

# Numeri primi e monete truccate

Corso di Formazione in  
Matematica e Comunicazione

Andrea Caranti

Trento, 3 dicembre 2003

# Numeri primi

Un numero intero positivo è *primo* se gli unici interi positivi per cui è divisibile sono 1 e il numero stesso.

Per esempio 2 e 3 sono primi.

(Per convenzione, 1 non è primo.)

Invece  $4 = 2 \times 2$  è *composto*.

Ogni numero intero si scrive come prodotto di numeri primi.

## **Perché studiare i numeri primi?**

1. Perché sono oggetti fondamentali dell'aritmetica.

Vero, verissimo, ma...

non lo sentirete da me oggi.

2. Perché servono, e come!

## **A cosa servono i numeri primi?**

1. Per la crittografia.
2. Per i codici a correzione d'errore.
3. Per fare soldi ;-)

Metodi quasi-Montecarlo per l'integrazione numerica, con applicazioni alla finanza.

Per quanto riguarda il primo punto, vedremo un esempio più avanti.

## Come verificare se un numero $N$ è primo?

Ma non basta provare a dividere per 2, 3, 4, ...?

Ahimè, no.

Supponiamo di voler mostrare che il numero

$$\begin{aligned} N &= 100\,000\,000\,000\,000\,000\,000\,000\,000\,000 \\ &\quad 000\,000\,000\,000\,000\,000\,000\,000\,151 \\ &= 10^{50} + 151 \end{aligned}$$

è primo.

Quante divisioni devo fare?

## Fattori di $N$

Generalmente si pensa di dover dividere per  $2, 3, 4, \dots, N - 1$ , o magari che basta fermarsi a  $N/2$ .

Una possibilità è che  $N$  sia un quadrato perfetto, ovvero

$$N = a^2 = a \cdot a, \quad \text{dunque } a = \sqrt{N}.$$

Se  $N$  non è primo, e *non* è un quadrato perfetto, allora

$$N = b \cdot c,$$

dove diciamo  $b < \sqrt{N}$ , mentre  $c > \sqrt{N}$ .

Dunque se  $N$  non è primo, ha un fattore  $b$  maggiore di 1, non più grande di  $\sqrt{N}$ .

## Quante divisioni?

$$\begin{aligned} N &= 100\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000 \\ &\quad 000\,000\,000\,000\,000\,000\,000\,000\,151 \\ &= 10^{50} + 151 \end{aligned}$$

Quindi verificando in questo modo se  $N$  è primo o no, potrei dover fare un numero di divisioni pari a

$$\begin{aligned} \sqrt{N} &\approx \sqrt{10^{50}} \\ &= (10^{50})^{\frac{1}{2}} \\ &= 10^{25} \\ &= 10\,000\,000\,000\,000\,000\,000\,000\,000. \end{aligned}$$

## Tanto i conti li fa il calcolatore. . .

Di chi è questa frase celebre? (La risposta fra poco. . .)

$10^{25}$  divisioni sono tante o poche?

E' stato calcolato (qualche anno fa) che da quando sono stati costruiti i primi calcolatori, tutti i calcolatori del mondo hanno eseguito in totale un numero di *operazioni elementari* non superiori a

$$10^{24}.$$

Quindi, semplicemente, *non c'è stato abbastanza tempo per una risposta.*

## Tutto il tempo di questo mondo

Un altro modo di rendersi conto che certi numeri sono troppo grandi segue dal fatto che l'età dell'universo è stimata in 10 miliardi di anni. Dunque il tempo finora trascorso in secondi è

$$\begin{aligned} 10^{10} \cdot 365 \cdot 24 \cdot 60 \cdot 60 &\approx \\ &\approx 10^{10} \cdot 400 \cdot 25 \cdot 10^2 \cdot 10^2 = 10^{18}. \end{aligned}$$

Oggi (fine 2003) un processore veloce per un PC va facilmente a 3GHz, cioè può fare  $\approx 10^{10}$  operazioni al secondo.

Se anche ogni individuo che vive oggi (e siamo meno di  $10^{10}$ ) avesse usato un PC dall'inizio dell'universo, finora avremmo fatto non più di

$$10^{18} \cdot 10^{10} \cdot 10^{10} = 10^{38}$$

operazioni elementari.

