Introduzione a Pari/GP

Progetto Lauree Scientifiche 2005-2006

di Andrea Centomo

Liceo Statale "F. Corradini" di Thiene (VI)

Email: andrea.centomo@istruzione.it

18 gennaio 2006

Prefazione

Queste pagine sono state realizzate per gli studenti frequentanti le lezioni del Progetto Lauree Scientifiche per il Veneto dedicate alla Crittografia. Buona parte del materiale è una mera rielaborazione, in chiave semplificata, delle dispense messe gentilmente a disposizione dal dott. Alessandro Languasco per questo corso.

Quanto esposto richiede, almeno in parte, la conoscenza di concetti sviluppati durante le lezioni del corso di Crittografia.

Introduzione

Come suggerisce il nome Pari/GP è una combinazione di due ambienti:

1. Pari: una libreria di routines scritte in C, orientata ad applicazioni di Teoria dei Numeri, caratterizzata da una notevole velocità di esecuzione;
2. GP: un interprete che consente l'accesso alla libreria estesa Pari attraverso una serie di comandi da terminale. Ciò consente di disporre di un semplice ambiente di programmazione nel linguaggio di scripting[1] GP.

pari/GP è software libero (rilasciato con licenza gpl) e quindi usabile e ridistribuibile per qualsiasi scopo. Il fondatore del progetto pari/GP è H. Cohen dell'Università di Bordeaux 1 e il software è attualmente mantenuto da K. Belabas della stessa Università. Trattandosi di software libero è importante ricordare che tutte le informazioni e la documentazione per l'uso del programma sono disponibili in rete e, in particolare, al sito ufficiale del progetto

`http://pari.math.u-bordeaux.fr/`

Lanciare Pari/GP

Per lanciare Pari/GP è sufficiente aprire un terminale e digitare `gp`. Dopo un attimo verrà visualizzata la seguente schermata di benvenuto.

```
                GP/PARI CALCULATOR Version 2.2.11 (alpha)
           i686 running linux (ix86 kernel) 32-bit version
   compiled: Dec 29 2005, gcc-3.4.3 (Mandrakelinux 10.2 3.4.3-7mdk)
          (readline not compiled in, extended help available)

                  Copyright (C) 2000-2005 The PARI Group

PARI/GP is free software, covered by the GNU General Public License,
and comes WITHOUT ANY WARRANTY WHATSOEVER.
```

1. Per chi lo desidera è disponibile il compilatore gp2c che consente di trasformare uno script GP in un codice C.

```
Type ? for help, \q to quit.
Type ?12 for how to get moral (and possibly technical) support.

parisize = 4000000, primelimit = 500000 ?
```

Osserviamo in primo luogo che tramite il comando `?` si può accedere al manuale in linea e alle sue diverse sezioni, mentre per uscire dal programma è sufficiente digitare il comando `\q`.

Allo startup viene allocata una certa quantità di memoria e vengono eseguiti dei calcoli preliminari. Le variabili `parisize` e `primelimit` indicano rispettivamente che sono stati allocati, per il funzionamento del programma, 4000000 byte e che sono stati calcolati tutti i numeri primi fino a 500000. Questi valori influenzano il comportamento di diverse funzioni aritmetiche disponibili. Se tali valori non fossero sufficienti è possibile:

a) eseguire GP ricorrendo al comando `gp -s NUMERO` oppure, durante l'esecuzione di GP, usare il comando `allocatemem` che raddoppia la quantità di memoria preesistente;

b) eseguire GP ricorrendo al comando `gp -p NUMERO` per aumentare il numero di primi precalcolati.

1 Esempi elementari

Iniziamo risolvendo con PARI alcuni semplici esercizi di aritmetica.

Esempio 1. Posto $x = 3456789$ calcolare il Massimo Comun Divisore tra 23457 e x^3.

