Soluzioni del compito di Crittografia a chiave pubblica (CR1)

5 Aprile 2001 (2 ore)

1. **Se $n \in \mathbf{N}$, sia $\sigma(n)$ la somma dei divisori di n. Supponiamo che sia nota la fattorizzazione (unica) di $n = p_1^{\alpha_1} \cdots p_s^{\alpha_s}$. Calcolare il numero di operazioni bit necessarie per calcolare $\sigma(n)$. (**_Suggerimento: Usare il fatto che σ è una funzione moltiplicativa e calcolare una formula per_ $\sigma(p^\alpha)$)

   Soluzione. Dalla moltiplicatività di σ segue che $\sigma(n) = \sigma(p_1^{\alpha_1}) \cdots \sigma(p_1^{\alpha_1})$. Inoltre $\sigma(p^a) = \sum_{i \leq a} p^i = \frac{p^{a+1}-1}{p-1}$. Notare che $\mathbf{Time}(\sigma(p^a)) = O(a^2 \log^2 p)$ infatti l'operazione che richiede il maggior numero di operazioni bit è il calcolo di p^{a+1}. Questo implica che per calcolare $\sigma(p_1^{\alpha_1}), \ldots, \sigma(p_s^{\alpha_s})$ sono necessarie $O(\sum_{i \leq s} \alpha_i^2 \log^2 p_i) = O(\log^2 n)$ operazioni bit. Quindi, calcolando $\sigma(n)$ moltiplicando gli s numeri $\sigma(p_1^{\alpha_1}), \ldots, \sigma(p_s^{\alpha_s})$ che sono tutti minori di n^2, otteniamo (usando la formula standard) $\mathbf{Time}(\sigma(n)) = O(s^2 \log^2 n)$. Infine da $s = O(\log s)$ si ha $\mathbf{Time}(\sigma(n)) = O(\log^4 n)$. Con un pò di lavoro in più si può anche mostrare che $\mathbf{Time}(\sigma(n)) = O(\log^3 n)$.

2. **Mostrare che le moltiplicazioni nell'anello quoziente $\mathbf{Z}/n\mathbf{Z}[x]/(x^d)$ si possono calcolare in $O(\log^2 n^d)$ operazioni bit mentre le addizioni in $O(\log n^d)$ operazioni bit.**

   Soluzione. Scriviamo
   $$f_1 = \sum_{j=0}^{d-1} a_j x^j, \;\; f_2 = \sum_{j=0}^{d-1} b_j x^j \in \mathbf{Z}/n\mathbf{Z}[x]/(x^d).$$
   Allora $f_1 + f_2 = \displaystyle\sum_{j=0}^{d-1} (a_j + b_j) x^j$. Pertanto $\mathbf{Time}(f_1 + f_2)$=$\mathbf{Time}(a_0 + b_0, \ldots, a_{d-1} + b_{d-1}) = O(d \log n)$.
   Invece se scriviamo $f_1 f_2 = \sum_{k=0}^{d-1} c_k x^k$, allora $c_k = \sum_{i+j=k} a_i b_j$. Inoltre per ogni $k = 0, \ldots, d-1$, $\mathbf{Time}(c_k)) = O(d \log^2 n)$ (infatti si tratta di k moltiplicazioni e $k-1$ somme in $\mathbf{Z}/n\mathbf{Z}$. Infine
   $$\mathbf{Time}(f_1 f_2) = \mathbf{Time}(c_0, \ldots, c_{d-1})) = O(d^2 \log^2 n).$$

3. **Dato il numero binario $n = (10011100101)_2$, calcolare $[\sqrt{n}]$ usando l'algoritmo delle approssimazioni successive (Non passare a base 10 e non usare la calcolatrice!)**

   Soluzione. Applicando l'algoritmo delle approssimazioni successive otteniamo:
   $$\begin{array}{ll} x_0 = (100000) = 2^5 & \\ x_1 = (100000) & x_0 + 2^4 > n; \\ x_2 = (100000) & x_1 + 2^3 > n; \\ x_3 = (100000) & x_2 + 2^2 > n; \\ x_4 = (100010) & x_3 + 2 < n; \\ x_5 = (100011) & x_4 + 1 < n; \end{array}$$
   Quindi $[\sqrt{n}] = 100011$.

