

TEST DI PRIMALITÀ, FATTORIZZAZIONE E $\pi(N)$ CON FORME $6k \pm 1$

ing. Rosario Turco, dott. Michele Nardelli, prof. Giovanni Di Maria, Francesco Di Noto, prof. Annarita Tulumello.

Sommario

Nel seguito è mostrato un Teorema di Test di Primalità ed alcuni suoi corollari; poi una tecnica di Fattorizzazione ed il calcolo preciso di $\pi(N)$, il tutto basato sulle forme dei numeri $6k \pm 1$; infine si discutono e si allegano gli algoritmi necessari.

Premessa

Da vari riferimenti in appendice e diversi libri di Teoria dei Numeri è noto che la forma dei numeri che permette di ottenere il maggior numero di primi, ad eccezione del 2 e del 3, è $6k \pm 1$.

Eulero, ad esempio, affermava che per $k > 3$ le forme $6k-1$ e $6k+1$ erano quelle meno dispersive. Tali forme in passato sono state anche studiate per il solo fatto che il 6 costituisce un “numero perfetto”, ovvero è la somma dei suoi fattori 1,2,3: è considerato anche l'1, anche se esso è l'elemento neutro in termini di prodotto e di fattorizzazione.

Sono note, inoltre, diverse proprietà delle forme $6k \pm 1$, osservabili facilmente su una tabella, se su tre sue colonne si dispongono i valori interi k , $6k-1$, $6k+1$. La figura successiva riporta solo dieci valori di k e con un “pallino rosso” sono stati marcati i numeri composti. L'estensione della tabella e la verifica di quanto di seguito risulta, è lasciata al lettore.

k	6k-1	6k+1
1	5	7
2	11	13
3	17	19
4	23	25•
5	29	31
6	35•	37
7	41	43
8	47	49•
9	53	55•
10	59	61

Tabella 1

Proprietà 1

- I numeri composti della forma $6k-1$ sono del tipo $(6k-1)(6k+1)$ detto “prodotto esterno”.
- I numeri composti della forma $6k+1$ sono del tipo $(6k-1)(6k-1)$ o $(6k+1)(6k+1)$ detti “prodotti interni”.

Dalla Tabella 1.

Esempi: caso a: $5=6*1-1$; $7=6*1+1$; $35=7*5$ composto
caso b: $5=6*1-1$; $11=6*2-1$; $55=5*11$ composto

Proprietà 2

A parità di k , è maggiore il numero di composti esclusi della forma $6k+1$ rispetto alla forma $6k-1$ (differenza quantitativa).

La cosa è evidente dalla tabella. Alcuni approfondimenti di tale proprietà potrebbero portare a formule ulteriori (o equivalenti) per il calcolo del numero di primi fino ad un certo valore M fissato. Tuttavia allo stato attuale una eventuale formula del genere necessita di correttivi attraverso costanti o funzioni correttive.

Proprietà 3

La somma delle cifre dei numeri in forma $6k-1$ dà come risultato 2, 5 o 8.

La somma delle cifre dei numeri in forma $6k+1$ dà come risultato 1, 4 o 7.

Esempio: 4289 dà come somma 23 e risommandolo dà 5.

Proprietà 4

Un numero in forma $6k-1$ è primo se non è un composto della propria colonna, ovvero risulta diverso dal prodotto esterno $(6k'-1)(6k''+1)$.

Un numero in forma $6k+1$ è primo se non è un composto della propria colonna, ovvero risulta diverso dai due prodotti interni $(6k'-1)(6k''-1)$ e $(6k'+1)(6k''+1)$.

Teorema TP $6k \pm 1$

Condizione necessaria e sufficiente affinché un N , intero e maggiore di 4, sia un numero primo è che si verifichi una delle seguenti alternative:

- N è esprimibile in forma $6k-1$, ovvero $k = (N+1)/6$ sia un valore intero o in alternativa che il modulo $(N,6)=5$, e che non esistono valori di k' e k'' per cui sia vera l'uguaglianza $36k'k''+6k'-6k''-1=N$;
- N è esprimibile in forma $6k+1$, ovvero $k = (N-1)/6$ sia un valore intero o in alternativa che il modulo $(N,6)=1$, e che non esistano dei valori k' e k'' tali che almeno una delle due eguaglianze $36k'k''-6k'-6k''+1=N$ e $36k'k''+6k'+6k''+1=N$ sia vera.

Dimostrazione

Dato un N , si identifica subito di che forma N è se è vera una sola delle due condizioni:

- $K = (N+1)/6$ è un intero con N di tipo $6K-1$
- $K = (N-1)/6$ è un intero con N di tipo $6K+1$

Se è vera la prima, si tratta di una forma $6n-1$. Affinché N non sia un composto e quindi sia un numero primo deve essere vera la proprietà 4:

$$N \neq (6k'-1)(6k''+1)$$

La sola condizione $(N+1)/6$ valore intero non è da sola sufficiente a far sì che N sia un numero primo, ma è necessario rispettare la proprietà 4. D'altra parte per essere di forma $6n-1$, il modulo $(N,6)=5$, cioè cade nella colonna del 5.

Se è vera la seconda, si tratta di una forma $6n+1$. Affinché N non sia un composto e quindi sia un numero primo deve essere vera la proprietà 4:

$$N \neq (6k'-1)(6k''-1) \quad (\text{PE})$$

$$N \neq (6k'+1)(6k''+1) \quad (\text{SE})$$

Anche qui la sola proprietà 4 non è sufficiente, è necessario anche che N sia nella forma $6n+1$ p che K sia intero. D'altra parte per essere di forma $6n-1$, il modulo $(N,6)=1$, cioè cade nella colonna del 7.

