GENESI DEI NUMERI PRIMI

* Legge di Genesi e Struttura dei Numeri Primi

* Algoritmo detto “Metodo del compasso e della gittata numerica”

* Implementazione rudimentale dell'algoritmo detto “metodo del compasso e della gittata numerica” in linguaggio C/C++

ISMAELE ALONGI

Note sulla pubblicazione

Autore: ***Ismaele Alongi (nato a Caltanissetta nel 1977 in Italia)***
Nazionalità dell'autore: ***ITALIANA***
Luogo di pubblicazione: ***Sommatino (CL)***
Pubblicatore e stampatore: ***Ismaele Alongi – Via Angelo Alaimo, 21***
C.A.P. 93019 – Sommatino (CL)
ITALIA / ITALY
Opera pubblicata sui siti Internet: ***genesi-numeri-primi.ismael.it***
genesis-of-prime-numbers.ismael.it
Anno di pubblicazione: ***2006***
e-mail dell'autore: ***infoarte@libero.it***

GENESI DEI NUMERI PRIMI

Legge di Genesi e Struttura dei Numeri Primi

"La bellezza, la sapienza, la potenza e la gloria appartengono a Dio, l'Altissimo, Colui che fa divenire.

Egli ci dà il privilegio e la gioia di poter conoscere il Suo eterno proposito per l'uomo."

Ismaele Alongi.

Da un' analisi dei numeri primi ho dedotto la seguente Legge di Genesi e Struttura dei Numeri Primi:

Nella successione numerica dei numeri primi, ogni nuovo numero primo nasce nel primo varco numerico lasciato vuoto dalle progressioni numeriche dei precedenti numeri primi noti.

Originariamente avevo espresso questo concetto così:

Ogni nuovo numero primo viene generato nel primo spazio lasciato vuoto dalle serie numeriche inferiori.

La successione numerica dei numeri primi giace su una semiretta che ha origine nel campo del finito e punta all'infinito. Idealmente ho immaginato delle successioni numeriche sovrapposte che hanno origine da punti situati su una retta immaginaria ortogonale alla semiretta dei numeri naturali e passante per il punto zero. E' straordinario immaginare tutti i numeri primi allineati su questa retta immaginaria: virtualmente immaginiamo un immenso e infinito numero composto che va a colmare l'infinito... o potrebbe anche essere l'infinito numero primo... Comunque sia, ci sono interrogativi che per l'uomo sono imperscrutabili, proprio perché la mente umana opera nel campo del finito e, dato che l'infinito è incolmabile, cercare di chiudere il cerchio infinito porta in errore.

L'idea dei nuovi spazi vuoti che vengono via via colmati dai nuovi numeri primi, scaturisce dall'osservazione dello speciale numero primo pari: il numero due.

Se infatti osserviamo il comportamento della progressione del numero due, notiamo subito che va a colmare ½ infinito dei numeri naturali. Cosa sarebbe necessario per colmare il resto dell'infinito? Una progressione numerica in base uno con ragione due. Non è questa però la logica della successione dei numeri primi. L'infinito viene colmato gradualmente, e dato che l'infinito non si può colmare, non si colmerà mai, quindi per la mente umana esistono infiniti numeri primi. Resterà sempre infatti un primo varco numerico vuoto oltre le progressioni sovrapposte dei numeri primi noti: quel primo varco numerico vuoto è uno “zero”, per niente banale, come amava dire Bernhard Riemann, e quel vuoto numerico è occupato da un nuovo numero primo da conoscere, un nuovo pianeta numerico con una sua precisa traiettoria ciclica, una frequenza con una precisa lunghezza d'onda.

Non ci vuole tanto per azzardare l'idea che esplorando i numeri primi lungo la semiretta dei numeri naturali, stiamo esplorando solo metà dell'infinito, l'altra metà si trova allo specchio, sulla semiretta virtuale dei numeri negativi, quindi molto probabilmente siamo di fronte a un miraggio matematico se pensiamo di vedere sull'allineamento virtuale che passa per lo zero-origine un unico e infinito numero composto, semmai nel miraggio c'è un infinito numero primo.

Non so se queste mie ipotesi vanno a dimostrare l'ipotesi di Riemann, che mi sembra quasi un azzardo matematico riuscito, ma questo risultato che ho ottenuto nasce da un'altra ricerca che sto ancora sviluppando: la fattorizzazione dei numeri composti.