## **Aggiornamento**

Al momento di andare in stampa mi sono reso conto che la stima di 10 miliardi di anni per l'età dell'Universo è vecchia di 10 anni ;-D

Mi ricorda la vecchia barzelletta dei 10003 indiani. . .

# Algoritmi migliori?

Naturalmente si può pensare di ideare algoritmi per fattorizzare che siano più efficienti del metodo delle divisioni di prova. (Diremo qualcosa fra un attimo.)

Ma alcuni problemi saranno sempre infattibili per questioni di tempo.

Problema: stampare (in forma decimale) il numero

$$10^{(10^{24})}$$

Questo numero è formato semplicemente da una cifra 1, seguita da  $10^{24}$  cifre 0.

Anche se stampare una cifra ci costasse solo una operazione elementare, tutta la potenza dispiegata da tutti i calcolatori fino ad oggi sarebbe stata appena sufficiente. . .

## Spaziotempo

Un altro genere di impossibilità deriva da problemi di spazio.

Una stima del numero di particelle elementari nell'Universo è di  $10^{80}$ .

*Su che cosa* posso scrivere in forma decimale il numero

$$10^{(10^{80})}?$$

## Tanto i conti li fa il calcolatore. . .

Questa frase celebre è di N. C. (16 novembre 2003).

Per chi non l'ha letto, raccomando a questo proposito *The Hitchhiker's Guide to Galaxy*.

Il calcolatore Deep Thought ci mette un milione di anni a rispondere a una certa domanda ("Life, the Universe and Everything"), e la risposta alla fine è

42

Quando gli fanno la domanda, Deep Thought, che normalmente è piuttosto borioso, ci pensa un po' in silenzio, e poi commenta: "Tricky". Dal Merriam-Webster: "giving a deceptive impression of easiness, simplicity, or order".

## Deep Thought

Gli scienziati che assistono alla risposta sono ovviamente molto imbarazzati: sta per arrivare l'Imperatore Galattico, che gli dicono?

Cosí cercano in tutta fretta di preparare una domanda ragionevole a cui "42" sia la risposta. Dopo aver scartato per ovvie ragioni

Quanto fa  $6 \times 7$ ?

ripegano su

How many roads  
must a man walk down  
Before you call him a man?  
*Bob Dylan, Blowin' in the Wind*

## Solo un gioco?

Tutti i calcolatori del mondo hanno eseguito finora in totale un numero di operazioni elementari non superiori a  $10^{24}$ .

Per fare *Toy Story* ci sono volute circa

$$10^{17}$$

operazioni elementari, cioè

$$\frac{10^{17}}{10^{24}} = \frac{1}{10^7} = \frac{1}{10\,000\,000}$$

del totale.

## Algoritmi buoni e cattivi

Dunque il metodo delle *divisioni di prova* non è particolarmente efficiente.

Come si fa a sapere se un *algoritmo* è buono o cattivo?

Una misura della bontà di un algoritmo è la sua *complessità algebrica computazionale*.

Provare che un numero è primo dividendolo per 2, 3, 4, ... richiede  $\sqrt{N}$  tentativi. Si dice che ha complessità

$$\sqrt{N} = N^{1/2}.$$

Questo è cattivo.

## Massimo comun divisore

E se voglio calcolare il *massimo comun divisore* di due numeri  $a$  e  $b$ , grandi circa come  $N$ , quante operazioni devo fare?

Se cerco di applicare la regoletta

Fattori (comuni) presi col minimo esponente,

mi sto impegnando a scrivere  $a$  e  $b$  come prodotto di numeri primi, dunque potrei aver bisogno di  $2\sqrt{N}$  divisioni. (Male.)

## Massimo comun divisore

Ma già Euclide aveva scoperto che per calcolare il massimo comun divisore di due numeri  $a$  e  $b$ , grandi circa come  $N$ , esiste un algoritmo che richiede solo  $2 \cdot \log(N)$  divisioni.

Se devo trovare il massimo comun divisore fra  $a$  e  $b < a$ , grandi circa  $N$ , eseguo le *divisioni successive*

$$\begin{aligned}a &= bq + r, \\ b &= rq' + r', \\ r &= r'q'' + r'',\end{aligned}$$

e così via, ove ogni volta  $b > r > r' > r'' > \dots \geq 0$ .