Per il test è sufficiente che fissato N , si faccia variare k' e k'' e si verificano i valori assunti da (PE) e (SE); se (PE) e (SE) superassero N senza uguagliarlo, il test si può arrestare. Se una sola delle uguaglianze è verificata, allora il numero N in forma $6n+1$ non è primo. In caso contrario è primo.

Esempio

$$N=13=6 \cdot 2+1$$

$$(\text{PE}) = 25$$

$$(\text{SE}) = 49$$

Test arrestato, N primo.

Corollario 1

Dati due numeri $N1$ in forma $6n-1$ e $N2$ in forma $6n+1$, interi e maggiori di 4, se rispettano il Teorema TP $6k \pm 1$, allora $N1$ e $N2$ sono numeri gemelli.

Corollario 2

I numeri primi 3 e 5 non sono numeri gemelli di forma $6k \pm 1$, perché non afferiscono allo stesso k.

Corollario 3

I numeri N che presentano $\text{modulo}(N,6) \neq 5$ e $\text{modulo}(N,6) \neq 1$ sono sicuramente composti.

Teorema di Fattorizzazione con forme $6k \pm 1$

Condizione sufficiente per ottenere una scomposizione in fattori di un numero N è di dividerlo per uno o più dei seguenti divisori: 2, 3 $(6n-1)(6n+1)$, $(6n-1)(6n-1)$, $(6n+1)(6n+1)$.

Dimostrazione

Si può dimostrare il tutto attraverso l'operazione $N/\text{modulo}(N,6)$ e verificare in quali casi si ottiene un intero come risultato. Il $\text{modulo}(N,6)$ viene introdotto a causa del fatto che la maggioranza dei numeri primi è di forma $6k \pm 1$.

E' evidente che il $\text{modulo}(N,6)$ può dare come risultato solo i seguenti valori come resto:

- 0,2,4,8
- 1,3,5

Non si potrà mai ottenere resto 6. Ad esempio $60=6*10$ e divisibile ancora per 6, quindi resto 0.

Per i resti 0,2,4,8 si tratta sempre di numeri N pari e conviene dividerli per 2; mai per 0, 4 e 8 per evitare di incappare in un contro-esempio.

Ad esempio:

- $\text{modulo}(12,6)$ dà resto 0, ma $N/\text{modulo}(N,6)$ è indivisibile, per cui essendo pari conviene dividerlo per 2;
- $\text{modulo}(14,6)$ dà resto 2, quindi $N=14$ è divisibile per $\text{modulo}(N,6)=2$;
- $\text{modulo}(22,6)$ dà resto 4 ma $22/4$ non è un intero, conviene dividerlo per 2;
- $\text{modulo}(62,6)$ dà resto 8 ma $62/8$ non è un intero, conviene dividerlo per 2;

Quando $\text{modulo}(N,6)=5$ o $=1$ si possono trovare i fattori rispettivamente con:

- $(6n-1)(6n+1)$
- $(6n-1)(6n-1)$ o $(6n+1)(6n+1)$

Se il $\text{modulo}(N,6)=3$ è sempre possibile fare $N/\text{modulo}(N,6)$ perché otteniamo un intero.

Esempio $\text{modulo}(69,6)=3$ $69/3=23$

$\text{Modulo}(N,6)=7$ è impossibile che esca, perché in tal caso esce sempre resto 1 e si ricade nei casi precedenti.

$\text{Modulo}(N,6)=9$ è impossibile perché al massimo dal $\text{modulo}(9,6)=3$.

Esempio: $\text{modulo}(63,6)=3$.

Tutto questo è in pieno accordo con la Teoria dei Numeri primi; cioè i fattori o i divisori possibili di un N qualsiasi sono i numeri primi 2, 3 e tutti quelli ricavabili da forme $6k \pm 1$.

Forme $6k \pm 1$ e operazioni su esse

Proprietà 5 - somma

Sia $p=6m+1$, $q=6n-1$:

$$(1) \quad p+q=6(m+n) \quad [\text{segni discordi}]$$

La (1) è vera sia che p e q sono numeri primi, sia che sono composti o misti. In particolare se p e q sono primi gemelli nella (1) $m=n$ allora $p+q=12n$. I numeri primi gemelli sono legati anche al problema di Goldbach – vedi Teorema gruppo ERATOSTENE o lavoro su Goldbach su **Aladdin's Lamp** o su **CNR Solar** (vedi riferimenti).

Sia $p=6m+1$, $q=6n+1$:

$$(2) \quad p+q=6(m+n)+2 \quad [\text{segni concordi}]$$

La (2) è valida per primi sia per composti o misti

Proprietà 6 - differenza

Sia $p=6m+1$, $q=6n-1$:

$$(3) \quad p-q=6(m-n) +2 \quad [\text{segni discordi}]$$

La (3) è valida per primi e per composti o misti. In particolare se p e q sono primi gemelli nella (1) $m=n$ allora $p-q=2$; il che giustifica che sono gemelli.

Un particolare: 3 e 5 non sono gemelli; difatti è vero che la differenza $5-3=2$, ma la loro somma $5+3=8$ non è multiplo di 12.

Sia $p=6m+1$, $q=6n+1$:

$$(4) \quad p-q=6(m-n) \quad [\text{segni concordi}]$$

La (4) è valida per primi e per composti o misti

Proprietà 7 - prodotto

Sia $p=6m+1$, $q=6n-1$:

$$(5) \quad P= pq = (6m+1)(6n-1)=36mn-6m+6n-1 \quad [\text{segni discordi}]$$

Nel caso di numeri gemelli $m=n$ la (5) diventa:

$$(6) \quad P= pq=36m^2-1=6^2m^2-1=Q-1$$

La (5)(6) è valida per primi e per composti o misti.