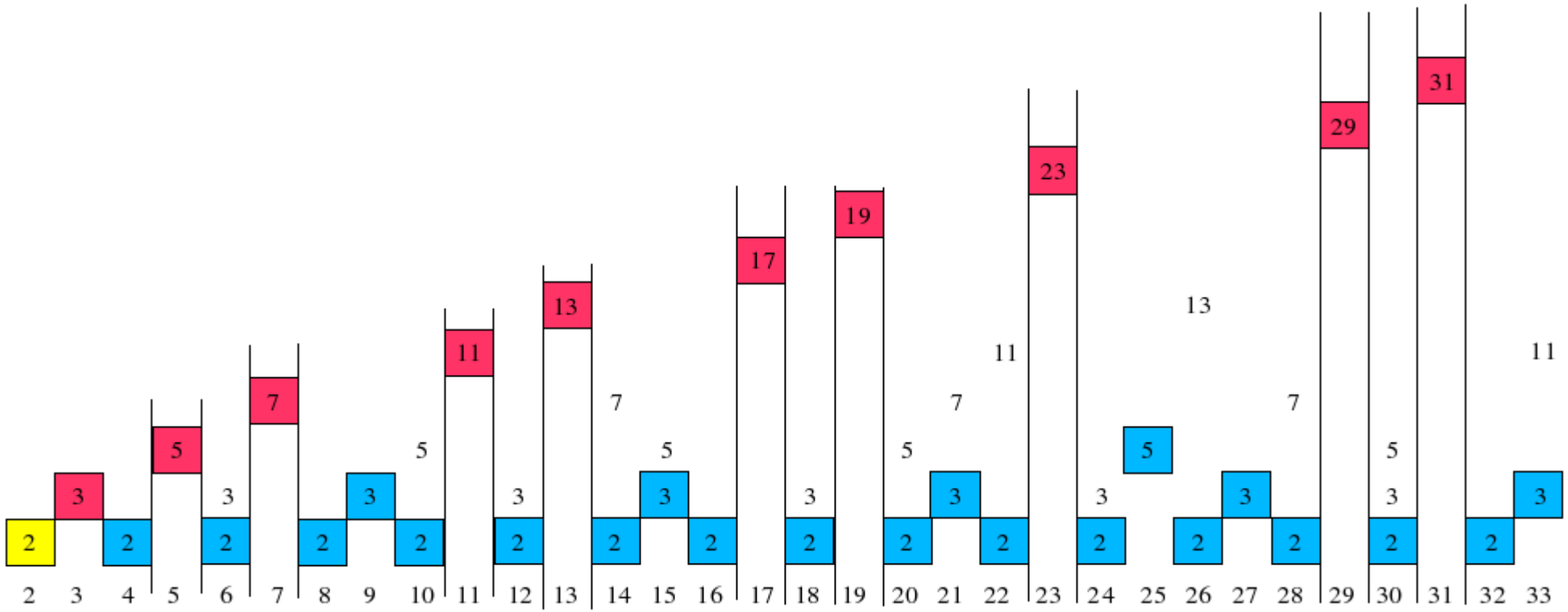
Dalla legge di successione numerica e genesi dei numeri primi che ho dedotto, ho preso spunto per sviluppare un sistema abbastanza efficace per calcolare i successivi numeri primi in un dato intervallo numerico, dati i precedenti numeri primi noti. Questo algoritmo l'ho denominato metodo del compasso numerico e della gittata numerica.

Basterebbe come elemento di base il fondamentale numero due, per ottenere tutti gli altri numeri primi, infatti se osserviamo la progressione del numero due, il primo spazio numerico vuoto è tra il due e il quattro, quindi il prossimo numero primo è il tre. Osserviamo poi le progressioni sovrapposte del due e del tre: il prossimo varco numerico è il cinque ecc.

Per visualizzare meglio il concetto ho realizzato un semplice grafico. La mia scoperta infatti è nata nel momento in cui ho iniziato a considerare per i vari numeri naturali che ho analizzato in successione, il minimo divisore maggiore dell'unità relativamente a ciascun numero, quindi ho iniziato a intravedere delle simmetrie che si ripetevano ciclicamente, quasi come una fantastica sequenza genetica numerica: il DNA dei numeri primi.

