Test di primalità deterministico

Corso di Sicurezza - A.A. 2006/2007

Luca Calderoni

Introduzione

- Test di primalità: procedura algoritmica che, dato un numero naturale *n* in input, restituisce PRIME se *n* è un numero primo, COMPOSITE se *n* è un numero composto.
- Questo algoritmo, noto già da secoli, è oggi utilizzato in svariate procedure di grande rilevanza, tipicamente quando è necessario manipolare numeri primi giganteschi. Algoritmi noti che si appoggiano a questa procedura, nel campo della crittografia, sono ad esempio RSA, El Gamal e altri.
- Fino al 2002 si credeva che l'algoritmo (detto PRIMES) appartenesse alla classe NP, poiché tutte le varianti proposte terminavano in un numero di operazioni esponenziali nella lunghezza dell'input.
- Nel 2002, Agrawal, Kayal e Saxena hanno dimostrato che PRIMES appartiene alla classe P.

Nozioni preliminari

- Un numero *n* è primo se e solo se ammette come unici divisori 1 e se stesso.
- Per convenzione 1 non è un numero primo.
- Dato *n* numero naturale, $x = log\ n$ rappresenta la lunghezza del numero in cifre rispetto alla base scelta.
- **Teorema**:
  Se *n* è un numero composto, sicuramente avrà un fattore compreso nell'intervallo $[2, \sqrt{n}]$.
  Prova: se esistessero due fattori *p* e *q* tali che $p, q > \sqrt{n}$, risulterebbe $p*q > n$, il che è assurdo.

PRIMES: algoritmo banale

Input: n
for(k=2; k≤ √n; k++)
if(k divide n) return COMPOSITE;

return PRIME;

Questo algoritmo termina in al più $\sqrt{n}$ passi, che rispetto alla lunghezza del numero *n* sono $\sqrt{10^x}$ passi.
Per questo motivo il metodo (chiaramente deterministico) è esponenziale.

Per ovviare al notevole dispendio di tempo calcolo imposto da questo algoritmo e da altri algoritmi esponenziali ottimizzati per la risoluzione di PRIMES si utilizza solitamente una procedura polinomiale di basso costo (che però è non deterministica).

Basi di aritmetica modulare (1)

- **Z** è l'insieme dei numeri interi
- **N** è l'insieme dei numeri naturali
- $\mathbf{Z}_n$ è l'anello degli interi modulo n
- Se $k = x \pmod n$ la divisione di k per n da resto x; in tal caso si dice che k appartiene alla classe di resto x modulo n, indicata con $[x]_n$.
- (a,b) = massimo divisore comune tra gli interi a e b; se $(a,b) = 1$ si dice che a e b sono primi tra loro.
- $k = O_r(a)$ è l'ordine di a modulo r, cioè il più piccolo k che verifica la congruenza $a^k = 1 \pmod r$, dove $(a,r) = 1$.

Basi di aritmetica modulare (2)

- Dato $n \in \mathbf{N}$, l'insieme $\{k_1, \ldots, k_t\}$ si dice sistema ridotto di residui modulo n se vale che $(k_i, n) = 1$ per $i = 1 \ldots t$ e $i \neq j \Rightarrow [k_i]_n \neq [k_j]_n$
- Funzione di Eulero: $\Omega(n) = |G|$, dove G è un sistema ridotto di residui modulo n.
- Dato che se p è primo tutti i numeri da 1 a p-1 sono primi con p e appartengono a classi di resto diverse, $\Omega(p) = p-1$.

Basi di aritmetica modulare (3)

- Piccolo teorema di Fermat:

  sia *p* un numero primo, per ogni *a* in **Z** risulta:

  $a^p = a \pmod p$

  inoltre, se $(p,a) = 1$, cioè *a* non è multiplo di *p*, vale anche:

  $a^{p-1} = 1 \pmod p$

- Problema: può accadere che la seconda congruenza venga soddisfatta anche da numeri *n* non primi.

- Ad esempio, $2^{340} = 1 \pmod{341}$ anche se $341 = 11 * 31$ non è primo.

