

Crittografia

Lettore del 28/09/2020

Ore 11:15



Esempi di algoritmi numerici

- ① Calcolo del MCD di due interi
- ② Test di primalità

① EUCLIDE

$a, b \in \mathbb{Z}, \quad a \geq b, \quad a > 0, \quad b \geq 0$

MCD(a, b)

```
if ( $b == 0$ ) return  $a$ ;  
else return MCD( $b, \underline{\underline{a \bmod b}}$ );
```

I: istanza di input: a, b

$$n = |\Pi| = \Theta(\log a + \log b) = \Theta(\log a)$$

stimare il # di chiamate ricorsive

$$q \geq b$$

OSS

$$a \bmod b < \frac{q}{2}$$

$$q = \frac{a}{b} \geq 1$$

$$a = qb + a \bmod b \geq b + a \bmod b > a \bmod b + a \bmod b$$

\uparrow
 $q \geq 1$ \uparrow
 $b > a \bmod b$

$$2a \bmod b < q$$

$$a \bmod b < \frac{q}{2}$$

1) $a \bmod b$

2) $b, a \bmod b$

3) $a \bmod b, b \bmod (a \bmod b) < \frac{q}{2}$

a (primo parentesi) si ridece almeno della metà ogni 2 chiamate
 $\rightsquigarrow O(\log a)$

Costo Calcolo del prodotto: $O(\log a * \log b) = O(\log^2 a)$

Complessità:

$$T(a, b) =$$

$$T(n) = O(\underbrace{\log^3 a}_{\uparrow}) = O(n^3)$$

$$\begin{aligned} n &= |I| \\ n &= \log a \end{aligned}$$

polinomiale nella dimensione dell'insieme di input
(\hookrightarrow dei dati)

polilogaritmico nel valore dei dati

Test di primalità (versione inefficiente)

Primo (N)

```
for (i = 2; i <  $\sqrt{N}$ ; i++)  
    if ( $N \% i == 0$ ) return FALSE;  
return TRUE
```

(se N non è primo, N possiede almeno un divisore $\leq \sqrt{N}$)

$$I = N$$

$$|I| = \Theta(\log N) = n$$

iterazioni: \sqrt{N}

costo del corpo $\Theta(\log^2 N)$

$$T(n) = \Theta\left(\sqrt{N} \cdot \underline{\log^2 N}\right) = \Theta\left(2^{\frac{n}{2}} n^2\right)$$

$$\sqrt{N} = \sqrt{2^n} = 2^{\frac{n}{2}}$$

$$n = \log N$$

pseudo polinomiale

Significato algoritmico della casualità
(Kolmogorov 1960)

$h: 1111111111 \dots 1$

$h': 101101101011\dots 0$

n : lunghezza
della
sequenza

$n = 20$

$$P(h) = \left(\frac{1}{2}\right)^{20}$$

$$P(h') = \left(\frac{1}{2}\right)^{20}$$

A_h : algoritmo che genera la sequenza h

: < genera $\underset{\textcircled{n}}{1}$ >

$$|A_h| = \log n + \text{costante}$$

n lo scrivo con $\log n$ bit

#bit di A_h^{\uparrow} codificato in binario

$$|h| = n \quad |A_h| = \Theta(\log n)$$

Intuizione: Una sequenza binaria è casuale se non ammette un algoritmo di generazione la cui rappresentazione binaria sia più corta di h .

$A_h = \text{Print} \ " \dots \text{sequenza } h' \dots \dots \ "$

$$|A_h| \geq |n| = |h'|$$

$$S_i(p) = h$$

Sistemi di calcolo

sono un'infinità numerabile $S_1, S_2, \dots, S_i, \dots$

S_i : programma che genera la sequenza h nel sistema S_i

DEF Complessità di Kolmogorov di h nel sistema S_i è

$$K_{S_i}(h) = \min \{ |P| \mid S_i(p) = h \}$$

OSS

se la sequenza h non obbedisce ad alcuna legge semplice,
il più caro programma che la genera contiene le
sequenze al suo interno, e la genera trasferendole in output

$$K_{S_i}(h) = |h| + \text{costante};$$

↳ costante che dipende da S_i (ma non da h)
e rappresenta le perte di programma che
trasformare h nel risultato / output

