Teoria elementare dei numeri con TI-92/TI-89

Giulio Cesare Barozzi – Università di Bologna

1. Le calcolatrici grafico-simboliche TI-92 e TI-89 forniscono buoni strumenti per una prima esplorazione della teoria elementare dei numeri, vale a dire dell'Aritmetica intesa nel senso classico del termine. Sono direttamente implementate alcune funzioni di sicuro interesse, come il calcolo del quoziente e del resto della divisione intera tra due naturali positivi e l'algoritmo euclideo per il calcolo del MCD: la sintassi dei comandi citati è, nell'ordine, `intDiv(n,m)`, `mod(n,m)` e `gcd(n,m)`, dove l'ultima scelta è ispirata all'inglese *Greatest Common Divisor*.

2. Quando vogliamo determinare la rappresentazione decimale del numero razionale *n*/*m*, eseguiamo la divisione di *n* per *m* e ci arrestiamo quando si presenta un resto ripetuto.
La "distanza" tra i due resti uguali tra loro fornisce la lunghezza del periodo della rappresentazione decimale, il numero di resti che (eventualmente) precedono il primo resto ripetuto fornisce la lunghezza dell'antiperiodo.
Ricordiamo che la lunghezza dell'antiperiodo è deducibile direttamente dalla fattorizzazione del denominatore *m*, supposti *n* ed *m* primi tra loro. Se

$$m = 2^{\alpha} \cdot 3^{\beta} \cdot 5^{\gamma} \cdot \ldots$$

è la scomposizione in fattori primi di *m*, allora la lunghezza dell'antiperiodo è data da

$$\max\{\alpha, \gamma\},$$

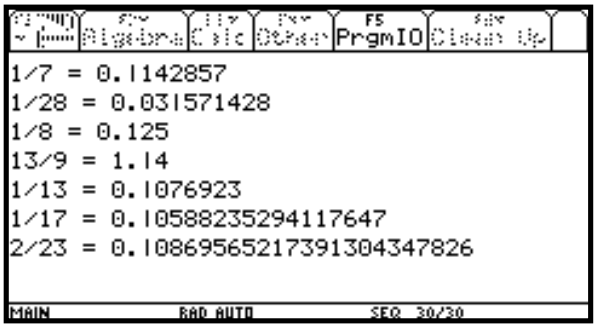
cioè dal più grande tra gli esponenti dei fattori 2 e 5. Ne segue che la rappresentazione decimale è periodica pura (cioè la lunghezza dell'antiperiodo è 0) se e solo se m è primo rispetto a 10:

$$\mathrm{MCD}(m, 10) = 1.$$

Quanto alla lunghezza del periodo, essa è necessariamente un divisore della quantità $\phi(m)$, dove ϕ è la funzione di Eulero che discuteremo nel punto seguente: essa conta i numeri interi positivi minori di *m* e primi rispetto ad esso.
Il programma `rd`, descritto nel listato seguente, verifica innanzitutto che il numeratore e il denominatore della frazione considerata siano primi tra loro, e, in caso contrario, provvede alla semplificazione. Successivamente viene calcolata la lunghezza dell'antiperiodo, determinando gli esponenti dei fattori 2 e 5 nella fattorizzazione del denominatore. Una volta calcolata tale lunghezza, si sa a priori quale resto si ripeterà: la sequenza dei resti, e di conseguenza la sequenza delle cifre decimali, viene arrestata quando tale resto si ripresenta.
Come elemento grafico di separazione tra l'antiperiodo (eventuale) e il periodo si è scelta una barra verticale.
Il programma è in tutto simile a ciò che si potrebbe fare in Pascal o in BASIC, salvo differenze nelle notazioni, ad esempio l'uso della freccia a destra → per indicare l'operazione di assegnazione.