PRIMES: algoritmo non deterministico (1)

Input: n
prendi a tale che (a,n)=1 e 1<a<n
if($a^{n-1} = 1 \pmod n$) return PRIME;
else return COMPOSITE;

Per quanto *n* sia grande, l'algoritmo di potenza modulare permette di calcolare $a^{n-1} = 1 \pmod n$ in $log^3\ n$ operazioni. Dunque l'algoritmo impiega in media $O(x^3)$ operazioni per terminare, se *x* è la lunghezza dell'input.

Problema: il "testimone" *a* potrebbe mentire ==> scegliamo *k* testimoni e ripetiamo per ognuno la procedura.

Su un numero di circa 100 cifre, la probabilità di errore è circa $(1 / 10^{13})^k$, che è inferiore alla probabilità di fallimento hardware, motivo per cui questo algoritmo è solitamente usato in pratica.

PRIMES: algoritmo non deterministico (2)

- Problema: esistono dei numeri, benchè molto rari, detti numeri di Carmichael, che soddisfano la seguente proprietà: $(a,n) = 1$ per ogni a, n non primo e $a^{n-1} = 1 \pmod{n}$.
- **Teorema**:
  I numeri di Carmichael sono infiniti.
- Applicando PRIMES non deterministico ad un numero n di Carmichael, l'algoritmo fallisce qualunque sia il numero k di testimoni scelto. Benchè le probabilità di fallimento siano minime, un algoritmo non deterministico è intrinsecamente inadeguato in una procedura che deve garantire la sicurezza totale dei dati. Per questo, trovare un algoritmo polinomiale deterministico per PRIMES è un obiettivo di notevole rilevanza in teoria dei numeri e nella crittografia.

PRIMES deterministico polinomiale nozioni di base (1)

- Un campo finito è un anello commutativo unitario in cui ogni elemento non nullo ammette inverso moltiplicativo.
- Se *p* è primo, $\boldsymbol{Z}_p$ è un campo finito con *p* elementi, cioè di ordine *p*, e si infica con $\boldsymbol{F}_p$.
- $\boldsymbol{F}_p[x]$ è l'isieme dei polinomi in una indeterminata a coefficienti in $\boldsymbol{F}_p$, detto anche anello di polinomi.
- Un polinomio non nullo *p(x)* in $\boldsymbol{F}_p[x]$ si dice *irriducibile* se non può essere scritto come prodotto di due o più polinomi di grado strettamente inferiore.

PRIMES deterministico polinomiale nozioni di base (2)

- E' possibile costruire una relazione di equivalenza del tutto analoga alla congruenza modulo *n* anche sugli anelli di polinomi; nella fattispecie, $\boldsymbol{F}_p[x]/(g(x))$ è l'anello quoziente dei polinomi in $\boldsymbol{F}_p[x]$ modulo il polinomio $g(x)$.
- Dato $\boldsymbol{F}_n[x]$ indichiamo con $p(x) = q(x) \ (mod\ g(x), n)$ l'equazione $p(x) = q(x)$ nell'anello $\boldsymbol{F}_n[x]/(g(x))$. Questo corrisponde ad una doppia mappatura: prima i coefficienti dei polinomi vengono ridotti nelle classi di resto dell'anello $\boldsymbol{Z}_n$ e poi i polinomi dell'anello $\boldsymbol{F}_n[x]$ vengono mappati nell'anello quoziente generato dalla divisione per il polinomio $g(x)$.
- **Lemma**: sia p primo, $h(x)$ irriducibile su $\boldsymbol{F}_p[x]$. Allora $\boldsymbol{F}_p[x]/(h(x))$ è un campo finito di ordine p^d, dove d è il grado di $h(x)$.

