8 3 7 2 5 4 3 6 8 2 2 5 6 5 3 9 9 7 8 4 3 0 6

Che è effettivamente lo stesso messaggio da cui eravamo partiti.

In informatica, il trucco va sotto il nome di "parità bidimensionale". La parola *parità* indica la stessa cosa di una somma semplice, quando si lavora con i numeri binari normalmente usati dai calcolatori; la parità poi è definita *bidimensionale* perché il messaggio viene disposto in una griglia a due dimensioni (righe e colonne). La parità bidimensionale è stata usata in alcuni sistemi di calcolo, ma non è efficace come certi altri trucchi di ridondanza. Ho scelto di parlarvene perché è molto facile da visualizzare e dà un po' il senso di come si possano trovare e correggere errori senza dover ricorrere agli strumenti matematici avanzati che stanno alla base dei codici più diffusi nei sistemi informatici odierni.

Correzione e rilevamento degli errori nel mondo reale

I codici a correzione di errore esistono dagli anni Quaranta, cioè da subito dopo la nascita degli stessi calcolatori elettronici. Con il senno di poi, non è difficile vedere perché: i primi calcolatori erano molto inaffidabili, e i loro componenti producevano spesso errori. Ma le vere radici dei codici a correzione d'errore risalgono ancora più indietro, a sistemi di comunicazione come telegrafo e telefono. Non sorprende affatto dunque che i due eventi principali che hanno favorito la creazione di quei codici si siano verificati entrambi nei laboratori di ricerca della Bell Telephone Company. I due eroi della nostra storia, Claude Shannon e Richard Hamming, erano ricercatori ai Bell Labs. Abbiamo già

incontrato Hamming: è stata la sua irritazione per i crash di un calcolatore aziendale nel corso dei fine settimana che l'ha portato direttamente a inventare i primi codici a correzione di errore, quelli che oggi chiamiamo codici Hamming.

I codici a correzione di errore però sono solo una parte di una disciplina più ampia, la *teoria dell'informazione*, campo la cui nascita risale, secondo il giudizio della maggior parte degli informatici, a un articolo del 1948 di Claude Shannon. Questo testo straordinario, intitolato "The Mathematical Theory of Communication" è descritto, in una biografia di Shannon, come "la Magna Carta dell'era dell'informazione". Irving Reed (coinventore dei codici Reed-Solomon di cui parleremo fra poco) ha detto dello stesso articolo: "Pochi altri lavori in questo secolo hanno avuto un impatto più forte sulla scienza e l'ingegneria. È una pietra miliare … Shannon ha modificato profondamente tutti gli aspetti della teoria e della pratica delle comunicazioni". Perché tutti questi elogi? Shannon ha dimostrato matematicamente che era possibile, in linea di principio, ottenere comunicazioni senza errori su canali rumorosi e non affidabili. Solo molti decenni più tardi gli scienziati si sono avvicinati nella pratica a quel massimo teorico previsto da Shannon.

Incidentalmente, Shannon era una persona con molti interessi diversi. Fu uno dei quattro organizzatori principali del convegno del 19556 a Dartmouth sull'intelligenza artificiale (di cui parleremo alla fine del Capitolo 6) e quindi strettamente coinvolto nella fondazione anche di quel campo. Ma non basta: era solito girare in monociclo e costruì un improbabile monociclo con una ruota ellittica, per cui il ciclista si alzava e si abbassava continuamente con il moto del suo monociclo!

Il lavoro di Shannon ha collocato i codici di Hamming in un contesto teorico più ampio e ha preparato la strada a molti progressi ulteriori. I codici di Hamming furono usati in alcuni dei primi computer e sono ancora ampiamente usati in alcuni tipi di sistemi di memoria. Un'altra famiglia di codici importanti è costituita dai codici di Reed-Solomon, che possono essere utilizzati per correggere un numero elevato di errori per parola di codice. (Vanno confrontati con il codice di Hamming (7, 4) della Figura 5.2, che può correggere solo un errore in ciascuna parola di codice di 7 cifre.) I codici di Reed-Solomon sono basati su una branca della matematica chiamata algebra dei campi finiti, ma molto rozzamente si può pensarli come la combinazione di caratteristiche della somma di controllo a scaletta e del trucco dell'estrazione bidimensionale. Sono utilizzati nei CD, nei DVD e nelle unità disco dei calcolatori.

Anche le somme di controllo sono ampiamente usate nella pratica, per lo più per rilevare e non per correggere gli errori. L'esempio forse più diffuso è Ethernet, il protocollo di rete usato oggi da quasi tutti i computer del pianeta. Ethernet utilizza un tipo di somma di controllo per il rilevamento degli errori che prende il nome di CRC-32. Anche il più diffuso dei protocolli Internet, il TCP (Transmission Control Protocol) usa somme di controllo per ciascun *pacchetto* di dati spedito. I pacchetti le cui somme di controllo sono errate vengono semplicemente scartati, perché TCP è stato progettato in modo da ritrasmetterli automatica in seguito, se è necessario. I pacchetti software pubblicati in Internet spesso vengono verificati con le somme di controllo, fra cui le più comuni sono la MD5 e la SHA-1, entrambe funzioni hash crittografiche, che proteggono dall'alterazione fraudolenta del software e da errori casuali di comunicazione. Le somme

di controllo MD5 hanno circa 40 cifre, le SHA-1 circa 50 cifre ed esistono nella stessa famiglia anche somme di controllo ancor più resistenti agli errori, come SHA-256 (che produce circa 75 cifre) e SHA-512 (circa 150 cifre).

La scienza dei codici che rilevano e correggono gli errori continua a svilupparsi. A partire dagli anni Novanta ha suscitato particolare attenzione un metodo denominato *codici a controllo della parità a bassa densità*. Questi codici sono usati ora in applicazioni che vanno dalla televisione satellitare alle comunicazioni con le sonde nello spazio profondo. Perciò la prossima volta che nel fine settimana vi godete qualche trasmissione televisiva satellitare ad alta definizione, mandate un pensiero a Richard Hamming: è stata la sua lotta nei fine settimana con uno dei primi computer che rende possibile a voi, nei vostri fine settimana, questa forma di intrattenimento.

---

1 Tr. it. *Saggio sull'intelligenza umana*, Laterza, Bari, 1972, Libro quarto, p. 109.

CAPITOLO 6

Riconoscimento di forme: apprendere dall’esperienza

> La Macchina Analitica non ha alcuna pretesa di *inventare* qualcosa. Può fare tutto quello che noi *sappiamo come ordinarle* di fare.
>
> Ada Lovelace, dalle sue note del 1843 sulla Macchina Analitica

In ciascuno dei capitoli precedenti abbiamo preso in considerazione un campo in cui le capacità dei computer superano di gran lunga quelle degli esseri umani. Un computer, per esempio, può cifrare o decifrare un file di grandi dimensioni nel giro di un secondo o due, mentre un essere umano potrebbe impiegare anni a svolgere gli stessi calcoli a mano. Un esempio ancora più estremo: immaginate quanto tempo sarebbe necessario a un essere umano per calcolare manualmente il PageRank di miliardi di pagine web secondo l’algoritmo descritto nel Capitolo 3. Un’impresa così grande da risultare praticamente impossibile. I computer delle società che gestiscono i motori di ricerca svolgono invece costantemente questi calcoli.

In questo capitolo, invece, esamineremo un settore in cui gli esseri umani hanno un vantaggio naturale: quello del *riconoscimento di forme* (*pattern recognition*), un sottoinsieme dell’intelligenza artificiale in cui rientrano attività come il riconoscimento di volti, il riconoscimento di oggetti, del parlato, della scrittura a mano. Esempi più specifici possono essere stabilire se una data fotografia è un’immagine di vostra sorella, o determinare città e indirizzo scritti a mano su una busta. Il riconoscimento di forme può essere definito più in generale quindi come ogni tentativo di fare in modo che i computer agiscano “in modo intelligente” sulla base di dati in ingresso che presentano una grande variabilità.

L’espressione “in modo intelligente” è fra virgolette per un buon motivo: se i computer potranno mai esibire una vera intelligenza è una questione assai controversa. La citazione in apertura di capitolo rappresenta una delle prime bordate del dibattito: Ada Lovelace commentava, nel 1843, il progetto di uno dei primi calcolatori meccanici, la Macchina Analitica. A volte si parla di Ada Lovelace come della prima programmatrice, date le sue profonde intuizioni sulla Macchina Analitica; in questa sua affermazione, però, sottolinea la mancanza di originalità dei computer, che debbono seguire pedissequamente le istruzioni dei loro programmatori umani. Non c’è consenso, oggi, fra gli informatici, sul fatto che i computer possano o meno mostrare, in linea di principio, intelligenza, e il dibattito diventa ancora più complesso se si tiene conto anche di filosofi, neuroscienziati e teologi.

Qui non dobbiamo risolvere i paradossi dell’intelligenza delle macchine. Per quel che

ci serve, possiamo tranquillamente sostituire “in modo intelligente” con “in modo utile”. Così il compito fondamentale del riconoscimento di forme è prendere dati con variabilità estremamente elevata (fotografie di volti diversi in condizioni di illuminazione diverse, campioni di parole scritte a mano da persone diverse) e ricavarne qualcosa di utile. Gli esseri umani senza alcun dubbio possono elaborare dati simili in modo intelligente: possiamo riconoscere volti con una precisione infallibile e leggere la calligrafia di praticamente chiunque senza aver visto prima dei campioni della loro scrittura. I computer sono estremamente inferiori agli esseri umani in queste attività, ma sono stati elaborati alcuni algoritmi ingegnosi che consentono ai computer di raggiungere buone prestazioni in certi compiti di riconoscimento di forme. In questo capitolo vedremo tre di questi algoritmi: i classificatori di vicinanza, gli alberi di decisione e le reti neurali artificiali. Prima, però, dobbiamo descrivere un po’ più scientificamente il problema che stiamo cercando di risolvere.

Qual è il problema?

Le attività di riconoscimento di forme possono sembrare, a tutta prima, assurdamente varie. I computer possono utilizzare un unico strumentario di tecniche di riconoscimento di forme per riconoscere la calligrafia, i volti, il parlato e altre cose ancora? Una possibile risposta ci guarda (letteralmente) in faccia: il nostro cervello mostra una straordinaria velocità e accuratezza in una gamma molto ampia di attività di riconoscimento. Possiamo scrivere un programma che faccia la stessa cosa?

Prima di discutere le tecniche che un programma del genere potrebbe usare, dobbiamo in qualche modo unificare la straordinaria varietà di attività e definire un singolo problema che vogliamo risolvere. Il modo canonico di affrontare la questione consiste nel vedere il riconoscimento di forme come un problema di *classificazione*. Assumiamo che i dati da elaborare siano suddivisi in blocchi sensati che chiamiamo *campioni*, e che ciascun campione appartenga a una di un numero finito e costante di *classi* possibili. Per esempio, in un problema di riconoscimento di volti, ciascun campione sarà l’immagine di un volto e le classi le identità delle persone che il sistema può riconoscere. In alcuni problemi, le classi sono solo due. Un esempio comune può essere la diagnosi medica rispetto a una particolare malattia, dove le due classi possono essere “sano” e “malato” e ogni singolo campione di dati può essere costituito dai risultati di tutti gli esami clinici relativi a uno stesso paziente (pressione sanguigna, peso, radiografie e molte altre cose, magari). Il compito del computer è elaborare nuovi campioni di dati mai visti in precedenza e *classificare* ciascun campione in una delle classi possibili.

Per rendere il discorso più concreto, concentriamoci per ora su un particolare compito di riconoscimento di forme, quello del riconoscimento di cifre scritte a mano. Alcuni campioni di dati tipici sono mostrati nella Figura 6.1. In questo caso le classi sono esattamente dieci: le cifre 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Il compito quindi è classificare campioni di cifre scritte a mano in queste dieci classi. È ovviamente un problema di grande importanza pratica, dato che la posta in molti paesi viene spedita utilizzando codici postali numerici. Se un computer può riconoscere rapidamente e con precisione questi codici postali, la posta può essere smistata da una macchina in modo molto più efficiente che da un essere umano.

Ovviamente, i computer non hanno una conoscenza innata di come si presentano le

cifre scritte a mano; non ce l'hanno in effetti neanche gli esseri umani: *impariamo* a riconoscere le cifre e in generale ciò che è scritto a mano attraverso qualche combinazione di insegnamento esplicito da parte di altri esseri umani e ispezione di esempi che utilizziamo per apprendere da soli. Queste due strategie (insegnamento esplicito e apprendimento da esempi) sono utilizzate anche nel riconoscimento di forme da parte dei computer. Tranne che per i compiti più semplici, però, si è visto che l'insegnamento esplicito per i computer è inefficace. Per esempio, possiamo pensare i controlli climatici di casa mia come un semplice sistema di classificazione. Un campione di dati è costituito dalla temperatura e dall'ora del giorno, e tre classi possibili sono "riscaldamento acceso", "condizionamento accesso" e "entrambi spenti". Dato che di giorno lavoro in un ufficio, programmo il sistema su "entrambi spenti" durante le ore del giorno; nelle altre ore invece su "riscaldamento acceso" se la temperatura è troppo bassa e "condizionamento acceso" se la temperatura è troppo alta. Programmando il mio termostato, in qualche senso del termine ho "insegnato" al sistema come compiere la classificazione in queste tre classi.

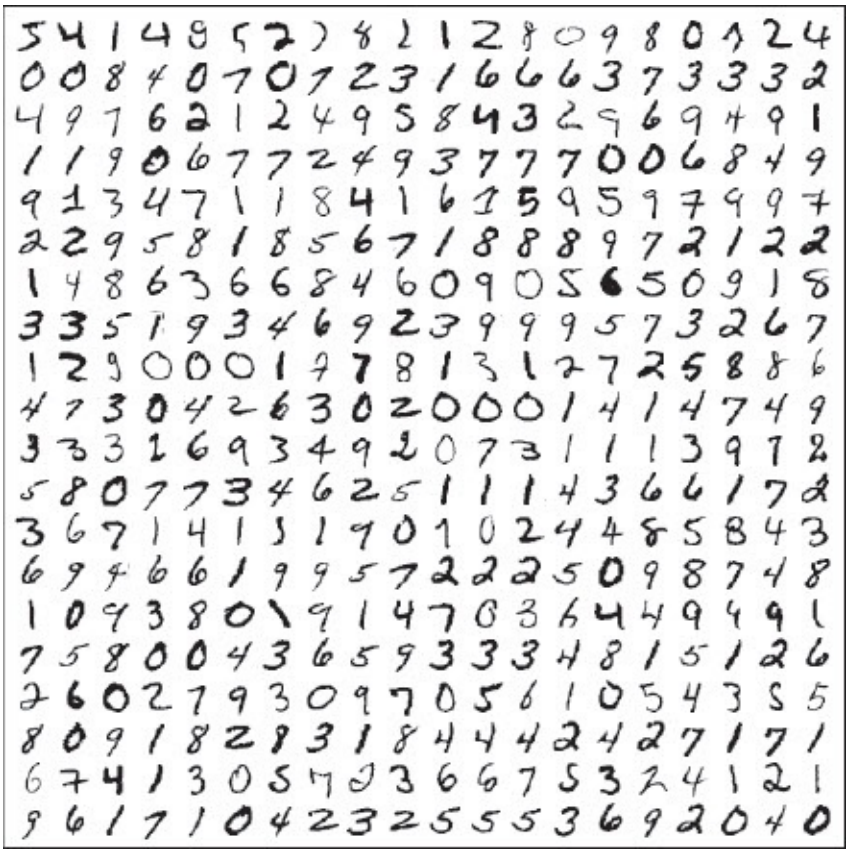


Figura 6.1 Per la maggior parte i compiti di riconoscimento di forme possono essere riformulati come problemi di classificazione. Qui, il compito è classificare ciascuna cifra scritta a mano come una delle dieci cifre 0, 1, …, 9. Fonte: *dati MNIST di LeCun* et al., *1998.*

Purtroppo, nessuno è mai riuscito a "insegnare" esplicitamente a un computer come risolvere compiti di classificazione più interessanti, come quello di distinguere le cifre manoscritte della Figura 6.1. Gli informatici si sono dedicati all'altra strategia disponibile: fare in modo che un calcolatore "apprenda" automaticamente come classificare campioni. La strategia di fondo è dare alla macchina una grande quantità di *dati etichettati*: campioni che sono già stati classificati. La Figura 6.2 mostra un esempio di dati etichettati per le cifre manoscritte. Ogni campione è fornito di un'etichetta (che è la sua classe), perciò il computer può utilizzare vari trucchi analitici per estrarre le caratteristiche di ciascuna classe. Quando in seguito gli viene presentato un campione non etichettato, il computer

può congetturare la sua classe scegliendo quella le cui caratteristiche sono più simili alle caratteristiche del campione non etichettato.

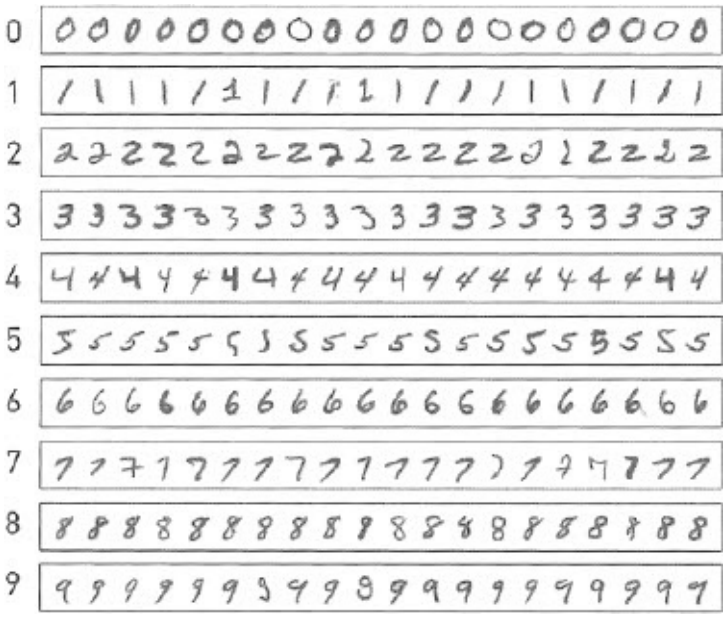


Figura 6.2 Per addestrare un classificatore, un computer ha bisogno di dati già etichettati. Qui ciascun campione di dati (una cifra scritta a mano) è fornito di un'etichetta che specifica una delle dieci cifre possibili. Le etichette sono a sinistra, e i campioni di addestramento sono nei riquadri a destra. Fonte: *dati MNIST di LeCun* et al., *1998*.

Il processo di apprendimento delle caratteristiche di ciascuna classe spesso viene chiamato "addestramento" e i dati etichettati stessi sono i "dati di addestramento". In sintesi, dunque, i compiti di riconoscimento di forme sono divisi in due fasi: prima, una fase di addestramento in cui il computer impara quali siano le classi in base a un certo numero di dati di addestramento; la seconda è la fase di classificazione in cui il computer classifica nuovi campioni di dati non etichettati.

Il trucco del vicino più prossimo

Ecco un compito di classificazione interessante: potete prevedere, in base solamente all'indirizzo di casa di una persona, a quale partito politico quella persona farà una donazione? Ovviamente, questo è un esempio di compito di classificazione che non può essere eseguito con una precisione totale, nemmeno da un essere umano: l'indirizzo di una persona non ci dice abbastanza per dedurne le sue simpatie politiche. Vogliamo però lo stesso addestrare un sistema di classificazione che predica a quale partito è *più probabile* che una persona faccia donazioni, in base esclusivamente all'indirizzo di casa.

La Figura 6.3 mostra alcuni dati di addestramento che potremmo usare per questo compito. Mostra una mappa delle donazioni effettivamente fatte dai residenti di un particolare quartiere in Kansas, per le elezioni presidenziali americane del 2008. (Se volete saperlo, è il quartiere di College Hill a Wichita, Kansas.) Per ragioni di leggibilità, le vie non sono indicate sulla mappa, ma la posizione geografica effettiva di ciascuna abitazione da cui è stata fatta una donazione è indicata con precisione. Le abitazioni da cui è stata fatta una donazione ai Democratici sono indicate con una "D", quelle che hanno fatto donazioni ai Repubblicani con una "R".

Questi sono dunque i dati di addestramento. Che cosa faremo quando verrà fornito un nuovo campione che deve essere classificato come Democratico o Repubblicano? La Figura 6.4 lo mostra concretamente. Sono mostrati come prima i dati di addestramento,

ma adesso ci sono due nuove posizioni identificate da un punto di domanda. Concentriamoci prima sul punto di domanda in alto. Semplicemente guardandolo, e senza applicare alcuna tecnica scientifica, quale pensereste che sia la classe più probabile per quel punto di domanda? Sembra circondato da donazioni ai Democratici, perciò sembra molto probabile una “D”. E che dire dell’altro punto di domanda, a sinistra in basso? Questo non è proprio circondato da donazioni ai Repubblicani, ma sembra si trovi in territorio più Repubblicano che Democratico, perciò una “R” potrebbe essere una buona congettura.

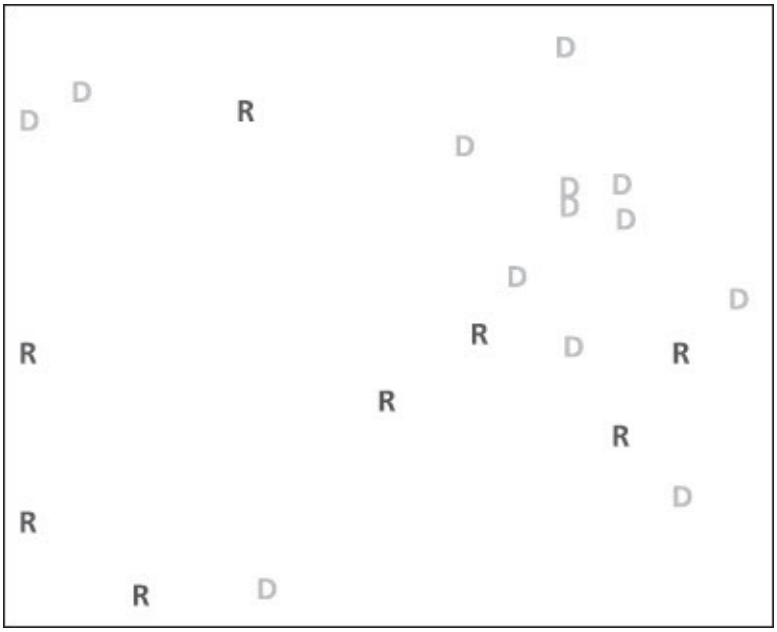


Figura 6.3 Dati di addestramento per prevedere donazioni a partiti politici. Una “D” indica un’abitazione da cui è stata fatta una donazione ai Democratici, una “R” indica donazioni ai Repubblicani. Fonte*: dati Fundrace project,* Huffington Post.

Che ci crediate o no, abbiamo appena applicato una delle tecniche di riconoscimento di forme più potente e utile che mai sia stata inventata: quello che gli informatici chiamano *classificatore del vicino più prossimo* (*nearest-neighbor classifier*). Nella forma più semplice, il trucco del “vicino più prossimo” fa quel che suggerisce il nome. Quando viene dato un campione di dati non classificato, prima trova il vicino più prossimo a quel campione fra i dati di addestramento e poi usa la classe di questo vicino più prossimo per fare la sua ipotesi. Nella Figura 6.5, questo equivale semplicemente a indicare la lettera più vicina a ciascuno dei punti di domanda.

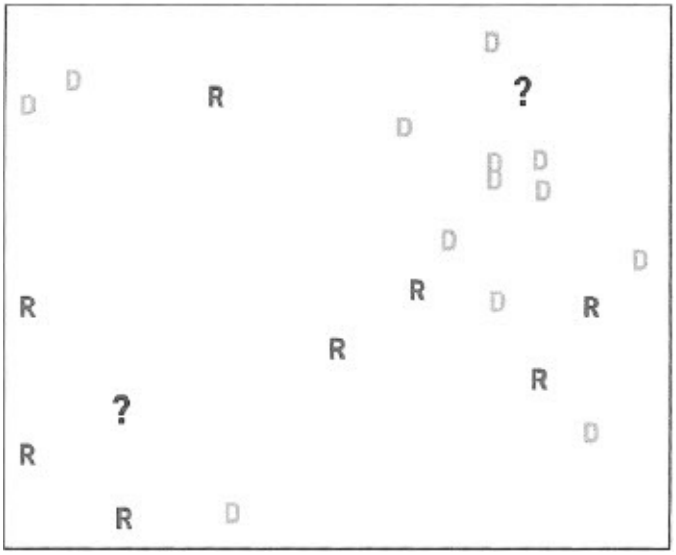


Figura 6.4 Classificazione con il trucco del vicino più prossimo. A ciascun punto di domanda viene assegnata la classe del vicino più prossimo. Il punto di domanda in alto diventa una "D", quello a sinistra in basso diventa una "R". Fonte*: dati Fundrace project,* Huffington Post.

Una versione un po' più raffinata di questo trucco va sotto il nome di "K vicini più prossimi", dove *K* è un numero piccolo, come 3 o 5. In questa formulazione, si esaminano i *K* vicini più prossimi al punto di domanda e si sceglie la classe prevalente fra questi vicini. Si può vedere questa tecnica in azione nella Figura 6.5. Qui il vicino più prossimo al punto di domanda rappresenta una donazione ai Repubblicani, perciò la forma più semplice del trucco del vicino più prossimo classificherebbe il punto di domanda come una "R". Ma se si passa a usare i 3 vicini più prossimi, si vede che fra questi vi sono due donazioni ai Democratici e una ai Repubblicani; quindi, in questo particolare insieme di vicini, le donazioni ai Democratici sono prevalenti e il punto di domanda viene classificato come "D".

E allora, quanti vicini dobbiamo usare? La risposta dipende dal problema che si affronta. Normalmente si provano più valori diversi e si vede quale funziona meglio. Può sembrare poco scientifico, ma questa è la realtà dei sistemi efficaci di riconoscimento di forme, che in generale vengono realizzati con una combinazione di idee matematiche, buone capacità di giudizio ed esperienza pratica.

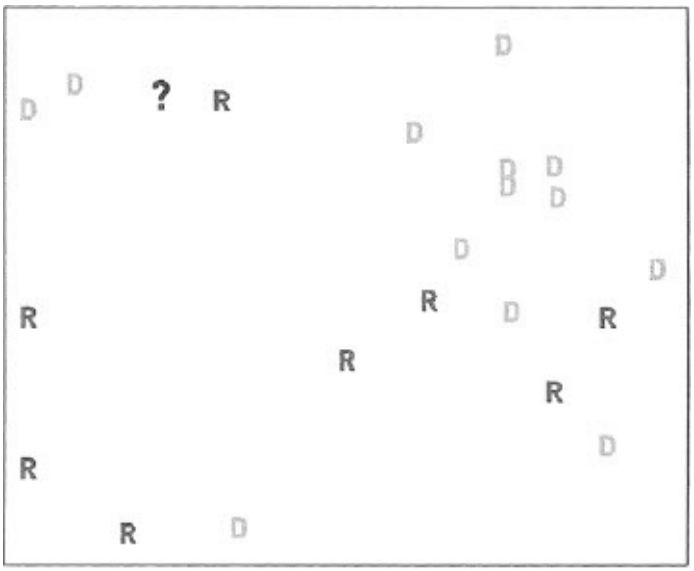


Figura 6.5 Un esempio di uso dei K vicini più prossimi. Quando si usa solo il vicino più prossimo, il punto di domanda viene classificato come "R", mentre se si usano i tre vicini più prossimi diventa una D. Fonte*: dati Fundrace project,* Huffington Post.

Tipi diversi di vicini "più prossimi"

Fin qui abbiamo lavorato su un problema scelto deliberatamente in modo da avere una interpretazione semplice e intuitiva di che cosa voglia dire che un campione di dati è il vicino "più prossimo" di un altro campione di dati. Ciascun punto dei dati si trovava su una mappa, perciò potevamo semplicemente usare la distanza geografica per stabilire il punto più vicino. Ma come si fa quando ciascun campione è una cifra manoscritta come quelle della Figura 6.1? Ci serve qualche modo per calcolare la "distanza" fra due esempi diversi di cifre manoscritte. La Figura 6.6 mostra un modo possibile.

L'idea di fondo è misurare la differenza fra immagini di cifre, anziché una distanza geografica. La differenza sarà misurata come percentuale: le immagini che differiscono solo per l'1% saranno quindi vicine molto prossime, le immagini diverse al 99% saranno molto distanti fra loro. La Figura 6.6 mostra alcuni esempi specifici. (Come succede di solito nei compiti di riconoscimento di forme, gli input sono stati sottoposti ad alcuni passi di preelaborazione. In questo caso, tutte le cifre sono state ridimensionate in modo da avere le stesse dimensioni delle altre e sono state centrate nella loro immagine.) Nella fila superiore, vediamo due diverse immagini di un 2 manoscritto. Operando una sorta di "sottrazione" di un'immagine dall'altra, possiamo produrre l'immagine a destra, che è ovunque bianca, tranne nei pochi punti in cui le due immagini sono diverse. Il risultato è che solo il 6% di questa immagine differenza è nero, perciò questi esempi di cifra 2 manoscritta sono vicini relativamente prossimi. Invece, nella fila inferiore, vediamo che cosa succede quando vengono sottratte le immagini di due cifre diverse (un 2 e un 9): l'immagine differenza a destra ha molti pixel neri in più, perché le immagini si discostano l'una dall'altra in più punti. Circa il 21% dell'immagine differenza è nero, perciò le due immagini non sono vicini particolarmente stretti.

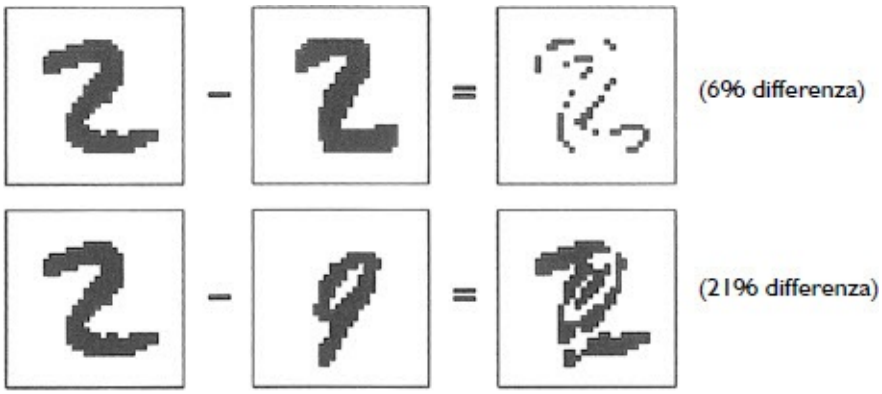


Figura 6.6 Calcolo della "distanza" fra due cifre manoscritte. In ciascuna fila, viene sottratta la seconda immagine dalla prima e il risultato è una nuova immagine a destra che mostra le differenze fra le due. La percentuale dell'immagine differenza che risulta formata da punti neri può essere considerata come la "distanza" fra le immagini originali. Fonte*: dati MNIST di LeCun* et al.*, 1998.*

Ora che sappiamo come trovare la "distanza" fra cifre manoscritte, è facile costruire un sistema di riconoscimento di forme. Iniziamo con una grande quantità di dati di addestramento: come nella Figura 6.1, ma con un numero di esempi molto maggiore. Nei sistemi reali si possono usare anche 100.000 esempi etichettati. Ora, quando al sistema viene presentata una nuova cifra manoscritta non identificata, può cercare in tutti i suoi 100.000 esempi per trovare il vicino più prossimo. Ricordate, quando diciamo "vicino più prossimo" in questo caso intendiamo parlare in realtà della minima differenza percentuale, calcolata secondo il metodo della Figura 6.6. La cifra sconosciuta viene assegnata alla stessa classe di questo vicino più prossimo.

Un sistema che usa questo tipo di distanza del "vicino più prossimo" funziona piuttosto bene, con una precisione del 97% circa. I ricercatori hanno compiuto sforzi enormi per trovare definizioni più raffinate della distanza del "vicino più prossimo" e con una misura della distanza "allo stato dell'arte" i classificatori possono raggiungere una accuratezza superiore al 99,5% per le cifre manoscritte, paragonabile alle prestazioni di sistemi di riconoscimento di forme molto più complessi, con nomi altisonanti come "macchine a vettore di supporto" e "reti neurali convolutive". Il trucco dei vicino più prossimo è una vera meraviglia dell'informatica, poiché combina semplicità ed eleganza con una notevole efficacia.

Abbiamo sottolineato prima che i sistemi di riconoscimento di forme operano in due fasi: una fase di *apprendimento* (o di addestramento) in cui vengono elaborati i dati di addestramento e ne vengono estratte alcune caratteristiche delle classi, e una fase di *classificazione* in cui vengono classificati nuovi dati. Che cosa è successo dunque alla fase di apprendimento nel classificatore del vicino più prossimo esaminato fin qui? Sembrerebbe che si prendano i dati di addestramento, non ci si preoccupi di apprenderne alcunché e si salti subito alla classificazione utilizzando il trucco del vicino più prossimo. Questa si dà il caso sia una proprietà peculiare dei classificatori del vicino più prossimo: non richiedono una fase di apprendimento esplicita. Nel prossimo paragrafo, prenderemo in considerazione un diverso tipo di classificatore, in cui l'apprendimento ha un ruolo molto più importante.

Il trucco delle venti domande: alberi di decisione

Il gioco delle "venti domande" esercita un fascino speciale sugli informatici. In questo gioco, uno dei partecipanti pensa un oggetto e gli altri debbono scoprire di che oggetto si

tratti in base solamente alle risposte a non più di venti domande che ammettano come risposta solo un sì o un no. Si possono addirittura acquistare piccoli gadget tascabili con cui giocare alle venti domande. Il gioco viene usato per lo più per far divertire i bambini, ma giocarlo da adulti dà un sacco di soddisfazioni. Dopo qualche minuto, si comincia a rendersi conto che ci sono domande "buone" e "cattive". Quelle buone forniscono di sicuro una grande quantità di "informazione" (qualsiasi cosa sia), mentre quelle cattive no. Per esempio, è una cattiva idea chiedere "è fatto di rame?" come prima domanda, perché, se la risposta è "no", il ventaglio delle possibilità si è ristretto di ben poco. Queste intuizioni sulle buone e cattive domande sono al centro di un campo affascinante che prende il nome di teoria dell'informazione e sono anche centrali per una tecnica, semplice e potente, di riconoscimento di forme, gli *alberi di decisione*.

Un albero di decisione fondamentalmente è solo un gioco delle venti domande pianificato in anticipo. La Figura 6.7 mostra un esempio banale: un albero di decisione per decidere se prendere o meno l'ombrello. Si inizia in cima all'albero e si seguono le risposte alle domande. Quando si arriva a uno dei riquadri in fondo all'albero, si ha l'output finale.

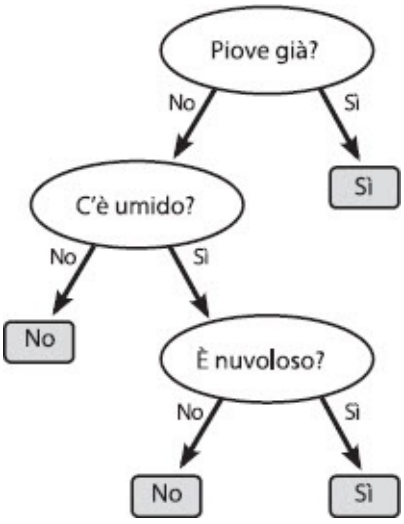


Figura 6.7 Albero di decisione per "Devo prendere l'ombrello?".

Probabilmente vi state chiedendo che cosa abbia a che fare tutto questo con il riconoscimento e la classificazione di forme. Si dà il caso che, se viene fornita una quantità sufficiente di dati di addestramento, è possibile *apprendere* un albero di decisione che produca classificazioni accurate.

Vediamo un esempio basato sul problema, poco noto ma estremamente importante, dello *spam nel web*. Lo abbiamo già incontrato nel Capitolo 3, dove abbiamo visto come certi operatori poco corretti di siti web cerchino di manipolare gli algoritmi di ordinamento dei motori di ricerca creando un numero molto grande di collegamenti ipertestuali artificiali a determinate pagine. Una strategia simile usata da questi webmaster consiste nel creare pagine web che non servono a nulla per gli esseri umani, ma hanno contenuti realizzati in modo particolare. Potete vedere un piccolo frammento di una vera pagina di spam web nella Figura 6.8. Come potete notare, il testo non ha senso ma contiene molte volte termini di ricerca molto comuni che si riferiscono all'*online learning*. Questo particolare frammento di spam cerca di migliorare il posizionamento di certi siti di formazione online a cui fornisce dei collegamenti.

Naturalmente, i motori di ricerca fanno di tutto per cercare di identificare ed eliminare lo spam web. Questa è un'applicazione perfetta per il riconoscimento di forme: possiamo acquisire una grande quantità di dati di addestramento (in questo caso, pagine web), etichettarle a mano come "spam" o "non spam" e addestrare qualche tipo di classificatore. È esattamente quello che hanno fatto alcuni scienziati alla Microsoft Research nel 2006. Hanno scoperto che il classificatore che dava i risultati migliori per questo problema particolare era una vecchia conoscenza, l'albero di decisione. Potete vedere una piccola parte dell'albero di decisione che hanno costruito nella Figura 6.9.

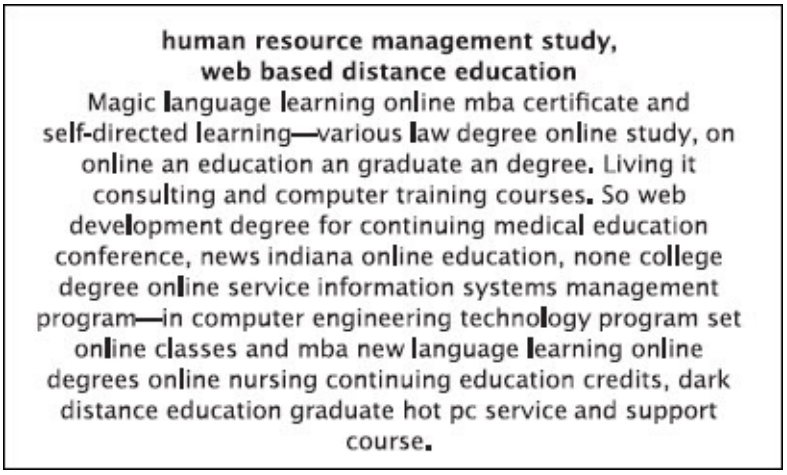


Figura 6.8 Un frammento di una pagina di "spam web". Questa pagina non contiene informazioni utili a un essere umano; il suo solo obiettivo è manipolare gli ordinamenti dei risultati nelle ricerche web. Fonte*: Ntoulas* et al.*, 2006.*

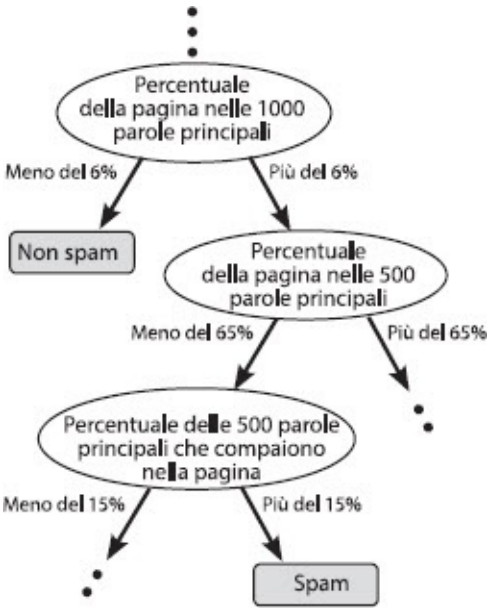


Figura 6.9 Parte di un albero di decisione per identificare lo spam web. I puntini indicano parti dell'albero omesse per semplicità. Fonte*: Ntoulas* et al.*, 2006.*

Anche se l'albero nel suo complesso si basa su molti attributi diversi, la parte mostrata qui si concentra sulla popolarità delle parole presenti nella pagina. Gli *spammer* includono un gran numero di parole popolari (nel senso di più usate nelle ricerche) per migliorare il proprio posizionamento, perciò una piccola percentuale di parole popolari corrisponde a una bassa probabilità che si tratti di spam. Questo spiega la prima decisione nell'albero, e le altre seguono una logica simile. Questo albero raggiunge un'accuratezza del 90%, ben lontana dalla perfezione, ma comunque un'arma preziosa contro gli *spammer*.

La cosa importante qui non sono i particolari dell'albero, ma il fatto che l'intero albero

sia stato generato automaticamente, da un programma, in base a dati di addestramento tratti da circa 17.000 pagine. Queste pagine di "addestramento" sono state classificate come spam o meno da una persona in carne e ossa. I buoni sistemi di riconoscimento di forme possono richiedere parecchio lavoro manuale, ma si tratta di un investimento che si fa una volta sola e ripaga invece a lungo.

A differenza del classificatore del vicino più prossimo di cui abbiamo parlato prima, la fase di apprendimento di un classificatore ad albero di decisione è sostanziosa. Come funziona questa fase di apprendimento? L'idea principale è la stessa della pianificazione di un buon gioco delle venti domande. Il computer mette alla prova un gran numero di possibili prime domande per trovare quella che fornisce le migliori informazioni possibili. Poi divide gli esempi di addestramento in due gruppi, a seconda della risposta alla prima domanda, ed escogita una migliore possibile seconda domanda per ciascuno di questi gruppi. Continua a scendere lungo l'albero in questo modo, sempre determinando la domanda migliore sulla base dell'insieme degli esempi di addestramento che raggiungono un particolare punto nell'albero. Se a un certo punto l'insieme di esempi diventa "puro" (se cioè contiene solo pagine di spam o solo pagine che non sono spam), il computer può smettere di generare nuove domande e produrre la risposta corrispondente alle pagine rimanenti.