Prima o poi viene un resto 0. Il resto immediatamente precedente è il massimo comun divisore.

## Quante divisioni?

Ho fatto

$$\begin{aligned}a &= bq + r, \\ b &= rq' + r', \\ r &= r'q'' + r'',\end{aligned}$$

e così via, ove ogni volta  $b > r > r' > r'' > \dots \geq 0$ .

Notate che  $r = r'q'' + r'' \geq r' + r'' \geq 2r''$ . Dunque ogni due passi il resto dimezza. Si arriva quindi a zero dopo aver fatto non più di  $2\log_2(N)$  divisioni.

Si può vedere facilmente che l'algoritmo richiede (“costa”) in totale  $\log(N)^3$  operazioni elementari, e con un po' più di attenzione si vede che ne bastano  $\log(N)^2$ .

In questo contesto i logaritmi si prendono tradizionalmente in base 2. (Abbiamo visto che vengono fuori naturalmente.) Ma non fa grande differenza prenderli in base 10.

## Algoritmi e logaritmi

L'algoritmo di Euclide è *molto* meglio rispetto a quello che richiede di fattorizzare i due numeri!

Confrontate il numero “impossibile”

$$\begin{aligned}\sqrt{N} &\approx \sqrt{10^{50}} \\ &= 10^{25} \\ &= 10\,000\,000\,000\,000\,000\,000\,000\,000\,000\end{aligned}$$

col numero “possibilissimo”

$$\log_{10}(N)^2 = \log_{10}(10^{50})^2 = 50^2 = 2\,500,$$

ovvero

$$\begin{aligned}\log_2(N)^2 &= (\log_2(10) \cdot \log_{10}(10^{50}))^2 \\ &\approx (4 \cdot 50)^2 \\ &= 40\,000.\end{aligned}$$

## Polinomiale/esponenziale

Gli algoritmi buoni sono quelli cosiddetti *polinomiali* che, per risolvere una classe di problemi che dipende da un numero  $N$ , richiedono un numero di operazioni che sia del tipo di  $\log(N)^t$  per qualche  $t$ .

Se invece occorre un numero di operazioni del tipo  $N^t$ , l'algoritmo si dice *esponenziale*.

Ci sono poi vie di mezzo. Vedremo un esempio più avanti.

## Polinomiale/esponenziale

Notate che il termine polinomiale/esponenziale si riferisce alla *lunghezza dell'input*, riconosciuto come il parametro giusto almeno fin dai tempi di Shannon.

Per scrivere un numero  $N$  in base 2 ho bisogno di

$$\log_2(N)$$

bit, ovvero zeri e uni. (Per la verità  $\lceil \log_2(N) \rceil$ .)

Dunque

$$\log(N)^2$$

conta come polinomiale, e quindi va bene, mentre

$$\sqrt{N} = N^{1/2} = 2^{\log(N)/2}$$

conta come esponenziale, e dunque va male.

# Fattorizzare

Consideriamo il problema

Dato un numero intero positivo  $N$ , scrivere  $N$  come prodotto di numeri primi.

Il metodo delle divisioni successive richiede almeno  $\sqrt{N} = N^{1/2}$  operazioni.

C'è di molto meglio. Per esempio il metodo  $\rho$  di Pollard richiede all'incirca

$$\sqrt[4]{N} \cdot \log(N)^3 = N^{1/4} \cdot \log(N)^3$$

operazioni.

Si tratta quindi sempre di un metodo di complessità esponenziale.

## Fattorizzare

Un recente metodo di Pomerance-Dixon richiede all'incirca

$$\exp(\sqrt{\log(N) \log(\log(N))})^{\sqrt{2}}$$

operazioni.

Abbiamo

$$\begin{aligned} \exp(\sqrt{\log(N) \log(\log(N))}) &= \\ &= e^{\sqrt{\log(N)} \sqrt{\log(\log(N))}} \\ &= e^{\log(N) \cdot \sqrt{\log(\log(N))} / \log(N)} \\ &= N^{\sqrt{\log(\log(N))} / \log(N)} \end{aligned}$$

## Verificare se un numero è primo

Cosa possiamo dire del problema

Dato un numero intero positivo  $N$ , dire se  $N$  è primo o no.