Ricorrendo alle funzioni predefinite possiamo eseguire immediatamente questo calcolo:

```
? x=3456789
%1 = 3456789

? gcd(23457,x^3)
%2 = 21
```

Il comando `x=3456789` assegna alla variabile `x` il corrispondente valore. Mentre la funzione predefinita `gcd` (Greatest Common Divisor) permette il calcolo del Massimo Comun Divisore cercato. Osserviamo che ogni risposta che PARI restituisce viene numerata attraverso un'etichetta del tipo `%n` dove `n` indica il numero di risposta.

Esempio 2. Dire se il numero 2354567 primo e in caso negativo scomporlo in fattori primi.

Utilizzando la funzione predefinita `isprime` possiamo vedere se il numero dato è primo:

```
? isprime(2354567)
%3 = 0
```

La risposta `0` è negativa. Possiamo allora procedere alla determinazione della scomposizione del numero in fattori primi utilizzando la funzione `factor`:

```
? factor(2354567)
%4 = [743 1]
     [3169 1]
```

da cui si ha che $235467 = 743 \cdot 3169$.

Esempio 3. Con riferimento all'esempio 1 calcolare x^{35}.

Possiamo calcolare immediatamente questa potenza:

```
%5 = 713757190005104555760075970012707062016650950 83
```

```
9854484099112071666416158997491385776748251944693234
4166105517766678674173034927927387709284305385367531
3743825384074810365829591053985481573296552445269477
228450400641174421655031 49
```

L'esempio ci mostra come il programma acceda ad una aritmetica intera a lunghezza arbitraria.

Esempio 4. Scrivere uno script per il calcolo del Massimo Comun Divisore di due numeri.

Uno degli aspetti più interessanti dell'ambiente PARI/GP è che esso può essere programmato includendo delle funzioni nuove definite dall'utilizzatore. Anche se la funzione predefinita `gcd` permette il calcolo del Massimo Comun Divisore è istruttivo scrivere uno script che esegua lo stesso calcolo autonomamente. L'algoritmo che utilizziamo è l'algoritmo di Euclide.

Siano $a, b \in \mathbb{N}$, con $a > b$, e vogliamo calcolare il loro Massimo Comun Divisore $\text{MCD}(a, b)$, allora l'algoritmo di Euclide consiste nei seguenti passaggi:

- dividiamo a per b ottenendo $a = bq_1 + r_1$, con $0 \leq r_1 < b$. Se $r_1 = 0$ allora $\text{MCD}(a, b) = b$, altrimenti:
- dividiamo b per r_1 ottenendo $b = r_1 q_2 + r_2$, con $0 \leq r_2 < r_1$. Se $r_2 = 0$ allora $\text{MCD}(a, b) = r_1$, altrimenti:
- dividiamo r_1 per r_2 ottenendo $r_1 = r_2 q_3 + r_3$, con $0 \leq r_3 < r_2$. Se $r_3 = 0$ allora $\text{MCD}(a, b) = r_2$, altrimenti si continua.

Osservato che al procedere del numero di iterazioni il resto della divisione decresce deve esistere un numero k tale che $r_k = 0$ allora avremo $\text{MCD}(a, b) = r_{k-1}$.

Per implementare questo algoritmo procediamo per passi:

1. con un editor di testo per la programmazione (ad esempio Emacs) scriviamo un codice GP che implementi l'algoritmo di Euclide:

```
/*********************************************************
*                  ALGORITMO EUCLIDEO
*       input: n,m - numeri di cui si calcola MCD
*       output: d - Massimo Comun Divisore tra m e n
*          A. CENTOMO per il PLS 2005-2006
*********************************************************/
{Euclide(m,n) = local (d,app, a, b, iterazioni);
/* se n=m il risultato e' banale */
if(n == m, d = n; print("I due numeri sono uguali");
   print("d=",d);
   return;
   );
/* se m < n scambio i due numeri */
if(m < n, app=m; m=n; n=app);
while (n!=0,
            r= m %n ; /* resto della divisione intera di m per n */
            m=n; /* scambio il ruolo di m e n, e di n e r per */
            n=r; iterazioni=iterazioni+1 /*conto il numero di iterazioni*/
      );
 d=m;
print("Massimo Comun Divisore");
 print(d);
print("Iterazioni");
 print(iterazioni);}
```

2. salviamo il codice con nome `euclide.gp`;

3. lanciamo PARI/GP e digitiamo il comando `\r euclide.gp` in modo che il file venga letto;
4. scriviamo ad esempio `Euclide(1255,55)` per calcolare il Massimo Comun Divisore tra i numeri 1255 e 55;
5. il risultato sarà 5 con 4 iterazioni.