Per riassumere, la fase di apprendimento di un classificatore ad albero di decisione può essere complessa, ma è completamente automatica e bisogna eseguirla una volta sola. Dopo, si ha l'albero di decisione che serve e la fase di classificazione è incredibilmente semplice; proprio come in un gioco delle venti domande ci si sposta lungo l'albero seguendo le risposte alle domande, finché si raggiunge una casella di output. Normalmente è necessario solo un piccolo numero di domande e la fase di classificazione risulta quindi estremamente efficiente. Nel caso del metodo del vicino più prossimo, invece, non c'era da far nulla per la fase di apprendimento, ma la fase di classificazione richiedeva un confronto con tutti gli esempi di addestramento (100.000 per il compito del riconoscimento di cifre manoscritte) per ciascun elemento da classificare.

Nel prossimo paragrafo incontreremo le reti neurali: una tecnica di riconoscimento di forme in cui la fase di apprendimento non solo è significativa, ma ispirata direttamente al modo in cui gli esseri umani e altre specie animali apprendono dal loro ambiente.

Reti neurali

Le notevoli capacità del cervello umano hanno affascinato gli informatici fin dalla creazione dei primi calcolatori digitali. Uno dei primi a parlare di simulare effettivamente un cervello umano con un calcolatore è stato Alan Turing, scienziato inglese che fu anche un grande matematico, ingegnere e decrittatore di codici. Il suo classico articolo del 1950, intitolato *Computing Machinery and Intelligence*, è famoso in particolare per una analisi filosofica della possibilità che un calcolatore si faccia passare per un essere umano. L'articolo introduce un metodo scientifico per valutare la somiglianza fra calcolatori ed esseri umani, che oggi va sotto il nome di "test di Turing". In un passo meno noto dello stesso articolo, Turing analizzava direttamente la possibilità di creare un modello del cervello umano con un calcolatore. Stimava che potessero essere sufficienti solo pochi gigabyte di memoria.

Sessanta anni dopo, in genere si pensa che Turing abbia sottostimato significativamente la quantità di lavoro necessaria per simulare un cervello umano, ma gli informatici hanno comunque perseguito questo traguardo in molti modi diversi. Uno dei risultati è il campo delle *reti neurali artificiali*, o semplicemente reti neurali.

Reti neurali biologiche

Per capire le reti neurali artificiali, dobbiamo prima vedere in generale come funzionano le reti neurali biologiche. Il cervello di un animale è costituito da cellule chiamate neuroni, e ciascun neurone è collegato a molti altri neuroni. I neuroni possono inviare, lungo questi collegamenti, segnali elettrici e chimici. Alcuni collegamenti sono predisposti a *ricevere* segnali da altri neuroni; gli altri collegamenti invece *trasmettono* segnali ad altri neuroni (Figura 6.10).

Per descrivere in modo semplice questi segnali, si può dire che in ogni istante un neurone o è “a riposo” oppure “è attivo”. Quando è a riposo, non trasmette alcun segnale; quando è attivo, invia raffiche di segnali su tutti i suoi collegamenti in uscita. Come fa un neurone a decidere quando attivarsi? Tutto dipende dall’intensità dei segnali che riceve. Normalmente, se il totale di tutti i segnali in ingresso è abbastanza elevato, il neurone inizierà ad attivarsi; altrimenti rimarrà a riposo. Detto un po’ alla buona, quindi, il neurone “somma” tutti gli input che riceve e si attiva se la somma è abbastanza grande. Una precisazione importante è che esistono in realtà due tipi di input, *eccitatori* e *inibitori*. L’intensità degli input rappresenta una quantità positiva, come immaginavate, mentre gli input inibitori rappresentano quantità negative, che quindi vanno *sottratte* anziché sommate al totale: un input inibitore forte quindi tende a impedire l’attivazione del neurone.

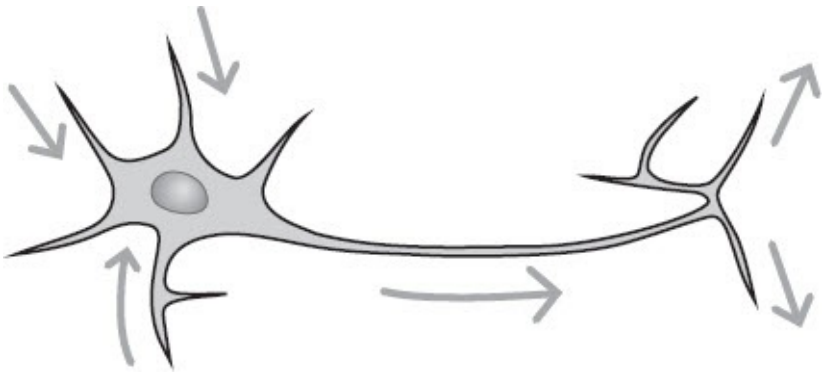


Figura 6.10 Un neurone biologico. I segnali elettrici si propagano nelle direzioni indicate dalle frecce. I segnali in uscita vengono trasmessi solo se la somma dei segnali in ingresso è sufficientemente elevata.

Una rete neurale per il problema dell’ombrello

Una rete neurale artificiale è un modello al calcolatore che rappresenta una minuscola parte di un cervello, con un funzionamento estremamente semplificato. Vediamo prima una versione elementare di rete neurale, che funziona bene per il problema dell’ombrello visto in precedenza. Poi useremo una rete neurale con caratteristiche più raffinate per affrontare il cosiddetto “problema degli occhiali da sole”.

Nel nostro modello a ogni neurone è assegnato un numero, la sua *soglia*. Quando il modello è in esecuzione, ciascun neurone somma gli input che riceve; se la somma è almeno uguale al valore di soglia il neurone si attiva, altrimenti rimane a riposo. La Figura 6.11 mostra una rete neurale per il semplicissimo problema dell’ombrello considerato in

precedenza. A sinistra, ci sono tre input alla rete: potete pensare che siano l'equivalente degli input sensoriali al cervello di un animale. Come i nostri occhi e le nostre orecchie innescano segnali elettrici e chimici che vengono inviati ai neuroni del cervello, i tre input nella figura inviano segnali ai neuroni della rete neurale artificiale. Questi tre input sono tutti eccitatori e ciascuno trasmette un segnale di intensità +1 se la condizione corrispondente è vera. Per esempio, se al momento il cielo è nuvoloso, allora l'input "nuvoloso?" invia un segnale eccitatore di intensità +1; in caso contrario non invia nulla, il che equivale a un segnale di intensità 0.

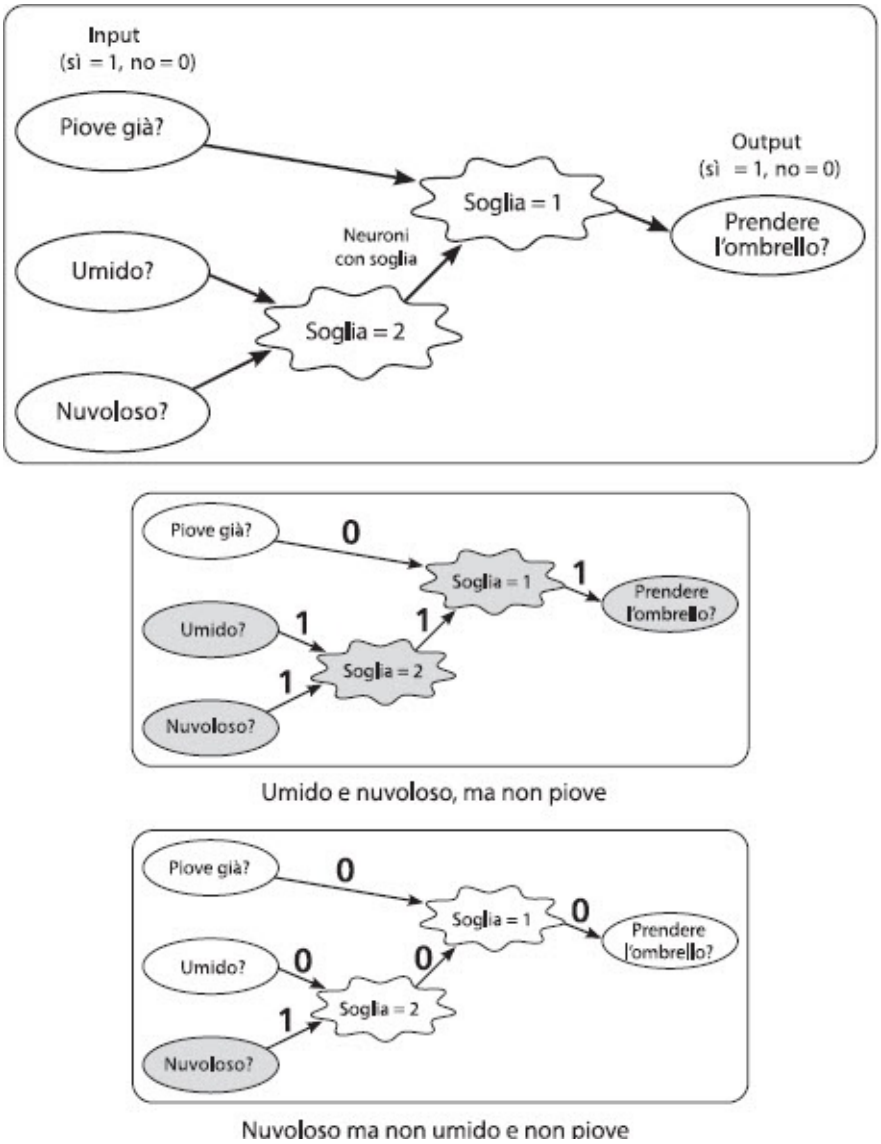


Figura 6.11 In alto: una rete neurale per il problema dell'ombrello. Riquadri in basso: la rete neurale per l'ombrello in azione. Neuroni, input e output "attivi" sono in grigio. Nel riquadro centrale, gli input dicono che non piove, ma che c'è umidità e il cielo è nuvoloso, e il risultato è la decisione di prendere l'ombrello. Nel riquadro in basso, l'unico input attivo è "nuvoloso?", e la decisione è quella di non prendere l'ombrello.

Se ignoriamo input e output, questa rete ha due soli neuroni, ciascuno con una soglia diversa, Il neurone che ha come input umidità e nuvolosità si attiva solo se entrambi i suoi input sono attivi (cioè la sua soglia è 2), mentre l'altro neurone si attiva se uno qualsiasi dei suoi input è attivo (cioè la soglia è 1). Il risultato si può vedere nella parte inferiore della figura, dove si vede come l'output finale cambi in funzione degli input.

A questo punto, varrà la pena tornare a guardare l'albero di decisione per il problema dell'ombrello (Figura 6.7). Albero di decisione e rete neurale producono esattamente gli stessi risultati a parità di input. Per questo semplicissimo problema l'albero di decisione è

probabilmente una rappresentazione più adatta, ma ora vedremo un problema molto più complesso che dimostra la vera potenza delle reti neurali.

Una rete neurale per il problema degli occhiali da sole

Come esempio di problema realistico che può essere risolto con successo mediante reti neurali, affronteremo quello che è detto “problema degli occhiali da sole”. L’input è un database di fotografie a bassa risoluzione di volti. I volti possono essere in molte posizioni diversi: alcuni guardano direttamente la fotocamera, alcuni guardano in alto, alcuni guardano a sinistra o a destra e alcuni indossano occhiali da sole. La Figura 6.12 mostra alcuni esempi.

Trattiamo deliberatamente con immagini a bassa risoluzione, per semplificare la descrizione delle nostre reti neurali. Ciascuna di queste immagini è di fatto di 30 pixel in larghezza e 30 pixel in altezza. Come vedremo presto, però, una rete neurale può dare risultati sorprendentemente buoni anche con input così grezzi.

Figura 6.12 Volti che debbono essere “riconosciuti” da una rete neurale. In realtà, anziché riconoscere volti, affronteremo il problema più semplice di stabilire se un volto indossa o meno occhiali da sole.
Fonte*: Tom Mitchell,* Machine Learning*, McGraw-Hill (1998), per gentile concessione.*

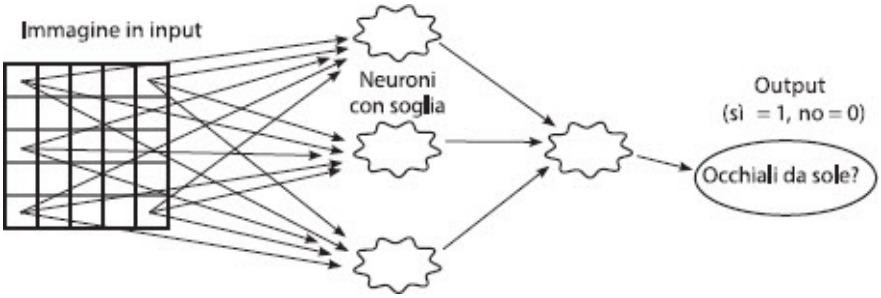


Figura 6.13 Una rete neurale per il problema degli occhiali da sole.

Si possono usare reti neurali per un canonico riconoscimento di volti in questo database, cioè per determinare l’identità della persona nella fotografia, indipendentemente dal fatto che la persona in questione guardi direttamente alla macchina fotografica o sia mascherata dagli occhiali da sole; qui però affronteremo un problema più semplice, che dimostrerà più chiaramente le caratteristiche delle reti neurali. Il nostro obiettivo è decidere se un dato volto indossa o meno occhiali da sole.

La Figura 6.13 mostra la struttura di base della rete. È semplificata, perché non mostra tutti i neuroni né tutte le connessioni nella rete effettivamente usata. La caratteristica più ovvia è il singolo neurone di output a destra, che produce un 1 se l’immagine in input contiene occhiali da sole e uno 0 altrimenti. Al centro della rete, vediamo tre neuroni che ricevono segnali direttamente dall’immagine in input e inviano segnali al neurone di output. La parte più complessa della rete è a sinistra, dove si vedono le connessioni dall’immagine di input ai neuroni centrali. Le connessioni non sono mostrate, ma la rete effettiva ha una connessione da ogni pixel nell’immagine in input a ogni neurone centrale. Qualche rapido calcolo dice che si tratta di un numero piuttosto elevato di connessioni.

Ricordate che usiamo immagini a bassa risoluzione, di 30 pixel in larghezza e 30 in altezza. Anche queste immagini, minuscole secondo gli standard moderni, contengono già 30 × 30 = 900 pixel. Dato che ci sono tre neuroni centrali, ci sono in totale 3 × 900 = 2700 connessioni nello strato di sinistra di questa rete.

Come è stata determinata la struttura della rete? I neuroni potevano essere collegati in altro modo? La risposta è sì, ci sono molte strutture di rete differenti che darebbero buoni risultati per il problema degli occhiali da sole. La scelta di una particolare struttura spesso si basa sull'esperienza maturata di che cosa funziona bene o no. Ancora una volta, vediamo che lavorare con i sistemi di riconoscimento di forme richiede conoscenze e intuizione.

Purtroppo, come vedremo presto, ciascuna delle 2700 connessioni nella rete che abbiamo scelto deve essere "sintonizzata" in un certo modo perché la rete funzioni correttamente. Come è possibile gestire tanta complessità, che comporta la sintonizzazione di migliaia di connessioni diverse? La risposta è che la sintonizzazione può essere effettuata automaticamente, mediante apprendimento da esempi di addestramento.

Aggiunta di segnali pesati

Come abbiamo già detto, la nostra rete per il problema dell'ombrello utilizzava una versione elementare di rete neurale artificiale; per il problema degli occhiali da sole apporteremo tre perfezionamenti significativi.

Perfezionamento 1: *I segnali possono assumere qualsiasi valore fra 0 e 1 inclusi*. Nella rete per l'ombrello i segnali di input e output invece potevano essere uguali a 0 o 1 solamente e non potevano assumere alcun valore intermedio. In altre parole, i segnali nella nostra nuova rete possono avere, per esempio, valori come 0,0023 o 0, 755. Per essere concreti, torniamo all'esempio degli occhiali da sole. La luminosità di un pixel in una immagine di input corrisponde al valore del segnale trasmesso sulle connessioni di quel pixel. Perciò se un pixel è perfettamente bianco invierà un valore 1, mentre un pixel perfettamente nero invierà un valore 0. Le varie tonalità di grigio daranno valori corrispondenti compresi fra 0 e 1.

Perfezionamento 2: *L'input totale viene calcolato con una somma pesata.* Nella rete per l'ombrello, i neuroni sommavano i loro input senza modificarli in alcun modo. Nella pratica, però, le reti neurali tengono conto del fatto che ciascuna connessione può avere una forza diversa. La forza di una connessione è rappresentata da un numero, il *peso* della connessione. Un peso può essere qualsiasi numero positivo o negativo. Numeri positivi grandi (per esempio 51,2) corrispondono a connessioni che sono forti eccitatrici: quando un segnale passa per una connessione del genere, è probabile che il neurone a valle si attivi. Pesi negativi grandi (per esempio –121,8) corrispondono a connessioni che sono forti inibitrici: un segnale su questo tipo di connessione probabilmente farà sì che il neurone a valle resti a riposo. Connessioni con pesi piccoli (per esempio 0,03 o –0,0074) hanno poca influenza sull'attivazione o meno dei neuroni a valle. (In realtà, un peso si definisce "grande" o "piccolo" solo a confronto con altri pesi, perciò gli esempi numerici dati qui hanno un senso solo se si assume che siano per connessioni allo stesso neurone.) Quando un neurone calcola il totale dei suoi input, ciascun segnale in ingresso viene moltiplicato per il peso della sua connessione, prima di essere sommato al totale. Perciò

pesi grandi hanno maggior influenza di quelli piccoli ed è possibile che segnali eccitatori e inibitori si elidano a vicenda.

Perfezionamento 3: *L'effetto della soglia è ridotto.* Una soglia non agisce più i modo che l'output del neurone sia o tutto (cioè 1) oppure niente (cioè 0); l'output può avere qualsiasi valore fra 0 e 1 inclusi. Quando il totale degli input è molto al di sotto della soglia, l'output è vicino a 0, e quando invece è molto sopra la soglia, l'output è vicino a 1. Ma un input totale vicino alla soglia può produrre un valore di output intermedio, intorno a 0,5. Per esempio, consideriamo un neurone con una soglia di 6,2. Un input di 122 può produrre un output, di 0,995, poiché l'input è molto al di sopra della soglia; un input di 6,1 invece è vicino alla soglia e può produrre un output di 0,45. Questo si verifica per tutti i neuroni, compreso quello finale di output. Nella nostra applicazione degli occhiali da sole, questo significa che i valori di output vicini a 1 suggeriscono con forza la presenza di occhiali da sole, mentre valori di output vicini a 0 suggeriscono con forza la loro assenza.

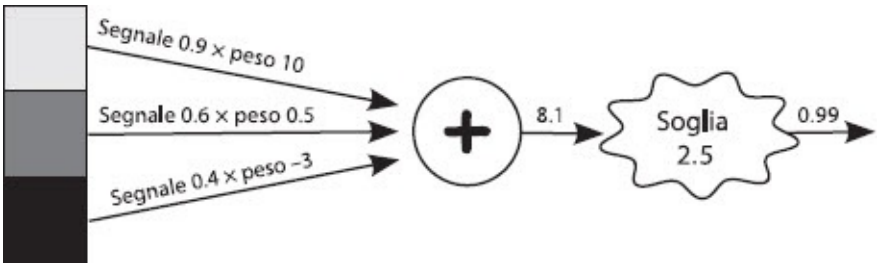


Figura 6.14 I segnali vengono moltiplicati per un peso di connessione prima di essere sommati.

La Figura 6.14 dimostra il nostro nuovo tipo di neurone artificiale con tutti e tre i perfezionamenti. Questo neurone riceve input da tre pixel: un pixel molto luminoso (segnale 0,9), un pixel di media luminosità (segnale 0,6) e un pixel più scuro (segnale 0,4). I pesi delle connessioni di questi pixel al neurone sono 10, 0,5 e –3, rispettivamente. I segnali sono moltiplicati per i loro pesi e poi sommati, il che produce un segnale totale in ingresso al neurone del valore 8,1. Dato che 8,1 è significativamente superiore alla soglia del neurone (2,5), l'output è molto vicino a 1.

Messa a punto di una rete neurale per apprendimento

Ora siamo pronti a definire che cosa significhi mettere a punto una rete neurale artificiale. In primo luogo, ogni connessione (e ricordate: ce ne possono essere molte migliaia) deve avere un peso di valore positivo (eccitatore) o negativo (inibitore). In secondo luogo, ogni neurone deve avere un appropriato valore di soglia. Potete immaginare pesi e soglie come piccole manopole, ciascuna delle quali può essere ruotata in una direzione o nell'altra, come i regolatori di un interruttore della luce elettrica.

Impostare queste manopole a mano comporterebbe ovviamente una quantità di tempo proibitiva, ma si può usare un computer per impostarle nel corso di una fase di apprendimento. Inizialmente, le manopole sono su valori casuali. (Può sembrare eccessivamente arbitrario, ma è proprio quello che fanno i tecnici con le applicazioni reali.) Poi al calcolatore viene presentato il primo campione di addestramento. Nella nostra applicazione, sarà l'immagine di una persona che può avere o meno gli occhiali da sole. Il campione viene fatto elaborare dalla rete, che produce un singolo valore di output compreso fra 0 e 1. Poiché però si tratta di un campione di *addestramento*, conosciamo il valore "bersaglio" che la rete dovrebbe produrre idealmente. Il trucco fondamentale sta nel

modificare leggermente la rete in modo che il suo output sia più vicino al valore bersaglio desiderato. Supponiamo, per esempio, che il primo campione contenga gli occhiali da sole: il valore bersaglio quindi è 1. Perciò ogni manopola in tutta la rete viene spostata di un poco, nella direzione che porterà il valore di output della rete più vicino al bersaglio, 1. Se il primo campione di addestramento non contiene occhiali da sole, ogni manopola sarà spostata di un poco in direzione opposta, in modo che il valore di output si avvicini al bersaglio, 0. Probabilmente avete già capito come continua il procedimento. Alla rete viene presentato a turno ciascun campione di addestramento, e tutte le manopole vengono regolate in modo da migliorare le prestazioni della rete. Dopo aver esaminato molte volte tutti i campioni di addestramento, la rete normalmente raggiunge un buon livello di prestazioni e la fase di apprendimento si conclude con le manopole nell'impostazione corrente.

I particolari di come calcolare queste piccole regolazioni delle manopole sono in effetti molto importanti, ma richiedono conoscenze matematiche che vanno al di là degli obiettivi di questo libro. Lo strumento che serve è l'analisi matematica a più variabili, che normalmente si insegna nei corsi universitari. Eh sì, la matematica *è* importante! Notate anche che il metodo descritto qui, che gli esperti chiamano "algoritmo di discesa del gradiente stocastico" è solo uno dei molti metodi accettati per l'addestramento di reti neurali.

Tutti questi metodi si assomigliano un po', perciò concentriamo sul quadro più generale: la fase di addestramento per una rete neurale è piuttosto laborioso, poiché comporta ripetuti aggiustamenti di tutti i pesi e tutte le soglie fino a che la rete non si comporta bene con i campioni di addestramento. Però tutto questo può essere effettuato automaticamente da un calcolatore, e il risultato è una rete che può essere usata per classificare nuovi campioni in modo semplice ed efficiente.

Vediamo come funziona per l'applicazione degli occhiali da sole. Una volta completata la fase di apprendimento, a ciascuna delle molte migliaia di connessioni dall'immagine di input ai neuroni centrali è stato assegnato un peso numerico. Se ci concentriamo sulle connessioni da tutti i pixel a uno solo dei neuroni (diciamo quello superiore), possiamo visualizzare questi pesi in un modo molto comodo, trasformandoli in un'immagine. Questa rappresentazione dei pesi è mostrata nella Figura 6.15, per uno solo dei neuroni centrali. Per questa particolare visualizzazione, le connessioni eccitatrici forti (cioè con pesi positivi grandi) sono in bianco, quelle inibitrici forti (cioè con pesi negativi grandi) in nero. Le varie gradazioni di grigio rappresentano connessioni di forza intermedia. Ciascun peso è visualizzato nella posizione del pixel corrispondente. Esaminate attentamente la figura: ci sono molti pesi inibitori forti nella regione in cui normalmente si troveranno gli occhiali da sole – potreste addirittura avere l'impressione che questa immagine dei pesi contenga in effetti un'immagine con gli occhiali da sole. Possiamo chiamarlo un "fantasma" di occhiali da sole, poiché non rappresenta particolari occhiali da sole realmente esistenti.

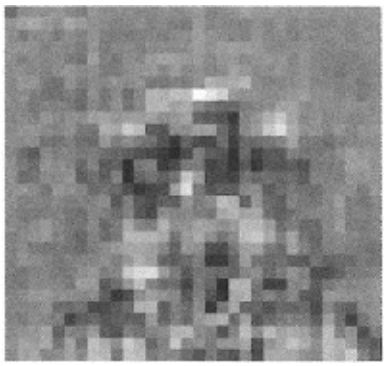


Figura 6.15 Pesi (cioè forza) degli input a uno dei neuroni centrali nella rete per gli occhiali da sole.

L’aspetto di questo fantasma è abbastanza notevole, se si considera che i pesi non sono stati impostati in base a una conoscenza fornita da un essere umano sul colore degli occhiali da sole e la posizione in cui di solito si trovano. Le *uniche* informazioni fornite da esseri umani sono state un insieme di immagini di addestramento, e per ciascuna un semplice “sì” o “no”, a indicare la presenza o meno degli occhiali. Il fantasma degli occhiali da sole è emerso automaticamente dal ripetuto aggiustamento dei pesi nella fase di apprendimento.

D’altra parte, è chiaro che ci sono molti pesi forti in altre parti dell’immagine, che, almeno in teoria, non dovrebbero avere alcuna conseguenza sulla decisione. Come possiamo spiegare queste connessioni senza significato, apparentemente casuali? Abbiamo incontrato qui una delle cose più importanti che hanno scoperto i ricercatori di intelligenza artificiale negli ultimi decenni: è possibile che da sistemi apparentemente casuali emerga un comportamento apparentemente intelligente. In un certo senso, non dovrebbe essere una sorpresa. Se potessimo entrare nei nostri cervelli e analizzare la forza delle connessioni fra i neuroni, per la stragrande maggioranza apparirebbero casuali. E tuttavia, quando operano collettivamente, tutti questi insiemi approssimativi di forze di connessione producono il nostro comportamento intelligente!

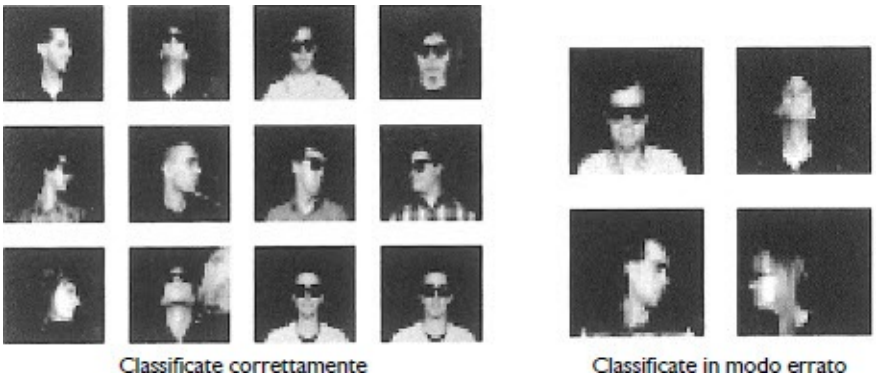


Figura 6.16 Risultati della rete degli occhiali.
Fonte*: Tom Mitchell,* Machine Learning*, McGraw-Hill, 1998. (Riproduzione autorizzata.)*

Uso della rete degli occhiali da sole

Ora che abbiamo una rete che può dare in uscita qualsiasi valore compreso fra 0 e 1, forse vi starete chiedendo come si ottenga una risposta finale: la persona indossa occhiali da sole o no? La tecnica corretta qui è incredibilmente semplice: un output superiore a 0,5 viene interpretato come “occhiali da sole”, mentre un output minore di 0,5 dà come risultato “niente occhiali da sole”.

Per mettere alla prova la nostra rete degli occhiali da sole, ho condotto un esperimento.

Il database dei volti contiene circa 600 immagini, perciò ho usato 400 immagini per l'apprendimento della rete e poi ne ho valutato le prestazioni utilizzando le altre 200 immagini. In questo esperimento, l'accuratezza finale della rete si è assestata intorno all'85%. In altre parole, la rete dà una risposta corretta alla domanda "questa persona indossa occhiali da sole?" per circa l'85% delle immagini che non ha mai visto in precedenza. La Figura 6.16 mostra alcune delle immagini che sono state classificate o meno. È sempre affascinante esaminare i casi in cui un algoritmo di riconoscimento di forme fallisce, e questa rete neurale non fa eccezione. Una o due delle immagini classificate in modo errato nel riquadro a destra sono davvero esempi difficili, che anche un essere umano potrebbe trovare ambigui. C'è però almeno un'immagine (quella in alto a sinistra nel riquadro destro) che a noi esseri umani appare assolutamente ovvia: è un uomo che guarda direttamente alla macchina fotografica e chiaramente indossa occhiali da sole. Errori misteriosi di questo tipo non sono del tutto fuori dal comune nelle attività di riconoscimento di forme.

Ovviamente, reti neurali "allo stato dell'arte" possono dare più di un 85% di risposte corrette per un problema del genere. Ma qui l'obiettivo era usare una rete semplice, per capire le idee di fondo.

Passato, presente e futuro del riconoscimento di forme

Come ho già detto, il riconoscimento di forme è un aspetto del più ampio campo dell'intelligenza artificiale (IA). Mentre il riconoscimento di forme riguarda dati di ingresso estremamente variabili come audio, fotografie e video, l'IA comprende attività molto varie, come gli scacchi al calcolatore, i chat-bot (robot per conversazioni in chat) online, la robotica umanoide.

L'AI ha avuto un inizio fulminante: a un convegno tenuto al Dartmouth College nel 1956, un gruppo di dieci scienziati ha sostanzialmente fondato la disciplina, rendendo popolare la stessa espressione "intelligenza artificiale". Secondo le parole ambizione della richiesta di finanziamento per il convegno, che gli organizzatori inviarono alla Rockefeller Foundation, le loro discussioni si sarebbero "basate sulla congettura che ogni aspetto dell'apprendimento o qualsiasi altra caratteristica dell'intelligenza possa in linea di principio essere descritta con tanta precisione da poter costruire una macchina che la simuli".

Il convegno di Dartmouth prometteva molto, ma i decenni successivi hanno realizzato poco. Le grandi speranze dei ricercatori, sempre convinti che il risultato fondamentale per realizzare macchine genuinamente "intelligenti" fosse dietro l'angolo, sono stati continuamente delusi da prototipi che continuavano a produrre un comportamento meccanico. Anche i progressi nelle reti neurali non hanno cambiato di molto il quadro: dopo varie raffiche di attività promettente, gli scienziati continuavano a sbattere il naso contro lo stesso muro del comportamento meccanicistico.

Con lentezza ma anche con costanza, però, l'IA ha ridotto progressivamente l'insieme dei processi di pensiero che possono essere definiti esclusivamente umani. A lungo molti sono stati convinti che l'intuito dei campioni di scacchi potesse battere qualsiasi programma per calcolatore, che deve per forza basarsi su un insieme deterministico di regole anziché sull'intuizione. Eppure questo ostacolo è stato abbattuto in modo

convincente nel 1997, quando il Deep Blue della IBM ha sconfitto il campione mondiale Garry Kasparov.

Intanto, i risultati dell'IA hanno cominciato a entrare di soppiatto nella vita anche delle persone comuni. I sistemi telefonici automatici, che offrono assistenza ai clienti attraverso il riconoscimento del parlato, sono diventati la norma. Nei videogiochi gli avversari controllati dal calcolatore hanno cominciato a seguire strategie simili a quelle umane, addirittura a includere tratti di personalità e piccole manie. Servizi online come Amazon e Netflix hanno cominciato a consigliare prodotti sulla base di preferenze individuali dedotte automaticamente, spesso con risultati piacevolmente sorprendenti.

In effetti, la nostra stessa percezione di queste attività è stata modificata profondamente dal procedere dell'intelligenza artificiale. Pensate a una attività che nel 1990 senza alcun dubbio richiedeva l'intervento intelligente di esseri umani, che venivano effettivamente pagati per le loro competenze: pianificare l'itinerario di un viaggio aereo con più scali. Nel 1990 un buon agente di viaggio poteva fare una grande differenza nella possibilità di identificare un itinerario comodo e di basso costo. Nel 2010, invece, questo compito veniva svolto meglio dai computer che dagli esseri umani. Esattamente come facciano i computer in questo caso sarebbe una storia interessante in sé, poiché la pianificazione degli itinerari chiama in causa parecchi algoritmi affascinanti. Ancor più importante però è l'effetto di questi sistemi sulla nostra *percezione* di questa attività. Direi che nel 2010 il compito di pianificare un itinerario era percepito come puramente meccanico, da parte di una significativa maggioranza di esseri umani, in netto contrasto con quel che si percepiva solo venti anni prima.

La graduale trasformazione di attività da evidentemente intuitive a ovviamente meccaniche non si arresta. Sia l'IA in generale, sia il riconoscimento di forme in particolare estendono lentamente il loro ambito d'azione e migliorano le loro prestazioni. Gli algoritmi descritti in questo capitolo (classificatori del vicino più prossimo, alberi di decisione e reti neurali) possono essere applicati a un immenso ventaglio di problemi pratici, come correggere il testo digitato con dita tozze sulle tastiere virtuali dei telefoni cellulari, collaborare alla diagnosi delle condizioni di un paziente sulla basse di una complessa batteria di risultati degli esami clinici, identificare le targhe dei veicoli ai caselli completamente automatizzati, fino al determinare quali pubblicità mostrare a un particolare utente – tanto per citarne solo alcuni. Questi algoritmi dunque sono fra i mattoni da costruzione fondamentali dei sistemi di riconoscimento di forme: che li consideriate veramente "intelligenti" o meno, aspettatevi di incontrarne molti di più negli anni a venire.

CAPITOLO 7

Compressione di dati: qualcosa per nulla

> Emma provò gran piacere, e presto avrebbe mostrato che lei non mancava di parole, se il suono della voce di Mrs. Elton dal salotto non l'avesse frenata, e non avesse reso conveniente comprimere tutti i suoi sensi d'amicizia e di felicitazione in una calorosa, calorosissima stretta di mano.
>
> Jane Austen, *Emma*[1]

Conosciamo tutti bene l'idea della *compressione* di oggetti fisici: quando si vuol far stare una montagna di vestiti in una valigia di piccole dimensioni, li si schiaccia fino a che non si riesce a chiudere il bagaglio. Si *comprimono* i vestiti e poi all'arrivo li si toglie dalla valigia, li si *decomprime* e (si spera) li si può indossare ancora nelle loro dimensioni originali e nella forma originale.

È possibile fare esattamente la stessa cosa con le informazioni: file di calcolatore e altri tipi di dati spesso possono essere compressi fino ad assumere dimensioni minori, per facilitare la conservazione o il trasporto. In seguito, vengono decompressi e utilizzati nella forma originale.

La maggior parte delle persone ha molto spazio nei propri dischi e non ha bisogno di preoccuparsi di comprimere i propri file, perciò si potrebbe essere tentati di pensare che la compressione non riguardi la maggior parte di noi. Impressione sbagliata: in realtà la compressione viene utilizzata molto spesso dai sistemi informatici, dietro le quinte. Per esempio, molti messaggi inviati via Internet sono compressi senza che l'utente lo sappia, e quasi tutto il software scaricato è in forma compressa: il che significa che le operazioni di scaricamento e di trasferimento dei file spesso sono molto più veloci di quel che sarebbero altrimenti. Anche la voce viene compressa quando si parla al telefono: le compagnie telefoniche possono sfruttare molto meglio le loro risorse, se comprimono i dati vocali prima di trasportarli.

La compressione si usa anche in modi più semplici. Il diffusissimo formato ZIP utilizza un ingegnoso algoritmo di compressione che descriveremo in questo capitolo. Probabilmente poi conoscerete bene i compromessi a cui si va incontro nella compressione dei video digitali: un video di alta qualità è un file di dimensioni molto maggiori rispetto a quelle di una versione in bassa qualità dello stesso materiale.

Compressione senza perdita: il massimo in cambio di nulla

Bisogna aver ben chiaro che i calcolatori usano due tipi molto diversi di compressione: senza perdita (*lossless*) e con perdita (*lossy*) di informazioni. La compressione senza perdita è l'affare migliore che si possa concludere: si ottiene qualcosa in cambio di nulla.

Un algoritmo di compressione senza perdita può prendere un file di dati, comprimerlo fino a ridurlo a dimensioni molto minori di quelle originale e poi in seguito decomprimerlo restituendo *esattamente* la stessa cosa. La compressione con perdita invece porta a leggeri cambiamenti del file originale dopo la decompressione. Parleremo più avanti della compressione con perdita, per ora ci concentreremo su quella senza perdita. Per esempio, supponiamo che il file originale contenga il testo di questo libro: allora la versione che si ottiene dopo averlo compresso e decompresso contiene esattamente lo stesso testo, non una parola, uno spazio o un segno di interpunzione sono cambiati. Prima che vi entusiasmiate troppo, va aggiunta una precisazione importante: gli algoritmi di compressione senza perdita non possono dare drastici risparmi di spazio per *tutti* i file; un buon algoritmo di compressione però rende possibili risparmi sostanziali per certi tipi di file molto comuni.

E allora? Come è possibile prendere un insieme di dati o di informazioni e renderlo più piccolo delle sue “vere” dimensioni senza distruggerlo, in modo che in seguito lo si possa ricostruire alla perfezione? In realtà, gli esseri umani lo fanno continuamente senza nemmeno pensarci. Pensate per esempio alla vostra agenda settimanale. Per semplicità, immaginiamo che lavoriate otto ore al giorno, cinque giorni alla settimana, e che dividiate la vostra agenda in intervalli di un’ora. Per ciascuno dei cinque giorni quindi ci sono otto possibili intervalli, per un totale di 40 intervalli settimanali. Per comunicare ad altri una settimana della vostra agenda dovete quindi comunicare 40 pezzi di informazione; ma se qualcuno vi telefona per fissare un appuntamento la prossima settimana, gli descrivete la vostra disponibilità elencando 40 informazioni distinte? Certo che no! Con tutta probabilità risponderete qualcosa come “lunedì e martedì sono già pieno, ho impegni dalle 13 alle 15 giovedì e venerdì, ma per il resto sono libero”. Ecco un esempio di compressione dei dati senza perdita! La persona con cui state parlando può ricostruire esattamente quando siete disponibili in tutti i quaranta intervalli della prossima settimana, ma non avete dovuto elencarli tutti esplicitamente.

A questo punto forse penserete che stiamo barando, perché questo tipo di compressione dipende dal fatto che molte parti dell’agenda sono uguali. Specificamente, tutti gli intervalli di lunedì e martedì sono già occupati, quindi potete descriverli rapidamente; il resto della settimana è tutto libero tranne due intervalli che sono altrettanto facili da descrivere. È vero anche che questo è un esempio particolarmente semplicemente, ma anche la compressione informatica funziona nello stesso modo: l’idea di fondo è trovare parti dei dati che sono identiche e usare qualche tipo di trucco per descriverle in modo più efficiente.

Questo è particolarmente facile quando i dati contengono delle ripetizioni. Per esempio, potete probabilmente immaginare un buon modo di comprimere i dati seguenti:

AAAAAAAAAAAAAAAAAAAABCBCBCBCBCBCBCBCBCAAAAAADEFDEFDEF

Se non vi risulta immediatamente ovvio, pensate a come dettereste questi dati al telefono. Invece di dire “A, A, A, A, …, D, E, F”, sono sicuro che escogitereste qualcosa del tipo “21 volte A, poi dieci volte BC poi ancora sei A, poi tre volte DEF”. Oppure, per annotarvi questi stessi dati rapidamente su un foglietto, potreste scrivere qualcosa come “21A,10BC,6A,3DEF”. In questo caso avreste compresso i dati originali, che sono 56 caratteri, a una stringa di 16 caratteri solamente: meno di un terzo, complimenti! Gli

informatici chiamano questo particolare trucco *run-length encoding* (RLE), perché si codifica una “serie” (*run*) di ripetizioni con la “lunghezza” (*length*) di quella serie.

Purtroppo, gli algoritmi RLE sono utili solo per la compressione di tipi di dati molto specifici: sono usati nella pratica, ma per lo più solo insieme con altri tipi di algoritmi di compressione. Per esempio, i fax utilizzano la codifica RLE insieme con un’altra tecnica, la codifica di Huffman, di cui parleremo più avanti. Il problema principale degli algoritmi RLE è che le ripetizioni dei dati debbono essere *adiacenti*, cioè non ci devono essere altri dati fra le parti ripetute. È facile comprimere ABABAB con la RLE (è semplicemente 3AB), ma impossibile comprimere ABXABYAB utilizzando lo stesso trucco.

Probabilmente potete intuire perché il fax può sfruttare questi algoritmi: si tratta per definizione di trasmettere documenti in bianco e nero, che vengono convertiti in un gran numero di punti, ciascun dei quali può essere o nero o bianco. Quando si leggono i puntini in ordine (da sinistra a destra, dall’alto verso il basso) si incontrano lunghe serie di puntini bianchi (lo sfondo) e piccole serie di puntini neri (il testo stampato o manoscritto), il che permette un uso efficace della codifica *run-length*. Ma, come abbiamo già detto, sono solo pochi i tipi di dati che possiedono queste caratteristiche.

