DAL TESTO
Ferragina - Luccio
CRITTOGRAFIA
Bollati Boringhieri 2007

4.3. Algoritmi randomizzati

Se un risultato è difficile da conseguire si può pensare di affidarsi al caso. Naturalmente la probabilità di avere successo non aumenta per aver osato tanto, a meno che il giorno sia particolarmente fausto o qualche evento esterno venga in nostro soccorso. Non volendo contare sulla prima situazione andiamo alla ricerca della seconda: ci occuperemo così di un nuovo paradigma algoritmico che prese originariamente il nome di «probabilistico», perché nell'analisi della complessità intervengono alcuni concetti di calcolo delle probabilità, ma che oggi è detto in gergo «randomizzato» a indicare come la struttura stessa dell'algoritmo si basi sull'intervento di eventi casuali.

Una prima utilizzazione delle sorgenti casuali all'interno di un algoritmo è volta ad aggirare alcune situazioni sfavorevoli in cui si possono presentare i dati, combinando questi con una sequenza casuale. È una situazione simile a quella in cui opera un crittografo, perché si agisce con lo scopo di confondere un avversario che in questo caso non è il crittoanalista ma il fato avverso. L'esempio più noto è Quicksort, algoritmo per l'ordinamento di dati riportato in tutti i testi. Anziché operare sui dati d'ingresso nell'ordine in cui appaiono, l'algoritmo ne costruisce una permutazione casuale e opera su questa: lo scopo è quello di garantire che la complessità sia, nel caso medio, quella prevista probabilisticamente su dati arbitrari. Gli algoritmi di questa famiglia sono familiarmente noti come *Las Vegas*: essi generano un risultato *sicuramente* corretto in un tempo *probabilmente* breve.

Più recente e interessante per noi è l'impiego della casualità a fini in qualche modo complementari ai precedenti. Gli algoritmi cui ci riferiamo, noti come *Monte Carlo*, generano un risultato *probabilmente* corretto in un

tempo *sicuramente* breve.[6] Illustreremo il nuovo paradigma in relazione al problema $\mathcal{P}_{primo}(N)$ che chiede di stabilire se un dato numero N è primo. Il motivo della scelta è duplice. Anzitutto la soluzione di $\mathcal{P}_{primo}(N)$ è importantissima in molti protocolli crittografici ma, come abbiamo visto nel paragrafo 3.4, non può essere ottenuta in tempo polinomiale per quanto a oggi noto:[7] dobbiamo quindi cercare un algoritmo efficiente per risolvere il problema pur correndo il rischio di dare una risposta errata, purché la probabilità che ciò avvenga sia trascurabile. Il secondo motivo è storico, poiché proprio a questo problema sono stati dedicati i primi algoritmi randomizzati tipo Monte Carlo.

I diversi algoritmi randomizzati per risolvere $\mathcal{P}_{primo}(N)$ differiscono sostanzialmente per le proprietà algebriche utilizzate, mentre la struttura generale è la stessa. Riportiamo per la sua semplicità ed efficienza l'*algoritmo di Miller e Rabin*, che prende il nome dai suoi inventori. Ammettiamo che N sia dispari, altrimenti è certamente composto, e poniamo $N-1=2^w z$ con z dispari (quindi $w \geq 1$ è il più grande esponente che si può dare a 2). Sia y un intero arbitrario tale che $2 \leq y \leq N-1$. Se N è primo devono essere veri i due predicati:

P_1. $mcd(N, y) = 1$
P_2. ($y^z \bmod N = 1$) **or** (esiste un valore $i, 0 \leq i \leq w-1$, tale che $y^{2^i z} \bmod N = -1$)

che derivano dalla definizione di numero primo (P_1), e da note proprietà dell'algebra modulare (P_2, vedi paragrafo 8.1). Si ha poi:

Lemma 1 (Miller e Rabin) *Se N è composto il numero di interi compresi tra 2 e $N-1$ che soddisfano entrambi i predicati P_1 e P_2 è minore di $N/4$.*