Sia $p=6m+1$, $q=6n+1$:

$$(7) \quad P=pq=(6m+1)(6n+1) \quad [\text{segni concordi}]$$

La (7) è valida per numeri primi e per composti o misti.

Quadrati perfetti

Sia $P=pq$ e $Q=6^2m^2$, con p e q primi gemelli, allora il prodotto di due numeri gemelli è nella forma $P=Q-1$ dove Q è un quadrato perfetto di un intero.

Però il quadrato perfetto è valido anche per casi di numeri non gemelli. Ad esempio 4095 è scomponibile in 63 e 65 che non sono numeri gemelli, perché non sono primi.

Fattorizzazione veloce col Quadrato perfetto

Dalla (6) si possono fare delle considerazioni.

Sia $P=323$, $Q=N+1=324=18^2$, allora $323=(18-1)(18+1)=(6*3-1)(6*3+1)=17*19$ (ovviamente gemelli con $m=3$). In particolare 18 è la media aritmetica di 17 e 19, tale fatto è sempre vero per definizione di numero gemello. In pratica dato un numero P, si somma 1 e si estrae la radice quadrata, se è un intero ed è un quadrato perfetto siamo di fronte a due numeri primi gemelli, a destra e a sinistra del numero trovato. Si potrebbe cioè introdurre un algoritmo che effettua dei controlli per stabilire se il numero è un quadrato perfetto; se lo è deve fornire i due valori (due numeri gemelli). Mentre se non fosse un quadrato perfetto, si può procedere, invece con un'altra tecnica, come vedremo. Ad esempio nell'algoritmo proposto successivamente il numero 1048575 è scomposto in soli 3 passi contro 6 passi di una fattorizzazione classica che cerca tutti i divisori a partire da 2 e con tempi leggermente più bassi. E' chiaramente altamente sconsigliabile, in ambito crittografico, che un numero RSA sia il prodotto di due numeri gemelli o il prodotto di due numeri che si possano scomporre attraverso il quadrato perfetto.

Algoritmo di Fattorizzazione F 6k± 1

E' possibile implementare un algoritmo di fattorizzazione mettendo insieme la teoria appresa dal "Teorema TP 6k±1", dal "Teorema di Fattorizzazione con forme 6k±1" e dalla "Fattorizzazione veloce col Quadrato perfetto". Il metodo è semplice però richiede diversi passi.

L'algoritmo deve tener conto del fatto che un numero scomposto in fattori può dare luogo a due ulteriori numeri, di cui il primo scomponibile ulteriormente e il secondo da considerare successivamente; ovviamente questo è vero ad ogni passo dell'algoritmo. Per cui è necessario disporre di uno "stack di memoria" da cui estrarre i fattori accantonati (primi o composti) e da processare successivamente.

La logica dell'algoritmo si può basare sui seguenti passi:

1. Si pone $nt \leftarrow N$; dove N è il numero da fattorizzare.
2. Se nt è minore di 12, nt si ritiene fattorizzato; altrimenti si prosegue al passo 3.
3. Si verifica se è un quadrato perfetto; se lo è si ricavano due valori. Il primo valore prosegue l'elaborazione, mentre il secondo è posto nello stack, per essere ripreso successivamente e si incrementa la posizione dello stack. Se non è un quadrato perfetto si prosegue al passo 4
4. Si ricava il modulo($nt,6$).
5. Se il risultato del modulo è 0,4,8 si divide nt per 2 e si memorizza il risultato ancora in nt ; intanto un fattore ricavato è il 2 e si ritorna al passo 2; altrimenti se il modulo non è 0, 4,8 si prosegue al passo 5
6. Se il modulo è diverso da 5 e da 1 (è quindi 2 e 3) si effettua la divisione $nt = nt/\text{modulo}(nt,6)$: un fattore trovato è proprio il valore dato dal modulo($nt,6$) e si ripete il passo 3; altrimenti se modulo($nt,6$) vale 5 o 1 si va al passo 6 o al passo 7 rispettivamente; .
7. Se il modulo($nt,6$)=5 occorre verificare se è possibile scomporlo ancora in $V=(6k'-1)(6k''+1)$

Per trovare k' e k'' , si pone $k' \leftarrow 1, k'' \leftarrow 1$;

- a) si fa un immediato controllo: se con $k'=1, k''=1, V > nt$ allora nt è sicuramente primo, altrimenti si prosegue al passo b;
- b) è noto nt , si fissa k' ad un valore e si ricava con formula K".

$$k'' = \frac{\frac{n_t + 1}{6} - k'}{6k' - 1}$$

e con i valori k' e k'' si calcola V . Ora se $V < nt$ si incrementa k' e si ritorna ad b; se $V > nt$ si va in c; se $V=nt$ allora si è trovata la giusta combinazione di k' e k'' e si passa in d.

- c) se si arriva in c, il numero è primo ma prima di uscire si controlla lo stack, se lo stack è vuoto si esce; se contiene fattori si riparte dal passo 2 con nt pari al valore dello stack;
 - d) se si arriva in d calcoliamo i fattori $(6k'-1)$ e $(6k''+1)$ e si prosegue in e; se rimane invece un solo numero si passa a f
 - e) si pone in uno stack il secondo dei due fattori e si continua a processare il primo ripetendo i passi precedenti a partire da 2
8. Se il modulo($nt,6$)=1 occorre verificare se è possibile scomporlo ancora secondo una delle due formule:
 - $(6k'-1)(6k''-1)=V$
 - $(6k'+1)(6k''+1)=V$

E' ovviamente sufficiente fattorizzare nt con almeno una sola delle due formule, se possibile. Se si è fortunati ne basta una, altrimenti occorre provare anche la seconda.