Legge di Genesi e Struttura dei Numeri Primi

Rappresentazione grafica



GENESI DEI NUMERI PRIMI

Algoritmo detto “Metodo del compasso e della gittata numerica”

Dati iniziali
Sequenza di numeri primi noti.
Conosciamo la sequenza dei numeri primi da 2 a “n”.

Fase di calcolo 0
Centro di simmetria e raggio di simmetria
Le progressioni generate dai vari numeri primi da 2 a “n” creano delle “simmetrie”, quindi decidiamo di considerare la posizione dell'ultimo numero primo conosciuto “n” come “centro della simmetria”, ovvero come centro della semicirconferenza tracciata avente come centro il punto “n”. Il valore dell'ultimo numero primo conosciuto lo considereremo anche come “raggio di simmetria”.

Elementi della progressione del numero 2 dentro il raggio di simmetria.
Quanti elementi avrà la progressione base 2 e ragione 2 all'interno del raggio di simmetria? Il numero di elementi (di seguito nominato “n_due”) sarà pari a: (“raggio_di_simmetria” – 1) / 2.
A quanto ammonterà il valore dell'ultimo elemento della progressione del 2 (di seguito nominato “ultimo_due”) all'interno del raggio di simmetria?
“ultimo_due” = 2 + (“n_due” – 1) * 2 .
Questo calcolo si basa sulla formula per le progressioni aritmetiche:

$$a_n = a_1 + (n - 1) * d;$$

dove a_n è l'ultimo elemento della progressione, a_1 è il primo elemento della progressione, n è il numero di elementi della progressione, d è la ragione.

Fase di calcolo 1
Moduli delle traiettorie incompiute
Per ogni numero primo conosciuto calcolare il modulo (o resto) della divisione tra “centro_di_simmetria” e il numero primo esaminato, quindi memorizzare il modulo che rappresenta, per il relativo numero primo a esso associato, la sua traiettoria incompiuta all'interno del raggio di simmetria.

Fase di calcolo 2
Traiettorie mancanti
Per ciascun numero primo calcolare la differenza tra il numero primo esaminato e il relativo modulo: questa differenza è la traiettoria mancante del relativo numero primo esaminato all'interno della “gittata numerica” oltre l'ultimo numero primo conosciuto, comunque prima di 2n .

Fase di calcolo 3
Creazione delle progressioni dei numeri primi dispari inferiori a "n" all'interno della "gittata numerica" in base alle traiettorie mancanti

Per ciascun numero primo calcolare gli elementi della propria progressione all'interno della gittata numerica (che è pari al raggio di simmetria) in base alle traiettorie mancanti e in base alle seguenti regole:

- se la traiettoria mancante è pari, la base della progressione equivale alla traiettoria mancante;
- se la traiettoria mancante è dispari, la base della progressione equivale alla traiettoria mancante + il relativo numero primo;
- calcolare la progressione fino a che l'ultimo elemento della progressione è minore o uguale all'ultimo elemento della progressione del due che abbiamo calcolato in precedenza ("ultimo_due"), secondo la seguente regola di calcolo che ci permette di saltare i numeri composti pari all'interno della gittata numerica: $k_y = k_x + 2 * p$; dove k_y rappresenta il successivo elemento della progressione numerica esaminata, k_x il precedente elemento della progressione in questione, p rappresenta il numero primo relativo alla progressione che stiamo esaminando;
- memorizziamo gli elementi delle varie progressioni calcolate all'interno della "gittata numerica": queste sono le intersezioni con la semiretta dei numeri naturali.

Fase di calcolo 4
Calcolo degli zeri, posizioni dei nuovi numeri primi

Calcoliamo ogni elemento della progressione numerica del 2 all'interno della gittata numerica fino al limite di simmetria, in base zero e ragione 2, quindi in progressione verifichiamo l'esistenza di intersezioni con la semiretta dei numeri naturali: se l'elemento considerato esiste nelle precedenti progressioni numeriche, vuol dire che esiste già l'intersezione con la semiretta dei numeri naturali, quindi siamo in presenza di un numero composto; se invece l'elemento della progressione del due - che stiamo esaminando - non esiste nelle precedenti progressioni, vuol dire che siamo in presenza di un vuoto numerico, uno spazio vuoto ovvero di uno "zero" non banale, come direbbe Riemann. La somma della posizione dello zero + il numero primo "n" (ovvero l'ultimo numero primo conosciuto che abbiamo nominato anche "centro di simmetria") dà come risultato il successivo numero primo nella successione dei numeri primi. Continuare con i calcoli fino al limite di simmetria, perché molto probabilmente troveremo ulteriori numeri primi all'interno della gittata numerica.