Esercizio 1. Analizzare il codice dello script.

1.1 Algoritmo di Euclide esteso

Iniziamo con un esempio, supponendo che l'algoritmo di Euclide termini dopo tre iterazioni:

a	b	q_1	r_1	$a = b\,q_1 + r_1$	$r_1 = a - b\,q_1$
b	r_1	q_2	r_2	$b = r_1\,q_2 + r_2$	$r_2 = b - r_1\,q_2$
r_1	r_2	q_3	0	$r_1 = r_2\,q_3$	

Tabella 1. Algoritmo di Euclide

Allora, sostituendo a ritroso, avremo:

$$r_1 = q_3 r_2 = q_3(b - r_1 q_2) = q_3 b - q_3 q_2(a - b\,q_1) = q_3[(1 + q_2 q_1)b - q_2 a] = q_3(ua + vb)$$

dove si è posto $u = -\,q_2$ e $v = (1 + q_2 q_1)$. Se ora ricordiamo che r_2 è uguale a $\text{MCD}(a, b)$ avremo anche $\text{MCD}(a, b) = ua + vb$. In altri termini il Massimo Comun Divisore tra a e b si può scrivere come una combinazione lineare di a e b. Non è difficile comprendere che questo risultato vale in generale.

Proposizione 5. Siano $a, b \in \mathbb{N}$, con $a > b$, allora esistono due numeri interi u e v tali che vale l'uguaglianza $\text{MCD}(a, b) = ua + vb$.

Dimostrazione. La dimostrazione segue sostituendo a ritroso le uguaglianze dell'algoritmo euclideo e scrivendo ogni volta il Massimo Comun Divisore in termine dei resti precedenti. □

L'uguaglianza $\text{MCD}(a, b) = ua + vb$ prende il nome di uguaglianza di Bezout e i numeri interi u e v prendono il nome di coefficienti di Bezout. Tramite la funzione predefinita `bezout` possiamo calcolare in PARI coefficienti di Bezout e Massimo Comun Divisore tra due numeri assegnati:

```
? bezout(600,125)
%1 = [-1, 5, 25]
```

Infatti: $\text{MCD}(600, 125) = 25$ e $25 = 5 \cdot 125 - 600$.

Esercizio 2. Ampliare lo script relativo all'algoritmo di Euclide includendovi il calcolo dei coefficienti di Bezout.

1.2 Metodo di Cesare

In questo paragrafo trattiamo un esempio di script più complesso del precedente ma interessante per la crittografia in quanto implementa il metodo di Cesare. Lo script si fonda sull'uso di tre funzioni predefinite:

- → `Vecsmall`: trasforma una qualsiasi stringa di caratteri nei corrispondenti numeri previsti dalla codifica ASCII;
- → `Strchr`: contrariamente alla precedente, trasforma numeri in caratteri;
- → `concat`: concatena i suoi argomenti.

Ad esempio:

```
? Vecsmall("crittografia")
%1 = Vecsmall([99, 114, 105, 116, 116, 111, 103, 114, 97, 102, 105, 97])
```

```
? Strchr(%)
%2 = "crittografia"
```

In questo esempio la stringa `crittografia` è stata trasformata usando `Vecsmall` in una sequenza numerica e, successivamente, il risultato è stato ritrasformato nel testo di partenza utilizzando `Strchr`.