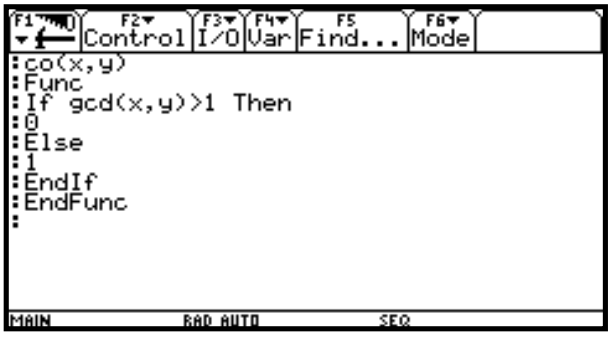
```
:rd(n,m)
:Prgm
:Local g,c,h,k,r,j,rr,ad
:gcd(n,m)→g
:If g >1 Then
:n/g→n: m/g→m
:EndIf
:string(n)&"/"&string(m)&" = "
 &string(intDiv(n,m))&"." → ad
:m→r: 0→h: 0→k
:While mod(r,2)=0
: h+1→h: r/2→r
:EndWhile
:While mod(r,5)=0
: h+1→h: r/5→r
:EndWhile
:max(h,k)→k
:0→j: 0→rr: mod(n,m)→r
:While r ≠ rr
: If j=k Then
:  r→rr: ad&"|" →ad
: EndIf
: j+1→j
: intDiv(10*R,m)→c
: mod(10*r,m)→r
: ad&string(c)→ad
:EndWhile
:Disp ad
:EndPrgm
```

Alcuni esempi di utilizzo della funzione `rd` :



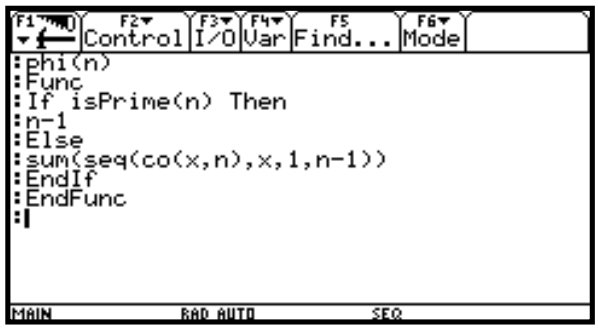
Si osservi, ad esempio, il numero 1/13 la cui rappresentazione decimale ha periodo di lunghezza 6; si ricordi che $\phi(13) = 13 - 1 = 12$, essendo 13 un numero primo.

3. Nel seguito ci farà comodo una funzione che si limita a verificare se due numeri naturali sono o non sono primi tra loro, o come anche si dice *coprimi*. Le possibilità di programmazione delle calcolatrici in esame rendono semplicissima la realizzazione di una funzione che adempie allo scopo, come mostrato dal listato seguente. La funzione `co`, che utilizza la funzione predefinita `gcd`, restituisce in uscita 1 se i due argomenti della funzione sono primi tra loro, 0 in caso contrario.

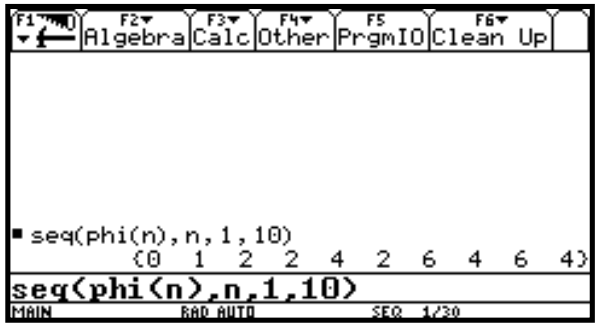


La funzione seguente, che utilizza il predicato predefinito `isPrime`, verifica se un numero naturale è primo: essa fornisce in uscita 1 se la condizione è verificata, 0 in caso contrario.

Diamo subito un esempio di utilizzazione delle due funzioni precedenti: la cosiddetta funzione ϕ di Eulero calcola, per ogni naturale positivo n, il numero dei numeri naturali inferiori ad n e primi rispetto ad esso. Essa può essere realizzata nel seguente modo:



Spiegazione: se n è primo, allora $\phi(n) = n-1$; in caso contrario si costruisce una lista di $n-1$ elementi, corrispondenti ai valori di x da 1 a $n-1$, i cui valori sono quelli forniti dalla funzione `co(x,n)`: sommando tali elementi, si vengono a contare i numeri inferiori a n e primi rispetto ad esso.
Ecco i risultati che si ottengono calcolando $\phi(n)$, per n da 1 a 10.