→ Consideriamo

$\overset{\circlearrowleft}{S_u}$

S_u : sistema di calcolo universale

può simulare qualsiasi altro sistema di calcolo

$$P: S_i(p) = h$$

P: programma che genera h in S_i

$$S_u(\langle i, p \rangle) = S_i(p) = h$$

$q = \langle i, p \rangle$ programma che genera h in S_u

$$|q| = |<i, p>| = |i| + |p| = \log_2 i + |p|$$

q: proposta
di generare h
in S_h

\downarrow dipende da i, ma non da h

f_h

F_i

$$k_{S_h}(h) \leq k_{S_i}(h) + c_i$$

=: volte per le sequenze generate per simulazione di S_i , non
crescendo per S_h altrimenti poi "brca"

<: volte per sequenze generalibili con parametri più corti (ad esempio
per simulazione di un altro sistema $S_j \neq S_i$)

DEF La complessità di Kolmogorov di una sequenza h
 $\bar{k}(h) = k_{S_h}(h)$

DEF. (Sequenza casuale)

Una sequenza h è casuale se

$$k(h) \geq |h| - \lceil \log_2 |h| \rceil$$

OSS

la casualità è una proprietà delle sequenze

(non dipende dalla sorgente che ha generato la sequenza)

CONTEGGIO delle SEQUENZE

$\forall n, \exists$ sequenze casuali (secondo kolmogorov) di lunghezza n .

Dim n $S = 2^n$ # sequenze binarie di lunghezza n

$T = \#$ sequenze di lunghezza n NON casuali

(obiettivo: $T < S$)

$N = \#$ sequenze binarie di lunghezza $< n - \lceil \log_2 n \rceil$

$$= \sum_{i=0}^{n-\lceil \log_2 n \rceil} 2^i = 2^{n-\lceil \log_2 n \rceil} - 1$$

tra queste N sequenze ci sono anche i raggruppamenti che generano le T sequenze non corrotte di lunghezza n .



$$T \leq N < S \Rightarrow T < S$$



$\forall n$, \exists certamente sequenze corrotte di lunghezza n
e sono estremamente più numerose di quelle non corrotte.

$$\frac{T}{S} \leq \frac{N}{S} = \frac{2^{n-\lceil \log_2 n \rceil} - 1}{2^n} = \frac{1}{2^{\lceil \log_2 n \rceil}} - \frac{1}{2^n} < \frac{1}{2^{\lceil \log_2 n \rceil}}$$
$$\lim_{n \rightarrow +\infty} \frac{T}{S} = 0$$

Th

Il problema di stabilire se una sequenza orbitaria è casuale (secondo K.)
è INDECIDIBILE.

Dimostrazione

per assurdo; supponiamo esiste un algoritmo RANDOM t.c.

$$\text{RANDOM}(h) = \begin{cases} 1 & h \text{ casuale} \\ 0 & h \text{ non casuale} \end{cases}$$

→ possiamo costruire l'algoritmo

PARADOSSO

for binary $h \leftarrow 1$ to ∞ do

 if ($|h| - \lceil \log |h| \rceil > |P|$)

 &&

 RANDOM(h) = 1) return h ;

①

②

}

// genera tutte le
// sequenze binarie in
// ordine di lunghezza crescente

P: sequenze binarie da rappresentare Paradosso e Random
↳ rappresentare il programma complesso
PARADOSSO + RANDOM

$$|P| = |\text{PARADOSSO}| + |\text{RANDOM}|$$

$|P|$ è una costante che non dipende da h

h appare nel programma come nome di variabile

(il valore è registrato nella memoria o fuori del programma)

→ Paradosso richiede come risultato la prima sequenza casuale h di lunghezza t.c.

$$|h| - \lceil \log_2 h \rceil > |P|$$

∃ sequenze casuali di qualsiasi lunghezza, certamente ne esistono una che soddisfa ① e ②

Mo:

①

il programma rappresentato da P è brave
e genera h

→ h non è costante

$$(k(h) \leq |P| < |h| - \lceil \log_2 |h| \rceil)$$

\uparrow
(Pgenera h)

②

h è costante

Y