Dunque se per un intero y scelto a caso tra 2 e $N-1$ almeno uno dei predicati P_1, P_2 è falso, N è certamente un numero composto; se entrambi i predicati sono veri possiamo solo concludere che N è composto con probabilità minore di 1/4 (quindi è primo con probabilità maggiore di 3/4). Diciamo che y è un *certificato probabilistico*, o *testimone*, del fatto che N è composto. La validità del certificato può essere controllata attraverso la funzione

[6] Il termine «Monte Carlo» è impiegato in analisi numerica per indicare una famiglia di algoritmi che forniscono soluzioni approssimate a problemi numerici. Il significato che noi diamo al termine è molto diverso: come vedremo la soluzione fornita dall'algoritmo potrà essere perfettamente giusta o completamente sbagliata, anche se il secondo caso si verificherà con bassissima probabilità.

[7] A parte la banale soluzione enumerativa indicata nel paragrafo 3.3, è oggi noto un algoritmo polinomiale (vedi nota 11 del paragrafo 3.4). Tuttavia il nuovo algoritmo benché polinomiale è molto lento e si preferisce ancora la soluzione Monte Carlo riportata nel seguito.

`Verifica` (N, y): se la funzione assume valore 1 allora N è certamente composto, se assume valore 0 allora N «probabilmente» non è composto:

```
Function Verifica (N, y):
  if (P1 = false) or (P2 = false)
    then return 1
    else return 0.
```

Un certificato è interessante se la sua verifica può essere eseguita in tempo polinomiale. Anzitutto partendo dall'intero $N-1$ e dimezzandolo per $O(\log N)$ volte consecutive si calcolano i valori w e z tali che $N-1=2^w z$, con z dispari. Nella funzione `Verifica` (N, y) il calcolo di P_1 si esegue in tempo polinomiale con l'algoritmo di Euclide, ma è critico il calcolo delle successive potenze y^z, y^{2z}, ..., $y^{2^{w-1}z} = y^{(N-1)/2}$ in P_2. Infatti, se eseguite in modo diretto mediante $(N-1)/2$ successive moltiplicazioni di y per sé stesso, queste operazioni richiedono tempo esponenziale nella dimensione di N. Tuttavia esiste un meccanismo polinomiale per calcolare l'elevamento a potenza che è bene studiare subito perché sarà implicitamente e spesso utilizzato nel seguito senza più farne menzione.

Il calcolo della potenza $x = y^z \bmod s$, con y, z, s interi dello stesso ordine di grandezza, viene eseguito in tempo polinomiale se si procede per successive esponenziazioni. Si decompone anzitutto z in una somma di potenze del due: $z = 2^{w_1} + 2^{w_2} + ... + 2^{w_t}$ con $w_1 < ... < w_t$, operazione immediata se si utilizza la rappresentazione binaria di z. Si eseguono ora tutte le operazioni mod s. Si calcolano in successione le potenze y^{2^j} per $j = 1, 2, ..., w_t$ ottenendo ciascuna di esse come quadrato della precedente. Si calcola quindi x come prodotto tra gli y^{2^j} con $j = w_1, w_2, ..., w_t$. Ad esempio per $z = 45 = 1 + 4 + 8 + 32$ si eseguono le esponenzianioni: $y^2 = y \times y$, $y^4 = y^2 \times y^2$, $y^8 = y^4 \times y^4$, $y^{16} = y^8 \times y^8$, $y^{32} = y^{16} \times \times y^{16}$, e si calcola infine y^{45} come $y \times y^4 \times y^8 \times y^{32}$. In questo modo si eseguono $\Theta(\log_2 z)$ moltiplicazioni, cioè un numero lineare nella lunghezza della rappresentazione di z, il che conduce a un algoritmo complessivamente cubico.