Per trovare k' e k'' , si pone $k' \leftarrow 1, k'' \leftarrow 1$; E si procede con la prima formula di cui sopra;

- a) si fa un immediato controllo: se con $k'=1, k''=1, V > nt$ allora nt è sicuramente primo, altrimenti si passa a b;
- b) è noto nt , si fissa k' ad un valore e si ricava con formula K".

$$k'' = \frac{\frac{n_t - 1}{6} + k'}{6k' - 1}$$

Con i valori k' e k'' si calcola V . Ora se $V < nt$ si incrementa k' e si ritorna ad b ; se $V > nt$ si va in c ; se $V = nt$ allora si è trovata la giusta combinazione di k' e k'' e si passa in d .

- c) se si arriva in c , il numero è primo ma prima di uscire si controlla lo stack, se lo stack è vuoto si esce; se contiene fattori si riparte dal passo 2 con nt pari al valore dello stack;
- d) se si arriva in d calcoliamo i fattori $(6k'-1)$ e $(6k''-1)$ e si prosegue in e
- e) Si pone in uno stack il secondo dei fattori ricavati e si continua a processare il primo, ripetendo i passi precedenti a partire dal passo 2; se rimane invece un solo numero si passa a f

Se con la prima formula non si riesce a trovare i valori di k' e k'' va provata anche la seconda formula. In particolare al passo b va sostituita la formula con:

$$k'' = \frac{\frac{n_t - 1}{6} - k'}{6k' + 1}$$

E al punto d va tenuto conto che vanno calcolati i fattori $(6k'+1)$ e $(6k''+1)$.

Se neanche con la seconda formula si ottiene nulla, allora nt è primo e costituisce il fattore finale. Se, invece, per una delle due formule, si sono trovati due fattori, essi vanno memorizzati e per ognuno di essi occorre verificare se vanno fattorizzati ulteriormente, per cui si ritorna al punto 2 per ognuno.

- 9. Se si è terminato con ogni elaborazione si controlla se lo stack è vuoto. Se lo stack presenta un elemento alla posizione corrente si pone nt pari all'elemento, si decrementa la posizione dello stack e si ritorna al passo 2

La strategia dell'algoritmo è quella di cercare innanzitutto una "riduzione veloce" del numero di partenza in pochi passi. Prima prova se ci si può ricondurre ad un quadrato perfetto per ricavare rapidamente due numeri; se non è possibile divide il numero per il modulo $(N,6)$, tra l'altro già calcolato per stabilire la tipologia della forma, senza tentare di cercare, come nel metodo classico, tutti i divisori a partire da 2.

L'algoritmo evita, nel contempo, di fattorizzare valori al di sotto del 12 che vengono considerati, comunque, dei fattori del numero (composti o primi); ovvero si traslascia la scomposizione banale di fattori elementari; esempio: $4=2*2, 6=2*3, 10=2*5$, anche perché nelle forme $6k \pm 1$ non è possibile ottenere 2,3. La probabilità di riuscita della "riduzione veloce" è buona perché è legata al fatto che la maggioranza dei composti cade fuori dalle colonne 5 e 7.

I casi limite di modulo uguale a 0, 4, 8 vengono superati dividendo semplicemente per 2 il numero pari, mentre nei casi di resto 5 e 1 si adotta una strategia legata alla forme $6k \pm 1$.

Competitività dell'algoritmo

Facciamo alcune considerazioni sulla velocità e la strategia dell'algoritmo.

L'algoritmo ha 3 casi favorevoli su 4 totali; cioè è competitivo in almeno tre casi costituiti dai casi Numeri gemelli ricavati da un quadrato perfetto, $N=6k-1$ o da $N=6k+1$ quando però risulta sufficiente una sola formula per fattorizzarlo. Nell'ultimo caso l'algoritmo, dovendo scegliere, prova prima con la formula $(6k-1)(6k-1)$ perché è la colonna 5 ad avere un maggior numero di primi. Sicuramente il caso peggiore è quando delle due formule deve arrivare alla seconda. La ricerca dei fattori è basata sul fatto che se nt è noto, K' si fissa e si calcola K'' ; ciò significa che si incrementa solo K' di 1. Inoltre viene sfruttato il modulo $(nt,6)$, sia per capire la tipologia del numero in gioco, sia per "ridurre più rapidamente il numero", cioè non viene necessariamente provato prima a dividerlo per 2. In una scomposizione in fattori, l'ordine dei fattori non cambia il risultato del prodotto. Una fattorizzazione " **$6k \pm 1$** ", come quella presentata, rispetto ad un algoritmo classico che incrementa di 1 una variabile provandone la sua divisibilità, ha buone probabilità, specie per numeri grandi, di essere più veloce.

L'implementazione dell'algoritmo in linguaggio C è presentata in APPENDICE.

Forme $6k \pm 1$, ultimo atto: una formula precisa per $\pi(N)$

Le forme $6k \pm 1$ danno un ulteriore contributo alla Teoria dei numeri primi; cioè permettono una formula precisa per il calcolo del numero di primi minore ad un fissato N , cioè $\pi(N)$. Vediamo come, iniziando dalle definizioni.