Note
Nel caso in cui non dovessero essere trovati nuovi numeri primi all'interno della gittata numerica (la quale è pari al raggio di simmetria), procedere con una nuova gittata numerica. In base al postulato di Bertrand, però, è alquanto ragionevole ipotizzare che all'interno della gittata numerica pari al raggio di simmetria (ovvero fino alla posizione 2n) troveremo sempre dei numeri primi.

Ho eseguito una rudimentale implementazione di questo algoritmo di calcolo in linguaggio C/C++ ed è possibile eseguire il programma per verificare in parte le funzionalità di questo metodo di calcolo, o meglio per chiarire le idee su come vanno formulati effettivamente i calcoli.

Esempio di applicazione dell'algoritmo denominato "metodo del compasso e della gittata numerica".

Data la successione dei numeri primi da 2 a 19.

Fase di calcolo 0
Centro di simmetria e raggio di simmetria.

```
n = 19;     // n = elemento[8];
centro_simmetria = 19;   // centro_simmetria = n;
```

Elementi della progressione del numero 2 dentro il raggio di simmetria.

```
n_due = (19 – 1) / 2;              //  n_due = (n – 1) / 2;
```

Fase di calcolo 1
Moduli delle traiettorie incompiute

```
// modulo[i] = n % elemento[i];

modulo[2] = 19 %  3;      // modulo[2] = 1;
modulo[3] = 19 %  5;      // modulo[3] = 4;
modulo[4] = 19 %  7;      // modulo[4] = 5;
modulo[5] = 19 % 11;      // modulo[5] = 8;
modulo[6] = 19 % 13;      // modulo[6] = 6;
modulo[7] = 19 % 17;      // modulo[7] = 2;
```

Fase di calcolo 2
Traiettorie mancanti

```
// tm[i] = elemento[i] - modulo[i];

tm[2] =  3 - modulo[2];    // tm[2] =  3 – 1;   -->   2
tm[3] =  5 - modulo[3];    // tm[3] =  5 – 4;   -->   1
tm[4] =  7 - modulo[4];    // tm[4] =  7 – 5;   -->   2
tm[5] = 11 - modulo[5];    // tm[5] = 11 – 8;   -->   3
tm[6] = 13 - modulo[6];    // tm[6] = 13 – 6;   -->   7
tm[7] = 17 - modulo[7];    // tm[7] = 17 – 2;   --> 15
```

Fase di calcolo 3
Creazione delle progressioni dei numeri primi dispari inferiori a "n" all'interno della "gittata numerica" in base alle traiettorie mancanti

- progressione del numero 3, in base alla traiettoria mancante:
  base = 2; // tm[i] è pari, quindi: base=tm[i];
  ragione = 2 * elemento[i];

  quindi otteniamo 2, 8, 14, ~~20;~~
- progressione del numero 5, in base alla traiettoria mancante:
  base = 1 + elemento [i]; // tm[i] è dispari, quindi: base=tm[i]+elemento[i];
  ragione = 2 * elemento[i];

  quindi otteniamo 6, 16, ~~26;~~

- progressione del numero 7, in base alla traiettoria mancante:
  2, 16, ~~30;~~

- progressione del numero 11, in base alla traiettoria mancante:
  14, ~~36~~ ;

- progressione del numero 13, in base alla traiettoria mancante:
  ~~20~~;

- progressione del numero 17, in base alla traiettoria mancante:
  ~~32~~ ;

Fase di calcolo 4
Calcolo degli zeri, posizioni dei nuovi numeri primi

In base alla fase di calcolo 3, abbiamo notato che esistono le seguenti intersezioni con la semiretta dei numeri naturali, all'interno della gittata numerica:
2,8,14;
6,16;
2,16;
14.