Perciò gli informatici hanno inventato una serie di trucchi più raffinati che usano la stessa idea di fondo (trovare ripetizioni e descriverle in modo efficiente), ma funzionano bene anche se le ripetizioni non sono adiacenti. Vedremo qui solo due di questi trucchi: il trucco dell’*uguale-a-quello-di-prima* e il trucco del *simbolo più corto*. Questi due trucchi sono gli unici necessari per produrre file ZIP, e il formato ZIP è il più diffuso per la compressione nei personal computer. Perciò, quando vi saranno chiare le idee di fondo di questi due trucchi saprete come il vostro calcolatore usa la compressione, nella maggior parte dei casi.

Il trucco dell’uguale-a-quello-di-prima

Immaginate di aver ricevuto il terribile incarico di dettare al telefono a qualcun altro questi dati:

VJGDNMQMYLH-KW-VJGDNMQMYLH-ADXSGF-O-

VJGDNMQMYLH-ADXSGF-VJGDNMQMYLH-EW-ADXSGF

I caratteri da comunicare sono 63 (ignoriamo i trattini: sono stati inseriti solo per facilitare la lettura). Possiamo fare qualcosa di meglio che dettare tutti i 63 caratteri, uno alla volta? Il primo passo può essere rendersi conto che ci sono molte ripetizioni, in questi dati. In effetti, la maggior parte dei “blocchi” che sono separati dai trattini vengono ripetuti almeno una volta. Perciò, nel dettare i dati, si può risparmiare molta fatica dicendo qualcosa come “questa parte è uguale a qualcosa che ti ho già dettato prima”. Per essere un po’ più precisi, dovrete dire quanto prima e quanto è lunga la parte da ripetere, magari qualcosa come “torna indietro di 27 caratteri e copia 8 caratteri da quel punto”.

Vediamo come possa funzionare in pratica questa strategia. I primi 12 caratteri non hanno alcuna ripetizione, perciò non avete altra scelta che dettarli uno per uno: “V, J, G, D, N, M, Q, M, Y, L, H, K W”. Ma i dieci caratteri successivi sono uguali ad alcuni dei precedenti, per cui potreste dire semplicemente “indietro 10, copia 10”. I sette successivi sono nuovi, e vanno dettati uno per uno: “A, D, X, S, G, F, O”, ma i 16 caratteri che seguono sono una lunga ripetizione, perciò potete dire “indietro 17, copia 16”. Anche i

dieci successivi sono una ripetizione di qualcosa di precedente, perciò “indietro 16, copia 10”. Poi ci sono due caratteri che non sono ripetizioni e vanno dettati esplicitamente: “E, W”. Infine, gli ultimi sei caratteri sono una ripetizione e potete comunicarli con “indietro 18, copia 6”.

Proviamo a riassumere il nostro algoritmo di compressione. Usiamo le abbreviazioni i per “indietro” e c per “copia”, in modo che un’istruzione come “indietro 18, copia 6” si abbrevia in i18c6. Allora le istruzioni da dettare possono essere riassunte in questo modo:

VJGDNMQMYLH-KW-i12c10-ADXSGF-o-i17c16-i16c10-EW-i18c6

Questa stringa è costituita da 44 caratteri soltanto, mentre quelli originali erano 63 caratteri: ne avete risparmiati 19, cioè quasi un terzo della lunghezza della stringa originale.

C’è un ulteriore aspetto interessante di questo trucco “ugualea-quello-di-prima”. Come usereste lo stesso trucco per comprimere il messaggio FG-FG-FG-FG-FG-FG-FG-FG? (Anche in questo caso i trattini non fanno parte del messaggio, ma sono stati aggiunti per favorire la leggibilità.) In questo messaggio ci sono 8 ripetizioni di FG, perciò potreste dettare singolarmente le prime quattro, poi usare un’istruzione “indietro e copia” in questo modo: FG-FG-FG-FGI8C8. Si risparmiano molti caratteri, ma si può fare anche di meglio. Serve un’istruzione “indietro e copia” che a prima vista può apparire priva di senso: “indietro 2, copia 14”, ovvero I2C14 nella nostra notazione stenografica. Il messaggio compresso diventa FG-I2C14. Ma come è sensatamente possibile copiare 14 caratteri quando ce ne sono solo due da copiare? In realtà, non è un problema, se si copia *il messaggio rigenerato* e non il messaggio compresso. Vediamo passo per passo. Dopo che sono stati dettati i primi due caratteri, abbiamo FG. Poi arriva l’istruzione i2c14, perciò torniamo indietro di due caratteri e cominciamo a copiare. Ci sono solo due caratteri disponibili (FG), perciò li copiamo: quando vengono aggiunti a quel che già c’era, si ottiene FG-FG. Ma ora ci sono altri due caratteri disponibili! Perciò copiamo anche questi e, dopo averli aggiunti al messaggio rigenerato esistente si ottiene FG-FG-FG. Ora sono disponibili altri due caratteri, perciò se ne possono copiare altri due. E si può andare avanti così fino a che non è stato copiato il numero richiesto di caratteri (in questo caso 14). Per vedere se vi è tutto chiaro, provate a scrivere la versione non compressa di questo messaggio compresso: AI1C250.[2]

Il trucco del simbolo più corto

Per capire il trucco di compressione che chiameremo “trucco del simbolo più corto” dobbiamo vedere un po’ più a fondo come vengono memorizzati i messaggi nei calcolatori. Come avrete sicuramente già sentito, i calcolatori non memorizzano realmente lettere come *a*, *b*, *c*. Tutto viene memorizzato in forma di *numero*, e poi viene interpretato come lettera in base a qualche tabella predefinita. (Questa tecnica per la conversione fra lettere e numeri è stata citata anche quando parlavamo di somme di controllo, a pagina xx.) Per esempio, possiamo convenire che “a” sia rappresentata dal numero 27, “b” dal 28, “c” dal 29. Allora la stringa “abc” sarà memorizzata nel calcolatore come “272829”, ma potrà essere facilmente ritradotta in “abc” prima di essere visualizzata sullo schermo o di essere stampata su un foglio di carta.

La Tabella 7.1 presenta un elenco completo di 100 simboli con il relativo codice a due cifre. Incidentalmente, questo particolare insieme di codici a due cifre non è utilizzato in

alcun sistema informatico reale, ma quelli usati normalmente sono molto simili. La differenza principale è che i calcolatori non usano il sistema decimale ma, come forse già sapete, usano un sistema di notazione diverso, il sistema binario. Questi particolari per noi non sono interessanti: il trucco del simbolo più corto funziona sia per il sistema decimale sia per quello binario, perciò faremo finta che i calcolatori usino il sistema decimale, per semplificarci la vita.

Date un'occhiata più attenta alla tabella. Notate che il primo elemento attribuisce un codice numerico, "00" allo spazio fra parole. Poi vengono le lettere maiuscole, *A* ("01") fino a *Z* ("26") e le minuscole da *a* ("27") a *z* ("52"). Nelle ultime colonne sono elencati vari segni di interpunzione e infine alcuni caratteri per le lingue diverse dall'inglese, a partire da *á* ("80") fino a *Ù* ("99").

Come verrebbero usati questi codici a due cifre da un calcolatore per memorizzare la frase "Meet your fiancé there."? Semplice: si traduce ciascun carattere nel suo codice numerico e poi si mettono in fila tutti i numeri:

M e e t y o u r f i a n c è t h e r e .

13 31 31 46 00 51 41 47 44 00 32 35 27 40 29 82 00 46 34 31 44 31 66

È importante tener presente che all'interno del calcolatore non c'è separazione fra coppie di cifre, perciò questo messaggio verrebbe memorizzato in effetti come una stringa continua di 46 cifre: "1 33131460051414744003235274029820046343144 3166". Ovviamente per un essere umano è più difficile da interpretare, ma non presenta alcun problema per un calcolatore, che può facilmente suddividere le cifre a coppie prima di tradurle in caratteri da visualizzare sullo schermo. Il punto fondamentale è che non esiste ambiguità nel processo di separazione dei codici, poiché ciascuno usa esattamente due cifre. In effetti, questo è il motivo per cui *A* viene rappresentato come "01" invece che semplicemente come "1" e *B* è "02" e non "2", e così via fino alla lettera *I* ("09" e non "9"). Se *avessimo* scelto di attribuire il valore "1" ad *A*, "2" a *B*, e così via, allora sì sarebbe stato impossibile interpretare senza ambiguità i messaggi. Per esempio, il messaggio "1123" avrebbe potuto essere suddiviso come "1 1 23" (che si traduce in AAW), oppure come "11 2 3" (KBC) o addirittura come "1 1 2 3" (AABC). Perciò ricordate questa idea importante: la traduzione fra codici numerici e caratteri deve essere priva di ambiguità, anche quando i codici sono memorizzati uno di seguito all'altro senza alcuna separazione. Questo problema tornerà a tormentarci molto presto!

Tabella 7.1 Codici numerici che un calcolatore potrebbe usare per memorizzare simboli

space	00	T	20	n	40	(	60	á	80
A	01	U	21	o	41	)	61	à	81
B	02	V	22	p	42	*	62	é	82
C	03	W	23	q	43	+	63	è	83
D	04	X	24	r	44	,	64	í	84
E	05	Y	25	s	45	-	65	ì	85
F	06	Z	26	t	46	.	66	ó	86
G	07	a	27	u	47	/	67	ò	87
H	08	b	28	v	48	:	68	ú	88
I	09	c	29	w	49	;	69	ù	89
J	10	d	30	x	50	<	70	Á	90
K	11	e	31	y	51	=	71	À	91
L	12	f	32	z	52	>	72	É	92
M	13	g	33	!	53	?	73	È	93
N	14	h	34	"	54	{	74	Í	94
O	15	i	35	#	55	|	75	Ì	95
P	16	j	36	$	56	}	76	Ó	96
Q	17	k	37	%	57	~	77	Ò	97
R	18	l	38	&	58	Ø	78	Ú	98
S	19	m	39	'	59	ø	79	Ù	99

Intanto, torniamo al trucco del simbolo più corto. Come molte delle idee apparentemente tecniche descritte in questo libro, il trucco del simbolo più corto viene applicato continuamente senza che nemmeno ce ne rendiamo conto. L'idea di fondo è che, se si usa qualcosa spesso, val la pena di averne una versione abbreviata. Tutti sanno che "USA" sta per "United States of America", e tutti ci risparmiamo tempo e fatica ogni volta che scriviamo il codice di tre lettere "USA" al posto dell'espressione completa per cui sta, con tutte le sue 24 lettere. Conoscete un'abbreviazione per "Il cielo è di colore blu", che è un'altra espressione di 24 lettere? Ovviamente no! Qual è la differenza fra "United States of America" e "Il cielo è di colore blu"? La differenza fondamentale è che una di queste espressioni viene usata molto più spesso dell'altra, e si può avere un risparmio notevole abbreviando un'espressione che si usa spesso anziché una che viene usata solo raramente.

Proviamo ad applicare questa idea al sistema di codifica della pagina precedente. Sappiamo già di poter risparmiare molto utilizzando abbreviazioni per cose che vengono usate spesso. Le lettere "e" e "t" sono quelle più comuni nella lingua inglese, perciò proviamo a usare un codice più breve per queste lettere. Ora "e" è 31 e "t" è 46, perciò ci vogliono due cifre per rappresentare ciascuna di esse. E se le riducessimo a una sola cifra? Diciamo che ora la "e" è rappresentata dalla singola cifra 8 e "t" da 9. Grande idea! Ricordate come abbiamo codificato prima la frase "Meet your fiancé there.", utilizzando in tutto 46 cifre. Ora possiamo fare così, usando solo 40 cifre:

M e e t y o u r f i a n c è t h e r e .

13 8 8 9 0051 41 47 44 0032 35 27 40 29 82 009 34 8 44 8 66

Purtroppo quest'idea ha un difetto fatale. Ricordate che il calcolatore non memorizza gli spazi fra le singole lettere, perciò la codifica non è del tipo "13 8 8 9 00 51 … 44 8 66" bensì "138890051…44866". Vedete il problema? Concentratevi sulle prime cinque cifre,

che sono 13889. Notate che il codice 13 rappresenta la “M”, 8 rappresenta “e” e 9 “t”, perciò un modo di decodificare le cifre 13889 consiste nel suddividerle come 13-8-8-9, che ci dà la parola “Meet”. Ma 88 rappresenta la lettera accentata “ú”, perciò le cifre 13889 si potrebbero suddividere anche come 13-88-9, che rappresentano “Mút”. In effetti, la situazione è anche peggio, perché 89 rappresenta la lettera accentata “ù” e quindi un’altra possibile suddivisione di 13889 è 13-8-89, che rappresenta “Meù”. Non esiste alcun modo per stabilire quale delle tre possibili interpretazioni sia corretta.

Disastro! La nostra idea astuta di usare codici più brevi per le lettere “e” e “t” ha portato a un sistema di codifica che non funziona per nulla. Per fortuna, si può sistemare la cosa con un altro trucco. Il problema reale è che ogni volta che vediamo una cifra 8 o 9 non c’è modo di stabilire se fa parte di un codice a una cifra (per “e” o “t”) o di uno dei codici a due cifre che iniziano con 8 o 9 (per le varie lettere accentate come “à” o “è”). Per risolvere il problema, dobbiamo fare un sacrificio: alcuni dei nostri codici diventeranno *più lunghi*. I codici ambigui a due cifre che iniziano con 8 o 9 diventeranno codici a tre cifre che *non* iniziano con 8 o 9. La Tabella 7.2 mostra un modo particolare per ottenere questo risultato. Ne vengono influenzati anche alcuni caratteri di interpunzione, ma alla fine abbiamo una situazione molto elegante: tutto ciò che inizia con un 7 è un codice a tre cifre, tutto ciò che inizia con 8 o 9 è un codice a una cifra e tutto ciò che inizia con 0, 1, 2, 3, 4, 5 o 6 è, come prima, un codice a due cifre. A questo punto esiste esattamente un modo solo per suddividere le cifre 13889 (13-8-8-9, che rappresenta “Meet”) e questo vale per qualsiasi altra successione di cifre codificata in modo corretto. Ogni ambiguità è stata eliminata e il nostro messaggio originale può essere codificato in questo modo:

M e e t y o u r f i a n c è t h e r e .

13 8 8 9 0051 41 47 44 0032 35 27 40 29 782 009 34 8 44 8 66

La codifica originale usava 46 cifre, questa ne usa solo 41. Può sembrare un risparmio da poco, ma con un messaggio più lungo il risparmio può essere molto significativo. Per esempio, il testo originale in inglese di questo libro (cioè le sole parole, escluse le immagini) occupa circa 500 kilobyte, mezzo milione di caratteri, ma se viene compresso con i due trucchi appena descritti, le sue dimensioni si riducono a solo 160 kilobyte, ovvero meno di un terzo dell’originale.

Riepilogo: chi ci ha fatto il regalo?

A questo punto dovrebbero essere chiari i concetti importanti alla base della creazione di normali file ZIP compressi. Ecco cosa succede:

Passo 1. Il file originale non compresso viene trasformato utilizzando il trucco dell’“uguale-a-quello-di-prima”, in modo che la maggior parte dei dati ripetuti nel file venga sostituita da istruzioni molto più brevi del tipo “torna indietro e copia i dati da un altro punto”.

Passo 2. Il file trasformato viene esaminato per vedere quali simboli si presentano più spesso. Per esempio, se il file originale è scritto in inglese, il calcolatore probabilmente scoprirà che “e” e “t” sono i due simboli più comuni. Allora il calcolatore costruisce una tabella come la 7.2, in cui ai simboli usati più spesso sono attribuiti codici numerici più brevi, ai simboli usati più raramente sono assegnati codici numerici più lunghi.

Passo 3. Il file viene trasformato nuovamente traducendolo direttamente nei codici numeri definiti al Passo 2.

Tabella 7.2 Codici numerici che utilizzano il trucco del simbolo più corto. Le variazioni rispetto alla Tabella 7.1 sono evidenziate in grassetto. I codici per le due lettere più usate sono stati abbreviati, al costo di un allungamento dei codici per molti simboli poco usati. Il risultato è una lunghezza complessiva minore per la maggior parte dei messaggi

space 00	T 20	n 40	(60	**á 780**
A 01	U 21	o 41	) 61	**à 781**
B 02	V 22	p 42	* 62	**é 782**
C 03	W 23	q 43	+ 63	**è 783**
D 04	X 24	r 44	, 64	**í 784**
E 05	Y 25	s 45	- 65	**ì 785**
F 06	Z 26	**t 9**	. 66	**ó 786**
G 07	a 27	u 47	/ 67	**ò 787**
H 08	b 28	v 48	: 68	**ú 788**
I 09	c 29	w 49	; 69	**ù 789**
J 10	d 30	x 50	< **770**	**Á 790**
K 11	**e 8**	y 51	= **771**	**À 791**
L 12	f 32	z 52	> **772**	**É 792**
M 13	g 33	! 53	? **773**	**È 793**
N 14	h 34	" 54	{ **774**	**Í 794**
O 15	i 35	# 55	| **775**	**Ì 795**
P 16	j 36	$ 56	} **776**	**Ó 796**
Q 17	k 37	% 57	~ **777**	**Ò 797**
R 18	l 38	& 58	**Ø 778**	**Ú 798**
S 19	m 39	' 59	**ø 779**	**Ù 799**

Anche la tabella dei codici numerici, calcolata nel Passo 2, viene memorizzata nel file ZIP, altrimenti sarebbe impossibile decodificare (e quindi decomprimere) il file in seguito. Notate che file non compressi diversi daranno tabelle di codici numerici diverse. In effetti, nel caso di un file ZIP reale, il file originale viene suddiviso in blocchi e ciascun blocco può avere una tabella di codici numerici diversa. Tutto questo può essere fatto in modo efficiente e automatico, ottenendo un'ottima compressione per molti tipi di file.

Compressione con perdita: non proprio gratis, ma è sempre un buon affare

Fin qui abbiamo parlato del tipo di compressione *senza perdita*, perché si può prendere un file compresso e ricostruire esattamente il file di partenza, senza modificare neanche un carattere o un segno di punteggiatura. A volte invece è molto più utile usare una compressione *con perdita*, che consente di prendere un file compresso e ricostruirne uno molto simile all'originale, ma non necessariamente esattamente identico. Per esempio, la compressione con perdita viene usata molto spesso per file che contengono immagini o dati audio: purché l'immagine *appaia* la stessa all'occhio umano, non importa se il file che memorizza quell'immagine sul vostro calcolatore è esattamente identico al file che la memorizza nella fotocamera. E lo stesso vale per i dati audio: se un brano risulta uguale all'orecchio umano, non è realmente importante se il file che memorizza quel brano sul lettore digitale è esattamente lo stesso del file che memorizza quel brano su un compact disc.

A volte, in effetti, la compressione con perdita viene usata in forme più estreme. Abbiamo visto tutti in Internet video e immagini di bassa qualità che appaiono poco definite o in cui la qualità audio è scarsa. È il risultato di una compressione con perdita utilizzata in forma più aggressiva per rendere molto piccole le dimensioni dei file video o di immagine. In questo caso non si vuole che il video appaia uguale all'originale all'occhio umano, ma che sia almeno riconoscibile. Regolando quanta sia la "perdita" della compressione, chi gestisce un sito web può trovare un compromesso fra file di grandi dimensioni e di alta qualità che risultano quasi perfetti all'occhio e all'orecchio, e file di bassa qualità che hanno difetti evidenti ma la cui trasmissione richiede molta meno banda. Forse avrete fatto la stessa cosa con una fotocamera digitale, dove di solito si possono scegliere impostazioni diverse per la qualità di immagini e video. Se scegliete una qualità elevata, il numero di immagini o di video che potete memorizzare nella fotocamera è minore rispetto a quello memorizzabile se si sceglie una qualità minore. Questo perché i file di media ad alta qualità richiedono molto più spazio di quelli a bassa qualità. E tutto si ottiene regolando quanta "perdita" di informazione produce la compressione. Vedremo qui qualche trucco per questa regolazione.

Il trucco dell'esclusione

Un trucco semplice e utile per la compressione con perdita consiste semplicemente nell'escludere parte dei dati. Vediamo come questo trucco dell'esclusione funzioni nel caso di immagini in bianco e nero. Prima dobbiamo capire un po' come vengano memorizzate in un calcolatore le immagini in bianco e nero. Un'immagine è costituita da un gran numero di puntini, chiamati "pixel". Ciascun pixel ha esattamente un colore, che può essere nero, bianco o una qualsiasi sfumatura intermedia di grigio. Ovviamente, in genere non ci accorgiamo di questi pixel perché sono molto piccoli, ma se guardate molto da vicino un monitor o uno schermo televisivo riuscirete probabilmente a vederli.

In una immagine in bianco e nero memorizzata in un calcolatore, ciascun possibile colore dei pixel è rappresentato da un numero. Per esempio, supponiamo che i numeri più elevati rappresentino colori più chiari, fino a un massimo di 100. Allora 100 rappresenta il bianco, 0 il nero, 50 una sfumatura intermedia di grigio, 90 un grigio chiaro e così via. I pixel sono disposti in una griglia rettangolare di righe e colonne, dove ogni pixel rappresenta il colore di una parte molto piccola dell'immagine. Il numero totale di righe e colonne è la "risoluzione" dell'immagine. Per esempio, molti televisori ad alta definizione hanno una risoluzione di 1920 per 1080, il che significa che hanno 1920 righe di pixel e 1080 colonne. Il numero totale dei pixel si calcola moltiplicando 1920 per 1080: il risultato è oltre 2 milioni di pixel. Le fotocamere digitali usano la stessa terminologia. "Megapixel" è solo un modo fantasioso di indicare un milione di pixel. Perciò una fotocamera da 5 megapixel ha un numero di righe e colonne di pixel tale che, quando si moltiplica il numero delle righe per il numero delle colonne, il risultato è superiore a 5 milioni. Quando un'immagine viene memorizzata in un calcolatore, è solo un elenco di numeri, uno per ciascun pixel.

L'immagine di una casa con una torre in alto a sinistra di Figura 7.1 ha una risoluzione molto inferiore a quella di un televisore ad alta definizione. Solo 320 per 240. Ciononostante, il numero dei pixel è ancora piuttosto elevato (320 × 240 = 76.800) e il file che memorizza questa immagine in forma non compressa usa oltre 230 kilobyte di spazio.

Un kilobyte, incidentalmente, equivale a circa 1000 caratteri di testo, all'incirca le dimensioni di un messaggio di posta elettronica di un paragrafo. Molto approssimativamente, quindi, l'immagine a sinistra in alto, quando memorizzata come file, richiede la stessa quantità di spazio su disco di circa 200 brevi messaggi di posta elettronica.

Possiamo comprimere questo file con questa tecnica estremamente semplice: possiamo ignorare, ovvero "escludere" una riga di pixel sì e una no, e una colonna di pixel e una no. Questo trucco è davvero semplicissimo! In questo caso ci dà un'immagine con una risoluzione minore, di 160 per 120, visibile al di sotto della immagine originale nella Figura 7.1. Le dimensioni del file sono solo un quarto dell'originale (circa 57 kilobyte), perché c'è solo un quarto dei pixel dell'originale: abbiamo ridotto sia la larghezza sia l'altezza dell'immagine alla metà. In effetti, le dimensioni dell'immagine sono state ridotte del 50% per due volte (una in orizzontale e una in verticale) e il risultato finale è un'immagine che è solo il 25% dell'originale.

Possiamo applicare questo trucco nuovamente. Prendete la nuova immagine 160 per 120 ed escludete una riga sì e una no, una colonna sì e una no: otterrete una nuova immagine, questa volta di 80 per 60 (il risultato è visibile in basso a sinistra nella Figura 7.1). L'immagine è stata ridotta ancora di un 50% (quindi è il 75% dell'originale) e il file finale è di soli 14 kilobyte. Cioè circa il 6% dell'originale, una compressione davvero notevole.

Ricordate però che stiamo usando una compressione con perdita, così questa volta c'è uno scotto da pagare, anche se non è molto. Osservate che cosa succede quando si prende una delle immagini compresse e la si decomprime riportandola alle dimensioni originali. Poiché alcune delle righe e delle colonne di pixel sono state cancellate, il calcolatore deve congetturare di quale colore fossero i pixel mancanti. Il metodo più semplice sta nell'attribuire a ogni pixel mancante lo stesso colore di uno dei suoi vicini. Si può scegliere qualsiasi vicino, ma per questi esempi è stato scelto il pixel immediatamente al di sopra e a sinistra del pixel mancante.

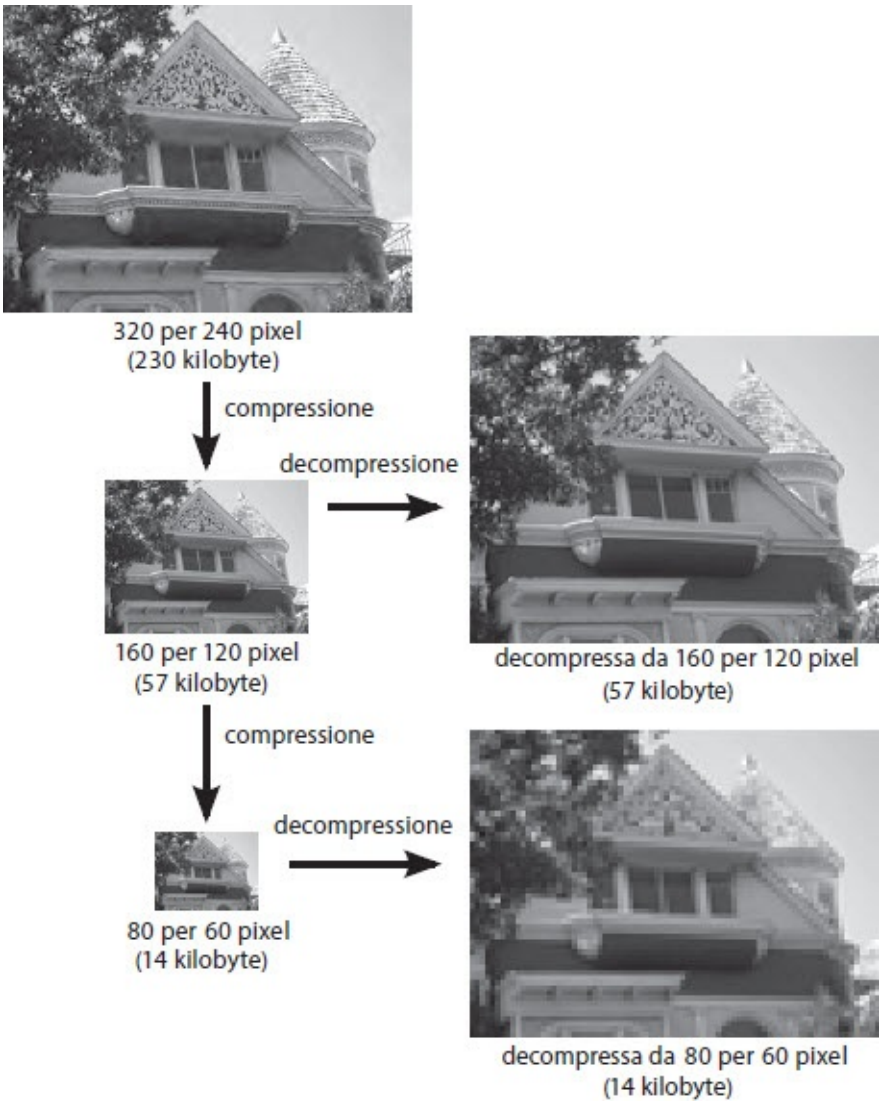


Figura 7.1 Compressione mediante il trucco dell'esclusione. La colonna a sinistra mostra l'immagine originale e due sue versioni più piccole, ridotte. Ciascuna immagine ridotta viene calcolata escludendo metà delle righe e delle colonne della precedente. Nella colonna di destra, si vede il risultato della decompressione dell'immagine ridotta, riportata alle stesse dimensioni dell'originale. La ricostruzione non è perfetta, ci sono alcune differenze ben visibili fra le ricostruzioni e l'originale.

Il risultato di questo schema di decompressione è visibile nella colonna di destra della Figura 7.1. Si può vedere che la maggior parte delle caratteristiche visive è stata conservata, ma che c'è una perdita chiara di qualità nei dettagli, in particolare nelle zone più complesse, come gli alberi, il tetto della torre e il fregio del frontone della casa. Inoltre, in particolare nella versione decompressa dall'immagine di 80 per 60, si possono vedere dei bordi sgradevolmente a zig-zag, per esempio nelle linee diagonali del tetto della casa. Sono i cosiddetti "artefatti da compressione": non semplicemente una perdita di dettagli, ma nuove caratteristiche visibili che sono introdotte da un particolare metodo di compressione con perdita, seguita da una decompressione.

Anche se è utile per capire l'idea fondamentale della compressione con perdita, il trucco dell'esclusione viene usato ben di rado nella forma semplice che abbiamo descritto qui. I calcolatori effettivamente "escludono" informazioni per ottenere una compressione con perdita, ma sono molto più attenti alle informazioni che escludono. Un esempio è il formato di compressione delle immagini JPEG, una tecnica di compressione che è stata studiata attentamente e che dà risultati molto migliori della semplice esclusione di metà delle righe e delle colonne. Date un'occhiata alla Figura 7.2 e paragonate qualità e

dimensioni delle immagini con quelle della Figura 7.1. In alto, si vede un'immagine JPEG di 35 kilobyte, che è praticamente indistinguibile dall'originale. Escludendo ancora più informazioni, ma sempre rimanendo nel formato JPEG, si scende all'immagine di 19 kilobyte al centro, che ha comunque una buona qualità, anche se si vede qualche punto confuso e una perdita di dettagli nel frontone. Anche il JPEG produce artefatti di compressione se la compressione è eccessiva: in basso si può vedere un'immagine JPEG compressa a 12 kilobyte: si notano alcuni punti squadrati e delle macchie sgradevoli nel cielo, proprio vicino alla linea diagonale della casa.

I dettagli della strategia di esclusione del formato JPEG sono troppo tecnici per poterli descrivere completamente qui, ma il senso di fondo della tecnica è molto chiaro. JPEG prima divide l'immagine complessiva in quadratini di 8 pixel per 8. Ciascuno di questi quadratini viene compresso separatamente. Senza compressione, ciascun quadrato sarebbe rappresentato da 8 × 8 = 64 numeri. (Immaginiamo che l'immagine sia in bianco e nero; se fosse a colori ci sarebbero tre colori diversi e quindi il triplo di numeri, ma è un particolare che qui possiamo trascurare.) Se il quadrato è tutto di un colore potrà essere rappresentato da un solo numero e il calcolatore può "escludere" 63 numeri. Se il quadrato è prevalentemente di uno stesso colore, con poche differenze minime (magari una regione di cielo che è quasi tutta della stessa sfumatura di grigio), il calcolare può decidere di rappresentare comunque il quadrato con un solo numero, con una buona compressione e solo una piccola quantità di errore nel momento della decompressione. Nell'immagine in basso della Figura 7.2, si possono vedere effettivamente alcun dei blocchi 8 × 8 del cielo che sono stati compressi in questo modo, e il risultato sono piccoli quadrati di colore uniforme.

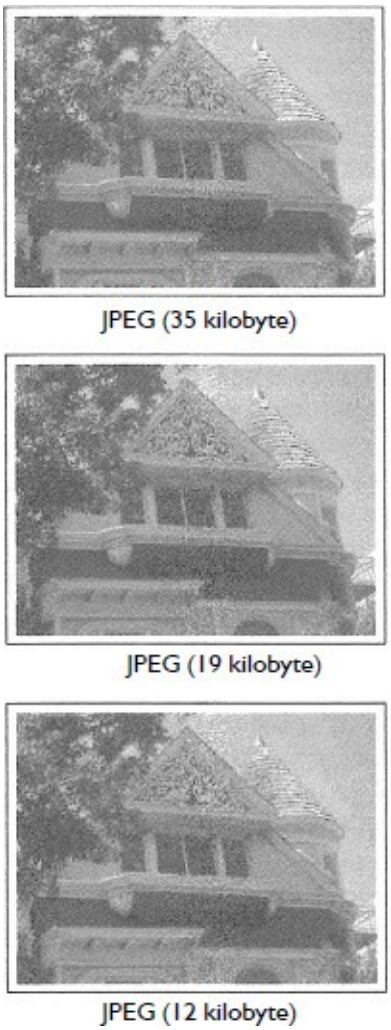


Figura 7.2 Con gli schemi di compressione con perdita di informazione, una compressione più alta dà una qualità minore. Qui si vede la stessa immagine compressa a tre diversi livelli di qualità in formato JPEG: in alto l'immagine di migliore qualità, che richiede maggiore spazio di memoria; in basso quella di qualità più bassa, che "pesa" circa la metà, ma presenta vari artefatti dovuti alla compressione, in particolare nel cielo e lungo i bordi del tetto.

Se il colore del quadrato varia gradualmente da una sfumatura all'altra (poniamo: dal grigio scuro a sinistra al grigio chiaro a destra) allora i 64 numeri possono essere ridotti a due soli: un valore per il grigio scuro e uno per il grigio chiaro. L'algoritmo JPEG non funziona esattamente così, ma usa la stessa idea: se un quadrato 8 × 8 è abbastanza simile a qualche combinazione di schemi noti come un colore costante o un colore che varia con continuità, allora la maggior parte delle informazioni può essere eliminata, memorizzando semplicemente il livello o la quantità di ciascuno schema.

JPEG funziona bene per le immagini, ma che dire dei file audio e della musica? Anche questi vengono compressi con una compressione con perdita, utilizzando la stessa filosofia di fondo: escludere informazioni che hanno poco effetto sul prodotto finale. I formati di compressione per la musica pop, come MP3 e AAC, in genere usano lo stesso approccio di JPEG. L'audio viene diviso in blocchi e ciascun blocco viene compresso separatamente. Come in JPEG, i blocchi che variano in modo prevedibile possono essere descritti con pochi numeri soltanto. I formati di compressione audio però possono sfruttare anche alcune particolarità note dell'orecchio umano: certi tipi di suoni, per esempio, hanno poco (o nessun) effetto sugli ascoltatori umani e possono essere eliminati dall'algoritmo senza ridurre la qualità dell'output.

Le origini degli algoritmi di compressione

Il trucco “uguale-a-quello-di-prima” descritto in questo capitolo, uno dei molti metodi di compressione utilizzati nei file ZIP, è chiamato dagli informatici algoritmo LZ77. È stato inventato da due informatici israeliani, Abraham Lempel e Jacob Ziv, ed è stato pubblicato nel 1977.

Per rintracciare le origini degli algoritmi di compressione, però, dobbiamo risalire più indietro di tre decenni nella storia della scienza. Abbiamo già incontrato Claude Shannon, lo scienziato dei Bell Labs che ha fondato la teoria dell’informazione con il suo saggio del 1948. Shannon è stato uno dei due protagonisti principali della nostra storia dei codici a correzione d’errore (Capitolo 5), ma con il suo saggio del 1948 ha un ruolo importante anche nella comparsa degli algoritmi di compressione.

Non è una coincidenza, perché in realtà codici a correzione di errore e algoritmi di compressione sono due facce della medesima medaglia. Tutto si riconduce all’idea di *ridondanza*, che è comparsa spesso nel Capitolo 5. Se un file possiede ridondanza, è più lungo del necessario. Per ripetere un esempio semplice tratto dal Capitolo 5, il file può usare la parola “cinque” al posto della cifra “5”: in questo modo un errore come “cinvue” può essere facilmente identificato e corretto. I codici a correzione d’errore perciò possono essere visti in linea di principio come un modo per *aumentare* la ridondanza di un messaggio o di un file.

Gli algoritmi di compressione fanno il contrario: *eliminano* ridondanza da un messaggio o da un file. È facile immaginare un algoritmo di compressione che noti l’uso frequente della parola “cinque” in un file e la sostituisca con un simbolo più corto (potrebbe essere addirittura il simbolo “5”), invertendo esattamente il processo di codifica per la correzione degli errori. In pratica, compressione e correzione degli errori non si elidono a vicenda in questo modo: i buoni algoritmi di compressione eliminano i tipi di ridondanza inefficienti, mentre i codici a correzione d’errore aggiungono un tipo diverso di ridondanza, più efficiente. Così succede molto spesso che un messaggio venga compresso e poi vi venga aggiunta una correzione degli errori.

Torniamo a Shannon. Il suo fondamentale saggio del 1948, fra i suoi molti contributi straordinari, includeva la descrizione di una delle prime tecniche di compressione. Un professore del MIT, Robert Fano, aveva scoperto la stessa tecnica all’incirca nello stesso periodo, così oggi si parla di codifica di Shannon-Fano. Si tratta di un modo particolare di realizzare il trucco del simbolo più corto descritto in precedenza. Come vedremo presto, la codifica di Shannon-Fano è stata presto soppiantata da un altro algoritmo, ma il metodo è molto efficace e sopravvive oggi fra i metodi di compressione opzionali per il formato di file ZIP.

Sia Shannon che Fano erano consapevoli che, anche se il loro metodo era pratico ed efficiente, non era il migliore possibile: Shannon aveva dimostrato matematicamente che dovevano esistere tecniche di compressione ancora migliori, ma non aveva ancora scoperto come realizzarle. Nel frattempo Fano aveva cominciato a tenere un corso avanzato di teoria dell’informazione al MIT e propose il problema di raggiungere una compressione ottimale come uno dei possibili temi per la relazione d’esame. Uno dei suoi studenti riuscì a risolvere il problema, producendo un metodo che dà la migliore

compressione possibile per ciascun singolo simbolo. Lo studente si chiamava David Huffman e la sua tecnica, oggi chiamata codifica di Huffman, resta un algoritmo di compressione fondamentale, ampiamente utilizzato nei sistemi di comunicazione e di memorizzazione dei dati.

---

1 Tr. it. di Mario Praz, Garzanti, Milano, 1982, p. 340.

2 La soluzione è: 251 volte la lettera A.

CAPITOLO 8

Database: alla ricerca della coerenza

> “Dati! Dati! Dati!”, esclamò con impazienza. “Non posso fare mattoni se non ho l’argilla.”
>
> Sherlock Holmes, in *The Adventure of the Copper Beeches* di Arthur Conan Doyle[1]

Pensate a questo misterioso rituale. Una persona prende dal cassetto un blocchetto di carta speciale stampata (nota con il nome di *libretto d’assegni*), ci scrive sopra dei numeri e aggiunge una firma con uno svolazzo. Poi strappa il foglio superiore del blocchetto, lo mette in una busta e ci appiccica sopra un altro pezzo di carta (noto con il nome di *francobollo*). Infine, la persona prende la busta, scende le scale e va fino a una grossa cassetta in cui deposita la busta.

Fino alle soglie del Ventunesimo secolo, era il rituale mensile di chiunque pagasse una bolletta: bollette del telefono, bollette della luce elettrica, della carta di credito e così via. Da allora, i sistemi di pagamento online e l’online banking si sono evoluti. La semplicità e la comodità di questi sistemi fanno sembrare al confronto ridicolmente elaborato e inefficiente il precedente metodo basato sulla carta.

Quali tecnologie hanno reso possibile questa trasformazione? La risposta più ovvia è l’arrivo di Internet, senza la quale qualsiasi comunicazione online sarebbe impossibile. Un’altra tecnologia cruciale è la crittografia a chiave pubblica, di cui abbiamo parlato nel Capitolo 4. Senza di essa, informazioni finanziarie delicate non si potrebbero trasmettere in modo sicuro in Internet. Esiste però almeno un’altra tecnologia che è essenziale per le transazioni online: il *database*. Da utenti di computer, la maggior parte di noi ne è del tutto inconsapevole, ma praticamente tutte le nostre transazioni online sono elaborate mediante raffinate tecniche di database, sviluppate dagli informatici a partire dagli anni Settanta del secolo scorso.

I database affrontano due dei problemi principali nell’elaborazione delle transazioni: efficienza e affidabilità. Contribuiscono all’efficienza con algoritmi che permettono a migliaia di clienti di condurre simultaneamente transazioni senza che si generino conflitti o incoerenze; forniscono affidabilità mediante algoritmi che consentono la sopravvivenza e l’integrità dei dati nonostante i guasti di componenti come le unità a disco, che solitamente porterebbero a gravi perdite di dati. L’online banking è un esempio canonico di applicazione che richiede estrema efficienza (per servire molti clienti contemporaneamente senza produrre errori o incoerenze) e affidabilità sostanzialmente perfetta. Per concretizzare le nostre analisi, torneremo spesso perciò all’esempio delle operazioni bancarie online.

In questo capitolo, parleremo di tre idee fondamentali, e molto belle, che stanno alla base dei database: il *write-ahead log*, il completamento in due fasi (*two-phase commit*) e i database relazionali. Queste idee hanno portato al predominio assoluto della tecnologia dei database per la conservazione di alcuni tipi di informazioni importanti. Come al solito, cercheremo di concentrarci sugli aspetti nodali di queste idee, identificando un singolo trucco che le fa funzionare. Il *write-ahead logging* si riduce al "trucco della lista delle cose da fare", che affronteremo per primo. Poi passeremo al protocollo del completamento in due fasi, descritto qui attraverso il semplice ma potente "trucco del prima-prepara-poi-finisci". Infine daremmo uno sguardo al mondo dei database relazionali esaminando il "trucco della tabella virtuale".

Prima però di passare a questi trucchi, vediamo di chiarire il mistero di che cosa sia effettivamente un database. In effetti, anche nella letteratura tecnica dell'informatica, la parola "database" può significare molte cose diverse, perciò è impossibile dare un'unica definizione corretta. Quasi tutti gli esperti però sarebbero d'accordo che la proprietà chiave dei database, quella che li distingue da altri modi di memorizzare le informazioni, è che le informazioni in un database hanno una struttura predefinita.