Un classico teorema di Eulero, che generalizza un precedente risultato di Fermat (si tratta del cosiddetto *piccolo teorema di Fermat*, v. bibliografia [1], p. 48, [4], p. 40) ci assicura che per calcolare il reciproco di n rispetto ad m, sempre n sia primo rispetto ad m, basta elevare n alla potenza $\phi(m) - 1$ e successivamente prendere il resto della divisione per m.
La funzione che segue calcola il reciproco di n modulo m, se n è primo rispetto a m; in caso contrario la funzione restituisce il valore 0.

```
:inv(n,m)
:Func
:If gcd(n,m)=1 Then
:mod(n^(phi(m)-1),m)
:Else
:0
:EndIf
:EndFunc
:
```

Ecco i risultati che si ottengono calcolando i reciproci rispetto ai moduli 9, 10 e 11 dei numeri interi positivi inferiori a ciascun modulo: gli zeri corrispondono agli elementi non primi rispetto al modulo, e dunque non invertibili rispetto ad esso.

```
■ seq(inv(x,9),x,1,8)
                {1  5  0  7  2  0  4  8}
■ seq(inv(x,10),x,1,9)
              {1  0  7  0  0  0  3  0  9}
■ seq(inv(x,11),x,1,10)
      {1  6  4  3  9  2  8  7  5  10}
seq(inv(x,11),x,1,10)
```

4. Abbandoniamo momentaneamente la funzione ϕ di Eulero, e torniamo all'algoritmo euclideo per il calcolo del MCD.
Oltre alla forma classica derivata direttamente dagli Elementi di Euclide (v. [5], Libro 7, prop. 1 e 2), esiste una notevole generalizzazione che consiste nel mostrare come

$$d := \mathrm{MCD}(n, m)$$

possa essere scritto come combinazione lineare a coefficienti interi, diciamo x e y, dei numeri n e m:

$$d = xn + ym.$$

L'identità scritta, spesso citata come formula di Bezout (dal nome del matematico francese Étienne Bezout, 1730-1783), mostra che d è il generatore dell'ideale di **Z** (anello degli interi) costituito dalle combinazioni lineari intere di n ed m.
La dimostrazione, che ci limitiamo ad accennare, si basa sul fatto che i numeri inizialmente dati, n ed m, si possono ovviamente scrivere come combinazioni lineari di se stessi:

$$n = 1 \cdot n + 0 \cdot m, \quad m = 0 \cdot n + 1 \cdot m; \qquad (*)$$

dopodiché si osserva che l'algoritmo consiste nel passare dalla coppia (n, m) alla coppia (d, 0), mediante una successione di coppie nelle quali il secondo elemento di una coppia diventa il primo elemento della coppia seguente, mentre il nuovo secondo elemento è il resto della divisione intera tra i due elementi considerati, resto che si può scrivere come combinazione lineare a coefficienti interi dei due elementi della coppia precedente.
La scrittura di un programma che realizzi l'identità di Bezout non comporta alcuna difficoltà, come mostra il listato seguente:

```
:xgcd(n,m)
:Prgm
:Local xv,xn,sv,sn,tv,tn,q,r
:n → xv: m → xn
:1 → sv: 0 → sn
:0 → tv: 1 → tn
:While xn > 0
: intDiv(xv,xn)→ q: mod(xv,xn)→ r
: xn → xv: r → xn
: sv-q*sn → r: sn → sv: r → sn
: tv-q*tn → r: tn → sv: r → tn
:EndWhile
:Disp string(xv)&" = "&
 string(sv)&"*" string(n)&" + "
 string(tv)&"*" string(m)
:EndPrgm
```

Le due schermate seguenti mostrano alcuni esempi di utilizzo del programma `xgcd` :

```
■ xgcd(89,144)                    Done
■ xgcd(20,144)                    Done
■ xgcd(13,121)                    Done
■ xgcd(15,21)                     Done
xgcd(15,21)
```

```
1 = -55*89 + 34*144
4 = -7*20 + 1*144
1 = 28*13 + -3*121
3 = 3*15 + -2*21
```

Le possibilità offerte dal linguaggio di programmazione delle calcolatrici che stiamo considerando, ci consente di dare una formulazione più elegante dell'algoritmo precedente. Possiamo utilizzare una matrice 2×3, che viene inizializzata ponendo nella prima riga la terna *n*, 1, 0 e nella seconda la terna *m*, 0, 1; queste due terne corrispondono alle formule (*) viste poco sopra, che consentono di esprimere i numeri di partenza *n* e *m* come combinazioni lineari a coefficienti interi dei numeri stessi. Dopodiché si procede sottraendo dalla prima riga un multiplo opportuno della seconda, e così si continua applicando il classico algoritmo euclideo ai numeri che costituiscono la prima colonna della matrice di servizio considerata.
Il tutto è espresso del seguente listato:

```
:mgcd(n,m)
:Prgm
:Local w,q,s
:[[n,1,0][m,0,1]] →w
:While w[2,1]>0
: intDiv(w[1,1],w[2,1]) →q
: w[1] →s: w[2] →w[1]
: s–q*w[2] →w[2]
:EndWhile
:Disp string(w[1,1])&" = "&
 string(w[1,2],)&"*" string(n)&
 " + "string(w[1,3],)&
 "*" string(m)
:EndPrgm
```

