

Block Notes Matematico

Scarabocchi sulla primalità

ing. Rosario Turco

Abstract

Questo articolo è una raccolta di appunti e considerazioni dell'autore, come rivisitazione di vecchie e nuove problematiche legate alle tecniche di primalità, usando come strumento di indagine PARI/GP. come punto di indagine di partenza vengono esaminate problematiche legate ai numeri di Carmichael, la funzione lambda e la funzione totiente.

Operatore Modulo ed algoritmi veloci

I test di primalità sono spesso una combinazione di due tecniche:

- evidenziano rapidamente se un numero è composto
- alternativamente devono avere un algoritmo rapido di primalità: deterministico (AKS, Lehmer, etc) o probabilistico (Miller-Rabin, Fermat, etc)

Oggi la "grande sfida numerica" per gli sviluppatori, i matematici e chi si interessa di hardware per tali tipi di problemi (primalità), è realizzare hardware e/o algoritmi più efficienti di quelli disponibili, anche attraverso lo sviluppo di nuovi Teoremi di Teoria dei numeri.

E' noto che per verificare la primalità di un numero primo o anche per generare le chiavi pubbliche e private (vedi RSA) nella crittografia, si fa uso dell'operatore aritmetico *modulo*, che restituisce il resto della divisione intera.

Ad esempio nel Piccolo Teorema di Fermat è noto che "se $a^{p-1} \bmod p = 1 \bmod p$ allora p è primo. Per evitare i *numeri di Carmichael* si testa per diverse basi per cui si individua un numero che *molto probabilmente* è primo.

Il problema che nasce è che se $p-1$ è un numero molto elevato allora $a^{p-1} \bmod p$ richiede un tempo notevole di calcolo.

Per tale motivo si usano "*tecniche di riduzione*" per velocizzare il calcolo, effetto che si nota soprattutto se il numero è di "grandissime dimensioni". Se ad esempio $a=2345$ $b=2345678$ $c=341$ allora $a^b \bmod c$ si può sostituire con $a^{b \bmod c} \bmod c$.

Se questo è conveniente perché adesso l'esponente è minore, col *Piccolo Teorema di Fermat* ai fini di un test di primalità non è granché conveniente; difatti da $a^{p-1} \bmod p$ si passa al test $a^p \bmod p-1$ come l'algoritmo che segue, ma abbiamo peggiorato le cose sostituendo $p-1$ con p :

Simple Method Algorithm

```
pMod(base,exponent,modulus)= local(ret=1); {  
    ret = lift(Mod(base, modulus)^exponent);  
    return(ret);  
}
```

Un'altra possibilità, se l'esponente è troppo grande, in $\text{Mod}(a,N)^n$ è di usare una riduzione del tipo $\text{Mod}(n, \varphi(N))$, dove $\varphi(N)$ è la *funzione totiente di Eulero*.

Una modalità più efficiente è di lavorare con gli esponenti in binario (tra l'altro sono possibili creazioni di processori che lavorano in questo modo): tale tecnica è nota come "*exponentiation squaring*", proposta da *Bruce Schneier nel suo Applied Cryptography, 2e, ISBN 0471117099*.

Questo metodo richiede un tempo costante indipendente dall'input, si dice che è a *complessità lineare*. Segue una mia implementazione concreta con PARI/GP.

Exponentiation squaring

```
ModExp(base,exponent, modulus) = local(ret=1,i=1); {  
    b = binary(exponent);  
    i = length(b);  
    while(exponent>0 & i>0,  
        if(b[i]==1,  
            ret = lift(Mod((ret * base), modulus));  
        );  
        exponent = exponent >> 1;  
        base = lift(Mod((base * base), modulus));  
        i--;  
        return(ret);  
    }  
};
```

La tecnica mette insieme anche l'altra tecnica dello "*Square & Multiply*" che vediamo di seguito, anche questa con esponenziazione binaria.

Square & Multiply

```
SqMod(base,exponent,modulus)= local(residue=1); {  
    b = binary(exponent);  
    len = length(b);
```

```

for(i=1,len,
  residue=(lift(Mod(residue,modulus)^2)*(base^b[i])) % modulus;
);
return(residue);
}

```

Lo 'square & multiply', usato nel RSA o altri come AES per le chiavi, potrebbe essere soggetto ad attacchi furbi per il crack delle chiavi stesse.