La progressione in base zero e ragione due fino al limite di simmetria n, produce 9 elementi:
2, 4, 6, 8, 10, 12, 14, 16, 18.

I seguenti numeri non esistono tra le intersezioni, sono i primi spazi vuoti o varchi numerici denominati zeri, ovvero le posizioni dei nuovi numeri primi dentro la gittata numerica: 4, 10, 12, 18.

La formula quindi è semplice: zero = n + posizione;

// elemento[i] = n + posizione;

elemento[9] = n + 4; // 23 = 19 + 4;
elemento[10] = n + 10; // 29 = 19 + 10;
elemento[11] = n + 12; // 31 = 19 + 12;
elemento[12] = n + 18; // 37 = 19 + 18;

ATTENZIONE !!! LEGGERE CON CURA LE SEGUENTI CONDIZIONI DI UTILIZZO PRIMA DI PROSEGUIRE NELLA LETTURA DEL DOCUMENTO.

Accordo di Licenza d'uso a titolo non esclusivo tra il proprietario della proprietà intellettuale e l'utente finale.

Il proprietario intellettuale è Ismaele Alongi (di seguito indicato come “l'autore”), nato nel 1977 in Italia, autore dell'opera oggetto della proprietà intellettuale dal titolo “Genesi dei Numeri Primi - Implementazione rudimentale dell'algoritmo detto “metodo del compasso e della gittata numerica” in linguaggio C/C++.

La suddetta opera oggetto della proprietà intellettuale è costituita dal programma per computer e l'algoritmo in esso espresso e rappresentato, entrambi ideati, creati e scritti dall'autore Ismaele Alongi (di seguito indicata come “il seguente programma per computer e l'algoritmo in esso rappresentato”), la quale opera si trova nelle pagine seguenti di questo documento.

L'utente finale è chiunque decide di leggere, utilizzare o impiegare il seguente programma per computer e l'algoritmo in esso rappresentato, accettando senza riserve le condizioni di questo accordo di licenza d'uso.

Il seguente programma per computer e l'algoritmo in esso rappresentato sono concessi in licenza d'uso a titolo non esclusivo “così per come sono” e senza alcuna garanzia implicita o esplicita, per un solo utente e per un solo computer. Il seguente programma per computer e l'algoritmo in esso rappresentato non sono freeware, bensì vengono semplicemente mostrati al pubblico per meglio esprimere e rendere meglio comprensibili le idee dell'autore al fine di comunicarle al pubblico.

Chiunque non è d'accordo con le condizioni di questo accordo di licenza non è autorizzato a leggere e/o utilizzare in alcun modo il seguente programma per computer e l'algoritmo in esso rappresentato.

Chi invece è d'accordo con le condizioni del presente accordo di licenza, concorda con l'autore in quanto l'utente finale non chiederà alcun tipo di rimborso per danni diretti o indiretti per l'uso proprio e/o improprio di quanto contenuto e/o eseguito dal seguente programma per computer e dall'algoritmo in esso rappresentato e per quanto in esso espresso e/o contenuto, né l'utente finale deve recare danni ad altri soggetti mediante l'uso del seguente programma per computer e dell'algoritmo in esso rappresentato. L'utente finale che è d'accordo con il presente accordo di licenza, il quale utente decide di leggere e/o eseguire e/o utilizzare e/o impiegare in pubblico il seguente programma per computer e l'algoritmo in esso rappresentato, deve comunicare alle persone che direttamente o indirettamente fruiranno dell'applicazione che ne deriva, le condizioni del presente accordo di licenza e mettere tali persone nelle condizioni di decidere individualmente e di esprimere la propria decisione, al fine di accettare o rifiutare le condizioni del presente accordo di licenza, quindi ne conseguirà l'utilizzo autorizzato o meno, in base alle condizioni espresse nel presente accordo di licenza.

L'autore si riserva qualsiasi forma di utilizzazione economica della propria proprietà intellettuale (in special modo per l'algoritmo) e l'utente finale che è d'accordo con il presente accordo di licenza acconsente esplicitamente a questa limitazione d'utilizzo.