Sia:

- (8) $K_{5p} := \{\max K: 6 \cdot K - 1 < N\}$
 (9) $K_{7p} := \{\max K: 6 \cdot K + 1 < N\}$
 (10) $K_{5c} := \{\text{num. coppie } (K1, K2): (6 \cdot K1 - 1)(6 \cdot K2 + 1) < N\}$
 (11) $K_{7c1} := \{\text{num. coppie } (K1, K2): (6 \cdot K1 - 1)(6 \cdot K2 - 1) < N\}$
 (12) $K_{7c2} := \{\text{num. coppie } (K1, K2): (6 \cdot K1 + 1)(6 \cdot K2 + 1) < N\}$

Per cui è:

$$(13) \quad \pi(N) = K_{5p} + K_{7p} - (K_{5c} + K_{7c1} + K_{7c2}) + 2, \text{ se } N > 3$$

In particolare nella (13) il termine $+ 2$ serve per tener conto dei numeri primi 2 e 3, non compresi nelle forme $6k \pm 1$.

Esempi per validare il metodo di sopra:

N	K_{5p}	K_{7p}	K_{5c}	K_{7c1}	K_{7c2}	+2	$\pi(N)$	Numeri Primi
12	2	1	0	0	0	+2	5	{2,3,5,7,11}
16	2	2	0	0	0	+2	6	{2,3,5,7,11,13}
27	4	4	0	1	0	+2	9	{2,3,5,7,11,13,17,19,23}
36	6	5	1	1	0	+2	11	{2,3,5,7,11,13,17,19,23,29,31}
61	9	10	1	2	1	+2	17	{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59}

E' possibile, quindi, scrivere un algoritmo che riesce, in un numero di passi finiti a trovare $\pi(N)$, qualunque sia N ed in un tempo finito dipendente dalla grandezza di N ; trattasi comunque di un **problema di classe P e non di classe NP**. La tecnica fin qui mostrata fornisce un valore preciso di $\pi(N)$ rispetto al metodo, ad esempio, indicato da Filippo Giordano, dove è richiesto un fattore di correzione o una funzione correttiva, ricadendo prima nel lavoro di Gauss, migliorato con la costante di Legendre e poi dal lavoro del gruppo **ERATOSTENE**.

Proprietà 8

Se $\pi(N)=c$ e $\pi(N+1)=c+1$ allora anche $N+1$ è numero primo.

Se $\pi(N)=c$, $\pi(N+k-1)=c$ e $\pi(N+k)=c+1$ allora anche $N+k$ è numero primo

Algoritmo di calcolo $\pi(N)$

La formula (13) se la interpretiamo in altro modo, ci dice che per il calcolo di $\pi(N)$ le formule (8)(9) non sono sufficienti perché considerano anche i valori composti; ecco quindi la necessità di sottrarre anche i valori delle formule (10)(11)(12). Inoltre occorre tener conto nel conteggio anche del 2 e del 3. In poche parole la (8) (9) sarebbero sufficienti se prendessero in conto tutti i numeri primi compreso il 2 ed il 3.

Esiste un metodo, funzionale matematico, per esprimere il tutto tramite un programma con poche righe di codice? Sì. Significa scrivere un algoritmo funzionale che trova tutti i numeri primi fino a N ; ovviamente servirà definirgli anche cosa sono i numeri primi e cosa sono i divisori di un numero; infine quello che interessa è il numero di elementi trovati. Un algoritmo è presentato in **APPENDICE**.

Vediamo il risultato che si otterrebbe con WinHugs.

Se si carica il programma NTK.hs con WinHugs si ottiene:

```
Main> inspidi 100
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
Main> pidi 100
25
Main> inspidi 1000
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103,
107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211,
223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331,
337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449,
457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587,
593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709,
719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853,
857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991,
997]
Main> pidi 1000
168
Main>
```

E' progettabile, comunque, un algoritmo in C basato sulla (13), che è più veloce di quanto possa fare WinHugs.

WinHugs già con N=10000 ha pessime prestazioni.

Un algoritmo in C tuttavia basato sulle formule (8)(9) richiederebbe almeno un ciclo per ognuna delle formule, conteggiando solo il massimo valore di K; mentre con le formule (10)(11)(12) l'algoritmo richiederebbe per ognuna delle formule due cicli per ricavare i valori di K1 e K2, e facendo attenzione agli arrotondamenti; il che fa aumentare il numero delle istruzioni e peggiorare le prestazioni. In conclusione l'algoritmo più veloce possibile in C è, proprio, quello "brute force", non basato su (13), ma che prevede un solo ciclo che cerca i divisori del numeri e conta i primi.

Vediamo qualche risultato che si otterrebbe col programma in C "brute force":

```
P(100)=25
P(1000)=168
P(10.000)=1229
P(100.000)=9592
P(2.000.000)=148.933
P(3.000.000)=216.816
P(10.000.000)=664.579
P(20.000.000)=1.270.650
```

Sui primi 20 milioni di numeri interi positivi solo circa la 20ma parte sono primi.

Anche qui, comunque, le prestazioni dipendono dalla grandezza di N. Fino a valori di qualche milione è abbastanza rapido. Per i valori di sopra col programma C bastano una ventina di secondi.

Riferimenti

- <http://www.geocities.com/tetractius83> "Perfetto 6, il file nascosto dei numeri primi" Filippo Giordano
- <http://xoomer.alice.it/smarter/matem/matem.htm> Maria Teresa Sica
- <http://www.gruppoeratostene.netandgo.eu/> Gruppo Eratostene
- <http://xoomer.alice.it/stringtheory/Home.html> Michele Nardelli
- <http://www.geocities.com/SiliconValley/Port/3264> R. Turco - Aladdin's Lamp (sezione MISC)

APPENDICE INFORMATICA

Il software è stato sviluppato a fini didattici con compilatore Dev-C++ (free), su un PC a 32bit con Pentium IV.