Esistono anche altri algoritmi basati sulla tecnica "*Montgomery reduction*" e varianti come la "*Barrett's reduction*" che usano un pre-processing e che sono interessanti per la produzione di HW ad hoc (sia basati sul binario ma preferibilmente sulle word). La "Montgomery reduction" è possibile anche sulle curve ellittiche GF(2^k).

Riduzione dell'esponente

Riassumiamo brevemente il Piccolo Teorema di Fermat (PTF) e la generalizzazione di Eulero, che qui riassumiamo brevemente:

$$a^{n-1} \bmod n = 1 \bmod n \quad (1)$$

$$a^{\varphi(n)} \bmod n = 1 \bmod n \quad (2)$$

dove $\varphi(n)$ è la funzione totiente di Eulero. Se $p = n$ allora $\varphi(n) = p-1$ e la (2) e la (1) sono equivalenti. La *funzione totiente* $\varphi(n)$ è l'insieme di numeri minori di n coprimi con n . Ad esempio $\varphi(8) = 4$, perché i numeri minori di 8 e coprimi con esso sono: 1,3,5,7. Se n è un numero primo ad esempio $\varphi(7) = 6$; infatti coprimi con 7 sono: 1,2,3,4,5,6.

La cosa funziona sia con a e n coprimi che non. Se non sono coprimi, si parla di pseudo-primalità di n rispetto alla base a .

Questo torna utile in vari tipi di problemi. Ad esempio se volessimo sapere qual è l'ultima cifra di 7^{222} ? Oppure per ridurre l'esponente nel calcolo del modulo? Qua è importante che a e n siano coprimi.

Ora $7^{222} \bmod 10$ ha 7 e 10 coprimi e $\varphi(10)=4$, difatti: {1,3,7,9}.

Dalla (2) allora è $7^4 \bmod 10 = 1 \bmod 10$ (3). Per cui tenendo conto della (3):

$$7^{222} \bmod 10 = 7^{4 \cdot 55 + 2} \bmod 10 =$$

$$7^{(4 \cdot 55)} \cdot 7^2 \bmod 10 = 1^{55} \cdot 7^2 \bmod 10 = 9 \bmod 10$$

Questa tecnica funziona se a ed n sono coprimi e si ottiene elevando la base a al valore $\varphi(n)$; difatti se $x = y \bmod \varphi(n)$ allora $a^x = a^y \bmod n$. Inoltre è quella più "rapida".

Cosa si risparmia? Facciamo il $\text{binary}(x)$:

? $\text{binary}(222)$

%1 = [1, 1, 0, 1, 1, 1, 1, 0]

? $\text{binary}(2)$

%2 = [1, 0]

Si risparmia molto, ma purtroppo è un metodo lento per il test di primalità; adesso vediamo perché.

Prima beffa

Nel caso di $2^{240} \bmod 241 = 1 \bmod 241$ possiamo fare $\varphi(241)=240$ [qua già sappiamo che 241 è primo], il problema è di avere una funzione $\varphi()$ che deve fare il MCD (algoritmo di Euclide) di tutti i numeri tra 1 e 241, contando quelli che hanno $\text{MCD}(x,241)=1$. Se il numero è molto più grande di 241, il metodo è lento e richiede, tra l'altro, una pre-elaborazione per la riduzione.

Seconda beffa

Nella primalità quanto si risparmia in bit? nulla : $2^{240} \bmod 241 = 2^{240} * 1 \bmod 241$.

Un'altra idea

L'idea allora è quella di ridurre l'esponente dividendolo per un divisore e trasformare il test $a^{n-1} \bmod n = 1 \bmod n$.

La relazione di sopra comporta che se n è il numero dispari sotto test, $n-1$ è un pari sicuramente divisibile per 2 ... Per grossi valori di n $(n-1)/2$ è intero; allora nel caso precedente avremo $2^{120} \bmod 241$.

Ma quanto abbiamo risparmiato? Facciamo anche qui il $\text{binary}(x)$.