Per quanto concerne la funzione `concat`, per comprenderne il comportamento è sufficiente il seguente esempio:

```
? x=32
%1 = 32
? y="ciao"
%2 = "ciao"
? concat(x,y)
%3 = "32ciao"
```

dove il numero `32` viene concatenato alla stringa `ciao`. A questo punto possiamo proporre il testo dello script che implementa il metodo di Cesare.

```
/*******************************************************
*            CRITTOSISTEMA DI CESARE
*         A. CENTOMO per il PLS 2005-2006
*******************************************************/

{alfabeto=["A","B","C","D","E","F","G","H","I",
"L","M","N","O","P","Q","R","S","T","U","V","Z"];}

{da_lettera_a_numero(lettera)=local(j);
 for(j=1,21, if(alfabeto[j]==lettera,return(j)));
 error("input non valido.")
}

{
 CIFRA(messaggio, chiave="", codifica="") = local(l,i,j,k,C,M);

 l=length(messaggio);
 M=Vecsmall(messaggio);

  for(j=1,l, C = da_lettera_a_numero(Strchr(M[j])) + 3;
   if( C % 21 == 0, C = 21, C = C % 21);
   codifica = concat(codifica,alfabeto[C]);
     );
  print("Il messaggio codificato e'");
  print(codifica);
  return(codifica);
}
{
 DECIFRA(codifica, messaggio="") = local(l,i,j,k,M,C);

 l=length(codifica);
 C=Vecsmall(codifica);

   for(j=1,l, M=da_lettera_a_numero(Strchr(C[j])) - 3;
    if (M % 21 == 0, M = 21, M = M % 21);
    messaggio = concat(messaggio,alfabeto[M])
      );
   print("Il messaggio decodificato e'");
```

```
print(messaggio);
return(messaggio); }
```

Esercizio 3. Studiare il codice dello script precedente. Utilizzare lo script per cifrare e decifrare messaggi con il codice di Cesare.

Esercizio 4. Estendere il codice precedente in modo da imlementare il codice di Vigenerè.

2 Aritmetica Modulare

Pari implementa un certo numero di operazioni modulari predefinite tra cui addizione e moltiplicazione. Le funzioni utili per eseguire calcoli modulo n sono `Mod(numero,n)` e `lift(numero)`.

Esempio 6. Calcolare in $\mathbb{Z}_{58}$ le seguenti: $49+57 \bmod 58$ e $49 \cdot 57 \bmod 58$.

Se scriviamo `Mod(49+57,58)` otterremo come risultato `48 mod58` mentre se si aggiunge il comando `lift` scrivendo `lift(Mod(49+57,58))` si ottiene direttamente `48`. L'azione di `lift` è quella di riportare il prodotto al suo valore in $\mathbb{Z}_{58}$. Analogamente si procede per il prodotto il cui risultato è `9`.

2.1 Tavola della moltiplicazione

In alcuni casi può essere utile disporre della tavola della moltiplicazione modulo n una volta assegnato un dato valore n. Per poter disporre della tavola della moltiplicazione eseguiamo le seguenti azioni:

1. con un editor di testo qualsiasi apriamo un file che chiamiamo `prodmod.gp` e copiamo al suo interno il seguente testo:

```
/*******************************************************
*            TAVOLA DELLA MOLTIPLICAZIONE MODULO n
*   input: n - intero che definisce la tabella
*   output: matrice dei risultati delle moltiplicazioni
*            A. CENTOMO per il PLS 2005-2006
*******************************************************/

{Modprod(n) = local(a,i,j);

        a=matrix(n,n);

        for(i=1, n,

                for(j=1, n, a[i,j]=lift(Mod(i*j-i-j+1,n)));

           );

print("Tavola della moltiplicazione modulo ", n);

return(a);
}
```

2. una volta salvato il file scriviamo `\r prodmod` in modo che il file `prodmod.gp` venga letto dal programma;
3. quindi, se ad esempio vogliamo la tabella della moltiplicazione in $\mathbb{Z}_7$, scriviamo il comando `Modprod(7)`.

Esercizio 5. Analizzare il codice dello script.