Nell'algoritmo `Verifica` (N, y) si calcola y^z (in modulo) per successive esponenziazioni, poi si calcolano in successione i valori y^{2z}, ..., $y^{2^{w-1}z}$ elevando al quadrato y^z per $w-1$ volte consecutive. Il tempo complessivo di `Verifica` (N, y) è quindi polinomiale nella dimensione di N. Possiamo ora costruire l'algoritmo randomizzato `Test_MR` (N) per la verifica di primalità di N con il metodo di Miller e Rabin. La variabile k che appare nell'algoritmo ha un valore scelto dall'utente, che può essere fissato una volta per tutte o passato come parametro.

```
Function Test_MR (N):
  for i ← 1 to k do
    scegli y a caso tra 2 e N − 1;
    if Verifica (N, y) = 1 then return 0;
  return 1.
```

Il risultato `Test_MR`$(N) = 1$ indica che N è primo e il risultato `Test_MR` $(N) = 0$ indica che N è composto, ma le cose non stanno esattamente così. L'algoritmo può infatti arrestarsi durante il ciclo di **for** se incontra una condizione che attesta che N è composto, altrimenti esaurisce il ciclo e dichiara che N è primo. Nel primo caso il risultato è certamente corretto; nel secondo caso il risultato può essere errato (N in verità è composto), ma ciò accede solo se l'algoritmo ha impiegato k valori di y per cui il certificato fallisce. Se le k scelte di y sono state casuali e indipendenti, la probabilità di errore complessiva è data dal prodotto delle probabilità che il certificato abbia fallito ogni volta: e poiché queste sono tutte minori di 1/4, la probabilità complessiva di errore è minore di $(1/4)^k$. Se per esempio si sceglie $k = 30$ l'algoritmo può dichiarare erroneamente primo un numero composto con probabilità al più $(1/4)^{30} \simeq 10^{-18}$, certamente inferiore a qualunque altra probabilità di errore per cause che possano ragionevolmente presentarsi.[8] Dunque possiamo legittimamente accettare il risultato come corretto dopo poche ripetizioni del ciclo, sempre che le scelte di y siano state eseguite con un buon generatore pseudo-casuale.

L'analisi di complessità dell'algoritmo è elementare. Poiché si ripete al massimo k volte un ciclo contenente operazioni di complessità polinomiale $p(n)$ nella dimensione n dei dati, la complessità è di ordine $O(k \cdot p(n))$, polinomiale se k è costante o polinomiale.

Il contributo degli algoritmi randomizzati alla crittografia è determinante, quanto meno nella *generazione* di numeri binari primi grandissimi che come vedremo è alla base di molti cifrari. Poiché non è noto un algoritmo polinomiale per generare un numero primo, potremo ripetere la generazione casuale di un intero alternata a un test probabilistico di primalità finché si incontra un numero dichiarato primo. La probabilità di errore di questo meccanismo è pari a quella del test, e valgono per essa i ragionamenti già fatti. Dobbiamo però sincerarci che il numero di interi da esaminare, fino a trovarne uno primo, sia ragionevolmente basso. Ora i numeri primi godono di una proprietà forte circa la loro densità sull'asse degli interi: il numero di interi primi e minori di N tende a $N/\log_e N$ per $N \to \infty$. Ciò significa che se N è grande, in un suo intorno di lunghezza $\log_e N$ cade mediamente un numero primo. E poiché $\log_e N$ è proporzionale alla dimensione (ovvero al numero di bit) n di N, il numero di prove attese è polinomiale in n.

Si può allora definire il seguente algoritmo `Primo` (n) per la generazione di un numero primo binario di $\geq n$ bit, ove n è un valore fissato dall'utente.

[8] In realtà si può dimostrare che per una grandissima maggioranza degli interi N composti, il numero di testimoni è molto maggiore del valore $3N/4$ derivante dal lemma di Miller e Rabin. Ne segue che scegliendo per esempio $k = 30$ la probabilità di errore è estremamente più piccola di $(1/4)^{30}$. A causa di tale rapidissima convergenza il test di Miller e Rabin è diventato uno standard.

Il calcolo è innescato da un seme N di n bit, in cui i due bit estrmi sono uguali a 1 e gli altri sono generati a caso.

`Function Primo` (n):
  $N \leftarrow 1S1$, ove S è una sequenza di $n-2$ bit prodotti da un generatore binario pseudo-casuale;
  while `Test_MR` $(N) = 0$ **do** $N \leftarrow N + 2$;
  return N.