? $\text{binary}(240)$

%9 = [1, 1, 1, 1, 0, 0, 0, 0]

? $\text{binary}(120)$

%10 = [1, 1, 1, 1, 0, 0, 0]

Si risparmia un solo bit!! Poco se vogliamo. Conviene quasi di lasciar fare al ModExp col numero intero. La discussione dimostra in modo semplice, senza tirar fuori la teoria della complessità che la velocità non si ottiene dalla riduzione dell'esponente ma dal numero di step necessari all'operazione modulo. Inoltre il test di primalità se vuole diventare più veloce necessita di ridurre molto il numero di basi di prova.

Piccolo Teorema di Fermat

Il *Piccolo Teorema di Fermat (PTF)*, dovuto a *Pierre de Fermat* e dimostrato successivamente da *Eulero* ha un fascino inesauribile pur provenendo dal lontano Seicento. E' quasi sempre tirato in ballo, anche nei nuovi metodi di primalità come AKS etc.

PTF:

Se $a^n \bmod n = a \bmod n$ allora n è primo (4)

alternativamente

Se $a^{n-1} \bmod n = 1 \bmod n$ allora n è primo. (5)

La versione di Eulero tiene conto della funzione totiente $\varphi(n)$ per cui:

$$a^{\varphi(n)} \bmod n = a \bmod n.$$

I numeri di Carmichael

Pierre de Fermat all'epoca pensava al solo test $2^n \bmod n = 2 \bmod n$, che se era verificato allora n era un numero primo. Solo che ci si accorse presto che alcuni composti, detti '*numeri di Carmichael*', superavano il test e per cui affinché il PTF fosse valido l'idea fu quello di testare n per vari valori di a , almeno un centinaio. In tal modo, visto che i numeri di Carmichael tendono al crescere di n ad essere rarefatti e molto distanti, allora il test era ancora valido e forniva una *pseudo-primalità a-SPRP*, cioè n risulta 'molto probabilmente' primo, se ha superato il test (4) per almeno cento valori della base a .

Se n molto è grande, testarne la primalità per cento basi è un test troppo lento, anche se di solito siamo abituati, purtroppo, a lasciare lavorare i computer per diversi giorni.

L'obiettivo di chi fa tali attività è trovare scorciatoie, per cercare di ottenere la primalità con poche risorse macchina e con poco tempo. Il che è difficilissimo, in molti casi impossibile!

Però a pensarci, possiamo ideare insieme un metodo. Poi dopo vediamo quanto è veloce e con quale complessità.

Il problema del PTF non è il PTF stesso ma i numeri di Carmichael. Una lista di numeri Carmichael è presente a [3]. Essa è stata individuata col PTF: cioè il numero di Carmichael è tale se supera o beffa il test del PTF (rispetto ad esempio a quello di Miller-Rabin).

Il bello della faccenda che il PTF funziona bene alla rovescia: certifica con certezza i composti ma fornisce una pseudo-primalità per i primi! Ma questo è vero per tutti i test probabilistici (Miller-Rabin etc).

Le domande che ci si può porre sono:

- 1) Potremmo rendere l'algoritmo che si basa sul PTF un po' meno di pseudo-primalità?
- 2) Possiamo fare solo un paio di test?

Forse sì, forse no. Solo due test: uno che dovrebbe ridurre di parecchio il fatto che n sia un numero di Carmichael ed uno costituito dal PTF con a=2. Un sogno finora.

Teorema di Chernick

Il Teorema di Chernick dice che i numeri di Carmichael generici sono tali da avere una forma $(6k+1)(12k+1)(18k+1)$ con ogni fattore primo.

Un'idea semplice è allora scrivere in PARI/GP un Carmichael's Test del tipo:

```
CT(number)=local(ret=0,x=0); {
/* Se lo trova esce 0 */
if( number == 561 | number == 1105, return(0));
trap( , return(1),
x=solve(k=1,number,(6*k+1)*(12*k+1)*(18*k+1) - number);
if(frac(x)!=0.0, ret=1); /* Only integer */
);
return(ret);
}
```

Fissato number l'idea è quella di risolvere una equazione cubica $(6*k+1)(12*k+1)(18*k+1) - \text{number} = 0$, se essa ha una soluzione intera abbiamo individuato un composto di Carmichael e sicuramente i tre fattori sono primi. Nello stralcio di algoritmo è stato introdotto il correttivo per i due numeri di Carmichael che l'equazione non individua cioè i due numeri 561 e 1105.