Esercizio 6. Implementare uno script che calcola la tabella dell'addizione modulo n.

2.2 Potenze

In questo paragrafo affrontiamo il calcolo delle potenze nell'aritmetica modulare. Data una potenza a^m, con $m \in \mathbb{N}$, osserviamo subito che il metodo ingenuo (svolgere m prodotti e riduzioni modulo n) è computazionalmente molto oneroso. Per ridurre il numero di calcoli un metodo consiste nello scrivere a^m come prodotto di potenze il cui esponente sia una potenza di 2. Ad esempio, per determinare a^{23}, è sufficiente calcolare a^2, a^4, a^8, a^{16} (quattro quadrati) e poi calcolare $a\,a^2a^4a^{16}$ risparmiando diverse operazioni rispetto alle 22 necessarie con il metodo ingenuo. Un algoritmo per il calcolo veloce di potenze è il seguente:

1. poniamo $P \leftarrow 1$, $M \leftarrow m$, $A \leftarrow a$;
2. calcoliamo q e r quoziente e resto della divisione di M per 2; se $r = 1$ poniamo $P \leftarrow PA$;
3. poniamo $A \leftarrow A^2$ e $M \leftarrow q$;
4. se $M = 0$ l'algoritmo termina e $P = a^m$ altrimenti si torna al secondo passo.

Vediamo come esempio il calcolo di a^{23} con questo algoritmo.

q	r	P	M	A
		1	23	a
11	1	$1 \cdot a = a$	11	a^2
5	1	$a \cdot a^2 = a^3$	5	a^4
2	1	$a^3 \cdot a^4 = a^7$	2	a^8
1	0	a^7	1	a^{16}
0	1	$a^7 \cdot a^{16} = a^{23}$	0	

Tabella 2. Algoritmo dei quadrati ripetuti

Possiamo scrivere uno script in PARI/GP che implementa questo metodo osservando che se è già disponibile l'espansione binaria dell'esponente, allora i resti delle divisioni per 2 sono esattamente le cifre di tale espansione. Faremo quindi uso della funzione `bittest` che consente di estrarre le varie cifre di un'espansione binaria di un intero.

```
/*****************************************************
*            QUADRATI RIPETUTI MODULO n
*     input: a,m,n - a intero, m > 1, n >= 2
*     output: a^m mod n
*            A. LANGUASCO per il PLS 2005-2006
*****************************************************/

{RepSquares(a,m,n) = local (P, A, L, i);

/* test sull'input */

if (((m<=0) || (n<=1)), print("input non valido"));

/* inizializzazioni */

P=1;   /* conterrà il risultato parziale */
A=a;   /* conterrà i vari quadrati */
i=0;

/* calcolo del numero di bit dell'espansione binaria di m */

L=floor((log(m)/log(2)))+1;

/* algoritmo dei quadrati ripetuti */

for (i=0, L-1, r=bittest(m, i);
```

```
    /*r è il resto della divisione di m per 2^i */

    if(r==1,P=lift(Mod(P*A,n)););
    A=lift(Mod(A^2,n));  /* calcolo il quadrato successivo */
    );

/* stampa dei risultati */

print(a," alla ", m," modulo ", n, " = ", P);
}
```

3 Numeri primi e fattorizzazione

Dato un numero naturale n è bene ricordare che sono da considerarsi fatti distinti lo stabilire se n è primo e determinare la scomposizione in fattori primi di n. Anche se potendo disporre della scomposizione in fattori primi di n si può stabilire immediatamente se esso è o meno primo, rimane il problema notevole che il calcolo della fattorizzazione di un numero può richiedere tempi di calcolo enormi. Quindi è importante sviluppare algoritmi che permettano di stabilire se un numero è primo evitando il ricorso alla sua fattorizzazione. Esula dagli scopi di queste pagine approfondire questo tema sicuramente interessante ma di una certa complessità. Ci limitiamo invece ad osservare come PARI/GP ci confermi la diversità dei tempi per la verifica della primalità e della fattorizzazione.