Per quanto detto precedentemente l'algoritmo `Primo` (n) controlla in media un numero di interi proporzionale a n prima di trovarne uno primo, e poiché ogni iterazione richiede tempo polinomiale in n l'intero algoritmo richiede anch'esso tempo polinomiale. Notiamo che il numero N generato conterrà al massimo $n+1$ bit perché passando da n bit a $n+1$ il valore di N raddoppia, e tra N e $2N$ cadono certamente molti numeri primi.

Concludiamo questa breve discussione sugli algoritmi randomizzati con alcune considerazioni sul loro ruolo nella gerarchia delle classi di complessità. Abbiamo visto nel paragrafo 3.4 che l'*esistenza* di un certificato polinomiale determina l'appartenenza di un problema alla classe **NP**, ma che solo per la sottoclasse **P** sono noti algoritmi polinomiali per *determinare* un certificato, ovvero per risolvere il problema. Per esempio per il problema $\mathcal{P}_{composto}(N)$, complementare di $\mathcal{P}_{primo}(N)$, un intero $\bar{y}$ per cui sia falso almeno uno dei predicati P_1, P_2 sopra riportati è un certificato polinomiale, ~~ma a tutt'oggi non è noto se sia possibile determinare $\bar{y}$ in un numero polinomiale di passi.~~ L'introduzione dei certificati probabilistici apre un modo nuovo di considerare i problemi. Senza addentrarci in questo argomento molto complesso, diamo solo una definizione che interessa il problema della primalità.

Dato un problema decisionale $\mathcal{P}$ e una sua arbitraria istanza x di lunghezza n, un certificato probabilistico y di x è un'informazione di lunghezza polinomiale in n estratta perfettamente a caso da un insieme associato a x. Il certificato è utile se esiste un algoritmo di verifica $\mathcal{A}$ polinomiale in n e applicabile a ogni coppia $\langle x, y \rangle$ che attesta che x possiede la proprietà richiesta dal problema con probabilità maggiore di 1/2, o attesta con certezza che non la possiede. Diciamo allora che $\mathcal{A}$ verifica $\mathcal{P}$ in tempo polinomiale randomizzato.

RP è la classe dei problemi decisionali verificabili in tempo polinomiale randomizzato.[9]

L'interesse per la classe **RP** deriva dalla possibilità di iterare l'applicazione dell'algoritmo di verifica fino a raggiungere una probabilità arbitra-

[9] L'acronimo **RP** sta per «random polinomiale».

riamente bassa di errore nella risposta. Per quanto detto sopra, e facendo molta attenzione perché il ragionamento è sottile, il lettore potrà convincersi che $\mathcal{P}_{composto}(N) \in \mathbf{RP}$.[10]

È possibile, ma difficile, dimostrare che anche $\mathcal{P}_{primo}(N) \in \mathbf{RP}$. L'algoritmo `Primo_MR` (N) fornisce una soluzione probabilistica per entrambi i problemi, ma i ruoli della probabilità di errore sono scambiati.

La relazione tra le principali classi diviene:

$$\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$$

e si ritiene che non valga mai l'uguaglianza. Passiamo così dai problemi che ammettono soluzione in tempo polinomiale (la classe **P**), a quelli per cui ciò può essere garantito solo in senso probabilistico (la classe **RP**), a quelli per cui neanche questo sembra possibile (la classe **NP**).

Nota: È stato recentemente trovato un algoritmo polinomiale per stabilire se N è primo. Quindi sia Primo che Composto appartengono alla classe P. Ciò non inficia i ragionamenti precedenti.

[10] Se N è composto, cioè possiede la proprietà richiesta da $\mathcal{P}_{composto}(N)$, vi è una probabilità $\geq 3/4$ che un intero y estratto a caso non verifichi uno dei predicati P_1, P_2 e che quindi l'algoritmo `Verifica` (N, y) ne scopra la natura. Se invece N è primo, cioè non possiede la proprietà richiesta dal problema, l'algoritmo `Verifica` (N, y) ne dichiara sempre la primalità.