Forme generali di numeri di Carmichael

Purtroppo sono possibili forme di Carmichael anche a più di 3 fattori:

$$(7k+1)(8k+1)(11k+1)$$

$$(6k+1)(12k+1)(18k+1)(36k+1)$$

$$(18k+1)(36k+1)(108k+1)(162k+1)$$

$$(24k+1)(72k+1)(192k+1)(288k+1)$$

$$(60k+1)(90k+1)(300k+1)(450k+1)$$

$$(60k+1)(240k+1)(300k+1)(600k+1)$$

$$(42k+1)(252k+1)(588k+1)(882k+1)$$

etc.

Inoltre il test di sopra andrebbe male ad esempio con 885 (altra forma del tipo $k*\lambda(n)+1$), anch'esso numero di Carmichael.

Versione generalizzata del Teorema di Chernick

Se un numero di Carmichael n è considerato nella sua forma canonica $n = k \lambda(n) + 1$ e se qualche divisore d di k è tale che il numero $p = d \lambda(n) + 1$ sia primo, allora p_n è un altro numero di Carmichael, dove $\lambda(n)$ è la *funzione ridotta totiente*, detta anche *funzione lambda di Carmichael*.

Definizione 1

$\lambda(n)$ è il più piccolo intero positivo tale che $x^m = 1 \pmod{n}$ per tutti gli x relativamente primi a n ; in tal caso $m = \lambda(n)$.

$\lambda(n)$ è anche l'esponente del gruppo moltiplicativo Z_n .

Proprietà

La funzione lambda di Carmichael può essere calcolata nel seguente modo:

$$\lambda(1) = 1, \lambda(2) = 1$$

$$\lambda(2^2) = 2, \lambda(2^e) = 2^{(e-2)} \text{ per } e > 2,$$

$$\lambda(p^e) = (p-1) * p^{(e-1)} \text{ per } p > 2 \text{ e primo,}$$

$$\lambda(q) = \varphi(q) \text{ se } q \text{ è una potenza di un primo.}$$

$$\text{Ad esempio } \lambda(2^{e*5^f}) = 2^{(e-2)} * 5^{(f-1)}, e > 3.$$

Definizione 2

$$\lambda(p_1^{e_1} \times \dots \times p_r^{e_r}) = \text{lcm}[\lambda(p_1^{e_1}), \dots, \lambda(p_r^{e_r})], \text{ dove lcm è il minimo comune multiplo.}$$

Lemma

La definizione 1 e 2 sono equivalenti.

Difatti facciamo un esempio e consideriamo 17 che è un numero primo: dalla prima definizione e le proprietà l'ordine di un numero modulo n primo è $n-1$ ovvero 16 in questo caso; mentre dalla seconda definizione $\lambda(17) = \text{lcm}(16) = 16$.

La definizione 2 è più comoda ed è sufficiente per calcolare il lambda di un composto uguale, secondo il Teorema fondamentale dell'Aritmetica (TFA) al prodotto di r primi.

Repunit e Carmichael

Se stiamo testando dei Repunit abbiamo meno problemi: un numero Repunit più difficilmente è un numero di Carmichael! Ma può esserlo? Forse si, forse no.

Per il Teorema fondamentale dell'aritmetica un composto qualunque $C = F_1 * F_2 * \dots * F_k$

Nel caso dei Repunit $(10^n - 1)/9$ deve quindi essere:

$$10^n = 9 * F_1 * \dots * F_k + 1 \quad (6)$$

Inoltre *condizione necessaria che il Repunit sia primo è che n sia primo.*

Dalla (6) se si passa al log in base 10 si ottiene che: $n = \log(9 * F_1 * \dots * F_k + 1)$

Il termine a destra del 'segno uguale' è il logaritmo in base 10 di un pari, poichè il prodotto di dispari è dispari e aumentato di 1 dà un pari. Quindi affinché n sia dispari e intero, allora l'argomento del logaritmo deve essere una 'potenza dispari di 10' e tale che n sia anche primo.