PRIMES deterministico polinomiale: l'idea (1)

- Dal piccolo teorema di Fermat deriva il seguente risultato:
  se $(a,n) = 1$, $n \geq 2$, allora n è primo se e solo se:
  $(X + a)^n = X^n + a \pmod n$

  Prova: se n è primo, i coefficienti $(n \text{ su } i) * a^{n-i}$ dello sviluppo di binomio, per $0<i<n$, sono del tipo $(n! * a^{n-i})/ k!*(n-k)! = n * (n-1)! * ... = n*w = 0 \pmod n$. L'equazione collassa perciò in $X^n + a^n = X^n + a \pmod n$ che è vera per n primo. Se al contrario poniamo n composto, esiste un fattore q tale che un coefficiente $(n \text{ su } q) * a^{n-q}$ è non zero e l'equazione risulta falsa.

- Questo risultato suggerisce un metodo deterministico per valutare se n è primo o meno, e cioè verificare l'equazione suddetta.

PRIMES deterministico polinomiale: l'idea (2)

- Problema: valutare *n* coefficienti impone un tempo esponenziale, proprio come l'algoritmo PRIMES banale.
- Idea: Spostiamo la valutazione all'anello quoziente di $\boldsymbol{Z}_n[x]$, trovando un polinomio irriducibile adatto.
- Valutiamo quindi se $(X + a)^n = X^n + a \pmod{X^r - 1, n}$, nell'anello quoziente $\boldsymbol{Z}_n[x] / (X^r - 1)$.
- Valutando entrambi i membri dell'equazione in modulo $X^r - 1$, il calcolo diventa trattabile se *r* è sufficientemente piccolo.
- Problema: si può dimostrare che, sebbene ogni primo *p* soddisfi la nuova congruenza, esistono dei numeri composti che la soddisfano.

PRIMES deterministico polinomiale: l'idea (3)

- Con una adeguata scelta di *r* è possibile ripristinare la corrispondenza biunivoca tra i numeri primi e la congruenza suddetta.
- Dunque, scelto *r* opportunamente, verificare se la congruenza è valida per *a* che varia in un certo intervallo è un metodo deterministico per implementare PRIMES.
- Inoltre, dato che *r* si può calcolare in tempo polinomiale e il suo valore è limitato da una funzione logaritmica in *n*, l'algoritmo che segue non solo è deterministico, ma è anche polinomiale rispetto alla lunghezza dell'input.

PRIMES: algoritmo deterministico polinomiale

Input: n

1- *Se* $n = a^b$ *per qualche a in* $\mathbf{N}$, $b>1$, *return COMPOSITE*

2- *Calcola il più piccolo r tale che* $O_r(n) > \log^2 n$

3- *Per ogni a nell'intervallo [2,r], valuta (a,n). Se esiste un (a,n) diverso da 1, return COMPOSITE*

4- *Se* $r \geq n$, *return PRIME*

5- *Se* $r < n$, *per ogni a nell'intervallo* $[1, \text{floor}(\sqrt{\Omega}(n) * \log n)]$
verifica: se $(X + a)^n \neq X^n + a \pmod{X^r - 1, n}$, *return COMPOSITE*

6- *return PRIME*

Complessità (1)

- **Lemma:** esiste $r \leq max\,[3, roof(\log^5 n)]$ tale che $O_r(n) > \log^2 n$
- Dal lemma si evince che il punto 4 dell'algoritmo è rilevante solo per $n \leq 5.690.034$
- Al passo 1, verifico che *n* non sia una potenza banale, il che costa $O(\log^3 n)$.
- Al passo 2 calcolo *r*, in $O(\log^2 n * \log r) ==> O(\log^7 n)$
- Al passo 3, utilizzando procedure ottimizzate per il calcolo di *(a,n),* ho $O(r * \log n) ==> O(\log^6 n)$
- Il passo 4 è trascurabile