Se sappiamo fattorizzare, si può rispondere anche a questa domanda.

Ma dato che fattorizzare è difficile, possiamo forse cercare di rispondere *direttamente* a questa domanda?

# Fermat

Un teorema di Fermat afferma che se  $p$  è un numero primo, e  $a$  è un intero non divisibile per  $p$ , allora

$$a^{p-1} \equiv 1 \pmod{p}.$$

Questo vuol dire che dividendo  $a^{p-1}$  per  $p$  si ottiene resto 1.

Prendiamo il numero  $N = 103082107$ . Abbiamo

$$2^{103082106} \equiv 73200281 \pmod{103082107},$$

dunque  $N$  non è primo.

Abbiamo imparato qualcosa sulla sua fattorizzazione? No.

(Incidentalmente  $103082107 = 10007 * 10301$ . Questo sarebbe stato molto facile da fattorizzare con un metodo dovuto proprio a Fermat.)

## Un criterio di non primalità

Se

$$a^{N-1} \not\equiv 1 \pmod{N}$$

anche per un solo  $a$  (che soddisfi qualche condizione. . . ) allora  $N$  *non* è primo.

Ci sono un paio di problemi con questo approccio.

Il primo è esemplificato da

$$2^{340} \equiv 1 \pmod{341},$$

mentre  $341 = 11 \cdot 31$ . Quindi se ottengo 1 non sono sicuro.

Però

$$3^{340} \equiv 56 \pmod{341}.$$

## Falsi primi

Un esempio più grave del primo problema è che

$$a^{560} \equiv 1 \pmod{561}$$

per tutti gli  $a$  il cui massimo comun divisore con  $N = 561$  sia 1.

(Su numeri veramente grandi, questa ultima condizione è irrilevante. . .)

Poiché

$$2^{340} \equiv 1 \pmod{341},$$

si dice che 341 è *uno pseudoprimo rispetto alla base 2*.

Si dice che 561 è un *numero di Carmichael*: vuol dire che è pseudoprimo rispetto a “tutte” le basi.

# Monete truccate

Supponiamo che  $N$  non sia un numero di Carmichael.

Dunque

- o  $N$  è primo,
- oppure esiste  $a$ , compreso fra 1 e  $p-1$ , tale che

$$a^{N-1} \not\equiv 1 \pmod{N}.$$

Si può vedere che nel secondo caso *almeno la metà* dei numeri  $a$  compresi fra 1 e  $p-1$  soddisfano

$$a^{N-1} \not\equiv 1 \pmod{N}.$$

C'è un metodo per cercare di verificare se  $N$  è primo basato su queste idee, di cui si può dare una ottima analogia mediante il lancio di monete.

## Monete truccate

C'è una moneta che può essere

- o una moneta usuale, con una faccia “te-  
sta” e l'altra faccia “croce” ,
- o una moneta *truccata*, con due facce “te-  
sta” .

Le monete truccate rappresentano i numeri pri-  
mi.

## Lanciare le monete

Per decidere fra i due casi, posso solamente ripetere più volte una stessa operazione.

Posso prendere la moneta (senza guardare), lanciarla in aria, e una volta che è caduta posso aprire gli occhi e vedere se è venuto testa o croce.

Se viene croce anche una sola volta, so per certo che è una moneta usuale testa/croce.

Ma che succede se lancio la moneta 10 volte, e viene sempre testa?

## Come riconoscere le monete truccate

Che succede se lancio la moneta 10 volte, e viene sempre testa?

Una possibilità è che si tratti proprio di una moneta testa/testa.

L'altra possibilità è che si tratti di una moneta testa/croce. In tal caso, però, ho assistito a un evento piuttosto raro, che capita solo una volta su  $2^{10} = 1024 \approx 1000$ .

Quindi se faccio l'affermazione

“Si tratta di una moneta testa/testa”,

Non ho assoluta certezza, ma so di aver ragione 999 volte su 1000.

## Abbastanza sicuro

Se voglio essere più sicuro, posso gettare la moneta 80 volte.