Se, poi, lo trovate un Repunit-Carmichael siete anche da record: è il primo Repunit Carmichael che si troverebbe; infatti gli attuali Repunit candidati a primi sono al momento considerati pseudo-primi. Finora i Repunit non primi trovati erano composti normali. Una lista è tenuta dal *prof. Di Maria Giovanni* al link [4].

Se i Repunit Carmichael esistessero dovrebbero soddisfare la forma $k * \lambda(n) + 1$. Forse è proprio questo un modo per setacciarli tra i composti oppure individuare un Teorema.

Un altro metodo

Nei primi 100 milioni di numeri ci sono solo 255 numeri di Carmichael e più si va oltre e più si rarefanno. Solo che è stato dimostrato il *Teorema: "I numeri di Carmichael sono infiniti."*

L'idea delle equazioni però è sempre alla base di queste cose ed ha fatto già accendere la lampadina¹ a qualcun altro: Miller e Rabin, a cui si ispira quello che vi mostriamo.

Se nella (2) poniamo che $n-1 = 2^t * u$ dove u è dispari e $t \geq 1$, allora la (2) è equivalente a:

$$a^{u * (2t)} = \text{mod } n$$

In sostanza la (3) suggerisce di calcolare $a^u \text{ mod } n$ e poi di elevare il risultato al quadrato per t volte!

Inizialmente è:

$$x_0 = a^u \text{ mod } n$$

ad ogni step invece:

$$x_i = x_{i-1}^2 \text{ mod } n$$

per cui alla fine abbiamo l'equazione $x_t^{n-1} = 1 \text{ mod } n$ (oppure $-1 \text{ mod } n$)

Poichè almeno *algebricamente l'unità mod n ha radici non banali se e solo se il numero non è un primo*, allora è possibile creare una *funzione witness(a,n)* che se durante l'elevazione al quadrato trova una radice banale dell'unità modulo n allora restituisce subito 0, se non trova radici non banali restituisce alla fine 1. Si deve però provare con diverse basi tra 1 e n: per tutte deve essere verificato.

¹ Gli americani la chiamano "*intelligenza ah!*", cioè l'intelligenza laterale, l'intuizione, l'Eureka!

Vediamo come la funzione witness si comporta con un numero primo. Supponiamo che $n=29$ allora $n-1 = 28 = 2^2 * 7$. Sia $a=10$. Qui $2^t=2^2$ $u=7$, $t=2$ allora è:

$$x_0 = 10^u \text{ mod } n = 10^7 \text{ mod } 29 = 17$$

$$x_1 = x_0^2 \text{ mod } n = 17^2 \text{ mod } 29 = 28$$

$$x_2 = x_1^2 \text{ mod } n = 28^2 \text{ mod } 29 = 1 \text{ STOP (la radice banale sintomo di primo!)}$$

Sia $a=2$. Qui $2^t=2^2$ $u=7$, $t=2$ allora è:

$$x_0 = 2^u \text{ mod } n = 2^7 \text{ mod } 29 = 12$$

$$x_1 = x_0^2 \text{ mod } n = 12^2 \text{ mod } 29 = 28$$

$$x_2 = x_1^2 \text{ mod } n = 28^2 \text{ mod } 29 = 1 \text{ STOP (la radice banale sintomo di primo!)}$$

Se si verifica tra 1 e n è sempre lo stesso discorso.

Allora è primo.

Vediamo $n=561$ che è un numero di Carmichael, allora $n-1 = 560 = 2^4 * 35$. Supponiamo che la base scelta sia 2:

$2^t=2^4$ $u=35$, $t=4$ allora è:

$$x_0 = 2^u \text{ mod } n = 2^{35} \text{ mod } 560 = 368$$

$$x_1 = x_0^2 \text{ mod } n = 368^2 \text{ mod } 560 = 464$$

$$x_2 = x_1^2 \text{ mod } n = 464^2 \text{ mod } 560 = 256$$

$$x_3 = x_2^2 \text{ mod } n = 256^2 \text{ mod } 560 = 16$$

$$x_4 = x_3^2 \text{ mod } n = 16^2 \text{ mod } 560 = 16$$

Per $a=10$:

$$x_0 = 10^u \text{ mod } n = 10^{35} \text{ mod } 560 = 320$$

$$x_1 = x_0^2 \text{ mod } n = 320^2 \text{ mod } 560 = 480$$

$$x_2 = x_1^2 \text{ mod } n = 464^2 \text{ mod } 560 = 240$$

$$x_3 = x_2^2 \text{ mod } n = 256^2 \text{ mod } 560 = 480$$

$$x_4 = x_3^2 \text{ mod } n = 16^2 \text{ mod } 560 = 240$$

E' composto.