Per capire che cosa voglia dire "struttura" in questo caso, vediamo prima il contrario, un esempio di informazione *non strutturata*:

> Rosina ha 35 anni ed è amica di Matt, che ne ha 26. Jingyi ne ha 37 e Sudeep 31. Matt, Jingyi e Sudeep sono tutti amici fra loro.

Questo è proprio il tipo di informazioni che un sito di social networking come Facebook o MySpace dovrebbe memorizzare a proposito dei suoi membri. Ovviamente, però, le informazioni non saranno memorizzate in questo modo non strutturato. Ecco le stesse informazioni in una forma strutturata:

nome	**età**	**amici**
Rosina	35	Matt
Jingyi	37	Matt, Sudeep
Matt	26	Rosina, Jingyi, Sudeep
Sudeep	31	Jingyi, Matt

Gli informatici definiscono questo tipo di struttura una *tabella*. Ogni riga della tabella contiene informazioni su una sola cosa (in questo caso, una persona). Ciascuna colonna della tabella contiene solo un particolare tipo di informazioni, per esempio l'età o il nome. Un database spesso è costituito da molte tabelle, ma i nostri esempi iniziali useranno, per semplicità, una sola tabella.

Ovviamente è molto più efficiente, per gli esseri umani come per i computer, manipolare dati nella forma strutturata di una tabella, anziché un testo non strutturato come nell'esempio precedente. Ma i database servono a molto più che a rendere più facile l'uso.

Il nostro viaggio nel mondo dei database inizia con un nuovo concetto: quello di *coerenza* o *consistenza*. Come vedremo presto, chi si occupa di database è ossessionato dalla coerenza, e per buoni motivi. In parole povere, "coerenza" vuol dire che le informazioni nel database non debbono contraddirsi. Se esiste una contraddizione nel

database, si verifica il peggior incubo dell'amministratore di database: incoerenza. Ma come potrebbe nascere un'incoerenza? Beh, supponiamo di modificare leggermente le prime due righe nella tabella precedente:

nome	età	amici
Rosina	35	Matt, Jingyi
Jingyi	37	Matt, Sudeep

Riuscite a vedere il problema? Secondo la prima riga, Rosina è amica di Jingyi; ma, per la seconda riga, Jingyi non è amico di Rosina. Questo viola la caratteristica fondamentale dell'amicizia: due persone sono simultaneamente amiche l'una dell'altra. Certo, questo è un esempio abbastanza innocuo di incoerenza. Per immaginare un caso più serio, supponiamo che il concetto di "amicizia" sia sostituito da quello di "matrimonio". Allora ci ritroveremmo con A sposato a B, ma B sposato a C – una condizione illegale in molti paesi.

In effetti, questo tipo di incoerenza è facile da evitare quando si inseriscono nuovi dati nel database. I computer sono bravissimi nel seguire regole, perciò è facile impostare un database che segua la regola "Se A è sposato a B, allora B deve essere sposato ad A". Se qualcuno cerca di inserire una nuova riga che viola questa regola, riceve un messaggio di errore e la riga non viene scritta. Garantire la coerenza sulla base di regole semplici dunque non richiede trucchi di particolare astuzia.

Ma ci sono altri tipi di incoerenza che richiedono soluzioni molto più ingegnose.

Le transazioni e il trucco della lista delle cose da fare

Le transazioni sono probabilmente l'idea più importante nel mondo dei database, ma per capire che cosa sono e perché sono necessarie, dobbiamo accettare due fatti relativi ai computer. Il primo vi è probabilmente ben noto: i programmi vanno in tilt e, quando succede, il computer dimentica tutto quello che stava facendo. Solo le informazioni che sono state esplicitamente salvate nel sistema di file del computer sono conservate. Il secondo fatto è molto meno noto, ma estremamente importante: i dispositivi di memoria, come le unità disco e le penne di memoria flash, possono scrivere istantaneamente solo una piccola quantità di dati, normalmente intorno ai 500 caratteri. (Se vi interessa il gergo tecnico, sto parlando qui delle *dimensioni di un settore* di un disco, che normalmente sono di 512 byte. Nel caso delle memorie flash, la grandezza pertinente è la *dimensione di pagina*, che può essere di centinaia o anche di migliaia di byte.) Da utenti di computer, non notiamo mai questa piccola limitazione, perché le moderne unità a disco possono eseguire centinaia di migliaia di queste operazioni di scrittura di 500 caratteri ogni secondo, ma resta il fatto che i contenuti del disco vengono modificati solo a poche centinaia di caratteri alla volta.

Che cosa ha a che fare tutto questo con i database? Ha una conseguenza estremamente importante: il computer normalmente può aggiornare una sola riga di un database alla volta. Purtroppo il nostro esempio, minuscolo e semplicissimo, non è molto adatto, perché tutta la tabella contiene meno di 200 caratteri, perciò in questo caso particolare il computer potrebbe aggiornare due righe alla volta. Ma, in generale, per un database di qualsiasi dimensione realistica, per modificare due righe distinte sono necessarie due distinte

operazioni su disco.

Con questi fatti di fondo ben chiari, possiamo andare al nocciolo. Molte variazioni apparentemente semplici a un database richiedono in genere la modifica di due o più righe. E, come ora sappiamo, modificare due righe distinte non è possibile in un'unica operazione sul disco, perciò l'aggiornamento del database consisterà in qualche sequenza di due o più operazioni sul disco. Il computer però può andare in tilt in qualsiasi momento, e che cosa succede se va in tilt proprio *fra* due di queste operazioni? Il computer può essere riavviato, ma avrò dimenticato quali operazioni stava per eseguire, perciò è possibile che alcune delle modifiche necessarie non siano mai state fatte. In altre parole, il database potrebbe essere rimasto in uno stato incoerente!

A questo punto, tutto il problema dell'incoerenza dopo un crash può sembrare accademico, ma vediamo due esempi che ne chiariscono invece l'importanza. Cominciamo con un database ancora più semplice del precedente:

nome	**amici**
Rosina	nessuno
Jingyi	nessuno
Matt	nessuno

Questo banalissimo database elenca tre persone solitarie. Ora supponiamo che Rosina e Jingyi stringano amicizia: vogliamo aggiornare il database in modo che rifletta questo felice stato di cose. Come potete vedere, questo aggiornamento comporterà cambiamenti sia nella prima sia nella seconda riga della tabella e, come abbiamo detto, questo in generale richiederà due distinte operazioni sul disco. Supponiamo che prima venga aggiornata la riga 1. Subito dopo questo aggiornamento, e prima che il computer abbia avuto una possibilità di eseguire la seconda operazione che aggiornerà la riga 2, il database apparirà così:

nome	**amici**
Rosina	Jingyi
Jingyi	nessuno
Matt	nessuno

Fin qui tutto bene. Ora il programma deve aggiornare la riga 2, e così farà. Ma aspettate: che cosa succede se il computer va in tilt prima che il programma possa farlo? Quando il computer sarà riavviato, non avrà alcuna idea che la riga 2 debba ancora essere aggiornata. Il database rimarrà esattamente come l'abbiamo lasciato prima: Rosina è amica di Jingyi, ma Jingyi *non è* amico di Rosina. Questa è l'incoerenza che temevamo.

Ho già detto che chi si occupa di database è ossessionato dalla coerenza, ma a questo punto può darsi che non sembri una gran cosa. In fin dei conti, è importante se Jingyi risulta avere un amico da una parte e dall'altra no? Potremmo immaginare uno strumento automatico che ogni tanto esplori il database alla ricerca di discrepanze come questa e le sistemi. In effetti, strumenti del genere esistono e posso essere utilizzati nei database in cui la coerenza è di importanza secondaria. Forse ne avrete incontrato qualche esempio anche voi, perché alcuni sistemi operativi, quando vengono riavviati dopo un crash, controllano

tutto il sistema di file alla ricerca di incoerenze.

Ma esistono situazioni in cui un'incoerenza è davvero dannosa e non può essere corretta da uno strumento automatico. Un esempio classico è il trasferimento di denaro fra conti bancari. Ecco un altro semplice database:

nome	**tipo di conto**	**saldo**
Zadie	corrente	€ 800
Jingyi	risparmio	€ 300
Jingyi	corrente	€ 150

Supponiamo che Zadie abbia richiesto il trasferimento di 200 euro dal suo conto corrente al conto risparmio. Come nell'esempio precedente, dovranno essere aggiornate due righe, con una successione di due distinte operazioni sul disco. Prima il saldo del conto corrente di Zadie verrà ridotto a 600 euro, poi il saldo del suo conto risparmio verrà aumentato a 500 euro. Se siamo abbastanza sfortunati da avere un crash proprio fra queste due operazioni, il database si presenterà in questo modo:

nome	**tipo di conto**	**saldo**
Zadie	corrente	€ 600
Jingyi	risparmio	€ 300
Jingyi	corrente	€ 150

Un disastro completo per Zadie: prima del crash aveva nei suoi due conti un totale di 1100 euro, ma ora ne ha solo 900. Non ha mai prelevato denaro, ma in qualche modo 200 euro sono svaniti nel nulla. E non c'è modo di scoprirlo, perché il database è perfettamente coerente in sé, dopo il crash. Abbiamo incontrato un tipo più insidioso di incoerenza: il nuovo database è incoerente con il suo stato *prima* del crash.

Vale la pena analizzare dettagliatamente questo punto importante. Nel nostro primo esempio di incoerenza, ci siamo trovati con un database che era palesemente incoerente: A amico di B, ma B non amico di A. Questo tipo di incoerenza può essere rilevato semplicemente esaminando il database (anche se il procedimento può essere molto lungo, se il database contiene milioni, o addirittura miliardi di record). Nel nostro secondo esempio di incoerenza, il database rimane in uno stato di perfetta plausibilità, se lo si considera come un'istantanea scattata in un istante particolare. Non c'è regola che dica quali devono essere i saldi dei conti, né regola che stabilisca le relazioni fra quei saldi. Ciononostante, si può osservare un comportamento incoerente se si esamina lo stato del database nel tempo. Qui sono pertinenti tre fatti: (1) prima di iniziare il trasferimento, Zadie aveva 1100 euro; (ii) dopo il crash ne ha 900; (iii) nel tempo intercorso, non ha prelevato alcuna cifra. Presi insieme questi tre fatti mostrano un'incoerenza, che però non può essere rilevata esaminando il database solo in un particolare istante.

Per evitare entrambi i tipi di incoerenza, i ricercatori che si occupano di database hanno formulato il concetto di "transazione", cioè di un insieme di cambiamenti che debbono avvenire tutti perché il database rimanga coerente. Se vengono eseguiti solo alcuni dei cambiamenti previsti dalla transazione, ma non tutti, il database può rimanere in una situazione di incoerenza. L'idea è semplice ma estremamente potente. Un

programmatore di database può impartire un comando come “inizio transazione”, poi effettuare una serie di cambiamenti interdipendenti al database, e concludere con un “fine transazione”. Il database garantirà che i cambiamenti voluti vengano eseguiti tutti, anche se il computer che ospita il database va in tilt e si riavvia nel bel mezzo di una transazione.

Per essere assolutamente corretti, dovremo essere consapevoli di un’altra possibilità ancora: è possibile che dopo un crash e un riavvio il database ritorni esattamente nello stato in cui era prima dell’inizio della transazione. Ma se succede, il programma riceverà una notifica che la transazione è fallita e deve essere effettuata nuovamente, perciò non è stato fatto alcun danno. Parleremo più approfonditamente di questa possibilità più avanti, nel paragrafo sul *roll-back* delle transazioni, ma per il momento il punto cruciale è che il database rimane coerente indipendentemente dal fatto che una transazione venga completata o annullata.

Dalla descrizione può sembrare che siamo eccessivamente ossessionati dalla possibilità di un tilt del sistema che, in fin dei conti, è molto raro nei sistemi operativi moderni che eseguono applicazioni moderne. Ci sono due risposte. Innanzitutto l’idea di “crash” va presa in senso molto generale: comprende qualsiasi incidente che possa far sì che il computer smetta di funzionare e quindi perda dei dati. I casi possibili sono, fra gli altri, una caduta di tensione, un guasto nel disco o altri malfunzionamenti dell’hardware ed errori nel sistema operativo o nei programmi applicativi. In secondo luogo, anche se questi crash in senso generalizzato sono abbastanza rari, ci sono database che non possono correre rischi: banche, compagnie di assicurazioni e altre organizzazioni i cui dati rappresentano in realtà denaro non possono permettersi incoerenze nelle loro registrazioni, in nessuna circostanza.

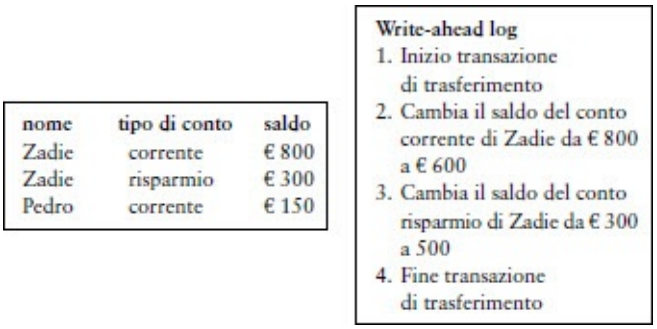
La semplicità della soluzione descritta prima (inizia una transazione, effettua tutte le operazioni necessarie, poi termina la transazione) può sembrare troppo bella per essere vera, ma in effetti può essere ottenuta con il trucco relativamente semplice della “lista delle cose da fare”.

Il trucco della lista delle cose da fare

Non tutti sono così fortunati da avere il dono di una buona organizzazione. Ma che siamo o meno bene organizzati, abbiamo visto tutti una delle grandi armi brandite dalle persone bene organizzate: la lista delle cose da fare. Forse non siete appassionati delle liste, ma è difficile sostenere che non siano utili. Se dovete fare dieci commissioni in una giornata, scriverle, preferibilmente in un ordine efficiente, è un ottimo inizio. Una lista di cose da fare è particolarmente utile se nel corso della giornata subentrano distrazioni (possiamo dire, dei “crash”?). Se per qualsiasi motivo dimenticate quali siano le commissioni ancora da sbrigare, un rapido sguardo alla lista ve le farà tornare in mente.

Nei database le transazioni vengono realizzate con un particolare tipo di lista di cose da fare: per questo abbiamo dato questo nome al nostro trucco, anche se gli informatici usano il termine *write-ahead logging* per indicare la stessa cosa. L’idea di fondo è conservare un registro (*log*) delle azioni che il database deve compiere. Il registro è conservato su un disco o su qualche altro dispositivo di memoria permanente, in modo che le informazioni del registro sopravvivano a crash e riavvii. Prima che qualsiasi azione di una data transazione venga effettuata, sono tutte registrate e così salvate su disco. Se la

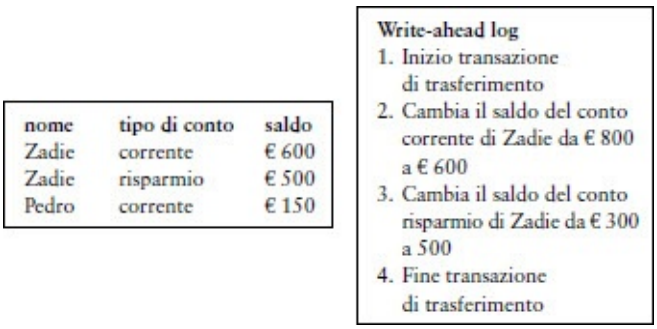
transazione va a buon fine, si può recuperare spazio cancellando dal registro la lista delle cose da fare della transazione. La transazione di trasferimento di denaro di Zadie vista prima allora si svolgerà in due passi fondamentali. Innanzitutto, la tabella del database rimane inalterata e si scrive la "lista delle cose da fare" della transazione nel registro:



nome	tipo di conto	saldo
Zadie	corrente	€ 800
Zadie	risparmio	€ 300
Pedro	corrente	€ 150

Write-ahead log
1. Inizio transazione di trasferimento
2. Cambia il saldo del conto corrente di Zadie da € 800 a € 600
3. Cambia il saldo del conto risparmio di Zadie da € 300 a 500
4. Fine transazione di trasferimento

Dopo aver verificato che le voci del registro siano state salvate in qualche dispositivo permanente come un disco, apportiamo i cambiamenti programmati alla tabella stessa:

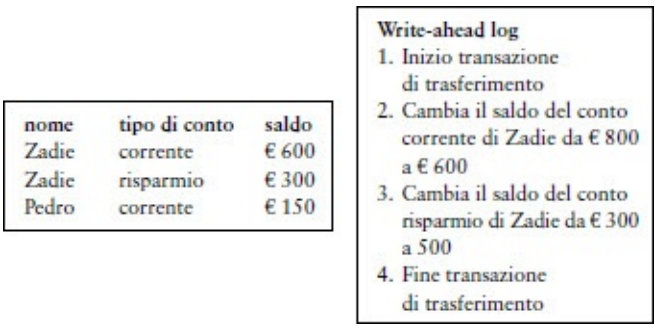


nome	tipo di conto	saldo
Zadie	corrente	€ 600
Zadie	risparmio	€ 500
Pedro	corrente	€ 150

Write-ahead log
1. Inizio transazione di trasferimento
2. Cambia il saldo del conto corrente di Zadie da € 800 a € 600
3. Cambia il saldo del conto risparmio di Zadie da € 300 a 500
4. Fine transazione di trasferimento

Se i cambiamenti sono stati salvati su disco, le voci del registro possono essere cancellate.

Ma questo è il caso facile. Che cosa succede se il computer si blocca improvvisamente nel bel mezzo della transazione? Come prima, ipotizziamo che il crash si verifichi dopo l'addebito sul con-to corrente di Zadie, ma prima dell'accredito sul suo conto risparmio. Il computer si riavvia e il database riparte, trovando sul disco queste informazioni:



nome	tipo di conto	saldo
Zadie	corrente	€ 600
Zadie	risparmio	€ 300
Pedro	corrente	€ 150

Write-ahead log
1. Inizio transazione di trasferimento
2. Cambia il saldo del conto corrente di Zadie da € 800 a € 600
3. Cambia il saldo del conto risparmio di Zadie da € 300 a 500
4. Fine transazione di trasferimento

Ora il computer può stabilire che si trovava nel mezzo di una transazione al momento del crash, perché il registro contiene alcune informazioni. Ma ci sono quattro azioni nella lista del registro: come si fa a stabilire quali siano state già eseguite sul database e quali siano ancora da fare? La risposta è piacevolmente semplice: non è importante! Il motivo è che ciascuna voce nel registro è costruita in modo tale da avere sempre lo stesso effetto, che sia eseguita una, due o un numero qualsiasi di volte.

Il termine tecnico è *idempotenza*: un informatico direbbe che ogni azione nel registro deve essere idempotente. Per esempio, vediamo la voce numero 2, "Cambia il saldo del conto corrente di Zadie da € 800 a € 600". Non importa quante volte il saldo di Zadie viene impostato a € 600, il risultato sarà sempre lo stesso. Perciò se il database sta effettuando un ripristino dopo un crash e vede questa voce nel registro può tranquillamente eseguire questa azione senza preoccuparsi se sia già stata eseguita prima del blocco.

Quindi, nel ripristino dopo un crash, un database può semplicemente rieseguire le azioni di qualsiasi transazione completa, ed è facile risolvere anche le transazioni incomplete. Qualsiasi insieme di azioni trascritte nel registro che non si concluda con un "fine transazione" semplicemente viene annullato in ordine inverso, lasciando il database come se la transazione non fosse mai iniziata. Torneremo a questa idea di *roll-back* di una transazione parlando della replica dei database.

Atomicità, in piccolo e in grande

Le transazioni si possono vedere anche in un altro modo: dal punto di vista dell'utente del database, ogni transazione è *atomica*. Anche se i fisici sanno da decenni come scindere gli atomi, il significato originale di "atomico" deriva dal greco, ed è "indivisibile". Quando gli informatici dicono di qualcosa che è "atomico", usano la parola nel senso originale. Una transazione atomica quindi non può essere divisa in operazioni più piccole: o l'intera transazione va a buon fine (viene completata con successo) o il database rimane nelle condizioni originali, come se la transazione non fosse mai nemmeno iniziata.

Perciò il trucco della lista delle cose da fare ci dà transazioni atomiche, che a loro volta garantiscono la coerenza. Questo è un ingrediente chiave nel nostro esempio canonico: un database efficiente e del tutto affidabile per l'online banking. Non siamo ancora alla fine, però. La coerenza da sola non significa efficienza o affidabilità adeguate. Quando è combinata con le tecniche di "chiusura" che vedremo fra breve, il trucco della lista delle cose da fare mantiene la coerenza anche quando migliaia di clienti accedono simultaneamente al database. Questo *produce* un'efficienza tremenda, perché è possibile servire contemporaneamente molti clienti. Il trucco della lista delle cose da fare offre anche una buona dose di affidabilità, perché impedisce le incoerenze; specificamente, preclude la *corruzione* dei dati, ma non elimina il rischio di *perdita* dei dati. Il successivo trucco, quello del "prima-prepara-poi-finisci" ci farà fare progressi significativi verso l'obiettivo di impedire qualsiasi perdita di dati.

Il trucco "prima-prepara-poi-finisci"
per i database replicati

Il nostro viaggio fra le tecniche ingegnose dei database continua con un algoritmo che chiameremo "trucco del prima-prepara-poifinisci". Dobbiamo però prima appurare due

altri fatti relativi ai database: in primo luogo, vengono spesso *replicati*, il che significa che più copie dello stesso database vengono conservate in luoghi diversi; in secondo luogo, le transazioni a volte debbono essere annullate, nel qual caso si parla di *roll back* della transazione, o di *abortire* la transazione. Vediamo brevemente questi due concetti prima di passare al nostro trucco.

Replica di database

Il trucco della lista delle cose da fare consente ai database il ripristino dopo certi tipi di crash, mediante il completamento o l'annullamento delle transazioni in corso al momento del blocco. Ma questo dà per scontato che tutti i dati che erano stati salvati prima del crash siano ancora lì. Ma che cosa succede se l'unità disco si è definitivamente guastata o se tutti i dati sono andati persi? Questo è uno dei molti modi in cui si può verificare una perdita permanente di dati nei computer. Altre cause possono essere errori nel software (nel programma di database stesso o nel sistema operativo) e guasti dell'hardware. Tutti questi problemi possono fare sì che il computer sovrascriva dati che si pensava fossero conservati in modo sicuro sul disco, cancellandoli e sostituendoli con spazzatura. Chiaramente, il trucco della lista di cose da fare in questo caso non ci può proprio aiutare.

La perdita di dati, però, è una cosa non tollerabile in certe circostanze. Se la vostra banca perde le informazioni sul vostro conto, ne sarete estremamente irritati e la banca andrà incontro a gravi misure legali e finanziarie. Lo stesso vale per una società di intermediazione finanziaria che esegua un vostro ordine ma poi perda i dettagli dell'operazione. In effetti, qualsiasi azienda che effettui molte vendite online (eBay e Amazon sono gli esempi che vengono subito in mente) semplicemente non può correre il rischio della perdita o della corruzione di informazioni sui propri clienti. In un centro con migliaia di computer, ogni giorno si guastano molti componenti (in particolare unità disco) e ogni giorno i dati che si trovano su quei componenti *vanno* persi. Come fa la vostra banca a conservare al sicuro i vostri dati, nonostante tutti questi disastri?

La soluzione ovvia, e ampiamente utilizzata, è quella di conservare due o più copie del database. Ogni copia è una *replica* del database, e l'insieme di tutte le copie è un *database replicato*. Spesso le repliche sono separate geograficamente (magari in centri che distano centinaia di chilometri) in modo che, se anche una di esse viene distrutta da una calamità naturale, ce ne sia sempre un'altra a disposizione.

Un dirigente di una società di informatica una volta ha descritto l'esperienza dei suoi clienti dopo gli attentati terroristici dell'11 settembre 2001 alle torri gemelle del World Trade Center di New York. La sua società aveva cinque grandi clienti nelle torri e tutti avevano database replicati. Quattro sono riusciti a continuare la loro attività sostanzialmente senza interruzioni, grazie alle repliche rimaste; il quinto cliente purtroppo aveva una replica in ciascuna torre e ha perso entrambe. Questo cliente ha potuto riprendere la sua attività solo dopo aver ricostruito il suo database da copie di backup che erano state archiviate altrove.

Un database replicato è una cosa molto diversa dal backup di dati. Un backup è una istantanea di certi dati *in un particolare istante*: nel caso di backup manuali, l'istantanea viene "scattata" nel momento in cui mandate in esecuzione il programma di backup, mentre i backup automatizzati spesso scattano un'istantanea di un sistema a un'ora

particolare settimanalmente o quotidianamente, per esempio tutti i giorni alle 2 di notte. In altre parole, un backup è un duplicato completo di certi file, o di un database o di qualsiasi altra cosa per cui vi serva una copia di riserva.

Ma un backup, per definizione, non è necessariamente aggiornato: se sono avvenuti dei cambiamenti dopo che è stato effettuato, quei cambiamenti non sono salvati in alcun altro posto. Un database replicato invece mantiene tutte le proprie copie sincronizzate costantemente. Ogni volta che viene effettuata una minima modifica in qualsiasi record del database, tutte le repliche debbono apportare immediatamente quella stessa modifica.

Chiaramente, la replica è un modo eccellente per proteggersi dalla perdita di dati, ma ha anche i suoi pericoli, perché introduce un altro tipo di possibile incoerenza. Che cosa si fa se una replica si ritrova per qualsiasi motivo con dati diversi da quelli di un'altra replica? Queste repliche sono incoerenti l'una con l'altra e può rivelarsi difficile o impossibile determinare quale replica abbia la versione corretta dei dati. Torneremo su questo problema dove aver visto come si effettua il *roll-back* delle transazioni.

Roll-back di transazioni

A rischio di essere un po' ripetitivi, proviamo a ricordare esattamente che cosa è una transazione: è un insieme di cambiamenti a un database che debbono essere *tutti* effettuati perché il database resti coerente. In precedenza, ci siamo preoccupati di fare in modo che una transazione si completasse anche se il database andava in tilt proprio nel bel mezzo della transazione stessa.

Qualche volta però succede che sia impossibile completare una transazione, per qualche motivo, Per esempio, magari la transazione comporta l'aggiunta di una grande quantità di dati al database e il computer esaurisce lo spazio su disco a metà della transazione. È una situazione molto rara, ma comunque importante.

Un motivo molto più comune per il fallimento di una transazione si riferisce a un altro concetto, quello del *locking* o *blocco*. In un database molto attivo, di solito ci sono molte transazioni in esecuzione allo stesso tempo. (Pensate a che cosa succederebbe se la vostra banca consentisse a un solo cliente alla volta di trasferire denaro: le prestazioni del sistema di banking online sarebbero davvero terribili.) Ma spesso è importante che qualche parte del database resti congelata durante una transazione. Per esempio, se la transazione A aggiorna una voce che registra il fatto che ora Rosina è amica di Jingyi, sarebbe disastroso se una transazione B, eseguita simultaneamente, cancellasse del tutto Jingyi dal database. Pertanto, la transazione A "bloccherà" la parte del database che contiene le informazioni di Jingyi. Questo significa che i dati sono congelati, e nessun'altra transazione può modificarli. Nella maggior parte dei database le transazioni possono bloccare singole righe o colonne, oppure intere tabelle. Ovviamente, solo una transazione può bloccare una particolare parte del database in ogni dato istante. Una volta che la transazione viene completata correttamente, "sblocca" tutti i dati che aveva bloccato e a quel punto altre transazioni sono liberi di modificare i dati in precedenza congelati.

Può sembrare una soluzione eccellente, ma può portare a una bruttissima situazione, che gli informatici chiamano *deadlock* o *stallo*, rappresentata nella Figura 8.1. Supponiamo che due lunghe transazioni, A e B, vengano eseguite simultaneamente. Inizialmente, come nel riquadro in alto della figura, nessuna delle righe del database è

bloccata. Poi, come si vede nel riquadro centrale, A blocca la riga che contiene le informazioni di Marie e B blocca quella che contiene le informazioni di Pedro. Dopo un po', A scopre di dover bloccare la riga di Pedro e B scopre di dover bloccare la riga di Marie, situazione rappresentata nel riquadro inferiore della figura. Notate che A ora deve bloccare la riga di Pedro, ma solo una transazione alla volta può bloccare una qualsiasi riga, e B ha già bloccato la riga di Pedro; perciò A dovrà aspettare che B finisca. Ma B non può finire se non blocca la riga di Marie, che in questo momento è bloccata da A. Quindi B deve aspettare che finisca A. A e B sono in una situazione di *stallo*, perché ciascuna delle due deve aspettare che l'altra proceda. Rimarranno in questa condizione per sempre e queste transazioni non verranno mai completate.

Gli informatici hanno studiato a fondo le situazioni di stallo e molti database periodicamente eseguono una procedura speciale per rilevarle. Quando si trova una situazione di stallo, semplicemente una delle due transazioni viene cancellata, in modo che l'altra possa procedere. Notate però che, come nel caso in cui si esaurisca lo spazio su disco nel mezzo di una transazione, questo richiede la capacità di effettuare il *roll-back* di una transazione già parzialmente completata. Ora dunque conosciamo almeno due motivi per cui può presentarsi la necessità di un *roll-back* di una transazione. Ve ne sono molti altri, ma non è necessario procedere oltre. Il punto fondamentale è che spesso le transazioni non vanno a buon fine per ragioni difficilmente prevedibili.

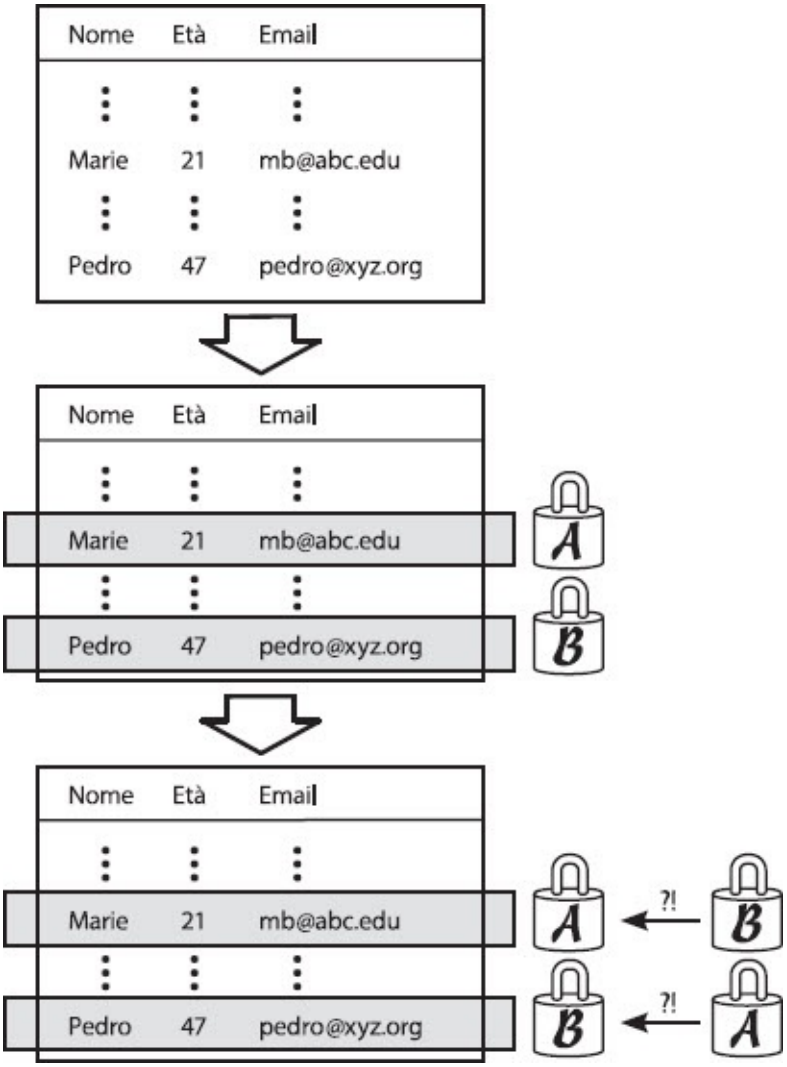


Figura 8.1 Stallo (deadlock): quando due transazioni, A e B, cercano di bloccare le stesse righe, ma in ordine inverso, si verifica una situazione di stallo e nessuna delle due può continuare.

Si può effettuare il *roll-back* con una piccola modifica del trucco della lista delle cose da fare: il *write-ahead log* deve contenere abbastanza informazioni ulteriori per consentire la *revoca* di ciascuna operazione, se necessario. (In precedenza invece avevamo messo in evidenza che ciascuna voce nel registro deve contenere abbastanza informazioni da *ripetere* l'operazione dopo un crash.) In pratica, non è una cosa difficile. In effetti, negli esempi semplici che abbiamo visto, le informazioni di revoca e quelle di ripetizione sono identiche. Una indicazione come "Modifica il saldo del conto corrente di Zadie da € 800 a € 600" può facilmente essere "revocata" semplicemente cambiando il saldo del conto di Zadie da € 600 a € 800. In breve: se una transazione deve essere annullata, il programma di database può semplicemente ripercorrere a ritroso il *write-ahead log* (cioè la lista delle cose da fare) invertendo ciascuna operazione.

Il trucco del "prima-prepara-poi-finisci"

Ora pensate al problema del *roll-back* di una transazione in un database *replicato*. Il problema grave qui è che una delle repliche può incontrare un problema che richiede il *roll-back*, mentre le altre no. Per esempio, è facile immaginare che una replica esaurisca lo spazio su disco mentre le altre hanno ancora spazio disponibile.

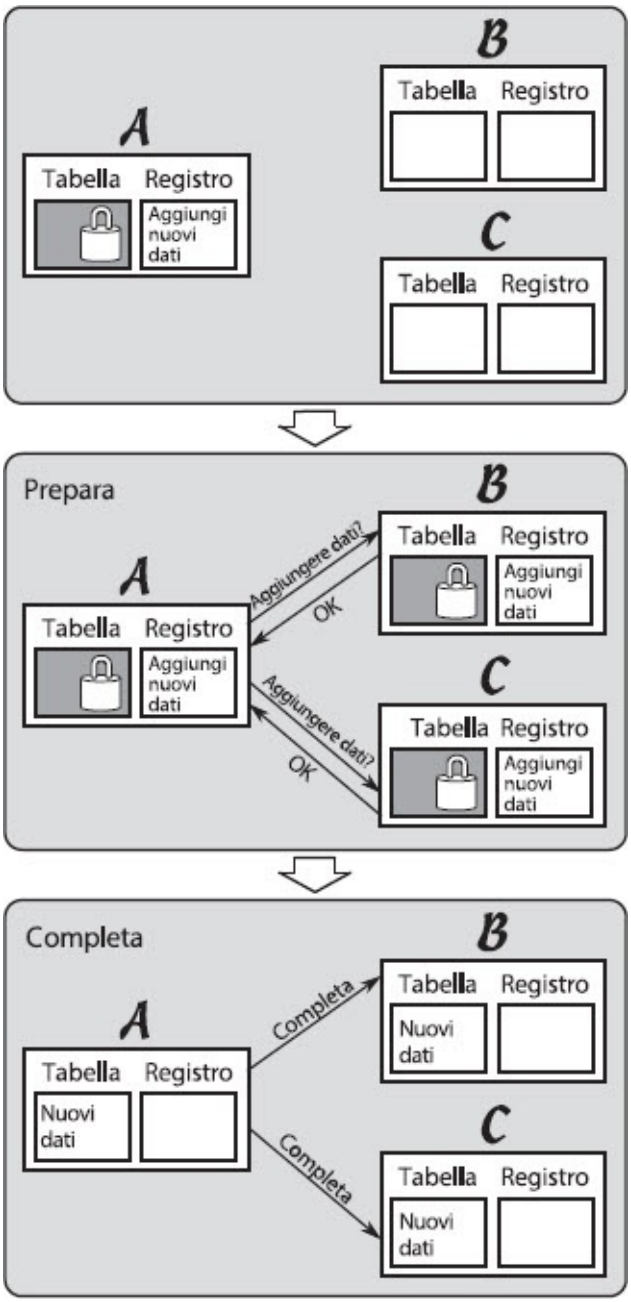
Sarà utile una semplice analogia. Supponiamo che voi e tre amici vogliate vedere insieme un film da poco uscito. Per rendere le cose interessanti, supponiamo che tutto questo succeda negli anni Ottanta, prima che esistesse la posta elettronica, così che l'uscita al cinema deve essere organizzata telefonicamente. Come fate? Uno dei metodi possibili è il seguente. Decidete un giorno e un'ora che vanno bene per voi e, per quanto ne sapete, è probabile vadano bene anche ai vostri amici. Supponiamo scegliate giovedì alle 20, Il passo successivo è chiamare uno degli amici è chiedergli se è libero giovedì alle 20. Se la risposta è positiva, gli direte qualcosa come "bene, scrivitelo, ti richiamo più tardi per conferma". Poi chiamate l'amico successivo e fate la stessa cosa. Infine chiamate il terzo e ultimo amico e gli fate la stessa domanda. Se tutti sono liberi giovedì alle 20, prendete la decisione definitiva e richiamate gli amici per confermare.

Questo è il caso facile. Ma che cosa succede se uno degli amici non è libero giovedì alle 20? In questo caso, dovete annullare tutto quello che avete fatto fino a quel punto e ricominciare da capo (è un *roll-back*). Nella realtà, probabilmente richiamereste ciascuno degli amici e proporreste subito un altro giorno e un'altra ora ma, per semplificare le cose, supponiamo invece che chiamiate ciascuno degli amici e gli diciate "Mi dispiace, giovedì alle 20 non va bene, cancella l'impegno dall'agenda e ti richiamo presto con una nuova proposta". Fatto questo, potete ricominciare tutto il processo.

Notate che ci sono due fasi distinte nella strategia per organizzare la serata al cinema. Nella prima fase, data e ora sono stati proposti ma non sono ancora certi al 100 per cento. Non appena scoprite che la proposta è accettabile per tutti, *voi* sapete che giorno e ora sono sicuri al 100 per cento, ma tutti gli altri no. Perciò c'è una seconda fase in cui richiamate tutti gli amici per confermare. Invece, se uno o più degli amici non sono disponibili, la seconda fase consiste nel richiamare tutti per annullare. Gli informatici lo chiamano *protocollo di completamento in due fasi*; noi lo chiameremo il "trucco del prima-prepara-poi-finisci". La prima è la fase di "preparazione", la seconda è una fase o di "completamento" o di "annullamento", a seconda che la proposta iniziale sia stata accettata da tutti oppure no.

È interessante il fatto che in questa analogia sia implicita l'idea del blocco del database. Anche se non ne abbiamo discusso esplicitamente, ciascuno degli amici effettua una promessa implicita quando annota in agenda la serata al cinema: promette di non pianificare altri impegni per giovedì alle 20. Finché non ricevono da voi una nuova telefonata con la conferma o l'annullamento, quell'intervallo in agenda è "bloccato" e non può essere modificato da alcuna altra "transazione". Per esempio, che cosa succederebbe se qualcun altro chiamasse il vostro amico, un po' dopo la prima fase ma prima della seconda, e gli proponesse di andare a una partita di basket giovedì alle 20? Il vostro amico potrebbe rispondergli "Mi dispiace, ma probabilmente ho un altro impegno a quell'ora. Finché quell'impegno non è confermato, non posso darti una risposta definitiva per la partita".

Vediamo ora come il trucco "prima-prepara-poi-finisci" funzioni per un database replicato. La Figura 8.2 illustra l'idea. Normalmente, una delle repliche è la "master" che coordina la transazione. Supponiamo che ci siano tre repliche, A, B e C, e che A sia la master. Supponiamo che il database debba eseguire una transazione che inserisce una nuova riga di dati in una tabella. La fase di preparazione inizia con A che blocca quella tabella, poi scrive i nuovi dati nel suo *write-ahead log*. Contemporaneamente, A invia i nuovi dati a B e C, che bloccano a loro volta le proprie copie della tabella e scrivono i nuovi dati nei loro registri. B e C poi riferiscono ad A se sono riuscite o meno a fare queste cose. Ora inizia la seconda fase.



