Il presente <u>Fascicolo B</u> contiene i seguenti argomenti

1) "Running time" dell'algoritmo euclideo.

2) "Running time" della potenza modulare (Powermod, algoritmo "square and multiply").

"Running time" dell'algoritmo euclideo

Dati due interi positivi a, b con $a > b$, vogliamo calcolare (a,b) mediante l'algoritmo euclideo. Vale, a questo proposito, il seguente importante risultato.

Teorema

Indichiamo con $\mathcal{N}$ il numero di operazioni elementari (nel nostro caso divisioni con resto) necessarie per passare dall'input (la coppia a, b) all'output (a,b). Vale allora la seguente maggiorazione

$$1) \quad \mathcal{N} \leq 2\lg_2 b + 2$$

Dimostrazione

Ponendo, per comodità di scrittura, $r_0 = a$ ed $r_1 = b$, ricordiamo che l'algoritmo euclideo è il seguente procedimento iterativo

$$2) \begin{cases} r_0 = r_1 q_1 + r_2 \\ r_1 = r_2 q_2 + r_3 \\ \text{---} \\ r_{J-2} = r_{J-1} q_{J-1} + r_J \\ \text{---} \\ r_{n-2} = r_{n-1} q_{n-1} + r_n \\ r_{n-1} = r_n q_n + 0 \end{cases}$$

dove la successione dei resti è strettamente decrescente, cioè

$$r_0 > r_1 > r_2 > \cdots > r_{n-1} > r_n$$

e, come sappiamo,

$$r_n = (a,b).$$

Dato che $q_j \geq 1$ $\forall j = 1, \dots, n$ si ha

3) $$r_{j-2} = r_{j-1} q_{j-1} + r_j \geq r_{j-1} + r_j > 2 r_j$$

da cui segue

4) $$r_j < \frac{1}{2} r_{j-2}$$

Ponendo $j = n$ ed iterando la 4) otteniamo

5) $$r_n < \frac{1}{2} r_{n-2} < \frac{1}{2^2} r_{n-2\cdot 2} < \frac{1}{2^3} r_{n-2\cdot 3} < \dots$$

Quando ci fermiamo?
Distinguiamo due casi, n pari ed n dispari.

- n pari
  Nella catena 5) ci fermiamo quando $n-2k=2$, cioè per $k = \frac{n-2}{2}$, ottenendo

6) $$r_n < \frac{1}{2} r_{n-2} < \frac{1}{2^2} r_{n-2\cdot 2} < \dots < \frac{1}{2^{(n-2)/2}} r_2$$

- n dispari
  Nella catena 5) ci fermiamo quando $n-2k=1$, cioè per $k = \frac{n-1}{2}$, ottenendo

7) $$r_n < \frac{1}{2} r_{n-2} < \frac{1}{2^2} r_{n-2\cdot 2} < \dots < \frac{1}{2^{(n-1)/2}} r_1$$

Dato che $\frac{1}{2^{(n-2)/2}} > \frac{1}{2^{(n-1)/2}}$ e $r_1 > r_2$, da 6) e 7) segue che la diseguaglianza

$$8) \qquad r_n < \frac{1}{2^{(n-2)/2}} \cdot r_1$$

vale per ogni n.

Ma $r_n \geq 1$ e quindi dalla 8) otteniamo

$$9) \qquad 1 < \frac{1}{2^{(n-2)/2}} \cdot r_1$$

da cui

$$10) \qquad 2^{(n-2)/2} < r_1$$

Considerando il logaritmo in base 2 di ambo i membri di 10) si ha

$$11) \qquad (n-2)/2 < lg_2(r_1)$$

Ricordando che $r_1 = b$, dalla 11) segue

$$12) \qquad n \leq 2\, lg_2(b) + 2$$

Ma (vedi formule 2)), il numero n rappresenta proprio il numero di divisioni con resto che abbiamo fatto per passare dall'input (cioè a, b) all'output (cioè $r_n = (a, b)$), e quindi $n = N$. Perciò la 12) dimostra la tesi, cioè la formula 1).

"Running time" della potenza modulare (Powermod, algoritmo "square and multiply")

In crittografia è fondamentale il calcolo di potenze modulari, cioè del tipo

$$13)\quad b^n \equiv r \pmod{m}$$

dove b, n ed m sono assegnati e si deve calcolare r (che è il minimo residuo positivo mod m).
Un metodo "veloce" per calcolarlo è il seguente.
Consideriamo la rappresentazione binaria dell'esponente n, cioè

$$14)\quad n = c_k 2^k + c_{k-1} 2^{k-1} + \dots + c_1 2^1 + c_0 \cdot 2^0$$

con $c_k = 1$, $c_j = 0, 1$ per $j = 0, 1, 2 \dots, k-1$.

Si ha evidentemente

$$15)\quad b^n = b^{c_k 2^k + c_{k-1} 2^{k-1} + \dots + c_1 2^1 + c_0 \cdot 2^0} =$$

$$\equiv \left(b^{2^k}\right)^{c_k} \left(b^{2^{k-1}}\right)^{c_{k-1}} \cdots \cdot \left(b^{2^1}\right)^{c_1} \cdot \left(b^{2^0}\right)^{c_0}$$

Come prima cosa si calcola $b^{2^0} = b^1$ e si pone

$$p_0 = \left(b^{2^0}\right)^{c_0} = \begin{cases} 1 \text{ se } c_0 = 0 \\ b^{2^0} = b \text{ se } c_1 = 1 \end{cases}.$$

Si calcola poi $b^{2^1} = b^2$, si riduce mod m, si pone

$$(b^{2^1})^{c_1} = \begin{cases} 1 \text{ se } c_1 = 0 \\ b^{2^1} (\text{mod } m) \text{ se } c_1 = 1 \end{cases} \quad \text{e} \quad (b^{2^1})^{c_1} \cdot (b^{2^0})^{c_0} = p_1,$$ sempre

riducendo mod m.

Si calcola $b^{2^2} \equiv (b^{2^1})^2 (\text{mod } m)$, si pone

$$(b^{2^2})^{c_2} = \begin{cases} 1 \text{ se } c_2 = 0 \\ b^{2^2} (\text{mod } m) \text{ se } c_2 = 1 \end{cases} \quad \text{e} \quad (b^{2^2})^{c_2} \left[(b^{2^1})^{c_1} \cdot (b^{2^0})^{c_0} \right] =$$

$$= (b^{2^2})^{c_2} \cdot p_1 = p_2$$

Si prosegue in questa maniera fino al coefficiente k, ottenendo quindi il risultato finale.

Quante operazioni elementari occorrono?
Ad ogni passo si debbono fare un quadrato (con relativa riduzione mod m), una trascrizione, un prodotto (con relativa riduzione mod m). Si tratta di k passi (se non contiamo il calcolo di $b^{2^0} = b$). Dato che

$$16) \quad 2^k \leq n = c_k 2^k + c_{k-1} 2^{k-1} + \dots + c_1 2^1 + c_0 \leq$$

$$\leq 2^k + 2^{k-1} + \dots + 2^1 + 1 = \frac{2^{k+1} - 1}{2 - 1} = 2^{k+1} - 1 < 2^{k+1}$$

dalla (16) segue $k \leq \lg_2 n < k+1$ e quindi il numero di operazioni elementari è

$$17) \quad N \leq 4 \lg_2 n$$

che è "polynomial-time" nell'input n.