Eventuali utilizzi diversi da quelli “didattici” devono considerare la sostituzione di determinate operazioni (modulo specialmente) con librerie che consentono di trattare gli interi con un numero elevato di cifre (qualsiasi). L’algoritmo fisico della fattorizzazione considera uno stack di 10 mila elementi interi sufficiente per scomporre numeri abbastanza grandi con almeno 20mila fattori tra primi e composti; ovviamente, in tal caso, il numero di cifre dell’intero devono essere molte e quindi occorre, a maggior ragione, utilizzare le librerie matematiche che superano il vincolo delle architetture hardware a 32/64 bit (lavoro presentato sul sito Aladdin’s Lamp – vedi riferimenti). In questo lavoro ci si è, invece, preoccupati di verificare la validità dell’algoritmo su tecniche $6k \pm 1$.

ALGORITMO DI FATTORIZZAZIONE

```
//
// R. Turco (C) 2008
// Fattorizzazioni con forme  $6n+1$ 
// stackprime.h
//

#ifndef PRIME_STACK
#define PRIME_STACK
static unsigned long int stack[10000];
// prototype
void fattp1(unsigned long int Num);
int vnm1(unsigned long int nt, unsigned long int *V1, unsigned long int *V2);
int vnp1(unsigned long int nt, unsigned long int *V1, unsigned long int *V2);
int quadrp(unsigned long int Num, unsigned long int *V1, unsigned long int *V2);
#endif

#include <stdio.h>
#include <stdlib.h>
#include "stackprime.h"

//
// R. Turco (C) 2008
// Fattorizzazioni con forme  $6n+1$ 
// Main
//

int main(int argc, char *argv[])
{
    unsigned long int Num=0;
    int i = 0;
    printf("\n n*** Fattorizzazione numeri naturali - R.Turco (C) 2008 ***");

    do{

        printf("\n nInserisci numero (0 per uscire): \n");

        scanf("%u",&Num);
        if( Num != 0 ){
            for(i=0; i<=10000; i++){
                stack[i]=0;
            }
            (void) fattp1(Num);
            system("PAUSE");
        }

    }while(Num != 0 );

    return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include "stackprime.h"

//
```

```

// R. Turco (C) 2008 v 1.1
// Fattorizzazione con forme  $6n+1$ 
// Abstract: L'algoritmo gestisce uno stack di 10mila fattori primi o composti
// e cerca anche se esistono quadrati perfetti (numeri gemelli)
//

void fattnp1(unsigned long int Num){
    unsigned long int nt = Num;

    unsigned long int V1 = 1;
    unsigned long int V2 = 1;
    unsigned long int *V3 = 0;
    unsigned long int *V4 = 0;

    int prime=0; /* Non è primo */

    int i=0; // indice dello stack
    long int T = -1; // Valore di confronto
    int quadrPerf=0; // Indicatore del quadrato perfetto

    printf("\n");

    if( nt < 12){
        printf("\nNumero < 12");
        printf("\nfatt-> %u \n", nt);
        printf("\n");
        system("PAUSE");
        exit(0);
    }

    V3=(unsigned long int*)malloc(sizeof(V1));
    V4=(unsigned long int*)malloc(sizeof(V2));

    do{

        prime=0;
        quadrPerf=0;

        T=nt%6;

        printf("Fattorizzo #: %u \n", nt);

        // Prova del quadrato perfetto

        quadrPerf=quadrpGen(nt, V3, V4);
        if( quadrPerf == 1 ){
            // Quadrato perfetto
            printf("\nQuadrato perfetto: %u %u \n", *V3, *V4);
            T=9; // passo per la zona di comodo dei q p
            nt = *V3;
            system("PAUSE");
        }

        // Prova del quadrato perfetto

        switch(T){
            case 0:
            case 2:
            case 4:
            case 8:
                T=2; // devo dividere per 2
                nt = nt/T;
                printf("\nfatt-> %u *\n", T);
                break;

            case 1:
            case 5:
            case 9: // il caso 9 è di comodo per i quadrati perfetti
                if( T == 1 ){
                    // Occorrerà verificare fattorizzazione di due numeri
                    // V1 e V2;
                    // Ricerca dei fattori k1 e k2
                    // su due equivalenze se serve  $(6n-1)(6n-1)$  o  $(6n+1)(6n+1)$ 
                    prime = vnp1(nt,V3,V4);
                }
        }
    } while (nt > 1);
}

```

```

}

if( T == 5){
// Occorrerà verificare fattorizzazione di due numeri
// V1 e V2;
// Ricerca dei fattori k1 e k2
// su una sola equivalenza (6n-1)(6n+1)
prime = vnm1(nt,V3,V4);
}

V1=*V3;
V2=*V4;

if(prime){ // è primo
printf("\ nfatt-> %u *\ n", nt);
nt=0;
i--;
if( i < 0 ) i=0;
if( stack[i] != 0 ){
// Controllo lo stack
nt = stack[i];
stack[i] = 0;
V2=0;
}
}

// Se ho V1 e V2, allora V2 va nello stack
if((V2>1) && (!prime)){
// Inserisco nello stack
stack[i]=V2;
i++;
}
// Se V1 già lo posso escludere dai successivi
// controlli, allora prendo un valore dallo stack
//
if((V1 < 12) && (V1>0) && (!prime)){
printf("\ nfatt-> %u *\ n", V1);
V1=1; /* per uscire se lo stack fosse vuoto*/
i--; // ritorno alla posizione precedente dello stack
if( i < 0 ) i=0;
if( stack[i] != 0 ){
nt=stack[i]; /* Prendo V2 e svuoto lo stack*/
stack[i]=0;
V2=0;
if( (nt < 12) && quadrPerf ) {
printf("\ nfatt-> %u *\ n", nt);
nt=1;
}
}
}

// adesso mi chiedo se il valore dello stack è da processare
// oppure ho finito già (i=0 e nt=0), visto che V1=1
//

if((i==0) && (V1==1) && (nt==0)){ // Ho finito V1 e lo stack nt==0 o < 12
printf("\ nfatt-> %u *\ n", nt);
nt=0;
i--;
if( i < 0 ) i=0;
if( stack[i] != 0 ){
nt=stack[i]; /* Prendo V2 e svuoto lo stack*/
stack[i]=0;
V2=0;
}
}
}
else{
if( (V1 > 1) && (!prime)){
nt=V1;
}
}
}
if( (V2 == 0) && (V1 == 1) && i<2){