[Figura 8.2](#) *Il trucco "prima-prepara-poi-finisci": la replica master, A, coordina due altre repliche (B, C) per l'inserimento di nuovi dati nella tabella. Nella fase di preparazione, la master controlla che tutte le altre repliche siano in grado di completare la transazione. Una volta che ne riceve conferma, la master dice a tutte le altre repliche di completare le operazioni sui dati.*

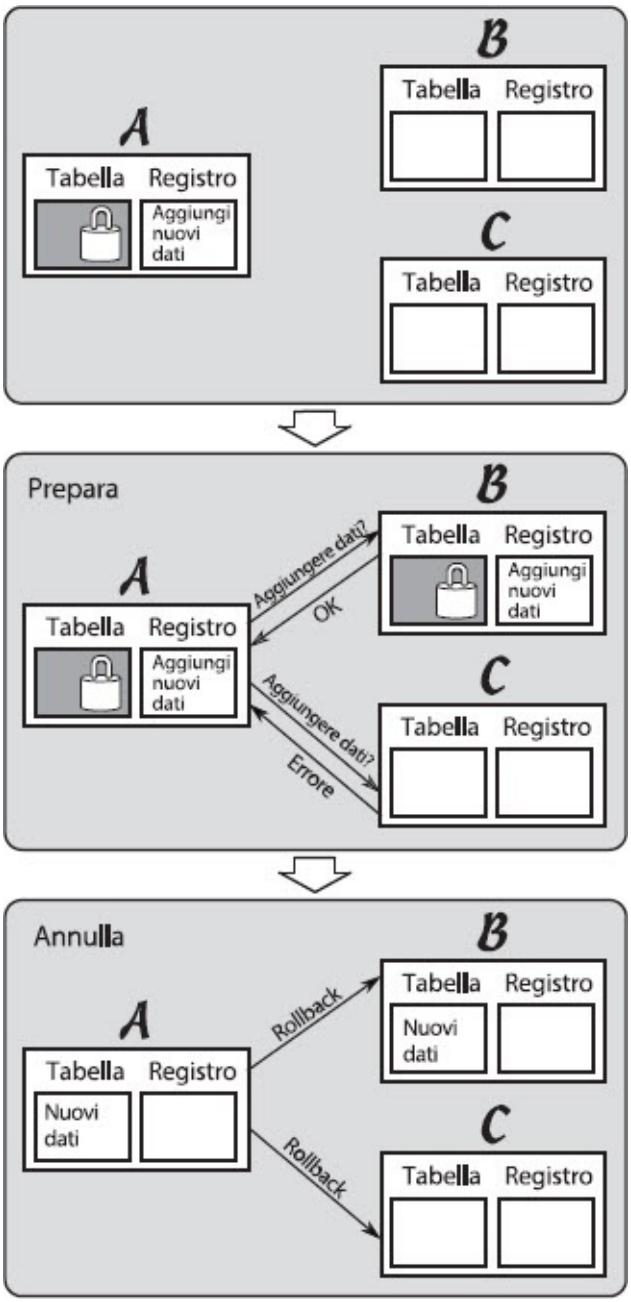


Figura 8.3 Il trucco del "prima-prepara-poi-completa" con roll-back*: il riquadro superiore è esattamente identico a quello della figura precedente ma, durante la fase di preparazione, una delle repliche incontra un errore. Di conseguenza, il riquadro inferiore è una fase di "annullamento" in cui ciascuna replica deve effettuare il* roll-back *della transazione.*

Se A, B o C hanno incontrato un problema (come l'esaurimento dello spazio su disco o l'impossibilità di bloccare la tabella) la master A sa che è necessario un *roll-back* della transazione e ne informa tutte le repliche. Ma se tutte le repliche hanno comunicato di aver portato a termine senza problemi le loro fasi di preparazione, A invia a ciascuna un messaggio che conferma la transazione, e quindi le repliche la completano (come nella Figura 8.2).

Fino qui abbiamo due trucchi di database a nostra disposizione: quello della lista delle cose da fare e il "prima-prepara-poi-finisci". Combinando i due trucchi, la vostra banca (e qualsiasi altra organizzazione online) può implementare un database replicato con transazioni atomiche. E questo consente di servire simultaneamente e in modo efficiente migliaia di clienti, con possibilità sostanzialmente nulle di incoerenza o perdita di dati. Però non abbiamo ancora guardato all'interno del database: come sono strutturati i dati, e

come si risponde alle interrogazioni? Il nostro ultimo trucco darà qualche risposta a queste domande.

I database relazionali e il trucco della tabella virtuale

Fin qui in tutti i nostri esempi il database era costituito da una sola tabella, ma la vera potenza della tecnologia moderna dei database sta nella possibilità di gestire molte tabelle. L'idea di fondo è che ciascuna tabella conserva un diverso insieme di informazioni, ma che gli elementi delle varie tabelle sono spesso collegati in qualche modo. Così il database di un'azienda può avere tabelle distinte per le informazioni sui clienti, per quelle sui fornitori e quelle sui prodotti; la tabella dei clienti però può fare riferimento a elementi presenti nella tabella dei prodotti, perché i clienti ordinano prodotti, e magari la tabella dei prodotti fa riferimento a elementi presenti nella tabella dei fornitori, perché i prodotti vengono realizzati a partire da semilavorati acquistati dai fornitori.

Diamo un'occhiata a un esempio semplice ma reale: le informazioni conservate da un'università, in cui si specifica quali corsi segue ciascuno studente. Perché l'esempio sia gestibile, considereremo solo pochi studenti e pochi corsi, ma spero sia chiaro che valgono gli stessi principi anche quando la quantità di dati è di gran lunga superiore.

In primo luogo, vediamo come possono essere memorizzati i dati nel metodo semplice, a una sola tabella, che abbiamo usato fin qui in questo capitolo. Lo si vede nel riquadro superiore della Figura 8.4: ci sono dieci righe e cinque colonne. Si può misurare in modo semplice la quantità di informazioni presenti nel database: possiamo dire che ci sono 10 × 5 = 50 elementi. Dedicate qualche istante a studiare il riquadro superiore della figura con maggiore attenzione. C'è qualcosa che vi dà fastidio, nel modo in cui i dati sono memorizzati? Per esempio, non ci sono delle ripetizioni di dati che non sono necessarie? Vi viene in mente un modo più efficiente per conservare le stesse informazioni?

Probabilmente vi siete resi conto che molte informazioni relative a ciascun corso sono duplicate per ogni studente che segue quel corso. Per esempio, tre studenti seguono ARCH101, e le informazioni specifiche relative al corso (titolo, docente, numero dell'aula) vengono ripetute per ciascuno dei tre studenti. Un modo più efficace di memorizzare queste informazioni sta nell'usare due tabelle: una per memorizzare quali corsi seguono i vari studenti e un'altra per memorizzare i particolari di ciascun corso. Questo metodo a due tabelle è visualizzato nel riquadro inferiore della Figura 8.4.

Potete edere immediatamente uno dei vantaggi di questo metodo a più tabelle: la quantità totale di spazio di memoria si riduce. Qui si usa una tabella con 10 righe e 2 colonne (cioè 10 × 2 = 20 elementi) e una seconda tabella con 3 righe e 4 colonne (cioè 3 × 4 = 12 elementi), per un totale di 32 elementi. Il metodo a una sola tabella invece aveva bisogno di 50 elementi per memorizzare esattamente le stesse informazioni.

Come è stato possibile questo risparmio? Deriva dall'eliminazione delle informazioni ripetute: invece di ripetere titolo, docente e aula di ciascun corso seguito da ciascuno studente, queste informazioni vengono archiviate esattamente una sola volta per ciascun corso. Abbiamo sacrificato qualcosa per ottenere questo risultato, però: ora i numeri dei corsi compaiono in due posti diversi, poiché c'è una colonna "numero corso" in entrambe le tabelle. Abbiamo scambiato una *grande* quantità di ripetizione (i particolari dei corsi)

con una *piccola* quantità di ripetizione (i numeri dei corsi). Nel complesso, un buon affare. Quel che si guadagna in un caso così semplice non è molto, ma nella realtà ci sono centinaia di studenti per ciascun corso, quindi i risparmi di spazio di memorizzazione sarebbero enormi.

Nome studente	Numero corso	Titolo corso	Docente	Numero aula
Francesca	ARCH101	Introduzione all'archeologia	Prof Black	610
Francesca	HIST256	Storia europea	Prof Smith	851
Susan	MATH314	Equazioni differenziali	Prof Kirby	560
Eric	MATH314	Equazioni differenziali	Prof Kirby	560
Luigi	HIST256	Storia europea	Prof Smith	851
Luigi	MATH314	Equazioni differenziali	Prof Kirby	560
Bill	ARCH101	Introduzione all'archeologia	Prof Black	610
Bill	HIST256	Storia europea	Prof Smith	851
Rose	MATH314	Equazioni differenziali	Prof Kirby	560
Rose	ARCH101	Introduzione all'archeologia	Prof Black	610

Nome studente	Numero corso
Francesca	ARCH101
Francesca	HIST256
Susan	MATH314
Eric	MATH314
Luigi	HIST256
Luigi	MATH314
Bill	ARCH101
Bill	HIST256
Rose	MATH314
Rose	ARCH101

Numero corso	Titolo corso	Docente	Numero aula
ARCH101	Introduzione all'archeologia	Prof Black	610
HIST256	Storia europea	Prof Smith	851
MATH314	Equazioni differenziali	Prof Kirby	560

Figura 8.4 In alto: un database a una sola tabella per i corsi seguiti dagli studenti. In basso: gli stessi dati organizzati, in modo più efficiente, in due tabelle.

Il metodo a più tabelle ha anche un altro vantaggio. Se le tabelle sono progettate correttamente, è più facile apportare cambiamenti al database. Supponiamo, per esempio, che il numero dell'aula per il corso MATH314 cambi e da 560 diventi 440. Nel metodo a una tabella (riquadro in alto della Figura 8.4) bisognerebbe aggiornare quattro righe diverse e, come abbiamo già visto, questi quattro aggiornamenti debbono essere inglobati in un'unica transazione per essere sicuri che il database resti coerente. Nel metodo a più tabelle (riquadro in basso della Figura 8.4) è necessario apportare un unico cambiamento, cioè aggiornare una sola voce nella tabella dei particolari dei corsi.

Chiavi

Vale la pena sottolineare che, mentre questo semplice esempio degli studenti e dei corsi può essere rappresentato in modo efficiente con due sole tabelle, i database reali incorporano spesso molte tabelle. È facile immaginare di estendere l'esempio con nuove tabelle. Per esempio, potrebbe esserci una tabella con le informazioni su ciascuno studente, come il numero di matricola, il numero di telefono, l'indirizzo di casa; potrebbe esserci una tabella con le informazioni sui docenti, con indirizzo di posta elettronica, il numero di stanza, gli orari di ricevimento. Ciascuna tabella è progettata in modo che la maggior parte delle sue colonne conservi dati che non vengono ripetuti da nessun'altra parte: l'idea è che, ogni volta che servono informazioni dettagliate su un certo oggetto si può andarle a "consultare" (*look up*) nella tabella pertinente.

Nella terminologia dei database, ogni colonna che viene utilizzata per "consultare" informazioni in una tabella, è una *chiave*. Per esempio, pensate come si potrebbero trovare le informazioni sull'aula in cui si tiene il corso di storia di Luigi. Utilizzando il metodo a una sola tabella, si esplorerebbero le singole righe fino a trovare il corso di storia di Luigi, si guarderebbe la colonna del numero di aula e si troverebbe che in questo caso è 851. Me nel metodo a molte tabelle si comincia a esplorare la prima tabella per trovare il numero del corso di storia di Luigi, e si scopre che è "HIST256". Poi si usa "HIST256" come *chiave* nell'altra tabella: si trovano i dettagli di questo corso identificando la riga che contiene "HIST256" come numero di corso, poi si cerca in quella riga il numero dell'aula (che è sempre 851). Il procedimento è visualizzato nella Figura 8.5.

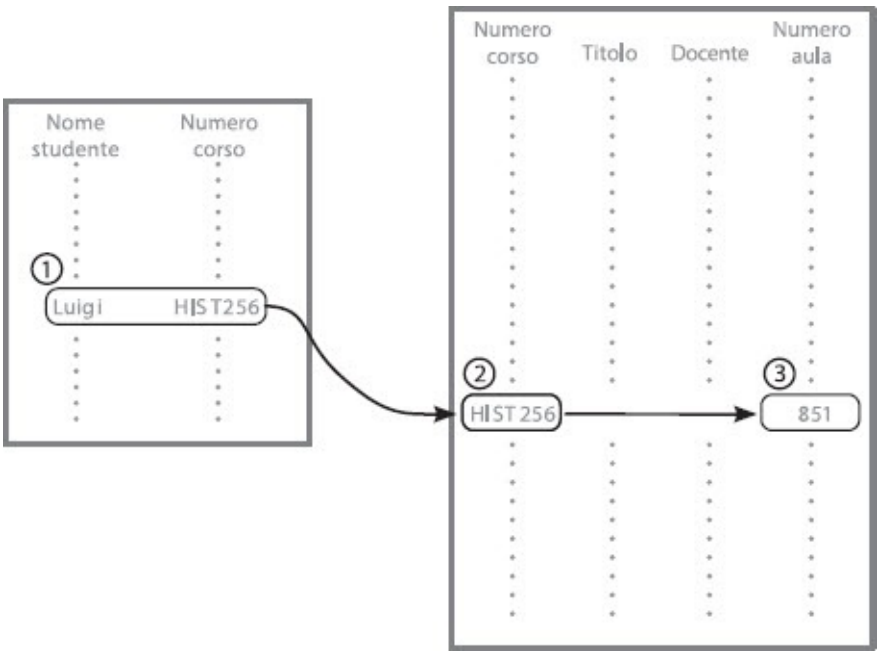


Figura 8.5 Consultazione di dati mediante una chiave: per trovare il numero dell'aula in cui si tiene il corso di storia di Luigi, prima si trova il numero del corso nella tabella a sinistra. Poi si usa quel valore, "HIST256", come chiave *nell'altra tabella. Poiché la colonna dei numeri dei corsi è ordinata alfabeticamente, si può trovare la riga giusta molto rapidamente, e poi ricavarne il numero dell'aula (851).*

Il bello delle chiavi è che i database possono utilizzarle con grande efficienza, in modo simile a come un essere umano cerca una parola in un dizionario. Pensate come cerchereste la parola "epistemologia" in un dizionario a stampa. Naturalmente, *non* iniziereste dalla prima pagina per esaminare ogni singola voce fino a trovare "epistemologia": vi avvicinereste rapidamente alla parola esaminando le testatine delle pagine, inizialmente saltando grossi blocchi di pagina e poi blocchi sempre più piccoli, man mano che vi avvicinate al traguardo. I database cercano le chiavi usando la stessa

tecnica, ma sono ancora più efficienti degli esseri umani, perché possono precalcolare i "blocchi" di pagine da saltare e possono conservare una indicazione delle testatine all'inizio e alla fine di ogni blocco. Un insieme di blocchi precalcolati per una ricerca veloce delle chiavi prende il nome di *B-albero* (*B-tree*). Si tratta di un'altra idea fondamentale e ingegnosa dei database moderni, ma un loro esame particolareggiato ci porterebbe purtroppo molto lontano.

Il trucco della tabella virtuale

Siamo quasi pronti per apprezzare il più ingegnoso dei trucchi utilizzati dai moderni database a più tabelle. L'idea è semplice: anche se tutte le informazioni di un database sono conservate in un insieme fisso di tabelle, un database può generare nuove tabelle temporanee ogni volta che ne ha bisogno. Le chiameremo "tabelle virtuali" per sottolineare il fatto che non vengono mai realmente memorizzate da qualche parte, ma il database le crea ogni volta che ne ha bisogno, per rispondere a una interrogazione, poi le cancella immediatamente.

Un esempio semplice illustrerà bene il trucco. Supponiamo di avere il database nel riquadro inferiore della Figura 8.4, e che un utente interroghi il database per sapere i nomi di tutti gli studenti che seguono i corsi del professor Kirby. Il database potrebbe procedere in molti modi diversi, ma esamineremo solo uno dei metodi possibili. Il primo passo consiste nel creare una nuova tabella virtuale in cui sono elencati studenti e docenti per tutti i corsi. Questo risultato si ottiene con una particolare operazione, denominata *join* o *unione* di due tabelle, in cui l'idea di fondo è combinare ciascuna riga di una tabella con ciascuna riga corrispondente dell'altra, dove la corrispondenza è definita da una colonna chiave che compare in entrambe le tabelle. Per esempio, quando uniamo le due tabelle del riquadro inferiore della Figura 8.4, il risultato è una tabella virtuale esattamente uguale a quella nel riquadro superiore della figura: il nome di ciascuno studente è combinato con tutti i dettagli del corso pertinente della seconda tabella, e questi dettagli vengono ricavati utilizzando il "numero del corso" come chiave. Ovviamente, l'interrogazione riguardava nomi di studenti e docenti, perciò non ci servono le altre colonne. Fortunatamente esiste anche un'operazione di *proiezione* che consente di eliminare le colonne che non interessano. Così, con un'operazione di unione per combinare le due tabelle, seguita da una operazione di proiezione per eliminare alcune colonne non necessarie, il database produce questa tabella virtuale:

Nome studente	**Docente**
Francesca	Prof Black
Francesca	Prof Smith
Susan	Prof Kirby
Eric	Prof Kirby
Luigi	Prof Smith
Luigi	Prof Kirby
Bill	Prof Black
Bill	Prof Smith
Rose	Prof Kirby
Rose	Prof Black

Poi il database un’altra operazione importante, quella di selezione (*select*), con cui si scelgono alcune delle righe da una tabella, sulla base di un criterio dato, e si eliminano le altre righe, producendo una nuova tabella virtuale. In questo caso, vogliamo sapere quali siano gli studenti che seguono i corsi tenuti dal professor Kirby, perciò ci serve un’operazione di selezione che estragga solo le righe in cui il docente è “Prof Kirby”. Questo ci dà la seguente tabella virtuale:

Nome studente	**Docente**
Susan	Prof Kirby
Eric	Prof Kirby
Luigi	Prof Kirby
Rose	Prof Kirby

La risposta all’interrogazione è quasi completa. Tutto quello che serve ora è un’altra operazione di proiezione, che elimini la colonna “Docente” e ci lasci con una tabella virtuale che risponde all’interrogazione originale:

Nome studente
Susan
Eric
Luigi
Rose

Val la pena aggiungere una annotazione un po’ più tecnica. Se per caso conoscete il linguaggio di interrogazione di database SQL, probabilmente la definizione precedente dell’operazione di selezione vi sarà sembrata un po’ strana, perché in SQL il comando “select” fa molto più che estrarre semplicemente alcune righe. La terminologia usata qui deriva da una teoria matematica delle operazioni su database, l’*algebra relazionale*, in cui “select” si usa solo per selezionare righe. L’algebra relazione contempla anche le operazioni di “unione” e “proiezione” che abbiamo utilizzato nella nostra interrogazione per trovare gli studenti del professor Kirby.

Database relazionali

Un database che conserva tutti i dati in tabelle interconnesse come quelle che abbiamo usato è un *database relazionale*. Il primo a propugnarne l’uso è stato E.F. Codd, ricercatore dell’IBM, in un articolo del 1970 che ha avuto un’influenza straordinaria, “A Relational Model of Data for Large Shared Data Banks”. Come molte grandi idee scientifiche, i database relazionali sembrano semplici, con il senno di poi, ma all’epoca hanno rappresentato un grande salto in avanti nella memorizzazione ed elaborazione di informazioni. Basta un piccolo numero di operazioni (come le operazioni di selezione, unione e proiezione dell’algebra relazionale che abbiamo visto prima) per generare tabelle virtuali che rispondono sostanzialmente a qualsiasi interrogazione. Quindi un database relazionale può conservare i suoi dati in tabelle strutturate nel modo migliore a fini di efficienza, e può usare il trucco della tabella virtuale per rispondere a interrogazioni che apparentemente richiedono che i dati siano in una forma diversa.

Grazie a tutte queste caratteristiche i database sono utilizzati in gran parte delle attività

di e-commerce. Ogni volta che acquistate qualcosa online, probabilmente interagite con una serie di tabelle di database relazionali in cui sono conservate informazioni su prodotti, clienti e singoli acquisti. Nel ciberspazio, siamo costantemente circondati da database relazionali, spesso senza rendercene conto.

Il volto umano dei database

Per l'osservatore casuale, i database forse saranno l'argomento meno attraente di questo libro: è proprio difficile essere affascinati dall'immagazzinamento di dati. Dietro le quinte, però, le idee ingegnose che fanno funzionare i database raccontano una storia diversa. Basati su hardware che può guastarsi nel bel mezzo di qualsiasi operazione, i database ci danno comunque l'efficienza e la solida affidabilità che ci siamo abituati a pretendere dall'online banking e da attività simili. Il trucco della lista delle cose da fare ci dà transazioni atomiche, che garantiscono coerenza anche quando migliaia di clienti interagiscono simultaneamente con lo stesso database. Questo grado enorme di concorrenza, insieme con la rapidità di risposta alle interrogazioni attraverso il trucco delle tabelle virtuali, rende efficienti i grandi database. Il trucco della lista delle cose da fare inoltre garantisce la coerenza anche in caso di guasti; quando è combinato con il trucco del "prima-prepara-poi-completa" per i database replicati, fa sì che i nostri dati abbiano sempre una coerenza ferrea e si conservino correttamente nel tempo.

Il trionfo eroico dei database sui componenti inaffidabili, quella che gli informatici chiamano "tolleranza ai guasti" o *fault-tolerance*, è opera di molti ricercatori nell'arco di molti decenni. Fra coloro che hanno dato i contributi più importanti c'è Jim Gray, eccellente informatico che ha definito l'elaborazione per transazioni. (Il suo libro *Transaction Processing: Concepts and Techniques* è stato pubblicato nel 1992.) Purtroppo la carriera di Gray si è conclusa prematuramente: un giorno del 2007 si è imbarcato sul suo yacht, è passato sotto il Golden Gate, ha attraversato la Baia di San Francisco e si diretto in oceano aperto diretto a una delle isole vicine. Non si è avuta più notizia né di Gray né della sua imbarcazione. I molti amici di Gray nella comunità dei database hanno usato i suoi stessi strumenti nel tentativo di salvarlo: immagini da satellite dell'oceano in prossimità di San Francisco sono state caricate in un database in modo che amici e colleghi potessero cercare le sue tracce. Purtroppo la ricerca non è approdata a nulla e il mondo dell'informatica ha perso uno dei suoi maggiori luminari.

---

1 Tr. it. di Nicoletta Rosati Bizzotto, Newton Compton, Roma, 1991, p, 452.

CAPITOLO 9

Firme digitali: chi ha *veramente* scritto questo software?

> A dimostrarvi il vostro errore e l'infondatezza della vostra asserzione, vi farò vedere un certificato […] Eccolo. Ah, guardate! Prendetelo in mano; non è falso.
>
> Charles Dickens, *Le due città*[1]

Tra tutte le idee che si incontrano in questo libro, il concetto di una "firma digitale" è forse il più paradossale. La parola "digitale", interpretata alla lettera, significa "costituito da una successione di cifre". Perciò, per definizione, qualsiasi cosa sia digitale può essere copiata: basta copiare le cifre una alla volta. Se posso leggerlo, posso copiarlo! Invece il senso di una "firma" è che possa essere letta, ma non copiata (cioè falsificata) da chi non ne sia l'autore. Come si può creare una firma che è digitale, ma non può essere copiata? In questo capitolo, scopriremo come si risolve questo intrigante paradosso.

Per che cosa si usano veramente le firme digitali?

Forse non sembrerà necessario porsi la domanda: per che cosa sono usate le firme digitali? Certo, penserete, possiamo usarle per le stesso tipo di cose per cui si usano le firme su carta: firmare assegni e altri documenti legali, come il mutuo per l'acquisto di una casa. Ma se ci pensate un attimo, vi renderete conto che non è vero. Ogni volta che effettuate un pagamento online, mediante carte di credito o un sistema di online banking, firmate qualcosa? No. Normalmente i pagamenti online mediante carta di credito non richiedono alcuna firma. I sistemi di online banking sono un po' diversi, perché vi chiedono di accreditarvi con una password che permetta di verificare la vostra identità. Ma se poi effettuate un pagamento durante la sessione, non vi viene chiesto alcun tipo di firma.

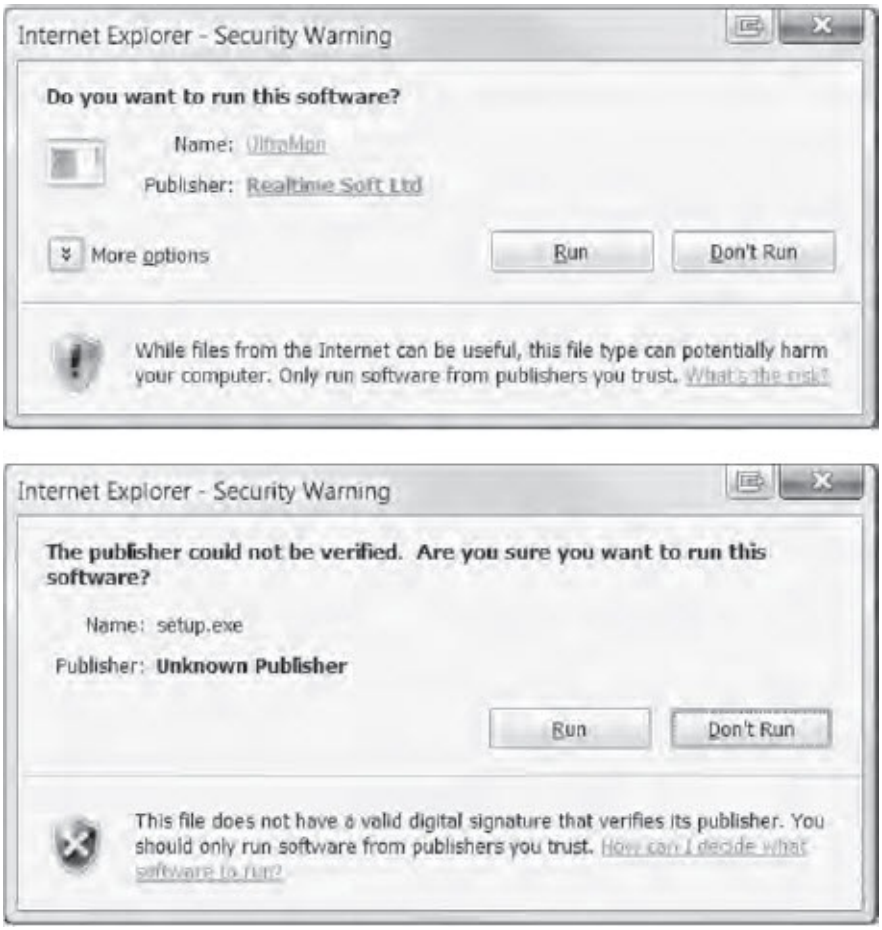


Figura 9.1 Il vostro computer controlla automaticamente le firme digitali. In alto: il messaggio che mi mostra il mio browser quando cerco di scaricare ed eseguire un programma che ha una firma digitale valida. In basso: il messaggio nel caso di firma digitale non valida o mancante.

Per che cosa si usano, in pratica, le firme digitali? La risposta è il contrario di quel che potreste pensare a tutta prima: non siete voi che firmate materiali che spedite ad altri, normalmente sono gli altri che devono firmare materiali prima di inviarli a voi. Il motivo per cui probabilmente non ne siete consapevoli è che le firme digitali vengono verificate automaticamente dal vostro computer. Per esempio, ogni volta che cercate di scaricare ed eseguire un programma, il vostro browser probabilmente controlla se il programma ha una firma digitale e se questa è valida o meno. Poi può presentarvi un avviso, variabile a seconda dei casi, come quelli nella Figura 9.1.

Come potete vedere, ci sono due possibilità. Se il software ha una firma valida, il computer può dirvi con assoluta fiducia il nome dell'azienda che ha scritto quel software. Ovviamente, questo non vi garantisce che il software sia privo di rischi, ma per lo meno potete prendere una decisione informata, in base alla fiducia che riponete in quell'azienda. Invece, se la firma non è valida o manca, non avete alcuna garanzia della reale provenienza del software. Anche se pensate di aver scaricato software da un'azienda di buona reputazione, è possibile che un hacker in qualche modo abbia sostituito un software dannoso a quello reale. Oppure il software è stato prodotto da un dilettante che non aveva il tempo o il motivo per creare una firma digitale valida. Sta a voi, gli utenti, decidere se fidarvi di quel software in tali circostante.

Anche se la firma del software è l'applicazione più ovvia delle firme digitali, non è affatto l'unica. Il vostro computer riceve e verifica firme digitali con incredibile frequenza, perché alcuni protocolli di Internet utilizzano le firme digitali per verificare l'identità dei

computer con cui interagite. Per esempio, i server sicuri il cui indirizzo web inizia con "https" normalmente inviano al vostro computer un certificato firmato digitalmente prima di stabilire una sessione sicura. Le firme digitali sono usate anche per verificare l'autenticità di molti componenti software, come i *plug-in* per i browser. Probabilmente avete visto messaggi di avvertimento relativi a queste cose mentre navigavate nel Web.

Esiste un altro tipo di firma online che può darsi abbiate incontrato qualche volta: ci sono siti che vi chiedono di scrivere il vostro nome come firma in un modulo online. A volte mi capita di doverlo fare quando compilo una lettera di raccomandazione online per uno dei miei studenti, per esempio. Questo *non* è ciò che un informatico intende quando parla di firma digitale! Ovviamente, questo tipo di firma può essere falsificato senza fatica, da chiunque sappia il vostro nome. In questo capitolo, vedremo come si crea una firma digitale che non può essere falsificata.

Firme su carta

Costruiremo gradualmente la nostra spiegazione delle firme digitali partendo dalle comuni firme su carta per procedere a piccoli passi verso le vere firme digitali. Per cominciare, torniamo a un mondo in cui non c'erano computer. In quel mondo, l'unico modo per autenticare un documento era mettere una firma su carta. Notate che in questo mondo un documento firmato non può essere autenticato isolatamente. Per esempio, supponiamo che troviate un pezzo di carta che dice "Prometto di pagare 100 dollari a Francoise. Firmato, Ravi", come nella Figura 9.2. Come potete verificare che proprio Ravi abbia firmato questo documento?

La risposta è che dovete avere un deposito fidato di firme, dove potete andare e verificare che la firma di Ravi sia autentica. Nel mondo reale, istituti come banche e uffici della pubblica amministrazione svolgono questo ruolo: conservano effettivamente documenti con la firma dei loro clienti, e quei documenti possono essere verificati fisicamente, ove si renda necessario. Nel nostro scenario fittizio, immaginiamo che un'istituzione fidata conservi la firma di tutti su appositi documenti: una "banca delle firme su carta" (Figura 9.3).

Per verificare la firma di Ravi sul documento che promette un pagamento a Francoise, dobbiamo solo andare alla banca delle firme su carta e chiedere di vedere la firma di Ravi. Ovviamente, qui facciamo due assunzioni importanti.

La prima è che della banca ci si possa fidare. In teoria è possibile che un impiegato della banca scambi la firma di Ravi con quella di un impostore, ma qui ignoreremo questa possibilità.

In secondo luogo, assumiamo che sia impossibile falsificare la firma di Ravi, un'assunzione, come tutti sanno, del tutto errata: un buon falsario può riprodurre facilmente una firma, e anche i dilettanti possono approssimarla sufficientemente bene. Ciononostante, abbiamo bisogno dell'assunzione della non falsificabilità, altrimenti la firma su carta è inutile. In seguito vedremo come sia essenzialmente impossibile falsificare le firme digitali: è uno dei loro grandi vantaggi rispetto alle firme su carta.

Figura 9.2 Un documento cartaceo con una firma scritta a mano.

Figura 9.3 Una banca che conserva documentazione delle identità dei suoi clienti con le loro firme manoscritte.

Firmare con un lucchetto

Il primo passo verso le firme digitali consiste nell'abbandonare completamente le firme su carta e adottare un nuovo metodo per autenticare i documenti che si basa su lucchetti, chiavi e forzieri. Tutti quelli che partecipano al nuovo schema (nel nostro esempio Ravi, Takeshi e Francoise) acquistano un'abbondante riserva di lucchetti. È fondamentale che ciascuno abbia lucchetti tutti uguali: così i lucchetti di Ravi, per esempio, sono tutti identici fra loro. Inoltre, per ciascun partecipante i lucchetti debbono essere *esclusivi*: nessun altro può costruire o ottenere un lucchetto come quello di Ravi. Infine, tutti i lucchetti di questo capitolo hanno una caratteristica fuori dal comune: sono dotati di sensori biometrici in modo che possano essere chiusi solo dal relativo proprietario. Così, se Francoise trova un lucchetto aperto che appartiene a Ravi, non può usarlo per chiudere alcunché. Ovviamente, Ravi ha anche una scorta di chiavi che aprono i suoi lucchetti. Poiché tutti i suoi lucchetti sono identici, sono identiche anche le chiavi. La situazione come si presenta fin qui è visualizzata schematicamente nella Figura 9.4. Questa è la configurazione iniziale per quello che possiamo chiamare il "trucco del lucchetto fisico".

Figura 9.4 Nel trucco del lucchetto fisico, ciascun partecipante ha una scorta esclusiva di lucchetti e chiavi, tutti identici fra loro.

Supponiamo ora che, come prima, Ravi debba 100 dollari a Francoise e Francoise

voglia documentare questo fatto in modo verificabile. In altre parole, Francoise vuole l'equivalente del documento di Figura 9.2, ma senza doversi basare su una firma a mano. Ecco come funziona il trucco. Ravi crea un documento che dice "Ravi promette di pagare 100 dollari a Francoise" e non lo firma. Crea una copia del documento e la colloca in un forziere. (Un forziere in questo caso è solo una scatolina molto robusta che può essere chiusa con un lucchetto.) Infine, Ravi chiude il forziere con uno dei suoi lucchetti e dà il forziere chiuso a Francoise. Il pacchetto completo è visualizzato nella Figura 9.5. In un senso che preciseremo presto, il forziere chiuso *è* la firma del documento. Notate che sarebbe una buona idea che Francoise, o qualche altro testimone fidato, osservi la creazione della firma; altrimenti Ravi potrebbe barare e mettere nel forziere un documento diverso. (Si potrebbe dire anche che questo schema funzionerebbe ancora meglio se i forzieri fossero trasparenti. In fin dei conti, le firme digitali garantiscono autenticità, non segretezza. Però un forziere trasparente è un po' controintuitivo, perciò non prenderemo in considerazione questa possibilità.)

Ora forse siete già in grado di vedere come Francoise possa autenticare il documento di Ravi. Se qualcuno, magari lo stesso Ravi, cerca di negarne l'autenticità, Francoise può dire: "Va bene, Ravi, per favore prestami una delle tue chiavi per un minuto. Adesso apro il tuo forziere con la tua chiave". In presenza di Ravi e qualche altro testimone (magari un giudice in un tribunale) Francoise apre il lucchetto ed espone i contenuti del forziere. Poi può continuare: "Ravi, poiché sei l'unico che possa accedere ai lucchetti che funzionano con questa chiave, nessun altro può essere responsabile dei contenuti del forziere. Perciò tu e solo tu hai scritto questa cambiale e l'hai messa nel forziere. Quindi *mi devi* 100 dollari!"

Anche se, la prima volta che lo si incontra, questo metodo di autenticazione sembra contorto, è sia pratico sia molto efficace. Ha però qualche svantaggio: il problema principale è che richiede la cooperazione di Ravi; per poter dimostrare qualsiasi cosa, Francoise deve convincere Ravi a prestarle una delle sue chiavi. Ravi potrebbe rifiutarsi o, peggio, potrebbe far finta di cooperare ma poi darle una chiave diversa, una chiave che non aprirà il suo lucchetto. Poi, quando Francoise non riesce ad aprire il forziere, Ravi può dire: "Vedi, non è uno dei miei lucchetti, un falsario può aver creato il documento e averlo messo qui dentro senza che io ne fossi a conoscenza".

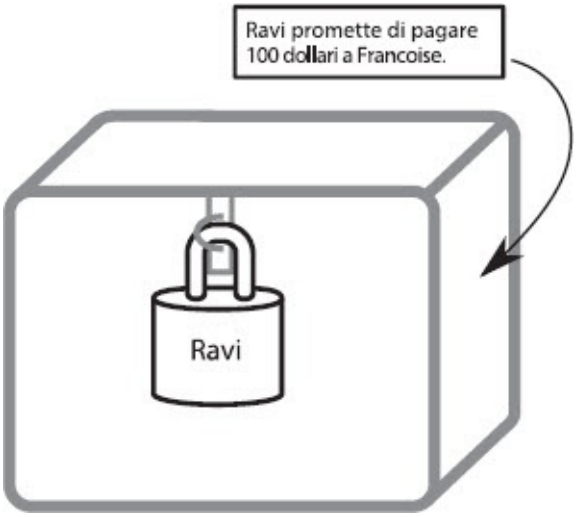


Figura 9.5 Per creare una firma verificabile con il trucco del lucchetto fisico, Ravi colloca una copia del documento in un forziere e lo chiude con uno dei suoi lucchetti.

Per impedire questa astuzia da parte di Ravi, dobbiamo ricorrere a un terzo fidato, come una banca. A differenza della banca delle firme su carta, la nostra nuova banca conserverà chiavi. Così, invece di dare alla banca una copia delle loro firme, i partecipanti ora danno alla banca una chiave fisica che aprirà i loro forzieri (Figura 9.6).

Figura 9.6 Una banca delle chiavi fisica conserva le chiavi che apriranno i lucchetti di ciascun partecipante. Notate che le chiavi sono tutte diverse fra loro.

Questa banca è il tassello finale del puzzle e completa la spiegazione del trucco del lucchetto fisico. Se Francoise dovesse dimostrare che Ravi ha scritto quel documento, non farebbe che portare il forziere alla banca con qualche testimone e aprilo lì con la chiave di Ravi. Il fatto che il lucchetto si apra dimostra che solo Ravi ha la responsabilità dei contenuti del forziere, e che questo contiene una copia esatta del documento che Francoise vuole autenticare.

Firmare con un lucchetto moltiplicativo

L'infrastruttura di chiave e lucchetto che abbiamo costruito è il modo giusto per affrontare le firme digitali. Ovviamente, però, non possiamo usare lucchetti fisici e chiavi fisiche per firme che debbono essere trasmesse elettronicamente, perciò il passo successivo sarà sostituirli con oggetti matematici analoghi che possano essere rappresentati digitalmente. Specificamente, lucchetti e chiavi saranno rappresentati da *numeri*, e l'atto di aprire o chiude un lucchetto sarà rappresentata dalla *moltiplicazione nell'aritmetica dell'orologio*. Se non vi trovate molto a vostra agio con l'aritmetica dell'orologio, vi conviene leggere nuovamente la spiegazione nel Capitolo 4, a pagina 60.

Per creare firme digitali non falsificabili, i calcolatori usano orologi di dimensioni assolutamente enormi, in genere pari a decine o centinaia di cifre. Però qui useremo un orologio di dimensioni irrealisticamente piccole, in modo che i calcoli siano semplici.

Specificamente, in tutti gli esempi di questa sezione useremo un orologio di dimensione 11. Dato che eseguiremo parecchie moltiplicazioni con queste dimensioni di orologio, la Tabella 9.1 elenca tutti i prodotti dei numeri minori di 11. Per esempio, calcoliamo 7 × 5. Manualmente, senza usare la tabella, dobbiamo prima calcolare il prodotto con la normale aritmetica: 7 × 5 = 35. Poi, dividiamo per 11 e consideriamo il resto. 11 nel 35 sta tre volte (il che ci dà 33) con un resto di 2. Quindi la risposta finale deve essere 2. Osservando la tabella, si vede che il numero all'incrocio della riga 7 e della colonna 5 è proprio 2. (Potete usare anche la colonna 7 e la riga 5, l'ordine non è importante: verificatelo.) Provate un altro paio di moltiplicazioni per essere sicuri che sia tutto chiaro.

Tabella 9.1 La tavola di moltiplicazione per l'aritmetica dell'orologio di dimensione 11

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	1	3	5	7	9
3	3	6	9	1	4	7	10	2	5	8
4	4	8	1	5	9	2	6	10	3	7
5	5	10	4	9	3	8	2	7	1	6
6	6	1	7	2	8	3	9	4	10	5
7	7	3	10	6	2	9	5	1	8	4
8	8	5	2	10	7	4	1	9	6	3
9	9	7	5	3	1	10	8	6	4	2
10	10	9	8	7	6	5	4	3	2	1

Prima di procedere, dobbiamo modificare leggermente il problema che stiamo cercando di risolvere. In precedenza, cercavamo dei modi in cui Ravi potesse "firmare" un messaggio (in realtà una cambiale, un "pagherò") a Francoise. Il messaggio era scritto in italiano corrente. Da ora in poi, sarà più comodo lavorare solo con i numeri. Perciò dobbiamo essere d'accordo che sia facile per un computer tradurre il messaggio in una successione di numeri che Ravi possa firmare. Poi, se e quando qualcuno avrà bisogno di autenticare la firma digitale di Ravi per questa successione di numeri, sarà semplice invertire la traduzione e trasformare i numeri di nuovo in italiano. Abbiamo incontrato lo stesso problema parlando delle somme di controllo (p. 78) e del trucco del simbolo più corto (p. 128). Se volete rivedere più particolareggiatamente questo aspetto, rileggete la discussione del trucco del simbolo più corto: la Tabella 7.1 presenta una semplice possibilità esplicita per tradurre lettere in numeri e viceversa.

Quindi, anziché firmare un messaggio scritto in italiano, Ravi dovrà firmare una successione di numeri, magari qualcosa come "494138167543 … 83271696129149". Per semplificare, cominceremo ipotizzando che il messaggio da firmare sia brevissimo: in effetti il messaggio di Ravi sarà costituito da una sola cifra, come "8" o "5". Non preoccupatevi: alla fine vedremo come firmare messaggi di lunghezza più sensata. Per il momento, però, meglio rimanere ai messaggi di una sola cifra.

Sistemati questi preliminari, siamo pronti ad affrontare il nocciolo di un nuovo trucco, il "trucco del lucchetto moltiplicativo". Come nel trucco del lucchetto fisico, Ravi avrà bisogno di un lucchetto e di una chiave che lo apra. Ottenere un lucchetto è facilissimo: Ravi sceglie una dimensione di orologio e poi sceglie sostanzialmente qualsiasi numero minore della dimensione dell'orologio come proprio "lucchetto". (In realtà, alcuni numeri funzionano meglio di altri, ma questi particolari ci porterebbero troppo fuori strada.) Per concretizzare, diciamo che Ravi sceglie 11 come dimensione di orologio e 6 come proprio lucchetto.

Ora, come fa Ravi a "chiudere" il suo messaggio in un forziere con il suo lucchetto? Per strano che possa risultare, Ravi userà la moltiplicazione: la versione "chiusa" del suo messaggio sarà il lucchetto moltiplicato per il messaggio (nell'aritmetica dell'orologio di dimensione 11, ovviamente). Ricordate, stiamo trattando il caso semplice di un messaggio di una sola cifra. Perciò supponiamo che il messaggio di Ravi sia "5". Quindi il messaggio "chiuso" sarà 6 × 5, che è 8, nell'aritmetica dell'orologio di dimensione 11. (Verificatelo

consultando la Tabella 9.1.) Il procedimento è riassunto nella Figura 9.7. il risultato finale, 8, è la firma digitale di Ravi per il messaggio originale.