Ma quante basi usare? Ce lo suggeriscono due teoremi.

Teorema 1

Se n è un numero dispari non primo, allora il numero di elementi per cui la funzione witness da esito positivo sono almeno $(n-1)/2$.

Teorema 2

Per ogni intero dispari $n > 2$ e per ogni intero positivo b (base), la probabilità che Miller-Rabin(n,b) si sbagli è al massimo 2^{-b} .

Quindi la scelta di $b \geq 40$ basi potrebbe essere sufficiente, grazie ad una probabilità migliore al solo PTF su 100 basi. Del software didattico sull'argomento è presente sul sito www.gruppoeratostene.com (sorgente modulus.txt) alla sezione Software.

Domanda ulteriore: Si possono abbassare ulteriormente il numeri di basi, fino a una sola? Significa cercare un nuovo algoritmo!

La funzione lambda'

E' possibile introdurre la funzione:

$$\lambda'(n) = \text{lcm}(p_1-1, \dots, p_r-1) \quad (7)$$

Questa funzione divide i composti in due insiemi: un insieme è quello per cui $\lambda'(n) | n-1$ e l'altro insieme è quello complementare ovvero quello per cui $\lambda'(n)$ non è un divisore di $n-1$.

Se indichiamo con il simbolo $\#_2(m)$ il numero di 2 in m , ciò è equivalente a cercare il massimo valore k tale che $2^k | m$.

Miller, ad esempio, suddivide i composti in due classi A e B secondo le definizioni che seguono.

Supponiamo

$$n = F_1 * F_2 * \dots * F_m \quad (8)$$

In generale poiché vale la (7) e avendo ipotizzato la (8) allora il $\#_2(m)$ corrisponde al numero di 2 in uno dei fattori della (8), un i -esimo fattore quindi. Per cui:

Si definisce n di tipo A se, per qualche i , è:

$$\#_2(\lambda'(n)) > \#_2(F_i-1)$$

Si definisce n di tipo B se è:

$$\#_2(\lambda'(n)) = \#_2(F_1-1) = \#_2(F_2-1) = \dots = \#_2(F_m-1)$$

Tali definizioni dividono effettivamente gli m in due categorie A e B. Di circa 2370 interi tra 1 e 20000 per i quali è $\lambda'(n) | n-1$ solo 576 di essi sono di tipo A, circa il 24%. Generalmente il rapporto B:A è 3:1.

Lo studio di queste proprietà può essere interessante per sviluppi ulteriori, che mostreremo eventualmente in un articolo successivo.

Riferimenti

- [1] <http://www.numericana.com/answer/modular.htm#carmlarge>
- [2] <http://mathworld.wolfram.com/Rabin-MillerStrongPseudoprimeTest.html>
- [3] <http://oeis.org/classic/A002997>
- [4] <http://www.gruppoeratostene.com/ric-repunit/repunit.htm>
- [5] http://en.wikipedia.org/wiki/Montgomery_reduction
- [6] http://en.wikipedia.org/wiki/Elliptic_curve_cryptography
- [7] <http://en.wikipedia.org/wiki/RSA>
- [8] <http://www.it.uu.se/edu/course/homepage/security/vt09/labs/lab2>
- [9] [*funzione totiente di Eulero*](#)
- [10] [*Piccolo Teorema di Fermat*](#)
- [11] [AKS](#)
- [12] [Miller-Rabin](#)
- [13] [RSA](#)
- [14] [curve ellittiche](#)
- [15] [complessità](#)
- [16] <http://www.dm.unito.it/~cerruti/doc-html/pubbb/a1.pdf>