```

```

        i--;
        if( i < 0 ) i=0;
        if( stack[i] != 0 ){
            nt=stack[i]; /* Prendo V2 e svuoto lo stack*/
            stack[i]=0;
            V2=0;
        }
    }
    break;
case 3:
    nt = nt/T;
    printf("\ nfatt-> %u *\ n", T);
    break;

default:
    nt = nt/T;
    printf("\ nfatt-> %u *\ n", T);
    break;
}
T = -1; // Lo setto ad un valore impossibile per il contesto
}while(nt >= 12); // Se c'è qualcosa nello stack devo ciclare

if( (nt < 12) && (nt>1) ){
    printf("\ nfatt-> %u *\ n", nt);
}
if( V3!= NULL)
    free(V3);
if( V4!= NULL)
    free(V4);
return;
}

#include <stdio.h>
#include <stdlib.h>

// fattnp1: gestisce situazioni mod 5
// fattnm1: gestisce situazioni mod 1
//
// Abstract:
// Indipendentemente dalla formula per mod 5 o mod 1
// se nt è noto, fissato K1 si calcola K2
// se il valore T temporaneo calcolato è minore di nt si aumenta K1
// e si ricalcola T (ciclo)
// se il valore T temporaneo calcolato è maggiore di nt è inutile
// proseguire
// altrimenti T=nt quindi sono stati trovati K1 e K2
//
// return value: VA e VB come puntaori
// return status: 1 se primo, 0 se non primo
//
//

int vnm1(unsigned long int nt,unsigned long int *VA, unsigned long int *VB){
    // Restituisce status = 1 se primo, 0 se non lo è
    // Caso modulo 5
    unsigned long int K1=0;
    unsigned long int K2=0;
    unsigned long int T1=0;
    unsigned long int V1=0;
    unsigned long int V2=0;
    unsigned long int A1=0;

    int fex=0;
    int status=0;

    K1=1;
    K2=1;
    if( nt==1 ) { /* Non va bene la formula: è primo */
        *VA=0;
        *VB=nt;
        return 1;
    }
}

```

```

A1 = (6*K1-1)*(6*K2+1);
if( A1 > nt ) { /* Non va bene la formula: è primo */
    *VA=0;
    *VB=nt;
    return 1;
}

printf("\ nMod 5 - Ciclo (6n-1)(6n+1)\ n");
do{

    K2 = ((nt+1)/6 - K1)/(6*K1-1);
    if( K2 == 0 ){
        fex=1; // E' primo
    }
    T1 = (6*K1-1)*(6*K2+1);
    if(T1 == nt){
        fex=1; // L'ho trovato
    }
    if( T1 < nt ){
        K1++;
    }

    if( T1 > nt ){
        fex=1;
    }

}while((nt != T1) && (fex==0));

if( nt == T1 ){
    /* E' un composto */
    V1=6*K1-1;
    V2=6*K2+1;

    if( K2 == 0 ){
        V2=0;
    }
    status=0;
}
else{
    /* E' un primo */
    V1=0;
    V2=0;
    status=1;
}
*VA=V1;
*VB=V2;
return status;
}

int vnp1(unsigned long int nt,unsigned long int *VA, unsigned long int *VB){
    // Restituisce status = 1 se primo, 0 se non lo è
    // Mod 1
    unsigned long int K1=0;
    unsigned long int K2=0;

    unsigned long int T1=0;
    unsigned long int T2=0;
    unsigned long int A1=0;
    unsigned long int A2=0;

    unsigned long int V1=0;
    unsigned long int V2=0;
    int fex=0;
    int status=0;

    K1=1;
    K2=1;

    if(nt==1){ /* Non vanno bene le formule: è primo */
        *VA=0;
        *VB=nt;
        return 1;
    }