La lettura e/o l'uso e/o l'impiego in qualsiasi forma e con qualsiasi mezzo del seguente programma per computer e dall'algoritmo in esso rappresentato, comportano da parte dell'utente finale l'accettazione esplicita e senza riserve delle condizioni espresse nel presente accordo di licenza d'uso a titolo non esclusivo tra il proprietario della proprietà intellettuale e l'utente finale.

GENESI DEI NUMERI PRIMI

Implementazione rudimentale dell'algoritmo detto “metodo del compasso e della gittata numerica” in linguaggio C/C++


```
// Copyright 2006 by Ismaele Alongi. All rights reserved.
// Tutti i diritti riservati in Tutti i Paesi.

// "La bellezza, la sapienza, la potenza e la gloria appartengono a Dio,
//  l'Altissimo, Colui che fa divenire. Egli ci dà la gioia e il
//  privilegio di poter conoscere il Suo eterno proposito per l'uomo."
//                                                    Ismaele Alongi.

// Una prima rudimentale implementazione di una verifica della "Legge di Genesi e
// Struttura dei Numeri Primi", ipotizzata da Ismaele Alongi, e dell'algoritmo
// del “compasso e della gittata numerica” dell'autore Ismaele Alongi.
// Altri nomi da proporre sono: "Legge di successione numerica dei numeri primi"
// oppure "ordine di progressione numerica dei numeri primi"

//-------------------------------------------------------------------------
// Note sul funzionamento del programma: il programma usa come input un
// file di testo con i numeri primi iniziali, quindi genera come output
// un file di testo con i numeri primi individuati nell'ordine successivo
// a quelli iniziali.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <limits.h>
#include <values.h>
#include <float.h>
#include <iostream>
#include <fstream>

#define FALSE 0
#define TRUE 1

using namespace std;

void inizializza_liste(int elementi[]);
void verifica_memorizza_esistenza(int ex, int esistenza[]);
void compasso_numerico(int elementi[]);

main ()
{
int primi_iniziali[1000]; // array dei numeri primi iniziali

inizializza_liste(primi_iniziali);// inizializzazione lista numeri primi

compasso_numerico(primi_iniziali);
                          // funzione di calcolo dei nuovi numeri primi:
                          // implementazione rudimentale, vedi dettagli.

} // fine main program
```

```
//---------------------------------------------------------------------------
void compasso_numerico(int elementi[])
{
// dichiarazione variabili
FILE *destinazione;

int moduli[1000], // array dei moduli
              tm[1000], // array delle traiettorie mancanti
              esistenza[1000]; // array delle intersezioni

int centro_simmetria, i, j,
              base, k, n,
              n_due, ultimo_due,
              posizione=0, zero;

short intersezione=FALSE;

esistenza[0]=0;

destinazione=fopen("nuovi_primi.txt", "w");

// fase calcolo 0: centro di simmetria
n=elementi[0]; // ennesimo primo conosciuto
centro_simmetria=elementi[n];// è l'ultimo numero primo conosciuto

cout << "Eseguito calcolo 0: centro di simmetria\n";

//elementi della progressione del numero due dentro il raggio di simmetria
n_due=(centro_simmetria - 1) / 2; // numero di elementi della
                                  // progressione del due

ultimo_due= 2 + (n_due - 1) * 2; // formula: An = A1 + (n - 1) * d;
                                 // dove An è il termine ennesimo cercato,
                                 // A1 è il primo elemento della progressione,
                                 // n è il numero di elementi della progr.,
                                 // d è la ragione.
cout << "Eseguito calcolo elementi della progressione del numero due dentro "
     <<"il raggio di simmetria.\n";

for (i=2; i<=n; i++)
    cout << "Numeri primi conosciuti, "<< i << " = " << elementi[i] <<"\n";
    cout << "altro "<< elementi[20]<< "altro " << elementi[0]<<"\n";

for (i=2; i<=n; i++)
    {
      // fase calcolo 1: moduli delle traiettorie incompiute
      moduli[i]=centro_simmetria%elementi[i];//esegue il modulo e memorizza
      cout << "Eseguito calcolo 1: moduli delle traiettorie incompiute\n"
           << "moduli[" << i << "] = " << moduli[i] << "\n";

      // fase calcolo 2: traiettorie mancanti
      tm[i]=elementi[i]-moduli[i];