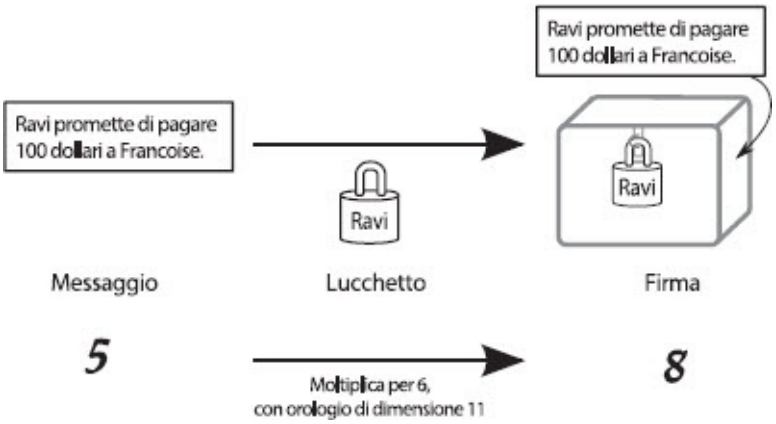


Figura 9.7 Come "chiudere" un messaggio numerico con un "lucchetto", creando una firma digitale. La riga superiore mostra come chiudere fisicamente un messaggio in un forziere con un lucchetto fisico. La riga inferiore mostra l'operazione matematica analoga, in cui il messaggio è un numero (5), il lucchetto è un altro numero (6) e il processo di chiusura del lucchetto corrisponde alla moltiplicazione dei due numeri in base a una determinata dimensione di orologio. Il risultato finale (8) è la firma digitale del messaggio.

Ovviamente, questo tipo di "chiusura con un lucchetto" matematica non avrebbe alcuna utilità se poi non si potesse aprire il messaggio con un qualche tipo di "chiave" matematica. Per fortuna, esiste un modo semplice per aprire i messaggi. Il trucco sta nell'usare di nuovo la moltiplicazione (come al solito nell'aritmetica dell'orologio), ma questa volta moltiplicando per un numero diverso – un numero scelto apposta in modo che apra il numerolucchetto scelto in precedenza.

Rimaniamo allo stesso esempio concreto: Ravi usa un orologio di dimensione 11, con 6 come numero-lucchetto. La chiave corrispondente è 2. Come facciamo a saperlo? Torneremo fra breve su questa domanda importante. Per il momento, affrontiamo il compito più semplice di verificare che la chiave funziona, non appena qualcuno ci ha detto il suo valore numerico. Come già detto, si apre il messaggio chiuso con il lucchetto moltiplicando per la chiave. Come abbiamo già visto, nella Figura 9.7, quando Ravi chiude il messaggio 5 con il lucchetto 6, ottiene il messaggio chiuso (o firma digitale) 8. Per aprirlo, prendiamo questo 8 e lo moltiplichiamo per la chiave 2, che ci dà 5 nella solita aritmetica dell'orologio. Come per magia, siamo tornati al messaggio originale, 5! Tutto il procedimento è visualizzato nella Figura 9.8, che dà anche un altro paio di esempi: il messaggio "3" diventa "7" quando chiuso con il lucchetto, e torna a essere "3" quando si applica la chiave. Analogamente "2" diventa "1" quando viene chiuso, ma la chiave lo riconverte in "2".

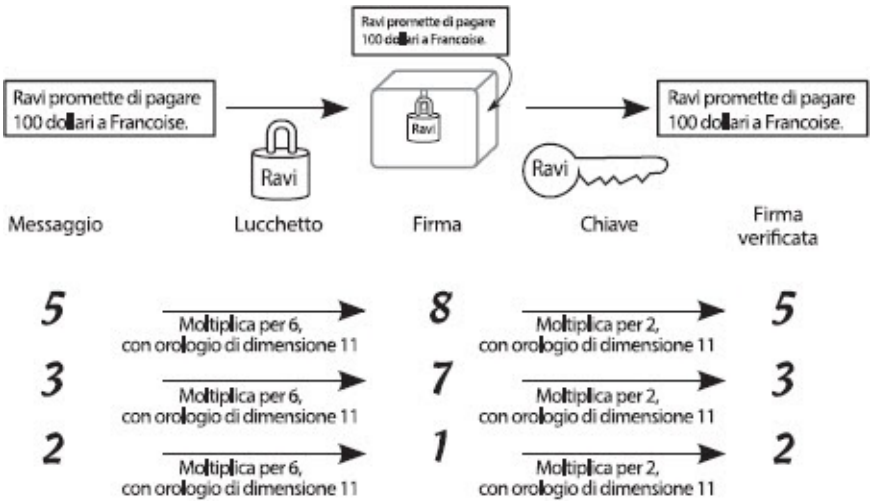


Figura 9.8 Come “chiudere” e poi “aprire” un messaggio mediante un lucchetto numerico e una corrispondente chiave numerica. In alto è illustrata la versione fisica del procedimento, mentre sotto sono mostrati alcuni esempi di chiusura e apertura dei messaggi mediante moltiplicazione. Notate che il processo di chiusura produce una firma digitale, mentre il processo di apertura produce un messaggio. Se il messaggio aperto corrisponde al messaggio originale, la firma digitale è verificata e il messaggio è autentico.

La Figura 9.8 spiega anche come verificare le firme digitali. Basta prendere la firma e aprirla con la chiave moltiplicativa del firmatario. Se il messaggio aperto risultante corrisponde al messaggio originale, la firma è autentica; altrimenti deve essere stata falsificata. Questo procedimento di verifica è visualizzato più particolareggiatamente nella Tabella 9.2. Anche lì la dimensione dell’orologio è 11, ma per mostrare che non c’è niente di speciale nel lucchetto e nella chiave numerici che abbiamo usato fin qui, useremo valori diversi. Qui il valore del lucchetto è 9 e la chiave corrispondente è 5. Nel primo esempio della tabella il messaggio è “4” con firma “3”. La firma restituisce “4”, che corrisponde al messaggio originale: la firma quindi è autentica. La riga successiva della tabella dà un esempio analogo per il messaggio “8” con firma “6”. L’ultima riga invece mostra che cosa succede quando la firma è stata falsificata. Qui, il messaggio è ancora “8” ma la firma è “7”. Questa firma ci dà il messaggio “2”, che non corrisponde al messaggio originale. Quindi la firma è falsa.

Tabella 9.2 Come identificare una firma digitale falsa. Questi esempi usano come lucchetto 9 e come chiave 5. Le prime due firme sono autentiche, ma la terza è falsa

Messaggio	**Firma digitale** (Per una firma autentica, moltiplica il messaggio per il lucchetto 9. Per fare una firma falsa, scegli un numero a caso.	**Firma aperta** (Per aprire la firma, moltiplica per la chiave 5.)	**Corrisponde al messaggio originale?**	**Falsa?**
4	3	4	Sì	No
8	6	8	Sì	No
8	7	2	No	Sì

Se ripensate alla situazione del lucchetto e della chiave fisici, ricorderete che i lucchetti hanno sensori biometrici per impedire l’accesso ad altri, altrimenti un falsario potrebbe usare uno dei lucchetti di Ravi per chiudere qualsiasi messaggio in un forziere, falsificando la sua firma. Lo stesso vale per i lucchetti numerici. Ravi deve mantenere *segreto* il suo numero-lucchetto. Ogni volta che firma un messaggio, può rendere pubblici il messaggio e la firma, ma non il numero-lucchetto usato per produrre la firma.

E la dimensione dell’orologio e la chiave di Ravi? Anche questi devono rimanere

segreti? No, Ravi può comunicare a tutti la dimensione dell'orologio e il valore della chiave, magari pubblicandoli su un sito web, senza compromettere lo schema per la verifica delle firme. Se Ravi pubblica dimensione dell'orologio e valore della chiave, chiunque può ottenere questi numeri e quindi verificare le sue firme. Questo metodo sembra davvero molto comodo, a prima vista; ma ci sono alcuni dettagli importanti di cui tener conto.

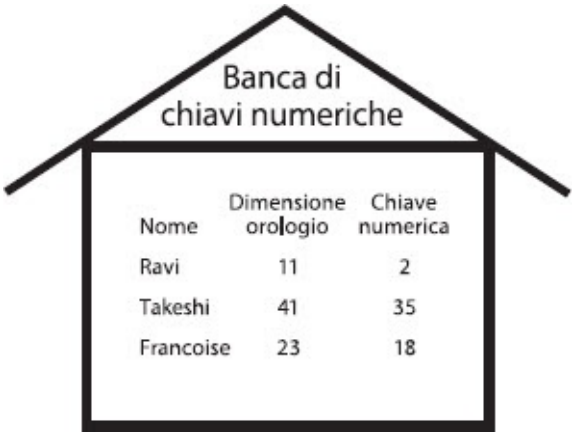


Figura 9.9 Una banca di chiavi numeriche. Il ruolo della banca non è quello di mantenere segrete le chiavi numeriche e le dimensioni dell'orologio. La banca è un'autorità fidata da cui si possono ottenere la chiave vera e la dimensione di orologio associate a ciascun individuo. La banca comunica liberamente queste informazioni a chiunque ne faccia richiesta.

Per esempio, questa impostazione elimina la necessità di una banca fidata, che invece è necessaria sia per la tecnica della firma su carta sia per quella di lucchetto e chiave fisici? La risposta è negativa: un terzo fidato come una banca è ancora necessario. In caso contrario, Ravi potrebbe distribuire una chiave falsa che farebbe sembrare non valide le sue firme; cosa ancora peggiore, un nemico di Ravi potrebbe creare un nuovo lucchetto numerico e la corrispondente chiave numerica, creare un sito web in cui annuncia che quella è la chiave di Ravi e poi firmare digitalmente tutti i messaggi che vuole con il nuovo lucchetto numerico. Chiunque pensi che la nuova chiave appartiene a Ravi crederà che i messaggi del nemico siano firmati da Ravi. Il ruolo della banca quindi non è mantenere segrete la chiave e la dimensione di orologio di Ravi; la banca è invece l'autorità fidata per il valore della chiave numerica di Ravi e la dimensione dell'orologio (Figura 9.9).

Si potrebbe sintetizzare in modo utile dicendo che i lucchetti numeri sono *privati*, mentre le chiavi numeriche e le dimensioni di orologio sono *pubbliche*. Certo è un po' controintutivo che una chiave sia "pubblica", perché nella vita quotidiana stiamo ben attenti a tenerci ben strette le nostre chiavi fisiche. Ripensate però al trucco del lucchetto fisico: in quel caso la banca conservava una copia della chiave di Ravi e la prestava tranquillamente a chiunque volesse verificare la firma di Ravi. Perciò la chiave fisica era, in un certo senso, "pubblica". Lo stesso ragionamento vale per le chiavi moltiplicative.

Questo è il momento giusto per affrontare un problema pratico importante: che cosa succede se vogliamo firmare un messaggio che sia lungo più di una cifra? Le risposte possono essere diverse. Una soluzione iniziale è usare un orologio di dimensione molto maggiore: se usiamo un orologio con dimensione di 100 cifre, per esempio, gli stessi metodi ci consentono di firmare messaggi di 100 cifre con firme di 100 cifre. Se il messaggio è più lungo, potremmo semplicemente dividerlo in blocchi di 100 cifre e

firmare ciascun blocco separatamente. Gli informatici però hanno un metodo migliore: i messaggi lunghi possono essere ridotti, ai fini della firma, a un singolo blocco (di, poniamo, 100 cifre) applicando un particolare tipo di trasformazione, una *funzione hash crittografica*. Abbiamo già incontrato queste funzioni nel Capitolo 5, dove erano usate come somme di controllo per garantire la correttezza di un messaggio di grandi dimensioni (come un pacchetto software). L'idea in questo caso è molto simile: un messaggio lungo viene ridotto a un blocco molto più piccolo prima di effettuare la firma. Questo significa che "messaggi" molto grandi, come i pacchetti software, possono essere firmati in modo efficiente. Per semplicità, ignoreremo il problema dei messaggi lunghi per il resto del capitolo.

Un'altra domanda importante è: da dove arrivano inizialmente questi lucchetti numerici e queste chiavi numeriche? Abbiamo detto prima che i partecipanti possono sostanzialmente scegliere qualsiasi valore per il proprio lucchetto. I particolari che si nascondono dietro quell'avverbio, "sostanzialmente", purtroppo richiedono un corso universitario di teoria dei numeri, ma, immaginando che non abbiate avuto occasione di studiare la teoria dei numeri, voglio darvi almeno questo assaggio: se la dimensione dell'orologio è un numero primo, allora qualsiasi valore positivo minore di quella dimensione funzionerà come lucchetto. In caso contrario, la situazione è più complicata. Un numero primo è un numero che non ha divisori all'infuori di 1 e di se stesso. 11, la dimensione di orologio usata fin qui in questo capitolo, è in effetti un numero primo.

La scelta del lucchetto quindi è la parte facile, in particolare se la dimensione dell'orologio è un numero primo. Una volta scelto il lucchetto, però, bisogna ancora determinare la chiave numerica corrispondente che lo può aprire. Questo si rivela un problema interessante, che i matematici conoscono da molto tempo. La soluzione in effetti è nota da secoli, e l'idea centrale risale addirittura ancora più indietro: è l'algoritmo di Euclide, una tecnica documentata oltre 2000 anni fa dal matematico greco Euclide. Non abbiamo bisogno però di seguire i particolari della generazione delle chiavi: ci basta sapere che, dato il valore di un lucchetto, il calcolatore può trovare il valore della chiave corrispondente utilizzando una tecnica matematica collaudata che si chiama algoritmo di Euclide.

Se questa spiegazione non vi soddisfa ancora, forse sarete più felici alla nuova svolta drammatica della nostra storia: tutto il metodo "moltiplicativo" a lucchetti e chiavi ha un difetto fondamentale e deve essere abbandonato. Nel prossimo paragrafo useremo un approccio numerico diverso a lucchetti e chiavi, che è poi quello usato nella pratica. Perché allora mi sono preso la briga di spiegarvi un sistema difettoso come quello moltiplicativo? Il motivo principale è che tutti conoscono bene la moltiplicazione, perciò la spiegazione non richiedeva troppe idee nuove tutte insieme; ma c'è anche qualche collegamento affascinante fra il metodo sbagliato della moltiplicazione e quello corretto che considereremo tra poco.

Prima di passare oltre, però, cerchiamo di capire quale sia il difetto del metodo moltiplicativo. Ricordate che i valori dei lucchetti sono *privati* (cioè segreti), mentre i valori delle chiavi sono *pubblici*. Come abbiamo appena visto, un partecipante a uno schema di firma sceglie liberamente una dimensione di orologio (che viene resa pubblica) e un valore di lucchetto (che rimane privato), poi genera il valore della chiave

corrispondente con un computer (mediante l'algoritmo di Euclide, nel caso particolare delle chiavi moltiplicative che abbiamo usato fin qui). La chiave viene conservata in una banca degna di fiducia, e la banca comunica il valore della chiave a chiunque ne faccia richiesta. Il problema di una chiave moltiplicativa è che lo stesso trucco (sostanzialmente l'algoritmo di Euclide) che si usa per generare una chiave da un lucchetto funziona perfettamente anche all'inverso: esattamente la stessa tecnica consente a un computer di generare il valore del lucchetto che corrisponde a un dato valore di chiave! Si vede subito che questo distrugge tutto il meccanismo della firma digitale. Poiché i valori delle chiavi sono pubblici, i valori dei lucchetti che dovrebbero essere segreti invece possono essere calcolati da chiunque. E, una volta conosciuto il valore del lucchetto di qualcuno, si può falsificare la sua firma digitale.

Firma con lucchetti a elevamento a potenza

Aggiorneremo ora il nostro sistema moltiplicativo, che è difettoso, a un sistema di firma digitale, l'RSA, che è effettivamente utilizzato nella pratica. Il nuovo sistema usa un'operazione matematica meno comune, l'*elevamento a potenza*, al posto della moltiplicazione In effetti siamo passati per la stessa successione di passi quando abbiamo cercato di capire la crittografia a chiave pubblica nel Capitolo 4: prima abbiamo studiato un sistema semplice ma difettoso che utilizzava la moltiplicazione, poi abbiamo considerato la versione reale che usa l'elevazione a potenza.

Dunque, se non siete molto a vostro agio con la notazione delle potenze, come 5^9 e 3^4, potete fare un ripasso veloce tornando a leggere a pagina 61. Come ripasso velocissimo, invece: 3^4 ("3 alla quarta potenza" o "3 alla quarta") significa $3 \times 3 \times 3 \times 3$. Abbiamo bisogno di qualche altro termine tecnico. In un'espressione come 3^4, il 4 è l'*esponente* o *potenza* e 3 è la *base*. L'operazione prende il nome di *elevamento a potenza*. Come nel Capitolo 4, combineremo l'elevazione a potenza con l'aritmetica dell'orologio. Tutti gli esempi in questo paragrafo del capitolo useremo un orologio di dimensione 22. Gli unici esponenti che ci serviranno sono 3 e 7, perciò nella Tabella 9.3 trovate i valori di n^3 e n^7, per tutti i valori di n fino a 20 (quando la dimensione dell'orologio è 22).

Tabella 9.3 I valori di n elevato alla terza e alla settima potenza quando la dimensione dell'orologio è 22

n	n^3	n^7	n	n^3	n^7
1	1	1	11	11	11
2	8	18	12	12	12
3	5	9	13	19	7
4	20	16	14	16	20
5	15	3	15	9	5
6	18	8	16	4	14
7	13	17	17	7	19
8	6	2	18	2	6
9	3	15	19	17	13
10	10	10	20	14	4

Verifichiamo un paio di voci della tabella, per essere sicuri che siano sensate. Guardate la riga che corrisponde a $n = 4$. Se non usassimo l'aritmetica dell'orologio, potremmo stabilire subito che $n^3 = 4 \times 4 \times 4 = 64$. *Applicando invece la dimensione di orologio 22, vediamo che 22 sta in 64 due volte (il che ci dà 44) con resto di 20. Questo spiega il 20 nella colonna* n^3. Analogamente, si può calcolare che $4^7 = 16.384$ (potete fidarvi di me

questa volta), che diviso per 22 dà resto 16 (22 × 744 = 16.368, nel caso foste interessati). Questo dunque spiega il 16 nella colonna n^7.

Ora siamo finalmente pronti a vedere una genuina firma digitale in azione. Il sistema funziona esattamente come nel metodo moltiplicativo già visto, con una eccezione: invece di chiudere e aprire i messaggi con la moltiplicazione, useremo l'elevazione a potenza. Come prima, Ravi sceglie inizialmente una dimensione di orologio che verrà resa pubblica, in questo caso 22. Poi seleziona un valore di lucchetto segreto, che potrà essere qualsiasi numero minore della dimensione dell'orologio (con qualche precisazione, che andremo a vedere poi). Nel nostro esempio, Ravi sceglie 3 come lucchetto. Poi usa un computer per determinare il valore della chiave corrispondente, per il dato lucchetto e la data dimensione dell'orologio. Vedremo qualche altro particolare poi, ma l'unico fatto importante è che un computer può calcolare facilmente la chiave partendo da lucchetto e dimensione dell'orologio, applicando una ben nota tecnica matematica. Qui si dà il caso che il valore della chiave, corrispondente al valore scelto per il lucchetto (3), sia 7.

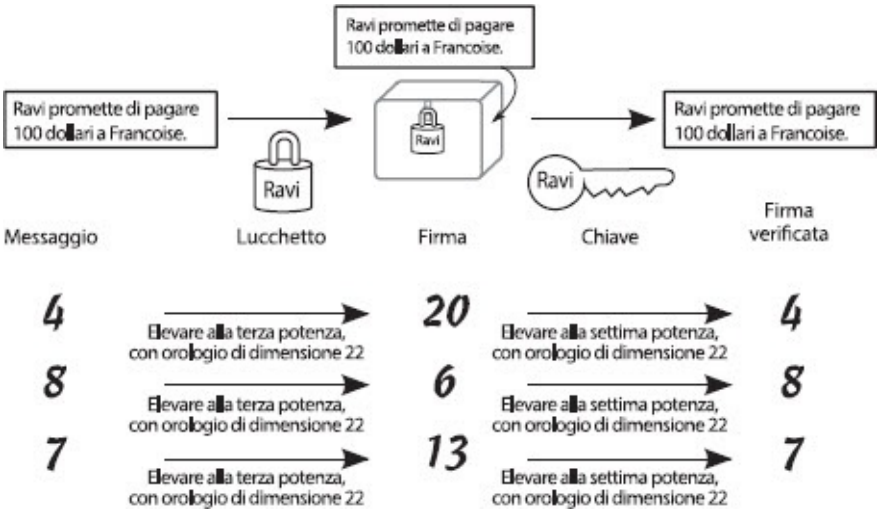


Figura 9.10 Chiusura e apertura di messaggi con l'elevamento a potenza.

La Figura 9.10 mostra qualche esempio concreto di come Ravi possa firmare messaggi, e di come altri possano aprire le firme per controllarle. Se il messaggio è "4", la firma è "20": la si ottiene elevando il messaggio a potenza con il lucchetto come esponente. Perciò si deve calcolare 4^3, che dà 20 (tenendo conto della dimensione dell'orologio: non dimenticate che tutti questi calcoli si possono facilmente verificare con la Tabella 9.3). Ora, quando Francoise vuole verificare la firma digitale "20" di Ravi, prima va alla banca per ottenere da una fonte autorevole i valori di dimensione dell'orologio e chiave di Ravi. (La banca è la stessa di prima, solo conserva numeri diversi: vedi la Figura 9.9.) Poi Francoise prende la firma, la eleva a potenza usando la chiave come esponente e applica la dimensione dell'orologio: questo dà $20^7 = 4$, sempre in base alla Tabella 9.3. Se il risultato è uguale al messaggio originale (e in questo caso lo è), la firma è autentica. La Figura 9.10 mostra calcoli simili per i messaggi "8" e "7".

La Tabella 9.4 mostra nuovamente il procedimento, ma questa volta mettendo in evidenza la verifica della firma. I primi due esempi in questo caso sono identici a quelli della Figura 9.10 (i messaggi "4" e "8", rispettivamente) e hanno firme autentiche; il terzo esempio usa il messaggio "8" e la firma "9". Aprendo, con la chiave e la dimensione di orologio date, si ottiene $9^7 = 15$, che non corrisponde al messaggio originale. La firma,

perciò, è falsa.

Tabella 9.4 Come scoprire una firma digitale falsa nel caso dell'elevamento a potenza. Questi esempi usano un lucchetto di valore 3, una chiave di valore 7 e una dimensione di orologio 22. Le prime due firme sono autentiche, la terza invece è falsa

Messaggio	**Firma digitale** (Per avere una firma autentica, elevare a potenza il messaggio, usando il valore 3 del lucchetto come esponente. Per falsificare, scegliere un numero a caso.)	**Firma aperta** (Per aprire la firma, elevare a potenza con il valore della chiave, 7, come esponente.)	**Il messaggio corrisponde all'originale?**	**Firma falsa?**
4	20	4	Sì	No
8	6	8	Sì	No
8	9	15	No	Sì

Questo schema basato sull'elevamento a potenza, con lucchetti e chiavi che fungono da esponenti, è denominato sistema di firma digitale RSA, dai nomi dei suoi inventori (Ronald Rivest, Adi Shamir e Leonard Adleman), che l'hanno pubblicato negli anni Settanta. Può sembrarvi vagamente familiare, perché abbiamo già incontrato l'acronimo RSA nel Capitolo 4, parlando di crittografia a chiave pubblica. In effetti RSA è sia uno schema di crittografia a chiave pubblica sia uno schema di firma digitale, e non è una coincidenza perché esiste una relazione teorica profonda fra questi due tipi di algoritmi. In questo capitolo abbiamo esplorato solo l'aspetto di RSA relativo alla firma digitale, ma probabilmente avrete notato alcune somiglianze significative con le idee del Capitolo 4.

I particolari di come si scelgono le dimensioni dell'orologio, i lucchetti e le chiavi nel sistema RSA sono proprio affascinanti, ma non sono necessari per capire il metodo in generale. Il punto più importante è che in questo sistema un partecipante può calcolare facilmente una chiave adeguata, una volta che è stato scelto il valore del lucchetto, ma per chiunque altro è impossibile invertire il procedimento: se si conoscono chiave e dimensione di orologio utilizzati da qualcun altro, non si può calcolare il corrispondente valore del lucchetto. Questo risolve il difetto che abbiamo visto prima nel sistema moltiplicativo.

Per lo meno, gli informatici pensano che lo risolva, ma nessuno lo sa per certo. Il problema, se RSA sia veramente sicuro, è una delle questioni più affascinanti e più dibattute di tutta l'informatica. Da un lato dipende sia da un antico problema matematico non risolto sia da un tema caldo molto più recente, che si trova all'intersezione fra fisica e informatica. Il problema matematico è quello della *scomposizione in fattori interi*; il tema di ricerca caldo è il *calcolo quantistico*. Esploreremo entrambi questi aspetti della sicurezza RSA ma, prima, dobbiamo capire meglio che cosa voglia veramente dire che un sistema di firma digitale come RSA è "sicuro".

La sicurezza di RSA

La sicurezza di qualsiasi sistema di firma digitale si riduce, alla fine, alla domanda: "Un truffatore può falsificare la mia firma?". Nel caso di RSA questo diventa: "Il truffatore può calcolare il valore del mio lucchetto privato, conoscendo la dimensione del mio orologio e il valore della mia chiave, che sono pubblici?". Sarete un po' scossi scoprendo che la risposta, molto semplice, a questa domanda è "Sì!". In effetti, lo sapevate già: è *sempre* possibile scoprire il valore del lucchetto di qualcuno andando per tentativi. In fin dei conti

abbiamo un messaggio, la dimensione dell'orologio e una firma digitale. Sappiamo che il valore del lucchetto è minore della dimensione dell'orologio, perciò possiamo provare tutti i possibili valori fino a che non troveremo quello che produce la firma corretta. È solo una questione di elevare a potenza il messaggio utilizzando ogni volta un diverso valore di lucchetto come esponente. Il fatto è che, nella pratica, gli schemi RSA usano dimensioni di orologio assolutamente enormi, nell'ordine delle migliaia di cifre, perciò anche con i più veloci supercomputer esistenti sarebbero necessarie migliaia di miliardi di anni per mettere alla prova tutti i possibili valori di lucchetto. Non siamo interessati a sapere se un malintenzionato possa calcolare il valore del lucchetto con uno strumento qualsiasi: vogliamo sapere se può farlo in modo *abbastanza efficiente* da costituire una minaccia pratica. Se il suo metodo di attacco migliore è il procedere per tentativi (*a forza bruta*, come dicono anche gli informatici) possiamo sempre scegliere un orologio di dimensioni tanto grandi da rendere impossibile nella pratica l'attacco. Se invece l'avversario ha una tecnica significativamente più veloce del procedere per tentativi, potremmo essere nei pasticci.

Per esempio, torniamo al sistema di lucchetto e chiave moltiplicativi: abbiamo visto che chi firma può scegliere un lucchetto e poi calcolare la chiave da quello utilizzando l'algoritmo di Euclide. Il difetto era che un possibile malintenzionato non aveva bisogno di ricorrere alla forza bruta per invertire il procedimento: l'algoritmo di Euclide può essere utilizzato anche per calcolare il lucchetto data la chiave, e questo algoritmo è enormemente più efficiente della forza bruta. Per questo il metodo moltiplicativo è considerato non sicuro.

Il collegamento fra RSA e scomposizione in fattori

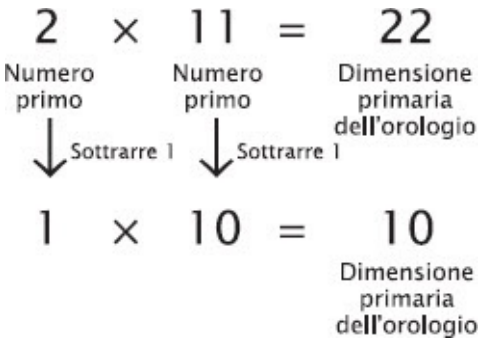
Ho promesso di svelare un collegamento fra la sicurezza di RSA e un antico problema matematico, quello della scomposizione in fattori primi. Ci serve però ancora qualche dettaglio su come viene scelta la dimensione dell'orologio di RSA.

Innanzitutto, ricordiamo la definizione di *numero primo*: è un numero che non ha divisori al di fuori di 1 e di se stesso. Per esempio, 31 è un numero primo perché 1×31 è l'unico modo di ottenere 31 come prodotto di due numeri (moltiplicazione e divisione sono operazioni inverse: i due fattori, 1 e 31, che permettono di ottenere il prodotto 31 sono, inversamente, gli unici divisori di 31, cioè gli unici numeri interi per cui 31 può essere diviso esattamente, con resto zero). 33 invece non è primo, perché si può ottenere anche come prodotto di 3 e 11: $3 \times 11 = 33$.

Ora siamo pronti per esaminare come il nostro vecchio amico Ravi possa generare una dimensione di orologio per RSA. La prima cosa che fa Ravi è scegliere due numeri primi molto grandi. Normalmente questi numeri saranno di centinaia di cifre, ma come al solito useremo numeri piccoli. Diciamo che Ravi sceglie 2 e 11 come suoi numeri primi. Poi li moltiplica fra loro, ottenendo la dimensione dell'orologio. Così, nel nostro esempio, la dimensione dell'orologio è $2 \times 11 = 22$. Come sappiamo, la dimensione dell'orologio può essere resa pubblica insieme con la chiave scelta da Ravi. I due fattori primi della dimensione dell'orologio, però (e questo è il punto cruciale) restano segreti, li conosce solo Ravi. La matematica alla base di RSA dà a Ravi il modo di usare questi due fattori primi per calcolare un lucchetto da una chiave e viceversa.

I particolari di questo metodo sono descritti nel riquadro della pagina successiva, ma non sono importanti per il nostro obiettivo principale. Tutto quello di cui dobbiamo renderci conto è che gli avversari di Ravi non possono calcolare il suo lucchetto segreto utilizzando le informazioni pubbliche (dimensione di orologio e chiave). Ma *se* i suoi avversari conoscessero anche i due fattori primi della dimensione dell'orologio, potrebbero facilmente calcolare il valore del lucchetto segreto. In altre parole, gli avversari di Ravi possono falsificare la sua firma se possono *scomporre in fattori primi* la dimensione dell'orologio. (Ovviamente, potrebbero esserci anche altri modi di forzare RSA: un procedimento efficiente di scomposizione in fattori primi della dimensione dell'orologio è solo uno dei possibili metodi di attacco.)

Nel nostro esempio semplice, scomporre in fattori la dimensione dell'orologio (e quindi forzare lo schema di firma digitale) è banalmente facile: tutti sanno che 22 = 2 × 11. Quando però la dimensione dell'orologio è lunga centinaia o migliaia di cifre, determinarne i fattori si rivela un problema estremamente difficile. In effetti, nonostante questo problema della scomposizione in fattori primi sia stato studiato da secoli, nessuno ha mai trovato un metodo generale di risolverlo che sia abbastanza efficiente da compromettere una tipica dimensione di orologio di RSA.



Ravi sceglie due numeri primi (2 e 11, nel nostro esempio semplicistico) e li moltiplica per ottenere la dimensione del suo orologio (22). La chiameremo la dimensione "primaria" dell'orologio, per ragioni che saranno presto chiare. Poi Ravi sottrae uno da ciascuno dei due numeri primi originari e moltiplica fra loro i due numeri così ottenuti. Il risultato è la dimensione "secondaria" dell'orologio di Ravi. Nel nostro esempio, i due nuovi numeri sono 1 e 10 (ottenuti sottraendo 1 da 2 e da 11 rispettivamente) e quindi la dimensione secondaria è 1 × 10 = 10. A questo punto, incontriamo un collegamento molto gratificante con il pur difettoso sistema moltilicativo di lucchetto e chiave descritto in precedenza: Ravi sceglie un lucchetto e una chiave in base al sistema moltiplicativo, ma utilizzando la dimensione secondaria dell'orologio al posto della primaria. Supponiamo che Ravi scelga 3 come lucchetto. Se si usa la dimensione secondaria dell'orologio, 10, la chiave moltiplicativa corrispondente è 7. Si può verificare rapidamente: il messaggio "8" moltiplicato per il lucchetto dà 8 × 3 = 24, ovvero "4" in base all'orologio di dimensione 10. Aprendo "4" con la chiave si ottiene 4 × 7 = 28, che è "8" in base alla stessa dimensione di orologio – esattamente il messaggio originale.

Adesso Ravi ha completato il suo lavoro: prende lucchetto e chiave moltiplicativi appena scelti e li usa direttamente come lucchetto e chiave nel sistema RSA a elevamento a potenza. Ovviamente, saranno utilizzati come esponenti tenendo conto della dimensione

primaria dell’orologio, 22.

Gli sporchi dettagli della generazione di dimensione d’orologio, lucchetto e chiave in RSA.

La storia della matematica è costellata di problemi irrisolti che hanno affascinato i matematici per le loro sole doti estetiche, motivando ricerche profonde nonostante l’assenza di qualsiasi applicazione pratica. Sorprendentemente, molti di questi problemi affascinanti ma fin qui apparentemente inutili si sono poi rivelati di grande importanza pratica; in qualche caso la loro importanza è stata scoperta solo dopo che il problema era stato studiato per secoli.

La scomposizione in fattori primi è proprio uno di questi problemi. Le prime ricerche serie sembra siano state quelle di Fermat e Mersenne nel diciassettesimo secolo. Eulero e Gauss (due fra le figure più grandi in tutta la storia della matematica) hanno dato dei contributi nei secoli successivi, e molti hanno continuato il loro lavoro. Ma solo con la scoperta della crittografia a chiave pubblica negli anni Settanta la difficoltà di scomporre in fattori numeri molto grandi è diventata il fiore all’occhiello di un’applicazione pratica. Come ora sapete, chiunque scopra un algoritmo efficiente per scomporre in fattori numeri molto grandi sarà in grado di falsificare tutte le firme digitali che vorrà!

Prima che vi preoccupiate troppo, debbo specificare che dagli anni Settanta sono stati inventati molti altri sistemi di firma digitale. Ciascuno dipende dalla difficoltà di qualche problema matematico fondamentale, ma ciascuno dipende da un diverso problema. Perciò la scoperta di un algoritmo efficiente di scomposizione in fattori primi permetterà di forzare solo i sistemi di tipo RSA.

Invece, gli informatici continuano a essere perplessi per una particolarità di tutti questi sistemi: di nessuno è stato possibile *dimostrare* che sia sicuro. Ciascuno dipende da qualche problema matematico evidentemente difficile e lungamente studiato, ma in nessun caso qualcuno è stato in grado di dimostrare che non possa esistere una soluzione efficiente. Così, anche se gli esperti lo considerano estremamente improbabile, è possibile in linea di principio che qualcuno di questi sistemi crittografici o di firma digitale venga forzato, in qualsiasi momento.

Il collegamento fra RSA e i computer quantistici

Devo ancora spiegare il collegamento fra RSA e un argomento di ricerca molto caldo, l’informatica quantistica. Per farlo, bisogna prima accettare il seguente fatto fondamentale. Nella meccanica quantistica, il moto degli oggetti è governato da *probabilità*, a differenza di quel che accade con le leggi deterministiche della fisica classica. Perciò, se si costruisce un computer con componenti soggette a effetti quantomeccanici, i valori che calcolerà saranno determinati da probabilità, invece che dalle successioni assolutamente certe di 0 e 1 prodotte da un computer classico. Lo si può vedere anche in un altro modo: un computer quantistico memorizza molti valori diversi allo stesso tempo; i diversi valori hanno probabilità diverse, ma finché non si costringe il computer a dare una risposta definitiva, tutti i valori esistono simultaneamente. Questo conduce alla possibilità che un computer quantistico possa calcolare molte risposte possibili, tutte tra loro diverse, contemporaneamente. Perciò, per certi tipi speciali di problemi, si potrebbe usare un metodo a “forza bruta” che tenti simultaneamente tutte le soluzioni possibili!

Questo non vale solo per certi tipi di problemi, ma la scomposizione in fattori primi è proprio uno dei compiti che si possono svolgere con efficienza enormemente maggiore con i computer quantistici che con quelli classici. Perciò, se si potesse costruire un computer quantistico in grado di manipolare numeri di migliaia di cifre, si potrebbero falsificare le firme RSA come si è spiegato prima: scomporre in fattori la dimensione pubblica dell'orologio, usare i fattori per determinare la dimensione secondaria dell'orologio e poi usare questa per calcolare il lucchetto privato dalla chiave pubblica.

Mentre scrivo queste parole nel 2011, l'informatica quantistica è molto lontana dall'applicazione pratica. I ricercatori sono riusciti a costruire veri computer quantistici, ma fin qui la scomposizione in fattori più complessa eseguita da un computer quantistico è $15 = 3 \times 5$: siamo ben lontani dalla possibilità di scomporre in fattori una dimensione di orologio RSA di migliaia di cifre! E ci sono poi problemi pratici formidabili da risolvere per poter costruire grandi computer quantistici. Perciò nessuno sa quando, e nemmeno se, i computer quantistici diventeranno abbastanza potenti da forzare il sistema RSA una volta per tutte.

Firme digitali nella pratica

All'inizio del capitolo, abbiamo visto che gli utenti finali come voi e me non hanno molto bisogno di apporre firme digitali. Alcuni utenti esperti firmano per esempio i messaggi di posta elettronica ma, per la maggior parte di noi, l'uso principale delle firme digitali sta nella verifica di contenuti scaricati. L'esempio più ovvio è quando si scarica un nuovo software: se il software è firmato, il computer "apre" la firma utilizzando la chiave pubblica del firmatari e confronta il risultato con il "messaggio" del firmatario, in questo caso il software stesso. (Come abbiamo già detto, nella pratica il software viene ridotto a un messaggio molto più piccolo, un cosiddetto *hash* sicuro, prima della firma.) Se la firma aperta corrisponde al software, si ottiene un messaggio incoraggiante; altrimenti si ottiene un avvertimento più severo: due esempi sono stati proposti nella Figura 9.1.

Come ho continuato a mettere in evidenza, tutti i nostri sistemi hanno bisogno di un qualche tipo di "banca" sicura in cui conservare le chiavi pubbliche e le dimensioni d'orologio di chi firma. Per fortuna, come probabilmente avrete notato, non c'è bisogno di fare un viaggio fino a una banca reale ogni volta che si deve scaricare del software. Nella vita reale, le organizzazioni fidate che conservano le chiavi pubbliche sono chiamate *autorità di certificazione*. Tutti questi enti gestiscono server che possono essere contattati elettronicamente per scaricare informazioni sulle chiavi pubbliche. Così, quando la vostra macchina riceve una firma digitali, questa sarà accompagnata da informazioni che precisano quale autorità di certificazione può certificare la chiave pubblica del firmatario.

Probabilmente avrete già visto un problema: certo, il vostro computer può andare avanti e verificare la firma con l'autorità di certificazione indicata, ma come si può avere fiducia nell'autorità stessa? Tutto quello che abbiamo fatto è prendere il problema della verifica dell'identità di una organizzazione (quella che vi ha inviato il software, diciamo la NanoSoft.com) e trasformarlo nel problema di verificare l'identità di un'altra organizzazione (l'autorità di certificazione, chiamiamola TrustMe Inc.). Che ci crediate o no, il problema normalmente viene risolto dall'autorità di certificazione (TrustMe Inc.) rimandandovi a un'altra autorità di certificazione (diciamo la PleaseTrustUs Ltd.) per una verifica, attraverso firma digitale. Questo tipo di catena può essere esteso all'infinito, ma

ci troveremo sempre con lo stesso problema: come possiamo fidarci dell'organizzazione che sta alla fine della catena? La risposta, come si vede nella Figura 9.11, è che certe organizzazioni sono state ufficialmente designate come autorità di certificazione *radice* (*root CA*): fra le più note vi sono VeriSign, GlobalSign e GeoTrust. Le informazioni per contattare (compresi indirizzi Internet e chiavi pubbliche) varie autorità radice sono preinstallate nel browser quando lo si acquista, e in questo modo la catena della fiducia per le firme digitali si àncora in un punto di partenza degno di fiducia.

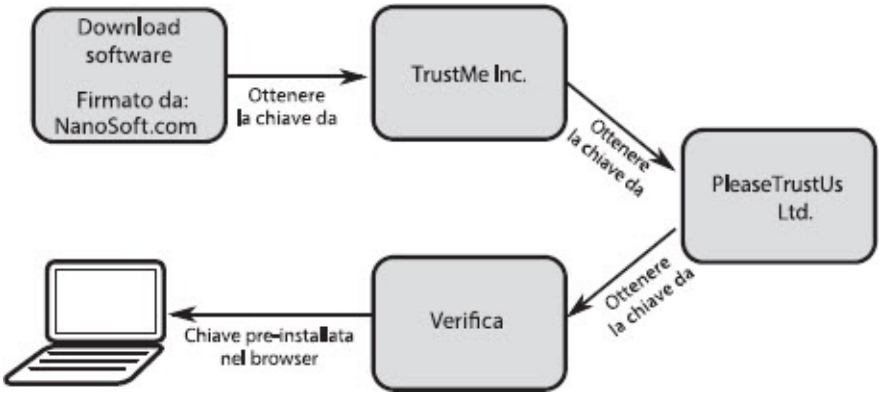


Figura 9.11 Una catena di fiducia per ottenere le chiavi necessarie alla verifica delle firme digitali.