4. **Calcolare il massimo comun divisore tra 240 e 180 utilizzando sia l'algoritmo euclideo che quello binario. Calcolare anche l'identità di Bezout.**

   Soluzione. L'esecuzione dell'algoritmo di Euclide per calcolare $(240, 180)$ è
   $$240 = 180 + 60; \quad 180 = 3 \cdot 60 + 0; \quad (240, 180) = 60.$$
   mentre quella di quello binario è
   $$(240, 180) = 4(60, 45) = 4(15, 45) = 4((45-15)/2, 15) = 4(15, 15) = 4(0, 15) = 4 \cdot 15 = 60.$$
   L'identità di Bezout è $60 = 240 - 180$.

5. **Dimostrare che se $n = p_1 \cdots p_{20}$ è un intero privo di fattori quadratici, e $f(x) \in \mathbf{Z}/n\mathbf{Z}[x]$ ha grado 10, allora la congruenza $f(x) \equiv 0 \bmod n$ è risolvibile se e solo se lo sono le 20 congruenze** $\begin{cases} f(x) \equiv 0 \bmod p_1 \\ \vdots \\ f(x) \equiv 0 \bmod p_{20}. \end{cases}$ **Dedurre che la prima congruenza $f(x) \equiv 0 \bmod n$ ha al più 10^{20} soluzioni. Sapreste dare un esempio in cui le soluzioni sono esattamente 10^{20}?**

   Soluzione. (Se) Per ogni $i = 1, \dots, 20$, sia α_i una soluzione di $f(x) \equiv 0 \bmod p_i$ e sia $\alpha \in \mathbf{Z}/n\mathbf{Z}$ una soluzione del sistema di congruenze

$$\begin{cases} X \equiv \alpha_1 \bmod p_1 \\ \vdots \\ X \equiv \alpha_{20} \bmod p_{20} \end{cases}.$$

   Si ha che $f(\alpha) \equiv f(\alpha_i) \equiv 0 \bmod p_i$. Quindi $f(\alpha)$ è divisibile per $p_1, \dots, p_{20}$ e quindi per n.
   (Solo se) Se $\alpha \in \mathbf{Z}/n\mathbf{Z}$ è una soluzione di $f(x) \equiv 0 \bmod n$, allora $\alpha \bmod p_i$ è una soluzione di $f(x) \equiv 0 \bmod p_i$. per ogni $i = 1, \dots, 20$.

   Se $f(x) = x(x-1)(x-2)\dots(x-9)$ e $n = 11 \cdot 13 \cdots 89$ (il prodotto dei 20 primi tra 11 e 89). Allora f ha esattamente 10^{20} soluzioni modulo n. Infatti per ciascuna delle 10^{20} scelte $i_1, \dots, i_{20} \in \{0, \dots, 9\}$, il sistema di congruenza

$$\begin{cases} X \equiv i_1 \bmod p_1 \\ \vdots \\ X \equiv i_{20} \bmod p_{20} \end{cases}$$

   da luogo (per il Teorema cinese dei resti) a esattamente una soluzione modulo n.

6. **Illustrare l'algoritmo dei quadrati successivi in un gruppo analizzandone la complessità. Fare anche un esempio.**

   Soluzione. Sia G un gruppo (o anche un monoide) abeliano indicato moltiplicativamente e sia $g \in G$. Per ogni $k \in \mathbf{N}$, idichiamo con $\mathtt{QS}(g, k)$ l'elemento $g^k \in G$. Allora scriviamo l'espansione binaria $k = \sum_{j=0}^{t} \epsilon_j 2^j$ e

$$\begin{array}{rcl} \mathtt{QS}(a, k) & = & A = g, B = 1 \\ & & \mathtt{for}\ j = 0\ \mathtt{to}\ t\ \mathtt{do} \\ & & \mathtt{if}\ \epsilon_j = 1\ \mathtt{then}\ B = B \cdot A\ \mathtt{fi} \\ & & A = A^2 \\ & & \mathtt{rof} \\ & & \mathtt{Print}(B). \end{array}$$

   Se T_G è il massimo numero di operazioni bit necessarie per moltiplicare due elementi di G, allora $\mathbf{Time}(\mathtt{QS}(g, k)) = O(tT_G)$.
   Esempio: $G = \mathbf{Z}/5\mathbf{Z}$, allora $2^5 \bmod 5 = 2 \cdot (2^2)^2 \bmod 5 = 2$ $(2^2 = -1, (2^2)^2 = -1)$.