Se viene sempre testa, la probabilità che sia una moneta testa/croce è una su

$$2^{80} = (2^{10})^8 \approx 1000^8 = (10^3)^8 = 10^{24}.$$

Vuol dire (più o meno) che se tutta la potenza di tutti i computer fosse stata impiegata a questo scopo, avrebbero sbagliato finora una volta sola.

## Monete truccate e numeri primi

Il paragone numeri primi/monete truccate è piuttosto stringente, dato che i risultati di

$$a^{N-1} \pmod{N}$$

per due numeri  $a$  *diversi* si possono considerare *eventi indipendenti* come due lanci di monete.

(Con buona pace dei ritardisti.)

Dunque calcolo

$$a^{N-1} \pmod{N}$$

per *tanti*  $a$ . Se viene anche solo una volta un numero  $\neq 1$ , so che il numero *non* è primo. Se viene sempre 1, ho una *ragionevole certezza* che il numero sia primo.

## Dal manuale di Maple

- The function `isprime` is a probabilistic primality testing routine.
- It returns `false` if `n` is shown to be composite within one strong pseudo-primality test and one Lucas test and returns `true` otherwise. If `isprime` returns `true`, `n` is ‘‘very probably’’ prime - see Knuth ‘‘The art of computer programming’’, Vol 2, 2nd edition, Section 4.5.4, Algorithm P for a reference and H. Reisel, ‘‘Prime numbers and computer methods for factorization’’. No counter example is known and it has been conjectured that such a counter example must be hundreds of digits long.

## Pseudoprimi forti

Un criterio più stringente rispetto a quello di Fermat è controllare che, calcolando

$$a^{(N-1)/2^s} \pmod{N}$$

per tutti gli  $s = 0, 1, \dots$  tali che l'esponente  $(N - 1)/2^s$  sia un intero, venga sempre 1, oppure il primo numero che non è 1 sia  $-1$ .

Qui solo i numeri primi  $N$  passano tutti i test: non ci sono falsi positivi. Ma comunque tutti i test non li posso fare per ragioni di tempo.

## Il test di Lucas

Se esistono un intero  $a$  e un divisore primo  $q$  di  $N - 1$  tale che

$$\begin{cases} a^{N-1} \equiv 1 \pmod{N} \\ a^{(N-1)/q} \not\equiv 1 \pmod{N}, \end{cases}$$

allora  $N$  è primo.

In sostanza, se  $N - 1$  è facile da fattorizzare, non è troppo difficile vedere se  $N$  è primo.

## Primi e giocattoli

La verifica che alcuni numeri siano primi è costata  $10^{17}$  operazioni elementari, cioè quanto fare Toy Story.

In altre parole, usare un decimilionesimo di tutta la potenza di calcolo mai dispiegata può avere risultati alquanto diversi:

- un film di 81 minuti;
- un “sì” o un “no”.

(Cfr. Deep Thought.)

## Come si fanno le potenze?

Il secondo problema di questo approccio è che finora abbiamo scritto allegramente qualcosa del tipo

$$a^k \pmod{N}.$$

Abbiamo già notato che se non si prende il resto *ogni volta che si fa un calcolo intermedio* ha poco senso prendere potenze con esponenti grandi.

Ma quanto ci costa calcolare una potenza  $a^k$ ?

## La potenza ha un suo costo

Quanto ci costa calcolare una potenza  $a^k$ ?

Se applico il metodo diretto, calcolando cioè

$$a^2 = a \cdot a,$$

$$a^3 = a^2 \cdot a,$$

...

$$a^k = a^{k-1} \cdot a,$$

sto facendo  $k - 1$  prodotti di numeri non più grandi di  $N$ , e questo ci costa

$$k \cdot \log(N)^2$$

operazioni. (Il costo di un prodotto di due numeri non più grandi di  $N$  è *al più*  $\log(N)^2$ , ma vedi dopo.)

Il “ $k$ ” non è sopportabile: ci da un algoritmo esponenziale, pensate al solito  $k = 10^{24}$ .