Esempio 7. Si consideri il numero di Jevons $J = 8616460799$. Stabilire se J è primo e in caso negativo fattorizzarlo calcolando i tempi computazionali per le diverse operazioni.

```
? J=8616460799
%1 = 8616460799

? isprime(J)
%2 = 0 ?
##
*** last result computed in 0 ms.

? factor(J)
%3 =
[89681 1]
[96079 1]
? ##
*** last result computed in 2 ms.
```

Come si può vedere J non è primo e il tempo impiegato per la fattorizzazione è superiore a quello per stabilirne la primalità.

3.1 Crivello di Eratostene

La procedura di Eratostene nota anche come crivello consiste nel passare i numeri interi positivi attraverso un opportuno setaccio, e quelli che restano sono solo i numeri primi. Il procedimento si può spiegare adeguatamente in forma algoritmica. Supponiamo di voler eseguire il crivello sugli interi nell'intervallo $[2, N]$ dove N è un numero naturale:

1. si scrivono tutti i numeri naturali da 2 ad N;
2. si parte da $p = 2$ (il più piccolo numero primo);
3. si cancellano tutti i multipli di p partendo da p^2 fino ad N;

4. si cerca il più piccolo intero $q > p$ ancora non cancellato. Se $p > N$ la procedura termina;
5. altrimenti si pone $p = q$ e si torna al passo 3.

Si noti che l'operazione di cancellazione di cui al punto 3. può essere effettuata in modo estremamente efficiente (ed è esattamente qui l'essenza dell'algoritmo) partendo da p^2 ed aggiungendo sempre p all'ultimo valore trovato, fino a superare N. Infatti i multipli di p compresi tra $2p$ e $p^2 - p$, estremi inclusi, sono stati cancellati tutti nei passi precedenti, poiché hanno almeno un fattore primo inferiore a p.

A questo punto il nostro obiettivo è dimostrare che i numeri "sopravvissuti" a questa e operazione sono tutti i numeri primi fino ad N. Il numero 2 non è stato cancellato (il più piccolo numero cancellato è il 4, alla prima iterazione). I numeri primi nell'intervallo $[1, N^{1/2}]$ non sono stati cancellati, perché il più piccolo multiplo del primo p che viene cancellato è $2p$, e i numeri primi nell'intervallo $[N^{1/2}, N]$ non sono stati cancellati perché non hanno divisori fra i numeri con i quali e abbiamo effettuato le cancellazioni. Infine, tutti i numeri composti contenuti nell'intervallo $[1, N]$ sono stati cancellati: infatti sia $n \leq N$ un numero composto. Esistono allora a e b, con $1 < a \leq b < n$, tali che $n = ab$. Se a e b fossero maggiori di $N^{1/2}$ allora il loro prodotto sarebbe maggiore di N. Dunque n è divisibile per almeno un numero primo minore o uguale a $N^{1/2}$ (ogni fattore primo di a va bene) e di conseguenza è stato eliminato.