L'algoritmo euclideo nella sua forma estesa consente il calcolo del reciproco di *n* modulo *m*, con una tecnica alternativa a quella già mostrata, che utilizza la funzione ϕ di Eulero. Infatti se

$$d = \text{MCD}(n,m) = 1,$$

esistono *x* ed *y* tali che

$$x\,n + ym = 1 \quad \Leftrightarrow \quad xn = 1 - ym$$

cioè

$$xn \equiv 1 \bmod m.$$

Dunque *x* è l'inverso cercato. Volendo ottenere un risultato compreso tra 1 e *m*–1 basta calcolare il resto della divisione dell'intero *x* (fornito dall'algoritmo euclideo) per *m*. Questo è precisamente quanto fa il programma `pinv` (ottenuto adattando il programma `xgcd)` presentato nel listato seguente.

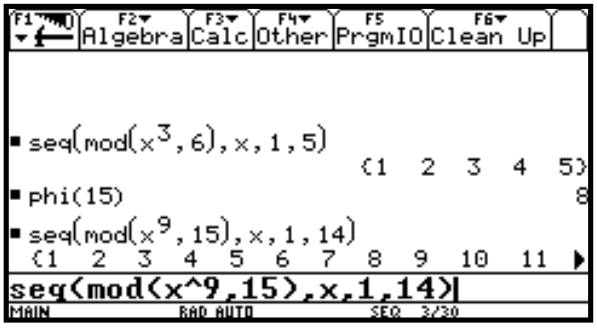
Esso applica l'algoritmo euclideo, trascurando il calcolo del coefficiente *y* che compare nella formula di Bezout, in quanto tale coefficiente non è necessario. Se *n* è primo rispetto ad *m*, viene stampato il resto della divisione di *x* per *m*; se così non è viene stampato un messaggio (abbiamo scelto per semplicità di stampare uno zero).

```
:pinv(n,m)
:Prgm
:Local xv,xn,sv,sn,q,r
:n →xv: m →xn
:1 →sv: 0 →sn
:While xn>0
: intDiv(xv,xn) →q
: mod(xv,xn) →r
: xn →xv: r →xn
: sv–q*sn →r: sn →sv: r →sn
:EndWhile
:If xv = 1 Then
:Disp string(mod(sv,m))
:Else
:Disp string(0)
:EndIf
:EndPrgm
```

5. Abbiamo ricordato poco sopra un teorema di Eulero relativo alla funzione φ: esso afferma che se *m* è un naturale (maggiore o uguale a 2) e *x* un naturale primo rispetto ad *m*, si ha (modulo *m*)

$$x^{\varphi(m)+1} \equiv x\,. \qquad (1)$$

Il punto di partenza del metodo crittografico RSA (dalle iniziali dei nomi dei matematici Rivest, Shamir e Adleman che lo proposero nel 1978; vedi bibliografia [1], [3], [4]) sta nella seguente domanda: che cosa accade se *x*, che d'ora in poi possiamo limitarci a supporre naturale minore di *m*, non è primo rispetto ad *m*?
Consideriamo il caso del modulo $m = 4$; si ha subito $\varphi(4) = 2$, in quanto 1 e 3 sono primi rispetto a 4. Calcoliamo 2 elevato a $\varphi(4) + 1$, cioè 8; esso è congruo a 0 modulo 4. Evidentemente tutte le potenze (proprie) di 2 sono congrue a 0 modulo 4.
Esaminiamo un modulo composto che sia prodotto di primi distinti, sia $m = 6 = 2 \cdot 3$. Si ha $\varphi(6) = 2$ in quanto i naturali minori di 6 e primi rispetto ad esso sono 1 e 5. Per ciascuno dei numeri da 1 a 5 calcoliamo la potenza di esponente $\varphi(6) + 1 = 3$ e passiamo al resto modulo 6; otteniamo i rusultati mostrati dalla schermata seguente.