## Divide (per due) et impera

Per fortuna c'è un algoritmo migliore. Ad esempio per calcolare  $a^{15}$ , invece di fare i 13 prodotti

$$\begin{aligned}a^2 &= a \cdot a, \\a^3 &= a^2 \cdot a, \\&\dots, \\a^{14} &= a^{13} \cdot a,\end{aligned}$$

posso calcolare prima i tre prodotti

$$a^2 = a \cdot a, \quad a^4 = a^2 \cdot a^2, \quad a^8 = a^4 \cdot a^4,$$

e poi altri tre prodotti

$$a^{15} = a^{1+2+4+8} = a \cdot a^2 \cdot a^4 \cdot a^8,$$

per un totale di 6 prodotti contro 13.

In generale, con questo sistema ho un costo computazionale polinomiale

$$2 \cdot \log(k) \cdot \log(N)^2.$$

## Ancora monete

Come giocare a testa o croce...

per telefono?

Anna e Bruno devono fare una mano di testa o croce per telefono.

Anna trova tre numeri primi  $p, q, r$ , con  $p < q$ , e  $N = p \cdot q < r$ . Fa in modo che uno dei due fra  $p$  e  $q$  sia un *residuo quadratico* modulo  $r$ , mentre l'altro no.

Esempio (troppo banale):  $p = 17$ ,  $q = 19$ , e  $r = 2003$ . Abbiamo  $N = 17 \cdot 19 = 323 < 2003$ .

Si ha

$$261^2 = 68121 \equiv 19 \pmod{r},$$

mentre non c'è nessun  $a$  tale che

$$a^2 \equiv 17 \pmod{r}.$$

(Spiegazione fra un attimo.)

## Testa o croce per telefono

Anna comunica a Bruno  $N$  e  $r$ .

Bruno non è in grado di fattorizzare

$$N = p \cdot q,$$

cioè di calcolare  $p$  e  $q$  a partire dalla sola conoscenza di  $N$ .

Deve comunque cercare di indovinare se il residuo quadratico è il più piccolo o il più grande dei due fattori primi di  $N$ . (Questa è la scelta “testa o croce” .)

Una volta che ha fatto la sua scelta, Anna gli dice se ha indovinato o no.

Poi magari gli comunica  $p$  e  $q$ , così Bruno può controllare da sé che Anna non stia cercando di approfittarsene un'altra volta :-)

## Residui quadratici

Un numero  $b$  è un residuo quadratico modulo  $r$  se e solo se

$$b^{(r-1)/2} \equiv 1 \pmod{r}.$$

Questo ci costa  $\log(r)^3$ .

Un altro metodo consiste nel calcolare il simbolo di Jacobi/Legendre  $\left(\frac{b}{r}\right)$ , che costa quanto il massimo comun divisore fra  $b$  e  $r$ , dunque  $\log(r)^2$ .

Ad esempio

$$17^{(2003-1)/2} = 17^{1001} \equiv 2002 \equiv -1 \pmod{2003}.$$

## Certamente primi

Recentemente M. Agrawal, N. Kayal e N. Saxena hanno ideato un algoritmo polinomiale e *deterministico* per decidere se un numero  $N$  è primo.

Le complessità è

$$\log(N)^{(3+\varepsilon)(1+\mu)}$$

ove  $\varepsilon$  è piccolo a piacere, e  $\log(N)^\mu$  è il costo di una moltiplicazione. Noi abbiamo usato  $\mu = 2$ , ma  $\mu$  si può rendere vicino quanto si vuole a 1 (da sopra).

# Tartaglia/Pascal

L'algoritmo è ingegnosissimo ed *elementare*. Il punto di partenza è la semplice osservazione che un numero  $N$  è primo se e solo se

$$N \text{ divide } \binom{N}{i}$$

per ogni  $0 < i < N$ .

1										
1	1									
1	2	1								
1	3	3	1							
1	4	6	4	1						
1	5	10	10	5	1					
1	6	15	20	15	6	1				
1	7	21	35	35	21	7	1			
1	8	28	56	70	56	28	8	1		
1	9	36	84	126	126	84	36	9	1	

**Questi lucidi, e ulteriori letture. . .**

`http://www-math.science.unitn.it/~caranti/`

o semplicemente cercate “Andrea Caranti” con Google, usando l’opzione Mi sento fortunato

**Grazie dell’attenzione!**