Lo schema teorico di base nel momento in cui si passa all'implementazione al computer deve essere rimaneggiato: ad esempio non è certo utile "scrivere" esplicitamente tutti i numeri da 1 a N! Oltre a questo ci sarebbero numerose altre osservazioni che permettono di aumentare la velocità dell'algoritmo di Eratostene. Una prima proposta, la più semplice, di implementazione dell'algoritmo in PARI/GP è la seguente.

```
/*******************************************************
*            CRIVELLO DI ERATOSTENE
*   input: L - limite entro cui ricercare i primi
*   output: vettore contenente tutti i primi tra 2 e L
*            A. LANGUASCO per il PLS 2005-2006
*******************************************************/

{Erat(L) = local(a, B, B1, i , j , k , l, primi, primiout);

 /* inizializziamo un vettore di L-1 componenti (1 non e' primo)
  * con gli interi tra 2 e L e definiamo un altro vettore della
  * stessa dimensione in cui memorizzeremo i primi */

B=L-1;
a=vector(B);
primi=vector(B);
for(i=1, B, a[i] = i+1);

 /* intendiamo come marcato un intero la cui corrispondente
  * componente di a e' zero. La componente viene marcata a
  * zero se l'intero contenuto e' un multiplo di uno degli interi
  * precedenti */

i = 1;
l=1;
B1=floor(sqrt(L));

while(i <= B1,

  /* "saltiamo" gli interi gia' marcati */

  while( a[i] == 0, i = i + 1 );
```

```
  /* alla fine del while, l'intero contenuto nella posizione a[i]
   * deve essere un primo perche' non e' diviso da nessun intero
   * più piccolo. Allora lo memorizzo nella prima posizione
   * disponibile del vettore primi     */

  primi[l]=a[i];
  l=l+1;

  /* marchiamo adesso i multipli dell'intero contenuto in
   * a[i]=i+1, ossia azzeriamo le posizioni corrispondenti agli
   * interi j(i+1), per j che varia fino a L/i+1. Queste posizioni
   * hanno indice j(i+1)-1       */

  for(j=2, floor(L/(i+1)), a[(i+1)*j-1] = 0);

   /* passiamo ora a studiare la primalita' dell'intero successivo
    * successivo incrementando la variabile che governa
    * il while piu' esterno */

  i=i+1;

   );

/* inseriamo nel vettore dei primi i primi più grandi di radice di L */

for (j=i+1,B, if ((a[j]<>0), primi[l]=a[j];l=l+1));

 /* Sappiamo anche quanti sono i primi minori o uguali
  * a L (sono l-1) e prepariamo l'output del vettore dei primi */

 print("Il numero di primi minori o uguali a ", L, " e': ", l-1);

 /* restituiamo in output il vettore dei primi */

 primiout=vector(l-1);

 for(j=1,l-1,primiout[j]=primi[j]);

 print("I numeri primi minori o uguali a ", L,
       " sono dati dal vettore: ");

 return(primiout);
}
```

Esercizio 7. Studiare il codice precedente e realizzarne una versione ottimizzata.

4 Il metodo RSA

Il metodo RSA è stato proposto nel 1978 da Rivest, Shamir e Adleman. L'idea alla base del metodo consiste nello sfruttare la diversità di complessità computazionale che sussiste tra il riconoscimento della primalità di un intero e la sua fattorizzazione. In generale il metodo funziona secondo le seguenti linee:

1. ciascun utente sceglie casualmente due numeri primi p e q distinti e grandi (circa trecento cifre) e pone $n = pq$;

2. si calcola la funzione di Eulero $\varphi(n) = (p-1)(q-1)$;
3. si sceglie casualmente un intero e tale che $1 < e \leq \varphi(n)$ e coprimo con n ossia tale che sia $\text{MCD}(e, \varphi(n)) = 1$;
4. si calcola $d \equiv e^{-1} \mod \varphi(n)$ (cosa "facile" per chi conosce sia p che q).

Importante 8. Con riferimento a quanto appena esposto osserviamo che:

a) la scelta casuale di alcuni parametri è fondamentale per la sicurezza del crittosistema;

b) come si è visto in precedenza il calcolo di d si ottiene come sottoprodotto della verifica al punto 3.

Per ogni utente vengono definite le seguenti:

→ **Chiave pubblica**: la chiave di cifratura viene resa disponibile a tutti gli utenti che vogliono comunicare con il possessore della chiave stessa. Matematicamente essa è rappresentata dalla coppia

$$K_E = (n, e)$$

→ **Chiave privata**: la chiave di decifratura viene mantenuta segreta in quanto solo tramite essa è possibile decifrare i messaggi ricevuti. Matematicamente essa è rappresentata da

$$K_D = d$$

→ **Funzione di cifratura**: è la funzione potenza $f: \mathbb{Z}_n \to \mathbb{Z}_n$ definita da

$$f(P) = P^e \mod n$$

→ **Funzione di decifratura**: è la funzione potenza $f^{-1}: \mathbb{Z}_n \to \mathbb{Z}_n$ definita da

$$f^{-1}(C) = C^d \mod n$$

Per chiarire il funzionamento del crittosistema analizziamo una situazione in cui sono coinvolti solo due utenti:

	Anna	Bruno
Pubblico	$K_{E_A} = (n_A, e_A)$	$K_{E_B} = (n_B, e_B)$
Privato	$K_{D_A} = d_A$	$K_{D_B} = d_B$

Tabella 3. Sistema RSA a due utenti

Anna per mandare un messaggio P a Bruno dovrà usare la chiave pubblica di Bruno e calcolare

$$C \equiv P^{e_B} \mod n_B$$

Bruno riceve C e applicando la sua funzione di decifratura segreta calcola

$$C^{d_B} = P^{e_B d_B} \mod n_B$$

Ora tuttavia $e_B d_B \equiv 1 \mod \varphi(n_B)$ da cui, per noti teoremi, consegue che il calcolo si riduce a

$$C^{d_B} \equiv P \mod n_B$$

e quindi Bruno è in grado di leggere il messaggio postato da Anna.

4.1 rsa con Pari/GP

Vediamo di ripercorrere il crittosistema RSA utilizzando Pari/GP:

- Generazione di p e q

```
? p=nextprime(random(10^40))
%1 = 72568255023458735500260032517794005975 61
```

```
? isprime(p)
%2 = 1
? q=nextprime(random(10^35))
%3 = 79214481285059248805111509508972537
? isprime(q)
%4 = 1
```

Si utilizza la funzione `nextprime(x)` che determina il più piccolo pseudoprimo maggiore o uguale a `x`.

- Calcolo di n

```
? n=p*q
%5 = 57484566794451788216542806860441459
25045762565894407353742497160289381822 57
```

- Calcolo di $\varphi(n)$

```
? phin=(p-1)*(q-1)
%6 = 57484566794451788216542806860441458
52476715397622821260994413527400286121 60
```

- Calcolo di e

```
e=random(n)
%7 = 47000017476545391300694041761909628
61130505176500129434920384429428824984 73
? while(gcd(phin,e)!=1,e=e+1)
? e
%8 = 47000017476545391300694041761909628
61130505176500129434920384429428824984 73
```

- Calcolo di $d \equiv e^{-1} \bmod \varphi(n)$

```
? d = lift(Mod(e,phin)^(-1));
? (e*d)%phin
%10 = 1
? d
%11 = 34816715732984455629550085271710811
19551727134699800742412302186794482098 17
```

- Calcolo della lunghezza di testo massimo codificabile

```
? log(n)/log(30)
%12 = 50.611654949776177874130565 28
```

Usando un alfabeto di 30 caratteri possiamo quindi codificare in un blocco unico un testo avente meno di 50 caratteri. Codifichiamo "PARMA":

- Calcolo dell'equivalente numerico in base 30 di PARMA usando l'alfabeto: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, ;, ,, ., '.

```
? m=15*30^4+0*30^3+17*30^2+12*30^1+0*30^0
%13 = 12165660
```

- Definizione della funzione di cifratura

```
E(x)=lift(Mod(x,n)^e)
```

- Definizione della funzione di decifratura

```
D(x)=lift(Mod(x,n)^d)
```

- Codifica di PARMA

```
? messaggio_segreto = E(m)
%14 = 23932693203095631249496654190604694
15928074248293277726170157793586311780 86
```

- Decodifica del messaggio

```
? D(messaggio_segreto)
%15 = 12165660
```

La procedura è stata sviluppata correttamente poiché l'equivalente numerico della decodifica è uguale a quello di PARMA.

Esercizio 8. Scrivere uno script che implementi il metodo RSA.

Bibliografia

1. A. Languasco, Progetto Nazionale Lauree Scientifiche 2005-2006 Regione Veneto Crittografia, Dipartimento di Matematica Pura e Applicata, Università di Padova, 2005.

Indice