                              // calcola le traiettorie mancanti e memorizza

      cout << "Eseguito calcolo 2: traiettorie mancanti\n"
           << "tm[" << i << "] = " << tm[i] << "\n";
```

```
      // fase calcolo 3: creazione delle progressioni numeriche
      // progressioni dei restanti numeri primi,
      // in base alle traiettorie mancanti

      if (tm[i] % 2 == 0)
         {base=tm[i];}
         // caso pari: base progressione = traiettoria mancante

         else base = tm[i] + elementi[i];
         // caso dispari:
         // base progr. = traiettoria mancante + n. primo relativo

      k=base;

      cout << "base = " << base << "\n";

      if (k == ultimo_due)
          verifica_memorizza_esistenza(k, esistenza);

      if (k < ultimo_due)
         { verifica_memorizza_esistenza(k, esistenza);

           while (k < ultimo_due)
                 { k = k + 2 * elementi[i]; // saltiamo le coincidenze con i
                                            // numeri composti pari
                   if (k<=ultimo_due)
                        verifica_memorizza_esistenza(k, esistenza);
                 }; // fine while
         }; // fine if (k< ultimo_due)

      cout << "Eseguito calcolo 3: creazione delle progressioni numeriche\n"
           << "progressioni dei restanti numeri primi, in base alle "
           << "traiettorie mancanti \n";

      cout << "indice del for " << i << "\n";

    }; // fine for delle fasi di calcolo 1, 2, 3

// fase 4: calcolo degli zeri, posizioni dei nuovi numeri primi
// - calcolare la progressione numerica del 2 fino al limite di simmetria
// - verificare l'esistenza degli elementi della progressione numerica:
//    -- se il numero non esiste, questo è lo "zero", quindi eseguire la
//       somma: centro_simmetria + posizione;
// - il ciclo termina quando si arriva al limite di simmetria
// N.B.: bisogna continuare a raffinare il programma per quando i nuovi
// numeri primi saranno più rari.... perché non so se all'interno della
// gittata della simmetria ci sono sempre nuovi numeri primi, anche se
// si potrebbe azzardare l'ipotesi che ci siano sempre.

for (i=1; i<=n_due; i++)
    {
     posizione=posizione + 2; // stiamo calcolando gli elementi della
                              // progressione del due, uno alla volta

     for (j=1; j<=esistenza[0]; j++)
         if (posizione==esistenza[j])
             { intersezione=TRUE;
               j=esistenza[0];
             };

     if (intersezione==FALSE)  // una volta chiuso il for annidato calcola
                               // lo zero se c'è la condizione di vuoto
        { zero=centro_simmetria + posizione;
          fprintf (destinazione,"%ld\n", zero);
        }
       else intersezione=FALSE;
```

```
      cout << "Eseguito calcolo 4: calcolo degli zeri, posizioni dei nuovi "
           << "numeri primi\n...indice del for "<< i <<"\n";

     }; // fine del "for" della progressione del due "a colmare"
        // dentro il raggio di simmetria

// N.B.: Se non vengono trovati nuovi numeri primi all'interno del raggio
// di simmetria, continuare la ricerca con una nuova "gittata numerica",
// fino a che non li troviamo. Questa opzione è da implementare.

fclose(destinazione);
} // fine funzione compasso_numerico

//---------------------------------------------------------------------------
void inizializza_liste(int elementi[])
{
FILE *origine;

int x=0;

origine=fopen("primi.txt", "r");

while (fscanf(origine, "%ld", &elementi[x+1]) !=EOF)
      x++;

fclose(origine);

elementi[0]=x; // indice dell'array

} // fine procedura inizializza_liste

//---------------------------------------------------------------------------
void verifica_memorizza_esistenza(int ex, int esistenza[])
{
short duplicato=FALSE;
int w;

if (esistenza[0]==0)
   { esistenza[1]=ex;
     esistenza[0]=1;
   }
   else
        { for (w=1; w<=esistenza[0]; w++)
             if (esistenza[w]==ex)
                { duplicato=TRUE;
                  w=esistenza[0];
                }; // fine for--if

             if (duplicato==FALSE)
                { esistenza[0]++;
                  w=esistenza[0];
                  esistenza[w]=ex;
                };
        }; // fine else
} // fine procedura verifica_memorizza_esistenza
```