Un paradosso risolto

All'inizio di questo capitolo, ho detto che l'espressione stessa "firma digitale" può essere considerata un ossimoro: qualsiasi cosa sia in forma digitale può essere copiata, mentre dovrebbe essere impossibile copiare una firma. Come si è risolto il paradosso? La risposta è che una firma digitale dipende sia da un segreto noto solo a chi firma e dal messaggio che viene firmato. Il segreto (che per tutto il capitolo abbiamo identificato con un lucchetto) rimane lo stesso per tutti i messaggi firmati da un particolare ente, ma la firma è diversa per ciascun messaggio. Quindi il fatto che chiunque possa copiare facilmente la firma è irrilevante: la firma non può essere trasferita a un altro messaggio, perciò il semplice copiarla non produce una falsificazione.

La risoluzione del paradosso non è semplicemente un'idea elegante e astuta. Le firme digitali sono anche di grandissima importanza pratica: senza di esse, l'Internet che conosciamo non esisterebbe. Si potrebbero comunque scambiare dati in modo sicuro con la crittografia, ma sarebbe di gran lunga più difficile verificare la *fonte* di qualsiasi dato ricevuto. Questa combinazione di un'idea profonda con conseguenze pratiche così estese significa che le firme digitali sono, senza alcun dubbio, uno dei risultati più spettacolari dell'informatica.

---

1 Tr. it. di Silvio Spaventa Filippi, Sonzogno, Milano, 1936 (edizione elettronica in www.liberliber.it).

CAPITOLO 10

Che cosa è calcolabile?

Permettetemi di ricordarvi alcuni problemi delle macchine elaboratrici.

Richard Feynman (Premio Nobel per la Fisica nel 1965)

Abbiamo visto diversi algoritmi brillanti, potenti ed eleganti, algoritmi che trasformano il nudo metallo di un computer in un genio sulla punta delle nostre dita. In effetti, sarebbe naturale domandarsi, stimolati dalla retorica rapsodica dei capitoli precedenti se ci sia *qualcosa* che i computer non possono fare per noi. La risposta è assolutamente chiara se ci limitiamo a considerare quello che i computer possono fare *oggi*: ci sono moltissime attività utili (soprattutto quelle che coinvolgono qualche forma di intelligenza artificiale) che al momento i computer non sanno compiere bene. Per esempio, una buona traduzione fra lingue come l'inglese e il cinese; il controllo automatico di un veicolo per poter viaggiare rapidamente in modo sicuro in un ambiente di città pieno di traffico; e (da insegnante, questo per me è un tema importante) valutare il lavoro degli studenti.

Eppure, come abbiamo visto, spesso è sorprendente quello che un buon algoritmo può fare. Forse domani qualcuno inventerà un algoritmo che guiderà alla perfezione un'automobile oppure saprà valutare in modo eccellente il lavoro dei miei studenti. Questi sembrano effettivamente problemi difficili, ma sono di una difficoltà insuperabile? C'è qualche problema che sia tanto difficile che nessuno possa mai inventare un algoritmo per risolverlo? In questo capitolo vedremo che la risposta è un deciso "sì": *ci sono* problemi che non potranno mai essere risolti dai computer. Questo fatto profondo, che alcune cose siano "calcolabili" e altre no, costituisce un interessante contraltare ai molti trionfialgoritmici visti nei capitoli precedenti. Per quanti algoritmi brillanti si inventeranno nel futuro, ci saranno sempre problemi le cui risposte sono "non calcolabili".

L'esistenza di problemi non calcolabili è notevole già di suo, ma ancor di più lo è la storia della loro scoperta. L'esistenza di problemi simili era nota già prima che venissero costruiti i primi computer! Due matematici, uno americano e l'altro inglese, hanno scoperto in modo indipendente problemi non calcolabili alla fine degli anni Trenta, parecchi anni prima che venissero costruiti dei veri computer nel corso della Seconda guerra mondiale. L'americano era Alonzo Church, il cui lavoro pionieristico sulla teoria della computazione resta fondamentale per molti aspetti dell'informatica. L'inglese era niente meno che Alan Turing, che è considerato solitamente la figura più importante per la nascita della *computer science*. Le ricerche di Turing hanno spaziato su tutto lo spettro delle idee relative al calcolo, dalle complessità della teoria matematica alla profondità della filosofia alla praticità dell'ingegneria. In questo capitolo, seguiremo le orme di Church e Turing in un viaggio che alla fine dimostrerà l'impossibilità di usare un

computer per una particolare attività. Il viaggio inizia nel prossimo paragrafo, con una discussione di bachi e crash.

Bachi, crash e affidabilità del software

L'affidabilità del software è aumentata enormemente in anni recenti, ma sappiamo tutti che ancora non conviene dare per scontato che un programma funzionerà sempre correttamente. Ogni tanto anche software di alta qualità, ben scritto, può fare qualcosa che non avrebbe dovuto fare. Nei casi peggiori il software andrà in tilt (quello che si definisce un *crash*) e vi farà perdere i dati o i documenti su cui stavate lavorando (o il videogioco con cui stavate giocando, cosa molto frustrante, come posso assicurarvi per esperienza). Ma come può testimoniare chiunque abbia avuto a che fare con i computer domestici degli anni Ottanta e Novanta, i programmi andavano in tilt molto più spesso di quel che succeda nel Ventunesimo secolo. Questo miglioramento ha molti motivi, ma principalmente è dovuto ai grandi progressi fatti nella progettazione di strumenti di verifica automatizzata del software. In altre parole, una volta che ha scritto un programma complesso di grandi dimensioni, una squadra di programmatori può utilizzare uno strumento automatico per verificare il nuovo software e trovare problemi che lo possano mandare in tilt. E questi strumenti automatizzati stanno diventando sempre più "bravi" a identificare i potenziali errori.

Perciò verrebbe naturale chiedersi: gli strumenti automatici di verifica del software arriveranno mai al punto da poter identificare tutti i potenziali problemi in tutti i programmi per computer? Sarebbe bello, perché eliminerebbe completamente la possibilità che un programma vada mai in tilt. La cosa notevole che scopriremo in questo capitolo è che questo nirvana del software non sarà mai raggiungibile: si può dimostrare che è impossibile che qualsiasi strumento di verifica del software possa identificare tutte le possibili cause di crash in tutti i programmi.

Vale la pena dire qualcosa di più su quel "dimostrare che è impossibile". Nella maggior parte delle scienze, come la fisica e la biologia, gli scienziati avanzano ipotesi sul modo in cui si comportano certi sistemi, e conducono esperimenti per vedere se le loro ipotesi sono corrette. Poiché però gli esperimenti hanno sempre un certo grado di incertezza, non è mai possibile essere sicuri al 100 per cento che le ipotesi fossero corrette, anche dopo che un esperimento abbia dato proprio i risultati sperati. In matematica e in informatica invece è possibile raggiungere, per certi risultati, una certezza del 100 per cento. Se si accettano gli assiomi fondamentali della matematica (come, per esempio, 1 + 1 = 2), la catena dei ragionamenti deduttivi utilizzati dai matematici dà la certezza assoluta che vari altri enunciati siano veri (per esempio: "qualsiasi numero che finisca con un 5 è divisibile per 5"). Questo tipo di ragionamento non coinvolge i computer: con carta e matita solamente, un matematico può dimostrare affermazioni indiscutibili.

Perciò, in informatica, quando diciamo che si può dimostrare che X è impossibile non vogliamo dire semplicemente che X risulta molto difficile, o che forse è impossibile realizzarlo in pratica. Vogliamo dire che è certo al 100 per cento che X non si potrà mai realizzare, perché qualcuno lo ha dimostrato con una catena di ragionamenti matematici deduttivi. Così, per esempio, "si può dimostrare che è impossibile che un multiplo di 10 termini con la cifra 3". Un altro esempio sarà la conclusione di questo capitolo: si può dimostrare che è impossibile che un programma di verifica del software identifichi tutte le

possibili cause di crash in tutti i programmi per computer.

Dimostrare che qualcosa non è vero

La nostra dimostrazione che i programmi di identificazione dei crash sono impossibili userà una tecnica che i matematici chiamano *dimostrazione per assurdo*. Anche se i matematici sostengono di avere la paternità di questa tecnica, è in realtà qualcosa che si usa normalmente nella vita quotidiana, spesso senza nemmeno pensarci. Vediamolo con un esempio molto semplice.

Per cominciare, dobbiamo essere d'accordo su questi due fatti, che non verrebbero contestati nemmeno dai più revisionisti fra gli storici:

1. La Guerra civile americana è avvenuta negli anni Sessanta dell'Ottocento.

2. Abramo Lincoln era presidente degli Stati Uniti durante la Guerra civile.

Supponiamo ora che io sostenga: "Abramo Lincoln è nato nel 1520". Vero o falso? Anche se non sapete nulla di Abramo Lincoln, a parte i due fatti elencati sopra, come potete stabilire rapidamente che la mia affermazione è falsa?

Con ogni probabilità, seguirete una catena di ragionamenti simile a questa: (1) nessuno vive più di 150 anni, perciò se Lincoln fosse nato nel 1520 dovrebbe essere morto al più tardi nel 1670; (ii) Lincoln era presidente durante la Guerra civile, perciò la Guerra civile deve essere stata prima della sua morte, cioè prima del 1670; (iii) ma questo è impossibile, perché tutti concordano che la Guerra civile si è verificata negli anni Sessanta dell'Ottocento; (iv) *perciò*, Lincoln non può essere nato nel 1520.

Esaminiamo più attentamente questo ragionamento. Perché è valida la conclusione che l'enunciato iniziale era falso? Perché abbiamo dimostrato che quell'enunciato contraddice qualche altro fatto che si sa essere vero. Specificamente, abbiamo dimostrato che l'enunciato iniziale comporterebbe che la Guerra civile si fosse verificata prima del 1670 – il che contraddice il fatto noto che abbia avuto luogo negli anni Sessanta dell'Ottocento.

La dimostrazione per assurdo è una tecnica estremamente importante, perciò vediamone un esempio un po' più matematico. Supponiamo che sostenga: "In media, un cuore umano batte circa 6000 volte in 10 minuti". Sarà vero o falso? Avrete subito qualche dubbio, ma come dimostrare che è falso? Dedicate qualche secondo ad analizzare il vostro processo di pensiero prima di continuare nella lettura.

Anche in questo caso usiamo una dimostrazione per assurdo. Innanzitutto, partiamo supponendo che l'enunciato sia vero: il cuore umano in media batte 6000 volte in 10 minuti. Se fosse vero, quante volte batterebbe in un minuto? In media, avremmo 6000 diviso per 10, cioè 600 battiti al minuto. Ora, non è necessario essere medici per sapere che questo è un numero di pulsazioni molto superiore a quello normale, che è fra i 50 e i 150 battiti al minuto. Perciò l'enunciato originale contraddice un fatto noto e deve essere falso: *non* è vero che il cuore umano batte circa 6000 volte in 10 minuti.

Per usare una terminologia più astratta, la dimostrazione per assurdo può essere presentata in questo modo: supponiamo che sospettiate che un enunciato *S* sia falso, ma che vogliate dimostrare la sua falsità al di là di ogni dubbio. In primo luogo, si assume che *S* sia vero. Applicando qualche ragionamento, si arriva a stabilire che di conseguenza qualche altro enunciato, chiamiamolo *T*, deve essere vero. Se però si sa che *T* è falso si è

giunti a una contraddizione. Questo dimostra che l'ipotesi iniziale (*S*) deve essere falsa.

Un matematico lo direbbe molto più sinteticamente: "*S* implica *T*, ma *T* è falso, perciò *S* è falso". Questa è la dimostrazione per assurdo in estrema sintesi. La tabella seguente mostra come collegare questa versione astratta della dimostrazione per assurdo ai due esempi precedenti.

	Primo esempio	**Secondo esempio**
S (enunciato originale)	Lincoln è nato nel 1520	Il cuore umano batte 6000 volte in 10 minuti
T (implicato da S, ma di cui si sa che è falso)	La Guerra civile si è verificata prima del 1670	Il cuore umano batte 600 volte in 1 minuto
Conclusione: S è falso	Lincoln non è nato nel 1520	Il cuore umano non batte 6000 volte in 10 minuti

Per ora la nostra deviazione nelle dimostrazioni per assurdo è finita. L'obiettivo del capitolo sarà dimostrare, per assurdo, che un programma che identifica tutti i possibili crash in altri programmi non può esistere. Prima di procedere verso il traguardo finale, dobbiamo fare la conoscenza di alcuni concetti interessanti in merito ai programmi per computer.

Programmi che analizzano altri programmi

I computer seguono pedissequamente le istruzioni nei loro programmi. Lo fanno in modo totalmente deterministico, perciò l'output di un programma è esattamente lo stesso ogni volta che lo si esegue. Giusto? O sbagliato? In effetti, non vi ho dato abbastanza informazioni per rispondere. È vero che certi programmi semplici producono sempre esattamente lo stesso output ogni volta che vengono eseguiti, ma la maggior parte dei programmi che usiamo ogni giorno hanno un aspetto diverso ogni volta che li eseguiamo. Pensate al programma di elaborazione testi che preferite: lo schermo appare sempre nello stesso modo quando lo lanciate? Ovviamente no, dipende dal documento che avete aperto. Se uso Microsoft Word per aprire il file "address-list.docx", lo schermo mostrerà un elenco di indirizzi che ho memorizzato. Se uso Microsoft Word per aprire il file "bank-letter.docx" vedrà il testo di una lettera che ho scritto ieri alla mia banca. (Se il ".docx" vi risulta misterioso, leggete il riquadro della pagina seguente a proposito dei suffissi dei file.)

Sia ben chiaro: in entrambi i casi eseguo esattamente lo stesso programma, che è Microsoft Word. Solo gli *input* sono diversi nei vari casi. Non fatevi ingannare dal fatto che tutti i sistemi operativi moderni consentono di mandare in esecuzione un programma facendo un doppio clic su un documento. Questa è solo una comodità che vi ha fornito il costruttore del computer o del sistema operativo (con tutta probabilità Apple o Microsoft). Quando fate un doppio clic su un documento, va in esecuzione un determinato programma e quel programma usa il documento come input. L'output del programma è quello che vedete sullo schermo, e naturalmente dipende da quale sia il documento che avete selezionato.

In tutto questo capitolo, userò nomi di file come "abcd.txt". La parte dopo il punto è il "suffisso" o "estensione" del nome del file, in questo caso l'estensione è "txt". La maggior parte dei sistemi operativi usa l'estensione di un nome di file per stabilire che

tipo di dati sia contenuto nel file. Per esempio, un file ".txt" normalmente contiene puro testo, un file ".html" normalmente contiene una pagina web, e un file ".docx" contiene un documento di Microsoft Word. Alcuni sistemi operativi nascondono queste estensioni per impostazione predefinita, perciò potreste non averli mai visti a meno che non abbiate attivato la funzione "nascondi le estensioni dei file". Con una veloce ricerca nel Web o nella Guida del vostro sistema operativo potete trovare le istruzioni per visualizzare le estensioni dei file.

Qualche dettaglio tecnico sulle estensioni dei nomi di file.

In realtà, input e output dei programmi sono un po' più complessi. Per esempio, quando fate clic su un menu o scrivete in un programma, gli fornite input ulteriori e, quando salvate un documento o qualche altro file, il programma crea un output ulteriore. Per amor di semplicità, immaginiamo che i programmi accettino un solo input, che è un file memorizzato nel computer; immaginiamo poi anche che i programmi producano esattamente un output, che è una finestra grafica sul monitor.

Purtroppo la comodità del doppio clic sui file oscura una questione importante. Il vostro sistema operativo utilizza vari trucchi per indovinare quale programma vogliate eseguire quando fate un doppio clic su un file. È importante però sapere che si può aprire *qualsiasi* file utilizzando *qualsiasi* programma. In altri termini, si può eseguire qualsiasi programma usando qualsiasi file come input. Come? Il riquadro della pagina seguente elenca vari modi in cui potete farlo. Questi metodi non funzioneranno su tutti i sistemi operativi, o su tutti i tipi di file di input: sistemi operativi diversi lanciano i programmi in modo diverso e a volte limitano la scelta del file di input per ragioni di sicurezza. Comunque, vi incoraggio a fare qualche esperimento per qualche minuto sul vostro computer, per convincervi che potete eseguire il vostro programma di elaborazione testi con vari tipi diversi di file di input.

Ecco tre modi in cui potete eseguire il programma Microsoft Word con stuff.txt come file di input:

- Fate un clic destro su stuff.txt, scegliete "Apri con…" e selezionate Microsoft Word.
- Utilizzate le operazioni ammesse dal vostro sistema operativo per creare un collegamento a Microsoft Word sul desktop. Poi trascinate stuff.txt su questo collegamento a Microsoft Word.
- Aprite direttamente l'applicazione Microsoft Word, andate al menu "File", scegliete il comando "Apri", attivate l'opzione per visualizzare "tutti i file", poi scegliete stuff.txt.

Vari modi per eseguire un programma con un file particolare come input.

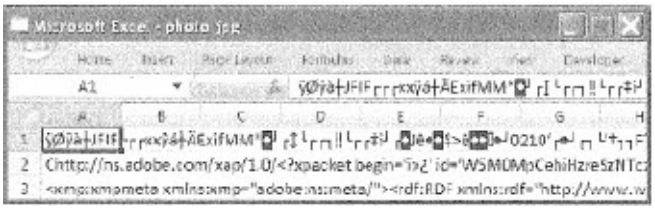


Figura 10.1 Microsoft Excel eseguito con il file "photo.jpg" come input. L'output è spazzatura, ma la cosa importante è che, in linea di principio, si può eseguire qualsiasi programma con qualsiasi input si voglia.

Ovviamente, se aprite un file con un programma diverso da quello che l'ha creato potete ottenere risultati imprevisti. Nella Figura 10.1 si vede che cosa succede se apro il file di immagine "photo.jpg" con il mio foglio di calcolo, Microsoft Excel. In questo caso l'output è spazzatura, inutile per chiunque. Ma il programma è andato in esecuzione e ha prodotto un output.

Già questo può sembrare ridicolo, ma possiamo fare anche un'ulteriore follia. Ricordate che i programmi sono a loro volta memorizzati sul disco del computer come file. Spesso, questi programmi hanno un nome con estensione ".exe", che sta per *executable*, eseguibile, il che significa semplicemente che si può eseguire, far girare il programma. Poiché dunque i programmi sono solo file sul disco, possiamo dare un programma come input a un altro programma. Per esempio, il programma Word è memorizzato sul mio computer come file "WINWORD.EXE". Se allora eseguo il mio foglio di calcolo con il file WINWORD.EXE come input, posso produrre il meraviglioso pasticcio visibile nella Figura 10.2.

Anche in questo caso, può valer la pena che facciate l'esperimento in prima persona. Dovete trovare il file WINWORD.EXE. Sul mio computer, si trova nella cartella "C:\Program Files\Microsoft Office\Office12", ma la posizione esatta dipende dal sistema operativo che usate e dalla versione di Microsoft Office che avete installato. Potreste forse anche dover abilitare la visualizzazione dei "file nascosti" prima di vedere la cartella. E, incidentalmente, potete condurre questo esperimento (e quello seguente) con qualsiasi foglio di calcolo e qualsiasi elaboratori di testi, perciò non c'è bisogno per forza di Microsoft Office.

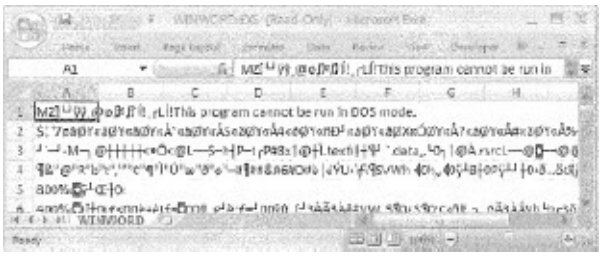


Figura 10.2. Microsoft Excel esamina Microsoft Word. Quando Excel apre il file WINWORD.EXE, il risultato è, come si poteva ben immaginare, spazzatura.

Qui è possibile raggiungere un ultimo livello di follia. E se eseguissimo un programma *su se stesso*? Per esempio, che cosa succederebbe se eseguissi Microsoft Word con il file WINWORD.EXE come input? L'esperimento si può condurre facilmente. La Figura 10.3 mostra il risultato che ho ottenuto quando ci ho provato sul mio computer. Come per gli esempi precedenti, il programma gira benissimo, ma l'output sullo schermo è solamente un pasticcio. (Anche questa volta, provateci di persona.)

Che senso ha tutto questo? Volevo farvi familiarizzare con alcune delle cose più oscure che si possono fare quando si esegue un programma. A questo punto dovreste essere a vostro agio con tre idee un po' strane che saranno molto importanti nel seguito. In primo luogo, qualsiasi programma può essere eseguito con qualsiasi file come input, ma l'output risultante di solito sarà spazzatura, a meno che il file di input non sia stato creato espressamente per il programma che si è scelto di eseguire. In secondo luogo, abbiamo visto che i programmi sono conservati come file sui dischi del computer, e perciò un programma può essere eseguito con un altro programma come file di input. In terzo luogo,

abbiamo visto che si può eseguire un programma dandogli come input lo stesso file in cui è conservato. Fin qui, la seconda e la terza attività hanno sempre prodotto spazzatura, ma nel prossimo paragrafo vedremo un caso affascinante in cui questi trucchi finalmente danno qualche frutto.

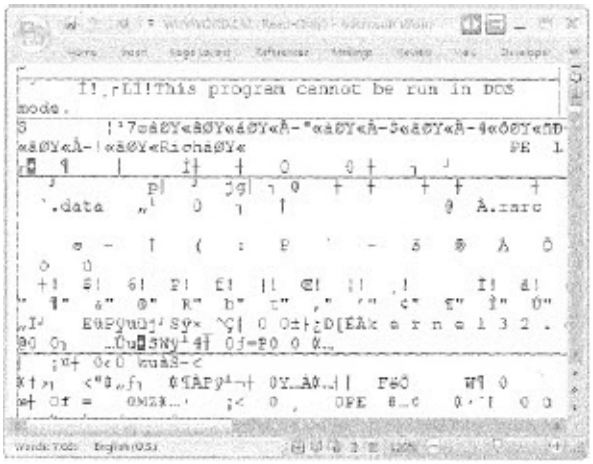


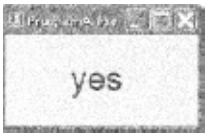
Figura 10.3 Microsoft Word esamina se stesso. Il documento aperto è il file WINWORD.EXE, che contiene il programma che viene eseguito quando si lancia Microsoft Word.

Alcuni programmi non possono esistere

I computer sono bravissimi a eseguire istruzioni semplici: in effetti, i computer moderni eseguono istruzioni semplici miliardi di volte al secondo. Perciò si potrebbe pensare che qualsiasi attività che possa essere descritta in termini semplici e precisi (in inglese o in italiano) possa essere trasformata in un programma ed eseguita da un computer. Il mio obiettivo in questo paragrafo è convincervi che è vero il contrario: esistono alcuni semplici enunciati che è letteralmente impossibile trasformare in un programma per computer.

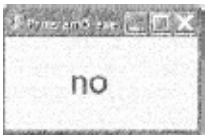
Alcuni semplici programmi “yes-no”

Per semplificare al massimo, considereremo solo un insieme molto noioso di programmi. Li chiameremo programmi “yes-no”, perché l’unica cosa che possono fare è far apparire una finestra di dialogo, la quale può contenere solo la parola “yes” o la parola “no”. Per esempio, qualche minuto fa ho scritto un programma che si chiama ProgramA.exe, che non fa altro che produrre questa finestra di dialogo:



Se guardate la barra del titolo della finestra di dialogo, potete vedere il nome del programma che ha prodotto questo output, in questo caso Program A.exe.

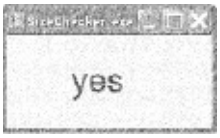
Poi ho scritto un altro programma che si chiama ProgramB.exe, che produce come output un “no” invece di uno “yes”.



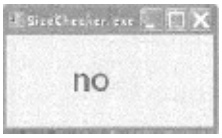
ProgramA e ProgramB sono estremamente semplici; tanto semplici che non hanno nemmeno bisogno di un input (se ricevono qualcosa in ingresso, lo ignorano). In altre

parole sono esempi di programmi che si comportano esattamente nello stesso modo ogni volta che vengono eseguiti, indipendentemente da qualsiasi input possa essere dato loro.

Come esempio un po' più interessante di questi programmi "yes-no", ho creato SizeChecker.exe. Questo programma prende un file come input e dà come output "yes" se le dimensioni del file sono superiori ai 10 kilobyte, "no" altrimenti. Se faccio un clic destro su un file video di 50 megabyte (poniamo mymovie.mpg), scelgo "Apri con…" e seleziono SizeChecker.exe, vedrò questo risultato:



Invece, se eseguo lo stesso programma su un piccolo messaggio di posta elettronica di 3 kilobyte (diciamo myemail.msg) ottengo ovviamente un output diverso:



Perciò SizeChecker.exe è un esempio di programma "yes-no" che qualche volta fornisce come risultato "yes" e qualche volta "no".

Ora consideriamo questo programma, leggermente diverso, che chiameremo NameSize.exe: esamina il *nome* del file in ingresso. Se il nome è lungo almeno un carattere, manda in uscita "yes", altrimenti "no". Quali sono i possibili output di questo programma? Beh, per definizione, il nome di qualsiasi file lungo almeno un carattere (altrimenti il file non avrebbe alcun nome e non si potrebbe nemmeno selezionarlo). Perciò NameSize.exe darà sempre come output "yes", quale che sia il suo input.

Incidentalmente, gli ultimi programmi che abbiamo visto sono i primi esempi di programmi che non producono spazzatura quando ricevono come input altri programmi. Per esempio, si dà il caso che le dimensioni del file NameSize.exe siano di solo 8 kilobyte, perciò se si esegue SizeChecker.exe con NameSize.exe come input, l'output è "no" (perché nameSize non ha più di 10 kilobyte). Possiamo addirittura eseguire SizeChecker.exe con se stesso come input: l'output questa volta sarà "yes", perché SizeChecker ha dimensioni superiori ai 10 kilobyte, circa 12 kilobyte per essere precisi. Analogamente, potremmo eseguire NameSize con se stesso come input e l'output sarebbe "yes" perché il nome "NameSize.exe" contiene almeno un carattere. Tutti i programmi yes-no visti fin qui sono, lo ammetto, abbastanza noiosi, ma è importante capire il loro comportamento, perciò passate in rassegna la Tabella 10.1 riga per riga, per essere sicuri di condividere ciascun output.

Tabella 10.1 Gli output di alcuni semplici programmi yes-no. Notate la distinzione fra programmi che inviano sempre in uscita "yes", indipendentemente dai loro input (per esempio ProgramA.exe, NameSize.exe) e i programmi che mandano in uscito "no" o qualche volta (come SizeChecker.exe) o sempre (come ProgramB.exe)

Programma	File di input	Output
ProgramA.exe	address-list.docx	yes
ProgramA.exe	ProgramA.exe	yes
ProgramB.exe	address-list.docx	no
ProgramB.exe	ProgramA.exe	no
SizeChecker.exe	mymovie.mpg (50 MB)	yes
SizeChecker.exe	myemail.msg (3 KB)	no
SizeChecker.exe	NameSize.exe (8 KB)	no
SizeChecker.exe	SizeChecker.exe (12 KB)	yes
NameSize.exe	mymovie.mpg	yes
NameSize.exe	ProgramA.exe	yes
NameSize.exe	NameSize.exe	yes

AlwaysYes.exe: un programma yes-no che analizza altri programmi

Ora siamo in grado di pensare a qualche programma yes-no un po' più interessante. Il primo che analizzeremo si chiama "AlwayYes. exe". Questo programma esamina il file di input che gli viene fornito e manda in uscita "yes" se il file di input è a sua volta un programma yes-no che manda *sempre* in uscita "yes". Altrimenti l'output di AlwaysYes.exe è "no". Notate che AlwaysYes.exe funziona perfettamente con qualsiasi tipo di file di input. Se gli si dà un input che non è un programma eseguibile (per esempio addresslist. docx) darà in uscita "no". Se gli si dà come input un file che *è* un programma eseguibile, ma non è un programma yes-no (per esempio WINWORD.EXE), darà come output "no". Se gli si dà come input un programma yes-no, che però qualche volta dà in uscita "no", allora AlwaysYes.exe darà in uscita "no". L'unico caso in cui AlwaysYes.exe può dare in output "yes" è quando il suo input è un programma yes-no che manda sempre in output "yes", indipendentemente dall'input che riceve. Fin qui abbiamo visto due esempi di programmi del genere: PrograA.exe e NameSize.exe. La Tabella 10.2 riassume l'output di AlwaysYes.exe per vari file di input, compreso AlwaysYes.exe stesso. Come si può vedere nell'ultima riga della tabella. AlwaysYes.exe manda in uscita "no" quando riceve se stesso come input, perché ci sono almeno alcuni file di input per i quali ha come uscita "no".

Nella penultima riga della tabella, avrete forse notato la comparsa di un programma che si chiama Freeze.exe, di cui finora non abbiamo parlato. Freeze.exe è un programma che fa una delle cose più seccanti che un programma possa fare: si "congela" (quale che sia il suo input). L'avrete sperimentato anche voi, quando un videogioco o un programma applicativo sembra semplicemente bloccarsi ("congelarsi") e rifiuta di rispondere a qualsiasi ulteriore input. A quel punto, l'unica possibilità è chiudere il programma. Se anche questo non funziona può addirittura essere necessario spegnere la macchina (qualche volta, quando si usa un laptop, bisogna addirittura togliere le batterie) e riavviare tutto. I programmi possono "congelarsi" per molti motivi diversi. Qualche volta la causa è una situazione di "stallo", di cui abbiamo parlato nel Capitolo 8. In altri casi, il programma può essere impegnato nell'eseguire un calcolo che non terminerà mai, per esempio cercare continuamente un dato che non esiste.

Tabella 10.2 Gli output di AlwaysYes.exe per vari input. Gli unici input che producono "yes" sono programmi yes-no che danno sempre "yes": in questo caso, ProgramA.exe e NameSize.exe

Gli output di AlwaysYes.exe

File di input	Output
address-list.docx	no
mymovie.mpg	no
WINWORD.EXE	no
ProgramA.exe	yes
ProgramB.exe	no
NameSize.exe	yes
SizeChecker.exe	no
Freeze.exe	no
AlwaysYes.exe	no

In ogni caso, non abbiamo bisogno di sapere tutto dei programmi che congelano. Abbiamo solo bisogno di sapere che cosa deve fare AlwaysYes.exe quando riceve un programma del genere come input. AlwaysYes.exe è stato definito in modo accurato perché le sue risposte fossero sempre chiare: dà “yes” se il suo input produce sempre in uscita “yes”; altrimenti dà “no”. Perciò, quando l’input è un programma come Freeze.exe, AlwaysYes.exe deve dare in output “no”, e questo è quel che vediamo nella penultima riga della Tabella 10.2.

YesOnSelf.exe: una variante più semplice di AlwaysYes.exe

Forse avrete già notato che AlwaysYes.exe è un programma decisamente abile e utile, poiché può analizzare altri programmi e prevederne l’output. Ammetto che non ho realmente scritto questo programma: ho semplicemente descritto come si sarebbe dovuto comportare, se l’avessi scritto. E ora descriverò un altro programma, YesOnSelf.exe, simile a AlwaysYes.exe, ma più semplice. Anziché calcolare “yes” se il file di input produce *sempre* “yes”, YesOnSelf.exe risponde “yes” se il file di input produce “yes” quando ha se stesso come input; in caso contrario YesOnSelf.exe dà in output “no”. In altre parole, se do SizeChecker.exe come input a YesOnSelf.exe, quest’ultimo effettuerà qualche tipo di analisi su Sizechecker.exe per determinare quale sarà il suo output quando SizeChecker.exe riceve se stesso come input. Come abbiamo già scoperto (vedi la Tabella 10.1), l’output di SizeChecker.exe, quando prende se stesso come input, è “yes”. Perciò l’output di YesOnSelf.exe per SizeChecker.exe è “yes”. Potete usare lo stesso tipo di ragionamento per calcolare l’output di YesOnSelf.exe per vari altri input. Notate che se il file di input non è un programma yes-no, YesOnSelf.exe automaticamente dà come risposta “no”. La Tabella 10.3 mostra alcuni degli output di YesOnSelf.exe: provate a verificare ciascuna riga della tabella, perché è molto importante avere ben chiaro il comportamento di questo programma prima di procedere.

Tabella 10.3 L’output di YesOnSelf.exe per vari input. Gli unici input che producono un “yes” sono programmi yes-no che danno in output “yes” quando prendono come input se stessi, in questo caso ProgramA.exe, NameSize.exe e SizeChecker.exe. L’ultima riga della tabella è un mistero, poiché sembrerebbe che entrambi gli output possibili siano corretti. Questo caso è analizzato nel testo

Gli output di YesOnSelf.exe

File di input	Output
address-list.docx	no
mymovie.mpg	no
WINWORD.EXE	no
ProgramA.exe	yes
ProgramB.exe	no
NameSize.exe	yes
SizeChecker.exe	yes
Freeze.exe	no
AlwaysYes.exe	no
YesOnSelf.exe	???

Dobbiamo notare due altre cose a proposito di questo programma YesOnSelf.exe, che è piuttosto interessante. In primo luogo, date un'occhiata all'ultima riga della tabella: quale dovrebbe essere l'output di YesOnSelf.exe, quando riceve se stesso come input? Ci sono solo due possibilità, perciò prendiamole in considerazione una alla volta. Se l'output è "yes", sappiamo che (in base alla sua definizione) YesOnSelf.exe deve mandare in output "yes" se ha se stesso come input. È un po' uno scioglilingua, ma se ci riflettete con attenzione vedrete che tutto è perfettamente coerente, perciò potreste essere tentati di concludere che "yes" sia la risposta giusta.

Ma non corriamo troppo. Che succederebbe se l'output di YesOnSelf.exe quando riceve se stesso come input fosse "no"? Vorrebbe dire (anche in questo caso in base alla sua definizione) che YesOnSelf.exe deve mandare in output "no" quando riceve come input se stesso. Anche questa affermazione è perfettamente coerente! Sembrerebbe quasi che YesOnSelf.exe possa scegliere quale debba essere il suo output. Basta che resti fedele alla sua scelta, e la sua risposta sarà corretta. La misteriosa libertà di comportamento di YesOnSelf.exe si rivelerà essere la punta innocua di un iceberg assai insidioso, ma per ora non esploreremo la questione più a fondo.

La seconda cosa da notare a proposito di YesOnSelf.exe è che, come nel caso di AlwaysYes.exe, non ho effettivamente scritto il programma, mi sono limitato a descriverne il comportamento. Notate però che, se *avessi* scritto AlwaysYes.exe. sarebbe facile creare YesOnSelf.exe. Perché? Perché YesOnSelf.exe è più semplice: deve esaminare solo un particolare input, non tutti gli input possibili.

AntiYesOnSelf.exe: l'opposto di YesOnSelf.exe

È venuto il momento di prendere un bel respiro e ricordare dove vogliamo andare a parare. L'obiettivo di questo capitolo è dimostrare che non può esistere un programma che identifica i crash; ma il nostro obiettivo immediato è meno nobile. In questo paragrafo stiamo solo cercando di trovare un esempio di un programma che non può esistere. Sarà un passo avanti molto utile sulla strada verso il nostro obiettivo ultimo, perché, una volta visto *come* dimostrare che un certo programma non può esistere, sarà ragionevolmente immediato usare la stessa tecnica per un programma che identifica i crash. La buona notizia è che siamo molto vicini al traguardo intermedio. Analizzeremo ancora un programma yes-no e ce l'avremo fatta.

Il nuovo programma si chiama "AntiYesOnSelf.exe". Come suggerisce il nome, è

molto simile a YesOnSelf.exe, anzi, è identico a YesOnSelf.exe, salvo che gli output sono invertiti: AntiYesOnSelf. exe dà come output “no” per quegli input per i quali YesOnSelf.exe produce “yes” e viceversa.

Ogniqualvolta il file di input è un programma yes-no, AntiYesOnSelf. exe risponde alla domanda:

Il programma in input, se eseguito con se stesso come input, darà in output un “no”?

Una descrizione concisa del comportamento di AntiYesOnSelf.exe.

Anche se questa è una definizione completa e precisa del comportamento di AntiYesOnSelf.exe, sarà bene esplicitare ancora meglio come si comporta il programma. Ricordate che YesOnSelf.exe risponde “yes” se il suo input produce “yes” quando riceve se stesso come input, e risponde “no” altrimenti. Perciò AntiYesOnSelf. exe risponde “no” se il suo input produrrebbe “yes” qualora avesse se stesso come input, e risponde “yes” in tutti gli altri casi. Detto in un altro modo ancora, AntiYesOnSelf.exe risponde a questa domanda relativa al suo input: “È vero che il file in input, quando riceve se stesso come input, non produce in output ‘yes’?”.

Bisogna ammetterlo, questa descrizione di AntiYesOnSelf.exe è un altro scioglilingua. Potreste pensare che sarebbe più semplice riformularla come “Il file di input, quando riceve se stesso come input, manderà in output ‘no’?”. Perché questa definizione non sarebbe corretta? Perché ci serve quella formulazione contorta sul non mandare in uscita “yes”, invece di dire più semplicemente che manda in output “no”? La risposta è che i programmi a volte possono fare qualcosa di diverso dal mandare in output un “yes” o “no”. Perciò, se qualcuno ci dice che un certo programma non dà come risposta “yes” non possiamo automaticamente concluderne che la sua risposta sia “no”. Per esempio, potrebbe mandare in output spazzatura, o addirittura congelarsi. Comunque, c’è una particolare situazione in cui possiamo trarre una conclusione più forte: se ci viene detto in anticipo che un programma è un programma “yes-no”, allora sappiamo che il programma non si congela mai e non produce mai spazzatura: arriva sempre al termine e produce come output “yes” oppure “no”. Perciò *per i programmi yes-no*, la formulazione contorta sul non mandare in output “yes” equivale alla formulazione più semplice sul mandare in output “no”.

Infine, possiamo dare una semplicissima descrizione del comportamento di AntiYesOnSelf.exe.Ogni volta che il file in input è un programma yes-no, AntiYesOnSelf.exe risponde alla domanda: “Il programma in input, se riceve se stesso come input, darà come output ‘no’?”. Questa formulazione del comportamento di AntiYesOnSelf.exe acquisterà così tanta importanza in seguito che l’ho evidenziata nel riquadro della pagina precedente.

Dato il lavoro già fatto nell’analizzare YesOnSelf.exe, è particolarmente facile costruire una tabella degli output di AntiYesOnSelf. exe. In effetti, possiamo semplicemente copiare la Tabella 10.3 e scambiare tutti gli output, facendo diventare “no” gli “yes” e viceversa. Tutto questo produce la Tabella 10.4. Come al solito, sarà una buona idea passare in rassegna ciascuna riga e verificare di essere d’accordo con la risposta

indicata nella colonna dell'output. Ogni volta che il file in input è un programma yes-no, si può usare la formulazione semplice nel riquadrato della pagina precedente, invece di quella più complessa e contorta data prima.

Tabella 10.4 Gli output di AntiYesOnSelf.exe per vari tipi di input. Per definizione, AntiYesOnSelf.exe dà la risposta opposta a quella di YesOnSelf.exe, perciò la tabella (tranne che per l'ultima riga) è identica alla 10.3, ma con gli output commutati, "yes" al posto di "no" e viceversa. L'ultima riga genera una difficoltà grave, come si analizza nel testo

Gli output di AntiYesOnSelf.exe

File di input	Output
address-list.docx	yes
mymovie.mpg	yes
WINWORD.EXE	yes
ProgramA.exe	no
ProgramB.exe	yes
NameSize.exe	no
SizeChecker.exe	no
Freeze.exe	yes
AlwaysYes.exe	yes
AntiYesOnSelf.exe	???

Come si può vedere dall'ultima riga della tabella, sorge un problema quando si cerca di calcolare l'output di AntiYesOnSelf.exe quando prende se stesso come input. Per analizzare questo caso, semplifichiamo ulteriormente la descrizione di AntiYesOnSelf.exe data nel riquadro della pagina precedente: invece di considerare tutti i possibili programmi yes-no come input, ci concentreremo su quel che accade quando AntiYesOnSelf.exe prende come input se stesso. Perciò la domanda in grassetto in quel riquadro, "il programma in input …" può essere riformulata come "AntiYesOnSelf. exe in input …" perché il programma in input *è* AntiYesOnSelf.exe. Questa è la formulazione finale che ci serve, perciò è evidenziata di nuovo nel riquadro qui sotto.

AntiYesOnSelf.exe, quando prende se stesso come input, risponde alla domanda:

AntiYesOnSelf.exe, se eseguito con se stesso come input, darà in output un "no"?

Una descrizione concisa del comportamento di AntiYesOnSelf.exe, quando prende se stesso come input. Si noti che questo riquadro è una versione semplificata del precedente, per il caso speciale in cui il file in input è lo stesso AntiYesOnSelf.exe.

Ora siamo pronti a ragionare sull'output di AntiYesOnSelf.exe quando prende se stesso come input. Ci sono solo due possibilità ("yes" e "no"), perciò non dovrebbe essere una cosa troppo difficile. Vediamo ciascuno dei due casi.

Caso 1 (l'output è "yes"): Se l'output è "yes", allora la risposta alla domanda in grassetto nel riquadro è "no". Ma la risposta alla domanda in grassetto è, per definizione, l'output di AntiYesOnSelf. exe (rileggete tutto il riquadro ancora una volta per convincervene) e perciò l'output deve essere "no". In breve, abbiamo appena dimostrato che se l'output è "yes" allora l'output è "no". Impossibile! Siamo arrivati a una *contraddizione*. (Se non avete ben presente questa tecnica di dimostrazione, la dimostrazione per assurdo, vi conviene tornare indietro e riguardare l'analisi che ne abbiamo fatto nella parte precedente del capito. Useremo questa tecnica varie volte nelle prossime pagine.) Poiché siamo arrivati a una contraddizione, la nostra ipotesi che l'output sia "yes" non può essere valida.