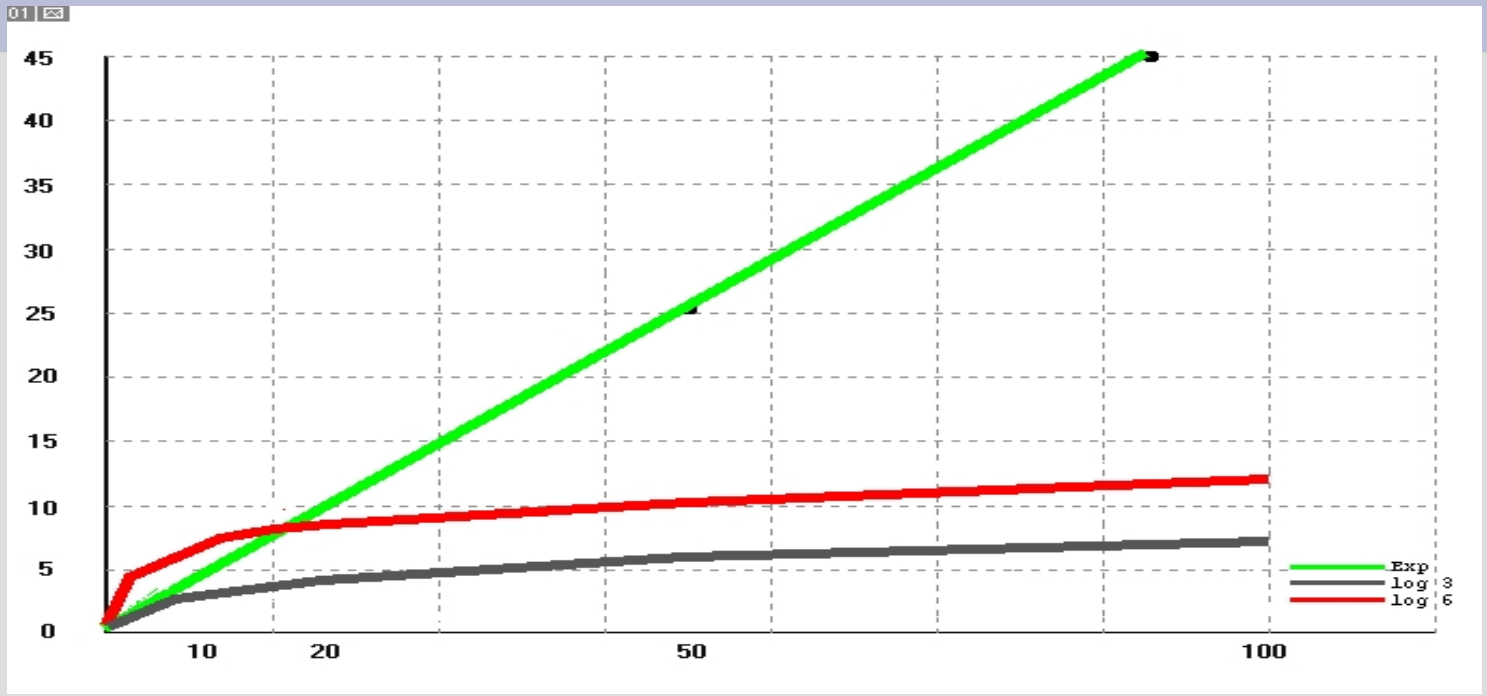
Complessità (2)

- Al passo 5, ogni equazione può essere verificata in $O(r * \log^2 n) ==> O(\log^7 n)$. Poiché verifico $roof(\sqrt{\Omega}(n) * \log n)$ equazioni, la complessità totale è $O(\log^{21/2} n)$.
- Essendo il passo 6 irrilevante, la componente che domina l'algoritmo è quella relativa al passo 5. Perciò si conclude che la complessità asintotica dell'algoritmo è $O(\log^{21/2} n)$.
- Sfruttando un risultato noto sin dall'anno 1996 grazie al contributo di C. Baker e G. Harman, è possibile ridurre il tempo calcolo dello step 2 per la valutazione di r a $O(\log^3 n)$.
- Quindi, la complessità asintotica dell'algoritmo è ridotta a $O(\log^{15/2} n)$.

Complessità (3)

- Nel 2003, H. Lenstra e C. Pomerance hanno dimostrato che è possibile ridurre ulteriormente la complessità della procedura a $O(\log^6 n)$.
- Nel seguente grafico mettiamo a confronto i due algoritmi deterministici e quello non deterministico, utilizzando una scala logaritmica sia in ascissa che in ordinata.