7. **Mettere in ordine di priorità e spiegare il significato di ciascuna delle seguenti operazioni:**

$$x\sim,\quad x \wedge y,\quad x\&y,\quad x{+}{+},\quad x\backslash y,\quad x = y,\quad x\%y,\quad x|y,\quad x \ll n.$$

   Soluzione. Ordine di priorità: 1 massima, 5 minima.

   1. $x++$: $x = x + 1$;
   2. $x = y$: assegna il valore dell'espressione y ad x;
   3. $x\tilde{}$: trasposta di x, vettore o matrice;
      $x\hat{}y$: x elevato ad y;

4. $x \backslash y$: restituisce q t.c. $x = q * y + r$ (quoziente euclideo);
   $x\%y$: restituisce r t.c. $x = q * y + r$ (resto euclideo);
   $x << n$: shift sinistro di x di n bits, pari a $x * 2^n$;
5. $x\&y$: x AND y;
   $x|y$: x OR y;

- 8. **Si dia la definizione di pseudo primo forte in base 2 e si mostri che se $n = 2^\alpha + 1$ è pseudo primo forte in base 2, allora $2^{2^\beta} \equiv -1 \bmod n$ per qualche $\beta < \alpha$.**

  Soluzione. $n \in \mathbf{N}$ si dice *pseudo–primo forte in base* 2 se, posto $n - 1 = 2^\alpha \cdot t$ con t dispari, si ha che

  $$(2^t, 2^{2t}, \ldots, 2^{2^\alpha t}) \equiv_n \begin{cases} (1, 1, \ldots, 1) \\ (2^t, 2^{2t}, \ldots, 2^{2^\beta t}, -1, 1, \ldots, 1) \end{cases}$$

  Nel caso in cui $n = 2^\alpha + 1$, allora $t = 1$ e quindi la prima possibilità non si presenta e quindi $\exists \beta < 1$ t.c. $2^{2^\beta} \equiv -1 \bmod n$.

- 9. **Scrivere un programma in Pari che produca due vettori v e w. In cui v contiene i primi** 100 *pseudo-primi composti* **in base** 2 **e il secondo i primi** 100 *pseudo primi di Eulero composti* **in base** 2**.**

  Soluzione.

```
n=1;
v=VECTOR(100,i,   UNTIL(k!=1 &   QUADSUCC(2,n-1,n)==1,
                                 n+=2;
                                 k=MATSIZE(FACTOR(n))[1]);
                  n);

n=1;
w=VECTOR(100,i,   UNTIL(k!=1 &   (Mod(QUADSUCC(2,(n-1)>>1,n),n)== Mod(JAC(2,n),n)),
                                 n+=2;
                                 k=MATSIZE(FACTOR(n))[1]);
                  n);
```

- 10. **Implementare RSA utilizzando il sistema Pari e creando tre funzioni distinte (una per generare le chiavi, una per cifrare e una per decifrare).**

  Soluzione.

  `KEYGEN` (n)

  – INPUT: n =numero di bits del modulo pubblico;
  – OUTPUT: stampa le chiavi pubbliche (e, m), ritorna la chiave privata d;

  `CIFRA` (P, e, M)

  – INPUT: P =messaggio in chiaro (formato numerico), (e, M) chiave pubblica;
  – OUTPUT: C =messaggio cifrato (formato numerico);

  `DECIFRA` (C, d, M)

  – INPUT: C =messaggio cifrato (formato numerico), (d, M) chiave privata;
  – OUTPUT: P =messaggio in chiaro (formato numerico);

```
KEYGEN(n,p,q,M,e,d,phi_M)   =
                            p=NEXTPRIME(RANDOM(1<<(n>>1)));
                            q=NEXTPRIME(RANDOM(1<<(n>>1)));
                            M=p*q;
                            phi_M=EULERPHI(M);
                            UNTIL(GCDBINARIO(e,phi_M)==1,e=RANDOM(phi_M));
                            d=LIFT(Mod(e,phi_M)^(-1));
                            PRINT(e,M);
                            RETURN(d);

CIFRA(P,e,M)=
                            RETURN(QUADSUCC(P,e,M));

DECIFRA(C,d,M)=
                            RETURN(QUADSUCC(C,d,M));
```