Come si vede, l'uguaglianza (1) è verificata per ogni x e non soltanto per gli x primi rispetto ad m. Un esperimento analogo con $m = 15 = 3 \cdot 5$ fornisce lo stesso risultato.
Nasce il sospetto che l'uguaglianza (1) sia valida per ogni x non solo se m è primo, ma anche se m è il prodotto di due primi distinti.
Diamo un'occhiata ravvicinata alla funzione φ. Abbiamo già detto che, se p è primo, $\varphi(p) = p–1$. Non è difficile convincersi che, per ogni k naturale ≥ 1, si ha

$$\varphi(p^k) = p^{k-1}\,(p–1).$$

Infatti i numeri minori di p^k e non primi rispetto ad esso sono $0, p, 2p, \ldots, p^k – p = p(p^{k-1} – 1)$, cioè in tutto p^{k-1}. Ne viene che

$$\varphi(p^k) = p^k – p^{k-1} = p^{k-1}\cdot (p–1).$$

La scomposizione in fattori primi di m ci consente di scrivere m stesso come prodotto di potenze di primi tra loro distinti. Si dimostra, senza troppa difficoltà, che la funzione φ è *moltiplicativa* nel senso che si ha $\varphi(n \cdot m) = \varphi(n) \cdot \varphi(m)$ se n e m sono primi tra loro. Combinando questo risultato con la scomposizione in fattori primi di m si perviene al calcolo di $\varphi(m)$ per ogni m.
Noi non abbiamo bisogno di un risultato così generale; ci basta il seguente:

$$\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q) = (p–1)(q–1) = \\ = pq – p – q + 1, \qquad (2)$$

dove p e q sono due primi distinti.
La dimostrazione della (2) è elementare: i naturali minori di pq e non primi rispetto ad esso sono lo 0 e i numeri > 0 che sono multipli di p aut di q, dunque $p, 2p, \ldots, (q–1)p$ e $q, 2q, \ldots, (p–1)q$.
In tutto $1+q–1+ p–1 = p+q–1$. Dunque

$$\varphi(p \cdot q) = pq – p – q + 1.$$

A questo punto possiamo enunciare il risultato principale agli effetti di ciò che segue:

Teorema. Sia $m = pq$ con p e q primi distinti; per ogni intero x si ha, modulo m,

$$x^{\varphi(m)+1} = x^{(p-1)(q-1)+1} \equiv x\,. \qquad (3)$$

Basta limitarsi ai naturali compresi tra 1 e $pq – 1$. Già sappiamo che la (3) vale per gli x che sono primi rispetto ad m. Quelli che non sono tali sono multipli di p aut di q. Sia x un multiplo di p; allora esso è primo rispetto a q.
Il resto di $x^{(p-1)(q-1)}$ rispetto a q è 1. Infatti

$$x^{q-1} \equiv 1 \mod(q),$$

in virtù del teorema di Fermat, e la stessa cosa vale per

$$x^{(p-1)(q-1)} = [x^{q-1}]^{p-1}.$$

Dunque $x^{(p-1)(q-1)} = 1 + hq$ con h intero, o, se si preferisce, $x^{(p-1)(q-1)} – 1$ è multiplo di q.
Scriviamo la nostra tesi nella forma

$$x^{(p-1)(q-1)+1} – x = \\ = x\,(x^{(p-1)(q-1)+1} – 1) \equiv 0 \mod(m).$$

Si tratta di verificare che l'ultimo prodotto scritto è multiplo di $m = pq$; ma questo è ovvio in quanto x è multiplo di p e $x^{(p-1)(q-1)+1}–1$ è multiplo di q. Il Teorema è dimostrato.