```

```

A1 = (6*K1-1)*(6*K2-1);
A2 = (6*K1+1)*(6*K2+1);

if((A1>nt) && (A2>nt)){ /* Non vanno bene le formule: è primo */
    *VA=0;
    *VB=nt;
    return 1;
}

if( A1 <= nt ){ /* Va provata la formula */
printf("\ nMod 1 - Ciclo (6n-1)(6n-1)\ n");

do{

    K2 = ((nt-1)/6 + K1)/(6*K1-1);
    if( K2 == 0 ){
        fex=1; // E' primo
    }

    T1 = (6*K1-1)*(6*K2-1);
    if(T1 == nt){
        fex=1; // L'ho trovato
    }
    if( T1 < nt ){
        K1++; // Devo cercare il valore di K1
    }
    if( T1 > nt ){
        fex=1; // E' inutile proseguire
    }
}while((nt != T1) && (fex==0));
}

if( (A2 <= nt) && (nt != T1) ){ /* Va provata la formula */
    K1=1;
    K2=1;
    fex=0;

    printf("\ nMod 1 - Ciclo (6n+1)(6n+1)\ n");

do{

    K2 = ((nt-1)/6 - K1)/(6*K1+1);
    if( K2 == 0 ){
        fex=1; // E' primo
    }
    T2 = (6*K1+1)*(6*K2+1);
    if(T2 == nt){
        fex=1; // L'ho trovato
    }

    if( T2 < nt ){
        K1++;
    }
    if( T2 > nt ){
        fex=1;
    }
}while((nt != T2) &&(fex==0));
}
if( (nt==T1) || (nt==T2) ){

/* E' un composto */
if( nt==T1 ){
    V1=(6*K1-1);
    V2=(6*K2-1);

    if( K2 == 0 ){
        V2=0;
    }
}
if( nt==T2 ){

    V1=(6*K1+1);
    V2=(6*K2+1);
}
}

```

```

    if( K2 == 0 ){
        V2=0;
    }
    status=0;
}
else{ /* E' un primo */
    V1=0;
    V2=0;
    status = 1;
}

*VA=V1;
*VB=V2;
return status;
}

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int quadrp(unsigned long int Num, unsigned long int *VA, unsigned long int *VB){
    // R.Turco (C) 2008
    // restituisce 0 se non è un quadrato perfetto
    // restituisce 1 se lo è con i valori gemelli V1 e V2
    //
    int status=0; // per default non è un quadrato perfetto
    double x=0.0;
    double y=0.0;
    double z=0.0;
    double *w;

    w= (double*)malloc(sizeof(double));
    x=Num+1; // P=Num=Q - 1 x=Q=P+1=Num+1
    y=sqrt(x);
    z=modf(y,w);

    if( z == 0.0 ){ // controllo la parte dopo la virgola quella frazionaria
        *VA= (int) y - 1;
        *VB= (int) y + 1;
        status = 1; // Quadrato perfetto
    }
    else{ // Nessun quadrato perfetto
        *VA=0;
        *VB=0;
        status = 0;
    }
    if( w!= NULL) {
        free(w);
    }
    return status;
}

```

ALGORITMO PER $\pi(N)$ – LINGUAGGIO FUNZIONALE HASKELL (WINHUGS)

```

-----
--
-- Package   : NT - Numbers' Theory
-- Module    : NTK.hs
-- Copyright : (c) 2007 by Rosario Turco
--
-- Maintainer : Rosario Turco
-- Stability  : developing
-- Portability : none
--
-- Copyright 2007 by Rosario Turco. All rights reserved.
-- Redistribution and use in source and binary forms,
-- with or without modification, are not permitted.
--
-----
--
-- Divisibilità di un numero
--

```

```

divide k n = mod n k == 0
--
-- Fattori di un numero
--
factors n = [k | k <- [1..n], divide k n]
--
-- Numeri primi
--
isprime n = factors n == [1,n]
--
-- Lista di primi fino a n
--
listprime n = [k | k <- [2..n], isprime k]
--
-- Lista di composti
--
listcomp n = [k | k <- [2..n-1], not (isprime k)]
--
-- P di N
--
inspidi n = [y | y <- [2..n-1], isprime y]
pidi n = length(inspidi n)

```

ALGORITMO PER $\pi(N)$ – LINGUAGGIO C

```

#include <stdio.h>
#include <stdlib.h>
//
// R. Turco (C) 2008
// Calcolo veloce P(n)
// Main
//

// prototype

int main(int argc, char *argv[])
{
    unsigned long int Num=0;
    unsigned long int pdn=0;

    printf("\nCalcolo Pi(N) - R. Turco (C) 2008");

    do{
        printf("\n\nInserisci numero (0 per uscire): \n");
        scanf("%d",&Num);
        if( Num != 0 ){
            pdn = CountPidiN(Num);
            printf("\n\nPi(%d) = %d\n\n", Num, pdn);
            system("PAUSE");
        }
    }while(Num != 0 );

    return 0;
}

#include "prototype.h"
unsigned long int CountPidiN(unsigned long int iValue){
    /* Conta il numero di primi a partire da 2 e minori di iValue */

    int iCounter=0, i=0;
    for(i=2; i<iValue; i++){
        if( prime(i,0) != 0 ){
            iCounter++;
        }
    }
    return (iCounter);
}

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

unsigned long int prime(unsigned long int iVal, int iDeb)

```

```

{
int i=0, iCount=0;
unsigned long int iSq=0, iPri=0;

/* iCount conta i Divisori */

float fVal = iVal;

/* Se 0,1,2,3 esco rapidamente */

switch(iVal){
case 0:
case 1:
return iPri;
break;
case 2:
case 3:
return 1;
break;
default:
break;
}

iSq = sqrt(fVal);

if( iDeb > 0 ){
printf("\n sqrt : %d\n", iSq);
}
if( iVal%2 == 0){
iCount++;
}
else{
/* Cerco i divisori con il modulo escludendo l'1 e il 2*/
for(i=3;i<=iSq;i=i+2){ /* Cerco i divisori dispari */
if( iVal%i == 0) {
iCount++; /* Esiste un divisore, allora non è primo */
break; /* Interrompo al primo Divisore trovato */
}
}
}
if( iCount == 0 ) iPri=1; /* Se iCount = 0 allora è primo */
return iPri;
}

// Prototype.h
#if !defined prototype_prime
#define prototype_prime
int prime(unsigned long int iVal, int iDeb);
unsigned long int CountPidiN(unsigned long int iValue);
#endif

```