Abbiamo dimostrato che l'output di AntiYesOnSelf.exe, quando prende se stesso come input, non può essere "yes". Perciò passiamo all'altra possibilità.

Caso 2 (l'output è "no"): Se l'output è "no", allora la risposta alla domanda in grassetto nel riquadro è "yes". Ma, come nel caso 1, la risposta alla domanda in grassetto è, per definizione, l'output di AntiYesOnSelf.exe, quindi l'output deve essere "yes". In altre parole, abbiamo appena dimostrato che se l'output è "no", allora l'output è "yes". Ancora una volta abbiamo ottenuto una contraddizione, perciò la nostra ipotesi che l'output sia "no" non può essere valida. Abbiamo dimostrato che l'output di AntiYesOnSelf.exe, quando prende se stesso come input, non può essere "no".

E adesso? Abbiamo eliminato le uniche due possibilità per l'output di AntiYesOnSelf.exe quando prende se stesso come input. Anche questa è una contraddizione: AntiYesOnSelf.exe è stato definito un programma yes-no, un programma che termina sempre e produce uno dei due output "yes" o "no". Noi invece abbiamo appena trovato un particolare input per il quale AntiYesOnSelf.exe non produce né l'uno né l'altro di questi output! Questa contraddizione comporta che la nostra assunzione iniziale fosse falsa: *non* è possibile scrivere un programma yes-no che si comporti come AntiYesOnSelf.exe.

Ora capite perché sono stato molto onesto e ho ammesso di non aver scritto realmente i programmi AlwaysYes.exe, YesOnSelf.exe o AntiYesOnSelf.exe. Tutto quel che ho fatto è descrivere come questi programmi si comporterebbero se li avessi scritto. Nell'ultimo paragrafo, abbiamo usato una dimostrazione per assurdo per dimostrare che AntiYesOnSelf.exe non può esistere. Ma possiamo dimostrare anche qualche cosa di più: la stessa esistenza di AlwaysYes.exe e di YesOnSelf.exe è impossibile! Perché? Come avrete già immaginato, lo si dimostra per assurdo. Ricordate quando abbiamo detto che, se AlwaysYes.exe esisteva, sarebbe stato facile apportarvi qualche piccolo cambiamento e produrre YesOnSelf.exe? E che, se esisteva YesOnSelf.exe, sarebbe stato facilissimo produrre AntiYesOnSelf.exe, dato che bisognava solo scambiare gli output ("yes" al posto di "no" e viceversa)? In breve, se esiste AlwaysYes.exe, esiste anche AntiYesOnSelf.exe. Ma sappiamo già che AntiYesOnSelf.exe non può esistere e, perciò, non può esistere neanche AlwaysYes.exe. La stessa argomentazione porta a concludere che anche YesOnSelf.exe è impossibile.

Ricordate, tutta questa sezione era solo un trampolino per raggiungere l'obiettivo finale, dimostrare l'impossibilità di programmi che identificano i crash. L'obiettivo, più modesto, di questa sezione era dare qualche esempio di programmi che non possono esistere e l'abbiamo raggiunto esaminando tre diversi programmi, ciascuno dei quali è impossibile. Dei tre, il più interessante è AlwaysYes. exe. Gli altri due sono un po' misteriosi, in quanto si concentrano sul comportamento di programmi che prendono se stessi come input. AlwaysYes.exe, invece, sarebbe un programma molto potente, perché, se esistesse, potrebbe analizzare qualsiasi altro programma e dirci se quel programma dà sempre in output "yes". Ma, come abbiamo visto, nessuno riuscirà mai a scrivere un programma tanto astuto e utile.

L'impossibilità di identificare i crash

Siamo finalmente pronti per iniziare una dimostrazione riguardante un programma che

riesce ad analizzare altri programmi e stabilisce se generano o meno un crash: specificamente, dimostreremo che un programma simile non può esistere. Dopo aver letto le ultime pagine, probabilmente avrete già indovinato che useremo una dimostrazione per assurdo. Cominceremo cioè con l'assumere che il nostro sacro graal esista: che esiste cioè un programma, CanCrash. exe che può analizzare altri programmi e dirci se possono andare in crash oppure no. Dopo aver fatto un po' di cose strane, misteriose e divertenti a CanCrash.exe, arriveremo a una contraddizione.

Uno dei passi della dimostrazione richiede che si prenda un programma e lo si modifichi in modo che vada in tilt sotto determinate circostanze. Come si può fare? È una cosa molto facile. I crash possono essere dovuti a molte cause diverse. Una delle più comuni è quando un programma cerca di eseguire una divisione per zero. In matematica, si dice che la divisione di un qualsiasi numero per zero è "non definita". Per un computer, "non definito" è un errore serio, e il programma non può continuare, perciò si verifica un crash. Un modo semplice perché un programma vada deliberatamente in crash è quindi quello di inserirvi un paio di istruzioni in più che dividano un numero per zero. In effetti, questo è proprio il modo in cui ho realizzato l'esempio di TroubleMaker.exe nella Figura 10.4.

Cominciamo dunque la dimostrazione principale dell'impossibilità di un programma che identifichi i crash. La Figura 10.5 riassume l'andamento dell'argomentazione. Iniziando ipotizzando l'esistenza di CanCrash.exe, un programma yes-no che termina sempre, dando come output "yes" se il programma che riceve in input può andare in crash in qualche caso, "no" se il programma in input non va mai in crash.

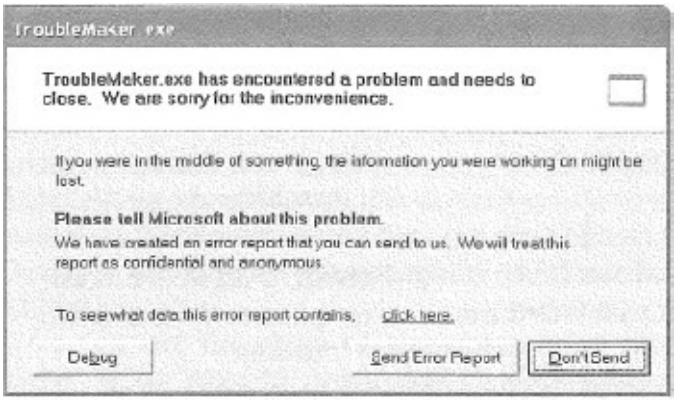


Figura 10.4 Il risultato di un crash in un particolare sistema operativo. Sistemi operativi diversi gestiscono i crash in modi diversi, ma se ve ne capiterà uno lo riconoscerete di certo. Questo programma, TroubleMaker.exe, è stato scritto apposta per provocare un crash, a dimostrazione che è facile mandare in tilt intenzionalmente un sistema.

Ora apportiamo una modifica un po' folle a CanCrash.exe: invece di dare in input "yes", faremo in modo che vada in crash. (Come abbiamo detto, è facile: basta produrre una divisione per zero.) Chiamiamo CanCrashWeird.exe questo programma. Questo programma dunque va deliberatamente in crash, provocando la comparsa di una finestra di dialogo simile alla Figura 10.4, se il suo input può andare in crash, mentre dà come output "no" se il suo input non va mai in crash.

Il passo successivo, come si vede nella Figura 10.5, è trasformare CanCrashWeird.exe in una bestia ancor più strana, CrashOnSelf. exe. Questo programma, proprio come YesOnSelf,exe nella sezione precedente, si occupa solo del comportamento dei programmi quando ricevono se stessi come input. Specificamente, CrashOnSelf.exe esamina il

programma che gli viene dato come input e va deliberatamente in crash se quel programma va in crash quando riceve se stesso in input; altrimenti calcola l'output "no". Si noti che è facile produrre CrashOnSelf.exe da CanCrashWeird.exe: la procedura è esattamente la stessa che trasformava AlwaysYes.exe in YesOnSelf exe (p. 214).

Il passo finale nella successione dei quattro programmi della Figura 10.5 è quello di trasformare CrashOnSelf.exe in AntiCrashOnSelf.exe. Si tratta semplicemente di invertire il comportamento del programma: se il programma che riceve in input va in crash quando riceve in input se stesso, AntiCrashOnSelf.exe manda in output "yes"; se invece il programma che riceve in input non va in crash quando riceve in input se stesso, AntiCrashOnSelf.exe va deliberatamente in crash.

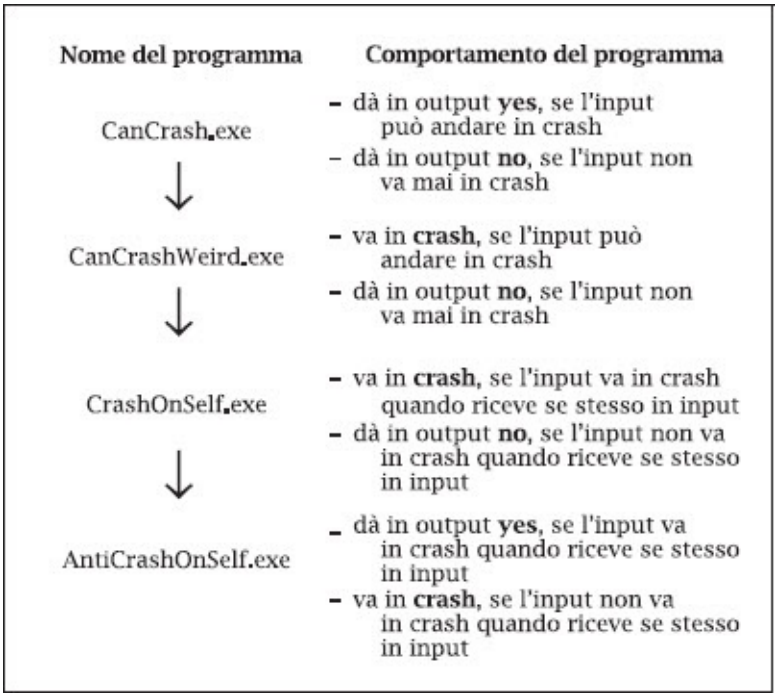


Figura 10.5 Una successione di quattro programmi di identificazione dei crash che non possono esistere. L'ultimo programma, AntiCrashOnSelf.exe, è ovviamente impossibile, poiché produce una contraddizione quando riceve se stesso come input. Ciascun programma però può essere prodotto facilmente apportando una piccola modifica al precedente (è quel che indicano le frecce). Perciò nessuno dei quattro programmi può esistere.

Ora siamo arrivati a un punto in cui possiamo produrre una contraddizione. Che cosa farà AntiCrashOnSelf.exe quando riceve se stesso come input? In base alla sua stessa descrizione, deve dare in output "yes" se va in crash (una contraddizione, perché non può terminare con l'output "yes" se è già andato in crash). E di nuovo, in base alla sua stessa descrizione, AntiCrashOnSelf.exe dovrebbe andare in crash se non va in crash, e anche questa è una contraddizione. Abbiamo eliminato entrambi i possibili comportamenti di AntiCrashOnSelf.exe, il che significa che il programma non può esistere.

Infine, possiamo usare la catena di trasformazioni indicata nella Figura 10.5 per dimostrare che non può esistere neanche CanCrash.exe. Se *esistesse*, potremmo trasformarlo, seguendo le frecce, in AntiCrashOnSelf.exe, ma sappiamo già che AntiCrashOnSelf. exe non può esistere. Questa è una contraddizione e, perciò, la nostra ipotesi che CanCrash.exe esista deve essere falsa.

Il problema della fermata e l'indecidibilità

Questo conclude la nostra esplorazione di uno dei risultati più raffinati e profondi della

computer science: abbiamo dimostrato l'assoluta impossibilità di scrivere un programma come CanCrash. exe, un programma che analizza altri programmi e identifica tutti i possibili errori che potrebbero mandarli in crash.

Quando Alan Turing, fondatore della *computer science* teorica dimostrò un risultato come questo negli anni Trenta del secolo scorso, non aveva in mente né bachi né crash: i fondo, i computer non erano stati ancora realizzati. Turing era interessato a scoprire se un dato programma alla fine producesse una risposta o no. Una domanda strettamente affine è: un dato programma si *fermerà*, oppure andrà avanti a calcolare per sempre, senza mai produrre una risposta? Questa domanda, se un dato programma alla fine terminerà o "si fermerà" è il Problema della fermata (o *Halting problem*, in inglese). Il grande merito di Turing è stato dimostrare che la sua variante del Problema della fermata è quello che si dice "indecidibile". Un problema indecidibile è un problema che non può essere risolto scrivendo un programma per computer, perciò un altro modo di formulare il risultato di Turing è: non si può scrivere un programma per computer, chiamiamolo AlwaysHalts. exe, che dà in output "yes" se il suo input termina sempre e "no" altrimenti.

Visto in questo modo, il Problema della fermata è molto simile al problema che abbiamo affrontato in questo capitolo, e che potremmo chiamare il Problema del crash. Abbiamo dimostrato l'indecidibilità del Problema del crash, ma si può utilizzare sostanzialmente la stessa tecnica per dimostrare anche l'indecidibilità del Problema della fermata. E, come avrete indovinato, esistono molti altri problemi informatici indecidibili.

Quali sono le conseguenze dei programmi impossibili?

Escludendo la Conclusione, questo è l'ultimo capitolo del libro. L'ho inserito apposta come contraltare ai capitoli precedenti. Mentre in quelli si esaltava un'idea notevole, che rende i computer ancor più potenti e utili per noi esseri umani, qui abbiamo visto uno dei loro limiti fondamentali. Abbiamo visto che esistono problemi che non è possibile risolvere con un computer, non importa quanto sia potente la macchina o quanto abile sia il suo programmatore umano. E fra questi problemi indecidibili ve ne sono anche alcuni che potrebbero svolgere attività molto utili, come analizzare altri programmi per scoprire se possano mai andare in tilt.

Qual è l'importanza di questo fatto strano, forse addirittura premonitore? L'esistenza di problemi indecidibili influenza il modo in cui usiamo i computer nella pratica? E che dire dei calcoli che noi esseri umani eseguiamo nei nostri cervelli, anche questi non possono affrontare problemi indecidibili?

Indecidibilità e uso dei computer

Vediamo prima gli effetti pratici dell'indecidibilità sull'uso dei computer. La risposta in breve è: no, l'indecidibilità non ha grandi conseguenze per la pratica quotidiana dell'informatica. I motivi sono due. In primo luogo l'indecidibilità riguarda solo il fatto che un programma produca una risposta, e non prende in considerazione il tempo che si deve aspettare per avere quella risposta. Nella pratica, il problema dell'efficienza (in parole povere, quanto si debba aspettare per avere una risposta) è estremamente importante. Esistono moltissimi problemi decidibili per i quali non è noto alcun algoritmo efficiente. Il più famoso è il Problema del commesso viaggiatore. Riformulato in termini moderni, il problema è questo: supponiamo che dobbiate visitare in aereo un gran numero

di città (diciamo 20, 30 o 100). In quale ordine dovete visitarle, in modo da spendere di meno nei viaggi aerei? Come abbiamo già detto questo problema *è* decidibile; in effetti un programmatore alle prime armi può scrivere un programma che trovi il percorso più economico fra le varie città. Il difetto è che il programma potrebbe richiedere milioni di anni per completare il suo lavoro. Nella pratica, non va bene. Quindi, il semplice fatto che un problema sia decidibile non significa che possiamo risolverlo nella pratica.

Il secondo motivo per cui l'indecidibilità ha pochi effetti pratici è che spesso si può trovare un buon modo per risolvere i problemi indecidibili *nella maggior parte dei casi*. L'esempio principale di questo capitolo ne è una illustrazione eccellente. Abbiamo seguito una dimostrazione complessa che dice che nessun programma sarà mai in grado di trovare tutti gli errori in tutti i programmi, però possiamo continuare a scrivere un programma per identificare i crash, nella speranza che trovi la maggior parte dei bachi nella maggior parte dei tipi di programmi. Questa, in effetti, è un'area di ricerca molto attiva nella computer science. I miglioramenti che abbiamo visto nell'affidabilità del software negli ultimi decenni sono dovuti in parte ai progressi fatti nei programmi per trovare i crash. Quindi spesso è possibile produrre soluzioni parziali molto utili a problemi indecidibili.

Indecidibilità e cervello umano

L'esistenza di problemi indecidibili ha conseguenze per i processi di pensiero umani. Questa domanda ci porta direttamente alle profondità oscure di alcuni problemi classici della filosofia, come la definizione della coscienza e la distinzione fra mente e cervello. Comunque una cosa può essere chiara: se siete convinti che il cervello umano in linea di principio possa essere simulato da un computer, allora anche il cervello è soggetto alle stesse limitazioni dei computer. In altre parole, ci saranno problemi che nessun cervello umano può risolvere, per quanto intelligente o ben addestrato sia. Questa conclusione segue immediatamente dal risultato fondamentale di questo capitolo. Se il cervello può essere imitato da un programma informatico, e il cervello può risolvere problemi indecidibili, si può usare una simulazione al computer del cervello per risolvere quei problemi indecidibili; il che contraddirebbe il fatto che i programmi per computer non possono risolvere problemi indecidibili.

Ovviamente, siamo ben lontani dall'aver dato una risposta definitiva alla domanda, se sarà mai possibile realizzare simulazioni accurate del cervello. Da un punto di vista scientifico, non sembra esistano barriere fondamentali, poiché conosciamo abbastanza bene nei particolari come avvenga la trasmissione dei segnali chimici ed elettrici nel cervello; d'altra parte, varie argomentazioni filosofiche fanno pensare che in qualche modo i processi fisici del cervello creino una "mente" che è qualitativamente diversa da qualsiasi sistema fisico simulabile con un computer. Queste argomentazioni filosofiche possono assumere varie forme e possono essere basate, per esempio, sulla nostra capacità di auto-riflessione e di intuizione, o su qualche appello alla spiritualità.

Esiste un collegamento affascinante con il saggio di Turing del 1937 sull'indecidibilità, saggio che molti considerano l'atto di fondazione della computer science come disciplina. Il titolo di quel saggio è purtroppo abbastanza oscuro: inizia con un'espressione che suona innocua, "On computable numbers…" [Sui numeri calcolabili], ma conclude con un terribile "… with an application to the Entscheidungsproblem" […con una applicazione all'Entscheidungsproblem]. (Non ci occuperemo qui della seconda

parte del titolo!) È fondamentale ricordare che negli anni Trenta la parola “computer” aveva un significato completamente diverso da quello contemporaneo. Per Turing un “computer” era un essere *umano*, che compiva qualche tipo di calcolo con carta e matita. Quindi i “numeri calcolabili” nel titolo del suo saggio sono i numeri che, in linea di principio, potrebbero essere calcolati da un essere umano. Per sostenere la sua argomentazione, Turing però descrive un particolare tipo di macchina (per Turing, una “macchina” era quel che oggi chiameremmo un “computer”) in grado di eseguire calcoli. Parte del saggio è dedicata a dimostrare che certi tipi di calcoli non possono essere eseguiti da queste macchine: è la dimostrazione di indecidibilità, di cui abbiamo già parlato ampiamente. Un’altra parte dello stesso saggio però espone un’argomentazione particolareggiata e convincente per cui la “macchina” di Turing (leggi: un computer) può eseguire qualsiasi calcolo eseguito da un “computer” (leggi: essere umano).

Forse comincia a esservi chiaro perché sia difficile sopravvalutare la natura fondante del saggio di Turing. Non solo definisce e risolve alcuni dei problemi più fondamentali della computer science, ma si fionda anche nel bel mezzo di un campo minato filosofico, sostenendo in modo persuasivo che i processi di pensiero umani possano essere emulati da computer (che, ricordate, non erano stati ancora inventati!). Nel gergo filosofico moderno, questa idea (che tutti i computer, e probabilmente anche gli esseri umani, abbiano una potenza di calcolo equivalente) è chiamata *tesi di Church-Turing*. Vengono abbinati i nomi di Alan Turing e Alonzo Church, che (come ho già detto) ha scoperto indipendentemente l’esistenza di problemi indecidibili. In effetti, Church ha pubblicato il suo lavoro qualche mese prima di Turing, ma la sua formulazione è più astratta e non parla esplicitamente di calcolo da parte di macchine.

Il dibattito sulla validità della tesi di Church-Turing è ancora aperto. Ma, se vale la sua versione forte, allora i nostri computer non sono gli unici toccati dai limiti dell’indecidibilità. Gli stessi limiti esisterebbero non solo per il genio sulla punta delle nostre dita, ma anche per il genio che sta dietro di esse, le nostre menti.

CAPITOLO 11

Conclusione: più genio sulla punta delle dita?

> Possiamo vedere nel futuro solo per un piccolo tratto, ma possiamo pure vedere che in questo piccolo tratto c'è molto da fare.
>
> Alan Turing, *Macchine calcolatrici e intelligenza*[1]

Nel 1991 ho avuto la fortuna di assistere a una conferenza pubblica del grande fisico teorico Stephen Hawking. Durante la conferenza, audacemente intitolata "Il futuro dell'universo", Hawking predisse fiducioso che l'universo avrebbe continuato a espandersi per almeno altri 10 miliardi di anni. Ha anche aggiunto: "Non penso che sarò ancora in circolazione nel momento in cui dovessi essere smentito". Per mia sfortuna, le previsioni sull'informatica non possono avere la stessa garanzia di 10 miliardi di anni su cui possono contare i cosmologi. Qualunque previsione farò è possibile venga smentita nel corso della mia vita.

Ma questo non può impedirci di pensare al futuro delle grandi idee della computer science. I grandi algoritmi che abbiamo esplorato resteranno "grandi" per sempre? Diventeranno mai obsoleti? Emergeranno nuovi grandi algoritmi? Per affrontare queste domande, dobbiamo pensare meno come cosmologi e più come storici. Questo mi ricorda un'altra esperienza. Molti anni fa ho seguito alcune conferenze televisive dello storico di Oxford A.J.P. Taylor, molto apprezzato ma anche molto discusso. Alla fine della serie di conferenze, Taylor affrontò direttamente il problema della possibilità di una terza guerra mondiale. Pensava che la risposta fosse positiva, perché gli esseri umani probabilmente "continueranno a comportarsi in futuro come hanno fatto in passato".

Seguiamo dunque le orme di A.J.P. Taylor e inchiniamoci alla grande onda della storia. I grandi algoritmi descritti in questo libro sono nati da accidenti e invenzioni che hanno punteggiato il Ventesimo secolo. Sembra ragionevole ipotizzare un ritmo analogo per il Ventunesimo, con un nuovo insieme importante di algoritmi ogni due o tre decenni. In qualche caso questi algoritmi potrebbero essere stupendamente originali, tecniche del tutto nuove immaginate dagli scienziati. La crittografia a chiave pubblica e i relativi algoritmi di firma digitale sono stati di questo genere. In altri casi è possibile che gli algoritmi siano stati presenti nella comunità di ricerca da un po' di tempo, in attesa dell'ondata tecnologica giusta per spiccare il volo e risultare ampiamente applicabili. Gli algoritmi di ricerca per l'indicizzazione e l'ordinamento rientrano in questa categoria: algoritmi simili sono esistiti per anni nel campo del recupero di informazioni, ma c'è voluto il fenomeno delle ricerche nel Web per farli diventare "grandi", nel senso di essere utilizzati quotidianamente dai comuni utenti di computer. Ovviamente, gli algoritmi sono evoluti anche per la nuova applicazione, e Page-Rank ne è un buon esempio.

L’emergere di una nuova tecnologia non porta necessariamente alla comparsa di nuovi algoritmi. Pensate per esempio alla crescita straordinaria dei laptop negli anni Ottanta e Novanta. I laptop hanno rivoluzionato l’uso dell’informatica, aumentando enormemente accessibilità e portabilità; hanno anche favorito progressi di estrema importanza in campi diversi, dalla tecnologia degli schermi alle tecniche di gestione dell’energia. Ma direi che nessun grande algoritmo è emerso dalla rivoluzione dei laptop. Internet è invece una tecnologia che ha portato a grandi algoritmi: fornendo un’infrastruttura in cui potevano esistere i motori di ricerca, ha consentito l’evoluzione degli algoritmi di indicizzazione e ordinamento verso la grandezza.

Perciò l’indubbia accelerazione della crescita tecnologica che continua intorno a noi non è in sé garanzia della comparsa di nuovi grandi algoritmi. Esiste anzi una grande forza storica che agisce in direzione opposta, facendoci immaginare che il ritmo dell’innovazione algoritmica, casomai, in futuro diminuirà. Sto pensando al fatto che l’informatica sta diventando una disciplina scientifica matura. Rispetto a campi come la fisica, la matematica e la chimica, è molto giovane: i suoi inizi risalgono solo agli anni Trenta. Si può pensare quindi che i grandi algoritmi scoperti nel Ventesimo secolo fossero i frutti dei rami più bassi, e che in futuro diventerà sempre più difficile trovare algoritmi ingegnosi e ampiamente applicabili.

Abbiamo quindi due effetti in concorrenza: nuove nicchie aperte da nuove tecnologie a volte preparano il terreno per nuovi algoritmi, mentre la progressiva maturazione del settore riduce le opportunità. Nell’insieme, penso che questi due effetti tenderanno a compensarsi, portando a una lenta ma costante comparsa di nuovi grandi algoritmi negli anni a venire.

Alcuni algoritmi potenzialmente grandi

Ovviamente alcuni di questi nuovi algoritmi saranno del tutto imprevisti, ed è impossibile dirne di più, ma ci sono invece nicchie e tecniche dotate di potenzialità evidenti. Una delle tendenze ovvie è l’uso crescente di intelligenza artificiale (in particolare di riconoscimento di forme) nei contesti quotidiani, e sarà affascinante vedere se in quest’area emergeranno nuove gemme algoritmiche che ci lasceranno a bocca aperta.

Un altro campo fertile è una classe di algoritmi definiti “protocolli a conoscenza zero”. Questi protocolli usano un tipo speciale di crittografia per ottenere qualcosa di ancor più sorprendente delle firme digitali: consentono a due o più enti di combinare informazioni senza comunicare nessuna delle informazioni parziali. Una possibile applicazione sono le aste online. Con un protocollo a conoscenza zero, chi partecipa all’asta può fare le proprie offerte cifrate in modo che venga determinata l’offerta vincente, ma senza che nessuno riceva alcuna informazione rispetto alle altre offerte. I protocolli a conoscenza zero sono un’idea così brillante che entrerebbero facilmente nel mio canone dei grandi algoritmi, se solo fossero utilizzati nella pratica. Fin qui, però, non sono mai stati molto utilizzati.

Un’altra idea a cui sono state dedicate grandi ricerche accademiche, ma che è di uso praticolo limitato è una tecnica definita “tabelle hash distribuite”. Queste tabelle sono un modo ingegnoso di memorizzare le informazioni in un sistema paritetico o *peer-topeer*, un sistema cioè che non possiede un server centrale che diriga il flusso delle informazioni. Mentre scrivo, però, molti dei sistemi che sostengono di essere paritetici di fatto utilizzano

server centrali per alcune delle loro funzionalità e quindi non hanno bisogno di affidarsi a tabelle hash distribuite.

La tecnica denominata “tolleranza bizantina ai guasti” ricade nella stessa categoria: un algoritmo sorprendente ed elegante, che non può essere ancora definito “grande” per mancanza di diffusione pratica. La tolleranza bizantina ai guasti consente a determinati sistemi di tollerare qualsiasi tipo di errore (purché non ce ne siano troppi simultaneamente). Questo a differenza della normale nozione di tolleranza ai guasti, in cui un sistema può sopravvivere a errori più innocui, come il guasto definitivo di una unità disco o il crash di un sistema operativo.

I grandi algoritmi possono scomparire?

Oltre a immaginare quali algoritmi possono assurgere in futuro alla grandezza, potremmo anche chiederci se qualcuno degli attuali “grandi” algoritmi, strumenti indispensabili che usiamo costantemente senza pensarci, possa perdere importanza. Anche qui può guidarci la storia. Se restringiamo l’attenzione a casi particolari, è sicuramente vero che gli algoritmi possono perdere importanza. L’esempio più ovvio è la crittografia, dove si svolge una costante corsa al riarmo fra ricercatori che inventano nuovi algoritmi e altri ricercatori che inventano modi per forzarli. Pensate alle cosiddette funzioni *hash* crittografiche. La funzione *hash* denominata MD5 è uno standard ufficiale di Internet ed è stata ampiamente usata dagli inizi degli anni Novanta, ma da allora sono stati individuati dei difetti significativi per la sicurezza e non è più consigliata. Analogamente, abbiamo visto che il sistema di firma digitale potrà essere forzato facilmente se e quando sarà possibile costruire computer quantistici di ragionevoli dimensioni.

Penso però che esempi come questi diano una risposta troppo limitata. Certo, MD5 è stata forzata (e, incidentalmente, lo è stata anche la successiva SHA-1), ma questo non significa che l’idea centrale delle funzioni *hash* sia diventata irrilevante. Al contrario queste funzioni sono usate molto spesso e ce ne sono moltissime che non sono mai state forzate. Perciò, se si mantiene un punto di vista abbastanza ampio sulla situazione e si è pronti ad adattare le specifiche di un algoritmo pur conservandone le sue idee principali, sembra improbabile che molti degli algoritmi che oggi sono grandi perdano la loro importanza nel futuro.

Che cosa abbiamo imparato?

Ci sono temi comuni che si possono estrarre dai grandi algoritmi presentati qui? Un tema, che è stato una grande sorpresa per me in quanto autore del libro, è che tutte le grandi idee possono essere spiegate senza dover far conto su una conoscenza precedente di programmazione o in generale di informatica. Quando ho cominciato a scrivere, davo per scontato che i grandi algoritmi si sarebbero divisi in due categorie: da una parte quelli che al fondo hanno qualche trucco semplice ma astuto, che può essere spiegato senza particolari conoscenze tecniche; dall’altra quelli che dipendono così strettamente da idee informatiche tanto avanzate da non poter essere spiegate a chi non ha una formazione in questo campo. Avevo in mente di includere algoritmi di questa seconda categoria presentando qualche aneddoto storico (speravo) interessante, illustrandone le applicazioni importanti e affermando con enfasi che l’algoritmo era ingegnoso anche se non ero in grado di spiegare come funziona. Immaginatevi la mia sorpresa e il mio piacere quando ho

scoperto che tutti gli algoritmi che avevo scelto ricadevano nella prima categoria! Certo, molti particolari tecnici importanti dovevano rimanere esclusi, ma il meccanismo centrale che fa funzionare il tutto poteva essere spiegato sfruttando nozioni non specialistiche.

Un altro tema comune a tutti i nostri algoritmi è che il campo della *computer science* comprende molto più che la sola programmazione. Ogni volta che tengo un corso introduttivo, chiedo agli studenti di dirmi che cosa pensano sia veramente la *computer science*: la risposta di gran lunga più comune è "programmazione", o qualcosa di equivalente, come "ingegneria del software". Quando vengono sollecitati a presentare qualche altro aspetto della disciplina, molti non sanno proprio che cosa dire. Spesso una risposta successiva è relativa all'hardware, come "progettazione dell'hardware". È evidente che è diffusa una concezione errata di quel che fa chi si occupa di *computer science*. Letto questo libro, spero che abbiate un'idea più concreta dei problemi su cui riflettono i ricercatori, e dei tipi di soluzioni che escogitano.

Un'analogia molto semplice potrà essere d'aiuto. Supponiamo che incontriate un professore il cui interesse principale di ricerca è la letteratura giapponese. È estremamente probabile che quel professore sappia parlare, leggere e scrivere in giapponese. Ma se vi venisse chiesto di indovinare su che cosa rifletta quel professore per la maggior parte del suo tempo mentre conduce le sue ricerche, non pensereste "la lingua giapponese". Certo, la lingua giapponese è una parte di ciò che bisogna conoscere per studiare i temi, la cultura e la storia della letteratura giapponese; ma qualcuno che parla perfettamente giapponese può ignorare completamente la letteratura giapponese (probabilmente milioni di persone in Giappone sono in questa condizione).

La relazione fra i linguaggi di programmazione e le idee centrali della *computer science* sono molto simili. Per implementare e sperimentare gli algoritmi, i ricercatori debbono convertirli in programmi per computer, e ciascun programma è scritto in un linguaggio di programmazione, come Java, C++ o Python. Quindi, la conoscenza di un linguaggio di programmazione è essenziale, ma è soltanto un prerequisito: la sfida principale sta nell'inventare, adattare e capire algoritmi. Dopo aver visto i grandi algoritmi in questo libro, spero che i lettori avranno più chiara questa distinzione.

La fine del nostro viaggio

Abbiamo raggiunto il termine del nostro viaggio nelle profondità del mondo quotidiano dell'informatica. Abbiamo raggiunto i nostri obiettivi? E quindi le nostre interazioni con gli apparecchi informatici saranno in qualche modo diverse?

È possibile che la prossima volta che visiterete un sito web sicuro sarete curiosi di sapere chi garantisce che è degno di fiducia, e andrete a controllare la catena dei certificati digitali che sono stati esaminati dal vostro browser (Capitolo 9). O magari la prossima volta che una transazione online fallisce per qualche motivo inspiegabile sarete grati invece che frustrati, sapendo che la coerenza dei database deve garantire che non vi venga addebitato qualcosa che non siete riusciti a ordinare (Capitolo 8). O un giorno vi direte: "Sarebbe bello se il mio computer potesse fare *questa cosa* per me!", rendendovi poi conto che è una cosa impossibile, perché si può dimostrare che quel compito è indecidibile, utilizzando lo stesso metodo che abbiamo usato per il programma che identifica i crash (Capitolo 10).

Sono sicuro che potrete trovare molti altri esempi in cui la conoscenza dei grandi algoritmi può modificare il modo in cui interagite con un computer; come però ho messo ben in chiaro nell'introduzione, non era questo l'obiettivo principale del libro. Il *mio* obiettivo principale era farvi conoscere un po' i grandi algoritmi, in modo da suscitarvi un senso di meraviglia per alcune delle comuni attività di elaborazione, un po' come un astronomo dilettante riesce ad apprezzare meglio il cielo notturno.

Solo voi, lettori, potete sapere se sono riuscito a raggiungere l'obiettivo. Una cosa però è certa: il vostro genio personale è sulla punta delle vostre dita. Usatelo liberamente.

---

1 Tr. it. in Vittorio Somenzi (a cura di), *La filosofia degli automi*, Boringhieri, Torino, 1965, p. 156.

RINGRAZIAMENTI

> Strada, m’avvio su di te, e in giro mi guardo, e ritengo che tu non sei solo ciò che vi è qui,
> Ritengo che molto invisibile vi sia anche qui.
>
> Walt Withman, “Canto della strada”[2]

Molti amici, colleghi e familiari hanno lento il manoscritto, in tutto o in parte. Fra loro Alex Bates, Wilson Bell, Mike Burrows, Walt Chromiak, Michael Isard, Alastair MacCormick, Raewyn MacCormick, Nicoletta Marini-Maio, Frank McSherry, Kristine Mitchell, Ilya Mironov, Wendy Pollack, Judith Potter, Cotten Seiler, Helen Takacs, Kunal Talwar, Tim Wahls, Jonathan Waller, Udi Wieder e Ollie Williams. I loro suggerimenti hanno portato a un gran numero di miglioramenti sostanziali; anche i commenti di due revisori anonimi hanno dato luogo a miglioramenti importanti.

Chris Bishop mi ha incoraggiato e consigliato. Tom Mitchell mi ha dato il permesso di usare i suoi disegni e il suo codice sorgente per il Capitolo 6.

Vickie Kearn, redattrice del libro, e i suoi colleghi alla Princeton University Press hanno fatto un lavoro eccellente, incubando il progetto e facendo in modo che arrivasse al termine.

I miei colleghi del Dipartimento di matematica e computer science al Dickinson College sono stati una fonte costante di sostegno.

Michael Isard e Mike Burrows mi hanno mostrato il piacere e la bellezza dell’informatica. Andrew Blake mi ha insegnato a essere uno scienziato migliore.

Mia moglie Kristine c’è sempre stata ed è ancora lì; *ritengo che molto invisibile vi sia anche qui*.

A tutte queste persone esprimo la mia più profonda gratitudine. Il libro è dedicato, con amore, a Kristine.

---

2 Da *Foglie d’erba*, tr. Di Enzo Giachino, Einaudi, Torino 1973, p. 183.

LETTURE CONSIGLIATE

Capitolo 1 . Consiglio le Royal Institution Christmas Lectures del 2008 tenute da Chris Bishop. I video di queste conferenze si trovano in rete: sono in inglese, ma non richiedono conoscenze di informatica. Il *New Turing Omnibus* di A.K. Dewdney (Henry Holt, 1993) amplia molti degli argomenti trattati qui. Se avete un po' di conoscenze matematiche, un'introduzione agli algoritmi è Juraj Hromkovic, *Algorithmic Adventures*, Springer, 2009. Fra i titoli di livello universitario: Dasgupta, Papadimitriou, Vazirani, *Algorithms*; Harel e Feldman, *Algoritmi. Lo spirito dell'informatica*, Springer, 2008; Cormen, Leiserson, Rivest, Stein, *Introduzione agli algoritmi*, Jackson Libri.

Capitolo 2 . Chi ha qualche conoscenza di informatica può leggere Croft, Metzler, Strohman, *Search engines: Information Retrieval in Practice*, Addison Wesley.

Capitolo 3 . La citazione iniziale di Larry Page è tratta da un'intervista di Ben Elgin, pubblicata su *Businessweek* del 3 maggio 2004. L'articolo "As We May Think" di Vannevar Bush è stato pubblicato in origine sulla rivista *The Atlantic*, nel luglio 1945. Il saggio originale che descriveva l'architettura di Google è "The Anatomy of a Large-Scale Hypertextual Web Search Engine", di Sergey Brin e Larry Page, presentato alla 1998 World Wide Web Conference. Una analisi più tecnica si trova in Lanville, Meyer, *Google's Page Rank and Beyond*, Princeton UP (esiste anche in versione digitale per il Kindle). Il libro di John Battelle, *Google e gli altri*, Cortina, Milano, 2006, inizia con una storia delle ricerche sul Web, fino all'ascesa di Google. Lo spam web è discusso in "Spam, Damn Spam, and Statistics: Using Statistical Analysis to Locate Spam Web Pages" di Fetterly, Manasse, Najork, pubblicato negli atti della 2004 WebDB Conference.

Capitolo 4 . Simon Singh, *Codici & segreti*, Rizzoli, Milano, 2006, tratta in modo accessibile e piacevole molti aspetti della crittografia, anche di quella a chiave pubblica.

Capitolo 5 . Gli aneddoti su Hamming sono documentati in Thomas M. Thomson, *From Error-Correcting Codes through Sphere Packins to Simple Groups*, Mathematical Association of America, 2004 (bel libro, ma richiede in varie parti buone conoscenze matematiche). Su Turing, Shannon e la nascita della teoria dell'informazione, si può leggere Gleick, *L'informazione*, Feltrinelli, Milano, 2012.

Capitolo 6 . Il database dei volti di cui si parla in questo capitolo è stato creato da Tom Mitchell della Carnegie Mellon University. Il libro di Mitchell, *Machine Learning*, McGraw-Hill (ultima edizione 1997) è stato molto importante per lo sviluppo di questo settore. Sul sito web che accompagna il libro, Mitchell presenta un programma per l'addestramento e la classificazione di reti neurali per il database dei volti. Daniel Crevier, *AI: The Tumultuous History of the Search for Artificial Intelligence*, Basi Books, 1994, offre un resoconto molto interessante del convegno al Dartmouth College. Il brano sulla domanda di finanziamento per il convegno è tratto dal libro di Pamela McCorduck, *Storia dell'intelligenza artificiale*, Muzzio, Padova, 1987.

Capitolo 7 . La storia di Fano, Shannon e della scoperta del codice di Huffman è tratta da una intervista del 1989 a Fano condotta da Arthur Norberg, disponibile nell'archivio di storia orale del Charles Babbage Institute. Un testo tecnico, di livello universitario, sulla compressione dei dati è David MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge UP, 2003. Una discussione più breve e più accessibile è nel libro di Dewdney già citato.

Capitolo 8 . Esistono molti libri di introduzione ai database, ma per lo più spiegano come usarli, non come funzionano, che era invece l'obiettivo del capitolo. Anche i manuali universitari tendono a concentrarsi sull'uso dei database, ma un'eccezione è la seconda parte di Garcia-Molina, Ullman, Widom, *Database Systems*, Prentice Hall, 2012.

Capitolo 9 . Sulle firme digitali, Gail Grant, *Understanding Digital Signatures*, 864.Com, 1997, offre molte informazioni ed è abbastanza accessibile anche a chi non ha una formazione informatica.

Capitolo 10 . La citazione di apertura del capitolo è tratta da una conferenza tenuta da Richard Feynman al Caltech il 29 dicembre 1959, dal titolo "There's Plenty of Room at the Bottom", poi pubblicata nella rivista *Engineering & Science* del Caltech (febbraio 1960). Una presentazione non convenzionale, ma molto interessante, dei concetti di computabilità e indecidibilità ha la forma di un romanzo (di fantasia): Christos Papadimitriou, *Turing (A novel about computation*), Mit Press, 2005 (disponibile anche in formato digitale). Molto bella e utile la biografia scritta da Andrew Hdges, *Alan Turing. Storia di un enigma*, Bollati Boringhieri, Torino, 2012.

Capitolo 11 . La conferenza Darwin tenuta da Stephen Hawking all'Università di Cambridge nel 1991, "The Future of

the Universe", è ristampata nel libro di Hawking, *Buchi neri e universi neonati. E altri saggi*, Rizzoli BUR, Milano, 2000. La serie televisiva curata da A.J.P. Taylor si intitolava *How Wars Begin*, e ne è stata pubblicata una versione in libro nel 1977 (Futura Publications) con lo stesso titolo.

INDICE