Il metodo RSA parte dall'idea che è possibile basare un sistema crittografico a chiave pubblica sulla difficoltà di fattorizzazione di un numero composto “abbastanza grande”, cioè un numero, che d'ora in poi chiameremo n, che sia prodotto di due primi distinti, $n = p \cdot q$, ciascuno composto da qualche decina di cifre decimali.
La dizione *a chiave pubblica* significa che lo strumento per la codifica dei messaggi da trasmettere può essere reso di pubblico dominio, senza pregiudicare la riservatezza del codice. Infatti, per la decodifica occorre possedere un'informazione aggiuntiva che, ancorché contenuta in linea di principio nelle informazioni necessarie per la codifica, di fatto richiederebbe tempi proibitivamente alti per essere dedotta da quest'ultime: centinaia di anni di tempo di calcolo con i mezzi attualmente a disposizione.
Osserviamo innanzitutto che qualunque messaggio può essere tradotto informa numerica: basta stabi-

lire una corrispondenza biunivoca tra le lettere dell'alfabeto, compresi i segni di interpunzione e lo spazio bianco, ed opportuni numeri naturali. Si può utilizzare il codice ASCII (*American Standard Code for Information Interchange*). Naturalmente il mittente ed il ricevente devono condividere tale codice.
Un qualunque messaggio viene così trasformato in una "stringa" di cifre: se necessario, tale stringa può essere suddivisa in sottostringhe, in modo tale che ciascuna di esse rappresenti, in forma decimale, un numero naturale non superiore ad un massimo prefissato.
Possiamo dunque sempre ridurci al problema di trasmettere un messaggio rappresentato da un numero naturale non superiore ad un assegnato n. Sia $n = pq$, con p e q primi distinti. Dal teorema dimostrato poco sopra sappiamo che, se $0 \le x < n$, abbiamo

$$x^{(p-1)(q-1)+1} \bmod n = x. \qquad (4)$$

Se riusciamo a fattorizzare l'esponente di x nella forma

$$b \cdot c = (p-1)(q-1) + 1, \qquad (5)$$

possiamo usare il seguente codice: per trasmettere l'informazione x, si calcola

$$r := x^b \bmod(n),$$

e si trasmette r. Il ricevente calcola

$$r^c \bmod(n) = x^{bc} \bmod(n)$$

e quest'ultimo, in virtù della (4), coincide con x.
In modo più formale: la funzione di codifica è l'applicazione f dell'insieme $\mathbf{N} \cap [0, n-1]$ in sé definita da

$$f : x \to x^b \bmod(n);$$

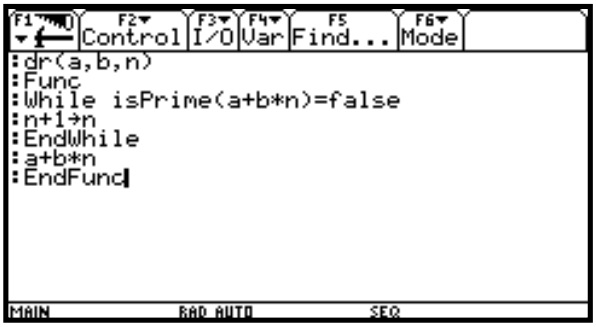
la funzione inversa è

$$f^{-1} : r \to r^c \bmod(n).$$

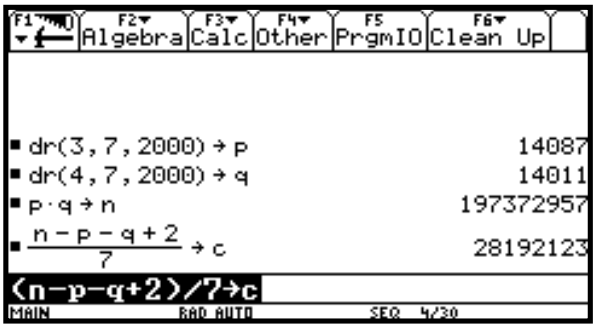
Per codificare è necessario conoscere b e n (le chiavi pubbliche); per decodificare è necessario conoscere la chiave privata c e n. Si osservi che

$$bc = (p-1)(q-1) + 1 = n - p - q + 2;$$

dunque per calcolare c non basta conoscere n e b, ma occorre conoscere i fattori p e q di n. Come abbiamo già detto, la fattorizzazione di n è un'impresa praticamente impossibile se n è grande.
La realizzazione pratica di un sistema crittografico a chiave pubblica pone alcuni problemi. Innanzitutto occorre procurarsi due primi p e q abbastanza grandi, ed in secondo luogo occorre soddisfare la (5). Per entrambi gli scopi è utile un teorema di Dirichlet (v. bibliografia [4] p. 32): esso afferma che ogni successione $k \to a+k\cdot b$, con a e b numeri primi tra loro, contiene infiniti numeri primi. Dunque esplorando una tale successione a partire da un valore di k "abbastanza grande" possiamo ottenere un primo grande e congruo ad a modulo b. La funzione `dir(a,b,k)` mostrata nella schermata seguente assolve a questo scopo.