Complessità (4)



- Come si nota, l'algoritmo deterministico polinomiale è conveniente rispetto a quello banale solo per numeri di 14 cifre o più; tuttavia, dovendo trattare numeri dell'ordine di 100 o più cifre, l'algoritmo PRIMES deterministico polinomiale è sempre molto conveniente rispetto a PRIMES esponenziale.

Correttezza (1)

- Per dimostrare la correttezza dell'algoritmo presentato, è sufficiente provare il seguente teorema.
- **Teorema:**
  l'algoritmo ritorna PRIME se e solo se *n* è un numero primo.
  Prova: è possibile dividere la dimostrazione provando separatamente le due implicazioni.
- **Lemma:** Se *n* è primo l'algoritmo ritorna PRIME.
  Prova: Se *n* è primo, il passo 1 e il passo 3 non possono chiaramente restituire COMPOSITE; inoltre, le equazioni di cui al passo 5 non sono mai verificate, come provato in precedenza. Dunque il passo 4 o il passo 6 restituiranno PRIME.

Correttezza (2)

- **Lemma:** Se l'algoritmo ritorna PRIME, *n* è primo.
  Prova: Se l'algoritmo risponde PRIME al passo 4, *n* deve essere primo in quanto altrimenti sarebbe stato trovato un fattore al passo 3. Resta da dimostrare che se l'algoritmo risponde PRIME al passo 6, *n* è effettivamente primo.
  Supponiamo dunque che al passo 6 l'algoritmo ritorni PRIME ma *n* sia un numero composto. Di certo esisterà un fattore primo *p* di *n* tale che $p > r$, poiché altrimenti il passo 3 e 4 avrebbero decretato correttamente lo stato di *n*. Dunque $(n,r) = 1$.

- Le seguenti equazioni sono tutte verificate, dato che *p* è un fattore primo di *n* e l'algoritmo non ritorna COMPOSITE: $(X + a)^n = X^n + a \pmod{X^r - 1, p}$; $(X + a)^p = X^p + a \pmod{X^r - 1, p}$; $(X + a)^{n/p} = X^{n/p} + a \pmod{X^r - 1, p}$

Correttezza (3)

- Diciamo che *m* è introspettivo per *f(X)* se $[f(X)]^m = f(X^m)$
- Detto ciò, sfruttando i residui modulo *r* e i polinomi ciclotomici su campi finiti è possibile definire due gruppi **G** e *G* tali che:
  |G| *= t*
  $|G| \geq$ *(t+l su t-1),* dove *l =* *floor(√Ω(n) *log n)*
- **Lemma:** se *n* non è una potenza di *p* allora $|G| \leq n^{\sqrt{t}}$
- Ora, se l'algoritmo ritorna PRIME deve valere $|G| \geq$ *(t+l su t-1),* che, tramite una serie di complessi passaggi matematici può essere ricondotto alla forma $|G| > n^{\sqrt{t}}$ *;* tuttavia, se *n* non è una potenza di *p*, e questo è certo dato che altrimenti il passo 1 avrebbe risposto COMPOSITE, deve valere $|G| \leq n^{\sqrt{t}}$ per il precedente lemma.

Correttezza (4)

- Se quindi $|G| > n^{\sqrt{t}}$ deve valere $n = p^k$ per qualche k > 0.
- Tuttavia, il passo 1 avrebbe risposto COMPOSITE per ogni $k > 1$, il che implica che deve valere $k = 1$, cioè $n = p$.
- Questo prova la correttezza della seconda implicazione e dunque di tutto il teorema, certificando la correttezza dell'intero algoritmo.

Riferimenti

- *Primes is in P* (Manindra Agrawal, Neeraj Kayal, Nitin Saxena), Department of Computer Science and Engineering, Indian Institute of Technology Kanpur
- *Primality testing with gaussian periods* (H.W. Lenstra, C. Pomerance)
- *Elementi di aritmetica modulare* (M. Barnabei, F. Bonetti)