Supponiamo, ad esempio, di voler usare la chiave pubblica $b = 7$. La condizione (5) richiede che $(p-1)(q-1) + 1$ sia un multiplo di 7, e dunque che $(p-1)(q-1)$ sia congruo a 6 modulo 7. Per far ciò è sufficiente che $p-1$ sia congruo a 2 (e dunque p sia congruo a 3) e $q-1$ sia congruo a 3 (e dunque q sia congruo a 4). Poiché tanto 3 quanto 4 sono coprimi con 7, possiamo determinare due primi p e q della forma rispettivamente $p = 3 + k' \cdot 7$ e $q = 4 + k'' \cdot 7$, come mostra la schermata seguente.

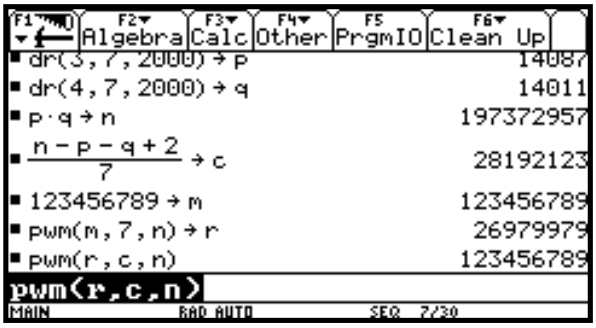


Un secondo problema riguarda il calcolo efficiente di potenze del tipo $a^b \bmod(n)$. Si tratta di ridurre il numero delle moltiplicazioni e di evitare il calcolo con numeri troppo grandi.

È noto in letteratura (v. [1], pag. 63) un algoritmo che riduce il numero delle moltiplicazioni all'ordine di grandezza di $\log_2 b$, ed evita la manipolazione di numeri superiori a n^2. La funzione seguente ne è una traduzione nel linguaggio della TI-92/89; gli argomenti della funzione sono, nell'ordine, la base, l'esponente e il modulo della congruenza..

```
:pwm(x,n,m)
:Func
:Local z
:1→z: mod(x,m)→x
:While n > 0
: If mod(n,2)>0 Then
:  n-1→n: z*x→z: mod(z,m)→z
: EndIf
: n/2→n: x*x→x: mod(x,m)→x
:EndWhile
:z
:EndFunc
```

L'ultima schermata mostra la trasmissione del "messaggio" 123456789.



Bibliografia

[1] G.C. Barozzi, *Aritmetica - Un approccio computazionale*, Zanichelli (Bologna), 1987;

[2] G.C. Barozzi, *Teoria elementare dei numeri con DERIVE*, L'insegnamento della Matematica e delle Scienze Integrate, Vol.20 A-B, n.6, pp.747-770;

[3] G.C. Barozzi, *I codici crittografici a chiave pubblica*, Atti del Convegno "Il fascino discreto della matematica", L'Aquila, 18-20 aprile 1996, pp. 9-22;

[4] H. Davenport, *Aritmetica superiore: un'introduzione alla teoria dei numeri*, Zanichelli (Bologna), 1995;

[5] R. Dvornicich, *Elementi di aritmetica e di algebra elementare*, in "L'algebra tra tradizione e rinnovamento - Seminario di formazione per Docenti", Lucca (1994), a cura del M.P.I. (Dir. Gen. Classica) e del Liceo Sc. Statale A.Vallisneri di Lucca;

[6] Euclide, *Elementi*, UTET (Torino), 1970 (a cura di A. Frajese, L. Maccioni);

[7] G.H. Hardy, E.M.Wright, *An Introduction to the Theory of Numbers*, 5th ed., Clarendon Press, Oxford, 1979;

[8] Knuth D.E., *The Art of Computer Programming*, Vol.1: *Fundamental Algorithms*, Addison Wesley, Reading MA, 1973;

[9] Knuth D.E., *The Art of Computer Programming*, Vol.2: *Seminumerical Algorithms*, Addison Wesley, Reading MA, 1980;

[10] M.R. Schroeder, *La teoria dei numeri*, Muzzio (Padova), 1986 [nel frattempo è uscita la terza edizione del testo originale: *Number Theory in Science and Communications*, Springer (New York), 1996];

[11] B. Scimemi, *Algebretta*, Decibel/Zanichelli (Bologna), 1989.